L'impact des outils utilisés pour l'apprentissage de la programmation et le développement de la pensée informatique au cycle 3

Kevin Sigayret

THÈSE DE DOCTORAT

L'IMPACT DES OUTILS UTILISÉS POUR L'APPRENTISSAGE DE LA PROGRAMMATION ET LE DÉVELOPPEMENT DE LA PENSÉE INFORMATIQUE AU CYCLE 3.

Présentée en vue de l'obtention du grade de docteur
en **Psychologie Cognitive** de **l'Université Paul-Valéry Montpellier 3**

Ecole Doctorale n°60 – Territoires, Temps, Société, Développement (TTSD)

Présentée par **Kevin SIGAYRET**

Co-dirigée par **Nathalie BLANC** et **André TRICOT**

Soutenue publiquement **le 01 DÉCEMBRE 2023** devant le jury composé de

Mireille BÉTRANCOURT Professeure des universités, Université de Genève	Rapporteure
Emmanuel SANDER Professeur des universités, Université de Genève	Rapporteur
Mônica MACEDO-ROUET Professeure des universités, CY Cergy Paris Université	Examinatrice
Pierre TCHOUNIKINE Professeur des universités, Université Grenoble-Alpes	Examinateur
Nathalie BLANC Professeur des universités, Université Paul-Valéry Montpellier 3	Co-directrice de thèse
André TRICOT Professeur des universités, Université Paul-Valéry Montpellier 3	Co-directeur de thèse

Remerciements

A l'heure où j'écris ces remerciements, mes premières pensées se portent bien évidemment sur mes deux directeurs de thèse André Tricot et Nathalie Blanc. Je savais en commençant ce travail de longue haleine que j'avais le privilège d'être encadré par deux chercheurs très compétents dans leur domaine, mais je pouvais ressentir un peu d'appréhension à l'idée de collaborer pendant trois ans (minimum) avec deux personnes que je ne connaissais pas encore très bien. C'est avec beaucoup de plaisir que j'ai pu découvrir vos qualités humaines, qui m'ont été indispensables pour mener à bien ce travail. Je mesure aujourd'hui la chance d'avoir bénéficié de ce soutien bienveillant et constant tout au long de ces quatre années. Vos conseils nombreux et avisés ont toujours été donnés avec beaucoup de sincérité et de gentillesse, sans aucune pression ou jugement, et c'est aussi en partie grâce à vous que j'ai pu traverser certaines épreuves. Merci infiniment d'avoir été, et d'être encore, les chercheurs et les personnes que vous êtes. Je souhaite à tous les doctorants d'être accompagnés comme je l'ai été.

Je tiens également à remercier les membres de mon jury : Mireille Bétrancourt, Emmanuel Sander, Mônica Macedo-Rouet et Pierre Tchounikine. C'est un honneur pour moi que vous ayez accepté d'expertiser ce travail et j'ai hâte de pouvoir échanger avec vous au cours de cette soutenance à venir qui ne pourra être qu'enrichissante. J'ai également eu la chance d'échanger avec Pierre Tchounikine et Emmanuel Sander pendant cette thèse, et je vous remercie des conseils et commentaires que vous avez pu faire sur mon travail, qui m'ont permis de progresser dans ma réflexion.

Je remercie bien sûr les personnes qui sont à l'origine du financement de cette thèse, sans lequel mes conditions de travail auraient été bien moins confortables, en particulier ceux qui ont porté le programme des groupes thématiques numériques (GTNum), Elie Allouche et Axel Jean, ainsi que les rectrices de l'Académie de Montpellier Béatrice Gille et Sophie Béjean,. Merci aussi à tous les membres de la DRANE avec qui j'ai eu le plaisir d'échanger pendant ces quatre années, en particulier Sabrina Caliaros, Sébastien Méjean, Ludovic Delorme, Régis Cazorla et Elodie Camo. Nos discussions m'ont permis de mieux comprendre les usages du numérique dans les classes et ont été essentielles pour m'aider à choisir un sujet qui soit à la fois intéressant pour moi et pertinent pour le développement des pratiques

éducatives. Je remercie tout particulièrement Sébastien Méjean pour le temps qu'il a consacré au recrutement des classes qui ont participé à nos expériences, au même titre qu'Elodie Camo, qui a par ailleurs été d'une aide indispensable pour la conception des séances mises en place dans les classes. Un grand merci évidemment à tous les enseignants qui ont bien voulu s'intéresser à nos recherches et qui ont accepté de mettre en œuvre nos expériences dans leur classe. Votre intérêt et votre motivation ont constitué un maillon fondamental de l'aboutissement de cette thèse.

Merci également aux membres de mon CSIT : Arielle Syssau Vaccarella, Sophie Bayard et Margarida Romero. Votre accompagnement a marqué chaque année d'une étape importante. Vos observations bienveillantes et pertinentes sur l'avancée de mon travail m'ont rassuré et encouragé à continuer sur cette voie. Je n'oublie pas non plus de remercier les enseignants-chercheurs, du laboratoire Epsylon et d'ailleurs, avec qui j'ai pu échanger régulièrement. Je pense en particulier aux personnes qui m'ont accompagnées dans ma volonté d'enseigner à l'université : Pascale Maury, Stéphanie Bellochi, Royce Anders, Pom Charras et Catherine Monnier. Un grand merci à Dan Priolo qui m'a été d'une aide précieuse pour mieux comprendre les modèles mixtes que je voulais exploiter pour analyser mes résultats. Merci bien sûr à Jean-Marc Lavaur qui m'a donné le goût de la recherche depuis la troisième année de Licence jusqu'à la fin du Master. Nos échanges agréables et enrichissants ne sont pas étrangers à ma volonté de poursuivre en thèse. Merci également à Stéphanie Roussel pour l'aide apportée lors de l'adaptation de l'échelle dédiée à la mesure de la charge cognitive. Enfin, merci à Lionel Brunel qui a accompagné et soutenu le développement de mes compétences de chercheurs, en étant présent dans la totalité de mes jurys de mémoire.

Je tiens bien évidemment à remercier l'ensemble de mes collègues doctorants pour les échanges professionnels, amicaux (et festifs) que j'ai pu avoir avec vous. Certains d'entre vous sont depuis devenus des amis destinés, je l'espère, à rester dans ma vie bien après l'achèvement de cette thèse. Je pense bien sûr à Lisa Sanchez, ma 2Li, la plus belle rencontre de ce doctorat. Ta loyauté, ta fidélité, ton humour et ton amitié m'ont souvent permis de tenir le cap, malgré la tempête. Hâte de vivre le prochain « week-end de foufous » avec toi. Merci également à Louis Bourgaux, petit frère de thèse, chaque discussion avec toi nourrit mon esprit (et parfois aussi mes compétences en stats) et ta bienveillance a été un appui très important pour moi ces derniers temps. Merci bien sûr à Priscillia Improvisato (« Pressiya »), je ne vais pas m'épancher sur tes nombreuses qualités, je vais faire comme toi et aller à l'essentiel, tu es géniale ! Et en plus, tu ne te vexes jamais, c'est trop bien. Merci à Philippe Servajean pour ces débats passionnants et

ta touche d'humour très personnelle. Sache que je ne t'en veux pas, même si tu te trompes sur la conscience. Merci à Hélène Buche pour ces « mitraillettes » partagés ensemble et, plus globalement, pour nos multiples discussions drôles ou émouvantes. Merci à Sarah Ottavi pour nos échanges pleins de bienveillance et de respect mutuel (non pas du tout, mais on a bien rigolé quand même). Merci à mon José Samaniego pour ton humeur positive et communicative dont j'ai pu bénéficier à chacune de tes « pauses pommes ». Merci à Florian Lecêtre pour ces multiples soirées plus ou moins arrosées (je n'en dirai pas plus, mais tu sais bien à laquelle je pense). Merci à Christine Sanchez, malgré la défaite que tu m'as fait subir à Dune : Imperium (je finis par la reconnaître…), je me consolerai en pensant à toi et à ce portail du M.O.C.O. Merci à Chloé Galli pour ta bonne humeur et ton vocabulaire moyenâgeux qui m'ont replongé maintes fois dans une époque passée. Merci à Anne Ray pour nos échanges trop rares, mais instructifs et toujours plein de légèreté et d'humour. Un grand merci à Méryl Donadey ! Et merci aussi aux doctorants que je n'ai pas cités : Clarisse, Dhia, Jasmine, Jérôme, Justine, Lola, Lorene, Lucie, Lynda, Marie-Julie, Mathieu, Mikael et tous les autres.

Merci aussi à tout le groupe des « doctorants d'André » : Matisse, Louis, Lucille, Lydia, Alexia et Sarah pour ces journées de travail studieuses, mais agréables et détendues, ponctuées du traditionnel repas au Broc. C'était un plaisir de vous connaître et de partager nos galères et nos succès de thèse. En particulier, je remercie chaleureusement Sarah Pariser, pour son énergie positive et ses encouragements constants. Ce voyage en Grèce aurait été bien moins agréable et amusant sans toi. J'espère qu'on pourra célébrer bientôt l'aboutissement de nos deux thèses ensemble !

Je remercie aussi tous mes amis qui m'ont apporté de la joie, du réconfort et des rires au cours de ces quatre années. Ils se reconnaîtront, mais impossible de ne pas te citer toi Laura, ma 2Low, pour le soutien mutuel et l'amitié sincère que nous avons développés cette dernière année. Je tiens également à remercier Clotilde pour tous ce qu'elle m'a apporté au cours des premières années de cette thèse. Je n'oublierai pas ce que tu as pu faire pour moi et les moments de complicité que nous avons pu vivre ensemble. Merci également à ma famille, en particulier à mon père et à mon frère. Votre présence dans ma vie suffit à m'apporter la joie et l'appui nécessaire pour continuer à avancer. Enfin, je ne peux retenir une pensée émue pour ma mère. Tes nombreuses qualités font de toi un exemple à suivre pour le reste de ma vie. Je conserve l'espoir que, là ou tu es, tu pourras être fière et heureuse de mon parcours.

RÉSUMÉ

L'enseignement de la programmation informatique a intégré progressivement les cursus scolaires européens. L'un des objectifs majeurs de cette discipline serait de permettre de développer la pensée informatique, dont la définition ne fait pas consensus, mais qui englobe certains processus cognitifs impliqués dans la résolution de problèmes, en s'appuyant sur des concepts issus de l'informatique. Différents outils sont disponibles pour enseigner la programmation et la pensée informatique, notamment les langages visuels par blocs, la robotique pédagogique ou les activités débranchées n'impliquant aucun outil numérique. Cependant, malgré les arguments théoriques pertinents en faveur de l'un ou l'autre de ces outils, le manque de données empiriques comparatives ne permet pas aujourd'hui de conclure quant à leur efficacité relative. Pour cette raison, nous avons réalisé quatre expériences, en contexte écologique, auprès d'élèves de niveau CM2, novices en programmation, afin de mesurer l'impact de ces outils sur les performances d'apprentissage des élèves, ainsi que sur leur motivation, leur sentiment d'auto-efficacité et leur attitude vis-à-vis des sciences.

Pour évaluer les connaissances et compétences liées à l'enseignement de la programmation et de la pensée informatique, nous avons conçu et validé deux nouveaux outils de mesure, centrés sur la maîtrise des concepts computationnels et la capacité à résoudre des problèmes algorithmiques. Notre première expérience a révélé de meilleures performances lorsque les élèves utilisent le langage de programmation visuel Scratch, sans robot pédagogique. Par conséquent, nos trois expériences suivantes ont tenté d'expliquer cet effet. Nous n'avons pas réussi à mettre en lumière l'effet bénéfique du feedback permis par les outils numériques sur les performances des élèves. Cependant, nos résultats signalent une augmentation de la charge cognitive extrinsèque lorsque le logiciel Scratch est utilisé en association avec un robot Thymio, ce qui pourrait expliquer les moins bonnes performances des élèves dans cette condition. Les robots produisent toutefois un effet positif fort sur la motivation. Quelques différences ont pu être observées selon le genre, en particulier un plus haut niveau de motivation des garçons par rapport aux filles, principalement lorsque ceux-ci utilisent le logiciel Scratch sans robot.

Mots-clés : programmation, pensée informatique, langages visuels par blocs, robotique pédagogique, activités débranchées, feedback, charge cognitive.

ABSTRACT

Computer programming has gradually integrated European school curricula. One of the major aims of this discipline is to develop computational thinking, which is not defined by consensus, but which encompasses certain cognitive processes involved in problem solving, based on concepts derived from computer science. Various tools are available for teaching programming and computational thinking, including block-based visual languages, educational robotics and unplugged activities involving no digital tool. However, despite the relevant theoretical arguments in favour of one or other of these tools, the lack of comparative empirical data means that it is not possible today to conclude on their relative effectiveness. For this reason, we carried out four experiments, in an ecological context, with grade 5 students, novices in programming, to measure the impact of these tools on learning performance, as well as on their motivation, self-efficacy and attitude towards science.

To assess knowledge and skills involved in teaching programming and computational thinking, we designed and validated two new measurement tools, focusing on mastery of computational concepts and the ability to solve algorithmic problems. Our first experiment revealed better performance when students used the visual programming language Scratch, without an educational robot. Consequently, our next three experiments attempted to explain this effect. We were unable to highlight the beneficial effect of feedback enabled by digital tools on students' performance. However, our results indicate an increase in extrinsic cognitive load when Scratch software is used in conjunction with a Thymio robot, which could explain the lower performance of students in this condition. Robots do, however, have a strong positive effect on motivation. Some gender differences could be observed, in particular boys' higher level of motivation compared to girls, mainly when using the Scratch software without a robot.

Keywords: programming, computational thinking, block-based visual languages, educational robotics, unplugged activities, feedback, cognitive load.

Title: The impact of tools used to teach programming and developing computational thinking at the end of primary school.

TABLE DES MATIÈRES

INTRODUCTION GÉNÉRALE 9

I) PARTIE THEORIQUE 18

CHAPITRE 1. L'APPRENTISSAGE DE LA PROGRAMMATION 18

1.1) Qu'est-ce que la programmation ? 18

1.2) Apprendre à programmer : un véritable défi en contexte scolaire ? 20

1.3) Pourquoi apprendre la programmation ? 23

CHAPITRE 2. LE DEVELOPPEMENT DE LA PENSEE INFORMATIQUE 30

2.1) Plusieurs conceptions de la pensée informatique 30

2.2) Pensée informatique, programmation et mathématiques 38

CHAPITRE 3. DIFFERENTS OUTILS POUR ENSEIGNER LA PROGRAMMATION 44

3.1) Enseigner la programmation à l'école 44

3.2) Les activités de programmation débranchées 48

3.3) Logiciels de programmation éducatifs et langages visuels par blocs 55

3.4) La robotique éducative 65

CHAPITRE 4. PSYCHOLOGIE ET APPRENTISSAGE DE LA PROGRAMMATION 75

4.1) Des connaissances et des processus cognitifs à maîtriser 75

4.2) La charge cognitive d'une situation d'apprentissage 79

4.3) Différentes implications du corps et de la motricité 82

4.4) Quel impact sur le vécu subjectif des élèves ? 84

4.5) Le paradoxe préférences / performances 88

4.6) L'impact du genre des élèves sur les apprentissages 90

CHAPITRE 5. PROBLEMATIQUE ET HYPOTHESES 94

5.1) Problématique 94

5.2) Hypothèses 96

II) PARTIE EXPÉRIMENTALE 99

CHAPITRE 1. NOUVEAUX OUTILS D'EVALUATION DE LA PENSEE INFORMATIQUE 99

1.1) Introduction 99

1.2) Test de Maîtrise des Concepts Computationnels 107
1.2.1) Méthode 108
1.2.2) Résultats 113
1.3) Test d'Algorithmique 119
1.3.1) Méthode (version 1 du test) 120
1.3.2) Résultats (version 1 du test) 123
1.3.3) Méthode (version 2 du test) 126
1.3.4) Résultats (version 2 du test) 130
1.4) Discussion des tests 133
CHAPITRE 2. COMPARAISON DE TROIS OUTILS D'APPRENTISSAGE 141
2.1) Introduction de l'expérience 1 141
2.2) Méthode de l'expérience 1 147
2.3) Résultats de expérience 1 154
2.4) Discussion de l'expérience 1 172
CHAPITRE 3. APPROFONDISSEMENTS 180
3.1) Introduction des expériences 2, 3 et 4 180
3.2) Expérience 2 188
3.2.1) Méthode de l'expérience 2 188
3.2.2) Résultats de l'expérience 2 191
3.3) Expérience 3 196
3.3.1) Méthode de l'expérience 3 197
3.3.2) Résultats de l'expérience 3 199
3.4) Expérience 4 204
3.4.1) Méthode de l'expérience 4 204
3.4.2) Résultats de l'expérience 4 207
3.5) Discussion des expériences 2, 3 et 4 210
DISCUSSION GÉNÉRALE 217
CONCLUSION 244
BIBLIOGRAPHIE 245
ANNEXES 280

Note liminaire

Certains passages qui constituent cette thèse sont directement issus de deux articles que nous avons publiés au cours de la thèse :

- Sigayret, K., Tricot, A., & Blanc, N. (2022). Unplugged or plugged-in programming learning: A comparative experimental study. *Computers & Education*, *184*, 104505. https://doi.org/10.1016/j.compedu.2022.104505.
- Sigayret, K., Blanc, N., & Tricot, A. (2022). L'apprentissage de la programmation: quels outils pour évaluer le développement de la pensée informatique à l'école?. *Enfance*, *4*(4), 479-500. https://doi.org/10.3917/enf2.224.0479.

Les passages en question sont traduits en français le cas échéant, placés entre crochets ([…]) et renvoient vers une note de bas de page indiquant lequel des deux articles a été cité. Ces articles sont présents en annexes, dans leur intégralité (Annexes A et B).

INTRODUCTION GÉNÉRALE

Enjeux actuels du numérique éducatif

Le terme « numérique », qui est aujourd'hui largement utilisé dans le domaine éducatif, s'est progressivement construit comme équivalent ou euphémisation de ce qui était autrefois désigné sous le terme plus générique d' « informatique » (Baron & Boulc'h, 2012). Par ailleurs, cette dénomination est traditionnellement associée à l'arithmétique et à la numération. Ici, elle met en évidence la notion de « numérisation », c'est-à-dire globalement de dématérialisation des ressources qui peuvent dorénavant être facilement échangées (Baron, 2014). Le numérique éducatif regroupe ainsi un grand nombre d'outils et de pratiques disparates qui s'appuient sur les technologies de l'informatique pour tenter de faciliter l'apprentissage ou l'enseignement. Avec la multiplication des écrans survenue ces dernières années, de nombreux mythes ou idées reçues se sont propagés sur la révolution numérique et sa capacité à bouleverser notre façon d'apprendre, à l'école ou ailleurs. Souvent, l'impact positif des outils est surestimé. Parmi ces mythes, on retrouve par exemple l'idée que le contact des écrans puisse transformer notre cerveau et notre façon d'apprendre. Or, aucune modification des structures cérébrales ou des processus mentaux n'a pu être mise en évidence. En réalité, les écrans ne transforment pas notre cerveau, mais notre accès et notre rapport à la connaissance (Gros et al., 2018).

Face à un véritable foisonnement des usages du numérique à des fins pédagogiques, les tentatives pour les classer et les analyser sont multiples. Bibeau (2005) propose par exemple une taxonomie qui distingue six catégories de ressources numériques pour l'éducation alors qu'une autre typologie isole huit différentes fonctions pédagogiques qui peuvent être attribuées à des logiciels éducatifs (De Vries, 2001). Les caractéristiques de ces fonctions sont détaillées par De Vries (2001) tout comme les théories psychologiques de l'apprentissage sur lesquelles ces fonctions prennent appui (behaviorisme, cognitivisme, constructivisme ou encore cognition incarnée et située). De Vries (2001) propose une approche qui consiste à associer un outil à une fonction pédagogique particulière et à une tâche. Tricot (2020) recense 24 fonctions pédagogiques principales pour lesquelles il convient d'étudier en quoi le numérique peut représenter un apport pour l'enseignement et l'apprentissage.

Au-delà des typologies qui peuvent être multiples – on en recensait déjà près d'une trentaine au début des années 2000 (Basque & Lundgren-Cayrol, 2003) – l'important reste de classer les technologies en fonction de leur positionnement par rapport à de grandes oppositions (Fluckiger, 2020). S'agit-il d'une technologie qui facilite l'apprentissage ou qui facilite l'enseignement ? A-t-elle été élaborée initialement pour servir dans un contexte scolaire ou l'école se l'est-elle appropriée par la suite ? La technologie est-elle l'objet même de l'apprentissage ou n'est-elle qu'un outil pour y parvenir ?

Dans le cadre de cette thèse, nous nous intéresserons particulièrement aux outils numériques censés faciliter l'apprentissage de certaines notions ou compétences. Selon Bloom (1979), un enseignement « efficace » peut comprendre plusieurs objectifs distincts : « une élévation de la moyenne de l'ensemble des résultats ; une réduction de la variance de l'ensemble des résultats ; une diminution de la corrélation entre l'origine sociale de chaque élève (et plus généralement ses caractéristiques initiales) et ses résultats ». Pour Bernard et al. (2018), les outils efficaces soutiennent et intensifient l'interaction entre l'élève et le contenu à apprendre.

Plus que la nature même de ces ressources et technologies numériques, les usages effectifs en classe peuvent présenter un intérêt certain pour initier la réflexion sur l'impact des outils numériques sur les apprentissages. Les deux tiers des professeurs des écoles estiment que la démocratisation du numérique et des ordinateurs a bouleversé leurs pratiques pédagogiques (Ravestein & Ladage, 2014). Le baromètre des usages du numérique éducatif (Gili, 2019), qui recense et analyse les usages de plus de 3000 enseignants de l'Académie de Montpellier, nous informe par ailleurs que le recours au numérique est particulièrement fréquent pour les fondamentaux du métier (plus de 90% des enseignants sondés l'utilisent fréquemment pour préparer les cours ou gérer la classe, plus de 70% pour animer les séances). En revanche, l'usage du numérique est nettement moindre en ce qui concerne les pratiques pédagogiques (faciliter la production écrite ou orale des élèves, faciliter le travail en groupe ou en autonomie, évaluer la production des élèves, etc.). Pour chacune des pratiques testées, moins d'un tiers des enseignants déclarait s'appuyer fréquemment sur des outils numériques (Gili, 2019).

Cette intégration plus difficile du numérique éducatif dans les pratiques pédagogiques s'explique peut-être en partie par l'abondance des ressources mises à disposition. Que celles-ci soient institutionnelles, personnelles ou issues d'échanges entre pairs, la prolifération des ressources numériques, loin de faciliter la tâche de l'enseignant, semble au contraire en accroître la difficulté (Fluckiger, 2020). Dans cet océan d'informations, il est nécessaire pour le

professeur de s'assurer de la fiabilité de ces sources ainsi que de leur adaptation au niveau des élèves ou de leur correcte intégration dans une séquence d'enseignement. Autant de difficultés qui peuvent rendre chronophage l'utilisation du numérique dans ce domaine. Cette recherche légitime de fiabilité et d'efficacité pousse notamment les enseignants à se tourner en priorité vers les manuels scolaires qui constituent un point d'appui non négligeable de par leur rôle d'opérationnalisation des instructions officielles (Choppin, 1992).

Pour Grugier (2016), la technologie numérique comme outil pédagogique ne saurait être dissociable de l'enseignement de contenus informatiques. Même en maternelle, la rencontre des enfants avec le tableau numérique interactif (TNI) leur permet déjà d'acquérir un vocabulaire spécifique et leurs premières connaissances techniques. Les outils technologiques éducatifs n'ont pas vocation à être uniquement des outils collectifs au service de la classe dans son ensemble et indissociables de celle-ci, tels que le TNI. Les équipements individuels et mobiles (ordinateurs portables, tablettes tactiles ou mêmes smartphones) constituent une autre catégorie d'équipement dont les usages diffèrent. En France, de nombreux collèges et lycées ont fait l'objet de campagnes massives visant à garantir à chaque élève un ordinateur individuel et personnel, susceptible d'être utilisé dans toutes les disciplines (Khaneboubi, 2009). D'autres dispositifs similaires ont été testés, les « classes mobiles » par exemple, à la différence que les machines ne sont cette fois plus individuelles, mais partagées entre les élèves d'une ou de plusieurs classes.

Limites du numérique éducatif

L'étude de l'opération « *One Laptop Per Child* » (« Un ordinateur portable par enfant ») qui a fourni 15 000 ordinateurs portables à des élèves d'école élémentaire aux États-Unis se révèle assez décevante (Warschauer et al., 2011). Le bénéfice de cette introduction massive de matériel informatique à visée pédagogique est considéré comme minimal par les chercheurs, les usages étant bien trop peu nombreux et majoritairement en dehors de la classe. Cette étude affirme qu'une approche centrée uniquement sur un équipement massif des établissements scolaires est résolument contre-productive, puisque la mise à disposition d'une technologie ne conduit pas nécessairement à son utilisation massive et à son intégration dans la pratique pédagogique. Les chercheurs insistent sur la nécessité de faire reposer toute réforme faisant appel au numérique sur des fondations pédagogiques solides (Warschauer et al., 2011).

Les tablettes numériques ne sont elles-mêmes pas dépourvues de contraintes. Malgré leur facilité d'utilisation et leur interface intuitive, les tablettes présentent des contraintes

ergonomiques, selon les enseignants (Villemonteix & Nogry, 2016). Par exemple, lors d'une tâche de production écrite ou graphique, il n'y a pas de traces intermédiaires d'élaboration, ce qui rend difficile le suivi des élèves et donc l'individualisation des apprentissages. La tablette est aussi jugée « trop ludique » et donc distractrice vis-à-vis de la tâche à accomplir. Enfin, il est observé que l'école primaire paraît plus « souple » et plus propice à la collaboration entre enseignants et aux échanges informels qui conduisent à une pratique effective de ces nouveaux outils, par rapport aux établissements du secondaire (Villemonteix & Khaneboubi, 2013). Pour Berry (2009), les univers numériques peuvent cependant être des espaces d'apprentissages informels, car ils offrent la possibilité à l'élève d'expérimenter librement.

Ce rapide tour d'horizon des usages du numérique nous a permis de prendre conscience des limites de certaines technologies lorsqu'elles sont utilisées dans un but pédagogique. Les limites techniques notamment sont encore un défi important (Karsenti, 2016), avec très peu d'enseignants qui parviennent à régler ces problèmes techniques seuls. Des contraintes logistiques sont aussi à déplorer. Par exemple, le paramétrage des tablettes tactiles bloque parfois l'utilisation de nouvelles applications (Villemonteix & Nogry, 2016). Par ailleurs, ces tablettes sont pensées pour un usage individuel et domestique, ce qui entraîne aussi des contraintes logistiques, le problème de la recharge notamment (Villemonteix & Khaneboubi, 2013). Ce constat pose la question de la scolarisation des technologies qui ne sont pas à l'origine pensées pour s'intégrer dans un cadre éducatif. D'ailleurs, on observe généralement que les outils ou technologies qui sont scolarisés perdent leur caractère excitant et innovant, en particulier chez les adolescents qui se construisent souvent sur la transgression des règles (Baron, 2014). Il est aussi important de noter que l'appropriation et la maîtrise d'un nouvel outil peuvent s'avérer particulièrement chronophages. Le baromètre des usages du numérique de l'Académie de Montpellier (Gili, 2019) recense l'ensemble des freins à l'adoption du numérique éducatif chez les enseignants. On peut notamment y observer que ces freins sont généralement liés aux conditions matérielles et professionnelles, parfois à des dimensions personnelles (manque de temps, connaissances insuffisantes...), mais rarement à des dimensions pédagogiques (usage du numérique non pertinent, ressources numériques inexistantes...) (Gili, 2019).

L'omniprésence du numérique dans la vie des élèves actuels ne garantit absolument pas qu'ils sachent s'en servir pour leurs apprentissages. Les *« digital natives »* qui sont nés ou ont grandi dans un monde où le numérique était déjà bien développé utilisent rarement ces technologies pour apprendre et l'usage domestique quotidien qu'ils en font développe des

compétences limitées, non transférables dans les domaines scolaires (Kirschner & De Bruyckere, 2017). De même, les enseignants manifestent des compétences numériques très variables. Généralement très habiles avec des outils simples (traitement de texte, moteurs de recherche...), ils reconnaissent souvent ne pas être tout à fait à l'aise lorsque les outils se complexifient (tableur, traitement d'images, vidéos...) (Ravestein & Ladage, 2014). Cependant, les enseignants ne souhaitant pas intégrer le numérique à terme dans leurs pratiques sont sous-représentés dans le baromètre des usages du numérique de Montpellier (Gili, 2019).

En outre, concernant le numérique éducatif, la question de l'efficacité se pose. Les revues de littérature et les méta-analyses peinent à répondre de manière claire et définitive en ce qui concerne l'efficacité de l'utilisation des outils numériques dans un contexte d'apprentissage académique. Les travaux de Bernard et al. (2018) recensent 52 méta-analyses réalisées entre 1982 et 2015 sur l'impact de l'introduction des outils technologiques en classe. Les 20 meilleures ont été sélectionnées et synthétisées, en se basant sur un instrument permettant d'évaluer la qualité des méta-analyses, conçu par Tamim et al. (2015). Un effet positif assez faible (g = 0.29) est rapporté. Concernant les six méta-analyses les plus récentes, la taille de cet effet chute légèrement (g = 0.26). Selon les auteurs, ce résultat traduit potentiellement le fait que l'évolution rapide des technologies ne s'accompagne pas nécessairement d'efforts de conception pédagogique et de pratiques éducatives adéquates. Globalement, cette efficacité des technologies éducatives dépend d'un très grand nombre de facteurs, notamment des caractéristiques de l'outil en question, des caractéristiques de l'élève (connaissances antérieures, niveau scolaire...) et des disciplines concernées. Les résultats sont pour le moins mitigés et ne permettent pas de fournir une réponse claire et tranchée quant à l'efficacité de ces outils. Il est souvent impossible d'apporter une réponse simple et définitive sur l'efficacité de telle ou telle technologie dans telle ou telle situation.

Il semblerait donc que la recherche ait encore beaucoup à faire sur ce sujet. D'autant plus que la notion d'efficacité est difficile à définir dans ce contexte particulier. Une pratique efficace doit-elle réduire les inégalités scolaires ? Doit-elle permettre d'augmenter la moyenne générale d'une classe ? Doit-elle améliorer encore davantage les performances des élèves les plus motivés ? Doit-elle permettre une économie de moyens (temps, argent...) ? Toujours est-il qu'il ressort des études sur les usages du numérique que l'innovation technologique n'est pas un moteur évident de l'innovation pédagogique (Amadieu & Tricot, 2014).

De plus, l'effet bénéfique supposé du numérique sur la motivation des élèves paraît limité et pas nécessairement uniforme. Parfois, la motivation et la satisfaction de l'élève sont confondues. Le niveau de motivation réel de l'élève et son engagement dans la tâche demandée ne sont pas nécessairement directement reliés à son envie d'utiliser un outil ou une technologie (Passey et al., 2004). De même, les préférences de l'élève, par exemple concernant l'utilisation ou non de tel ou tel outil numérique, ne présentent aucun lien évident avec sa capacité à être plus performant dans ses apprentissages lorsque ses préférences sont respectées (Amadieu & Tricot, 2014). Cependant, d'après le baromètre des usages du numérique de l'Académie de Montpellier, une assez large majorité d'enseignants reconnaissent une plus-value au numérique, notamment pour capter l'attention et pour améliorer la dynamique de classe et le travail collaboratif (Gili, 2019), c'est-à-dire globalement pour animer les séances. En ce qui concerne les apprentissages (mémorisation de notions, expression écrite et orale...), cette plus-value paraît moins évidente chez les enseignants de l'académie.

En outre, ce baromètre des usages du numérique rend compte d'un lien étroit entre usage du numérique éducatif et recours aux pédagogies actives. Ce recours est assez fréquent puisque plus de 86% des enseignants y font appel au cours de l'année scolaire et plus de la moitié le font plusieurs fois par mois (Gili, 2019). Les pédagogies actives sont appréciées, car elles favoriseraient la motivation et l'implication, l'autonomie et la responsabilisation, la coopération et la collaboration. Il existe cependant des freins à l'utilisation des pédagogies actives pour une minorité de répondants. Au-delà du manque de formation et d'accompagnement, elles sont parfois jugées inefficaces, inadaptées à une discipline ou pouvant entraîner une perte de contrôle sur le comportement des élèves (Gili, 2019).

L'apport de la psychologie expérimentale dans la recherche en éducation

De nos jours, la méthode expérimentale participe grandement aux avancées de la recherche en éducation, en particulier en ce qui concerne les apports du numérique éducatif. Cette méthode, fondement de la recherche scientifique basée sur des faits et des preuves, permet de comparer différentes situations d'apprentissage entre elles. Généralement, pour tester l'efficacité d'une approche ou d'un outil, un groupe d'élèves (au sein d'une même classe ou issus de plusieurs classes ou établissements différents) qui bénéficie de cette approche ou de cet outil est comparé à un autre groupe qui n'en bénéficie pas. Lorsque l'ensemble des paramètres qui peuvent influer sur les résultats en classes sont contrôlés (répartition équitable entre filles et garçons, même niveau scolaire, même âge, même milieu socio-économique…),

les différences observées peuvent être imputées à l'approche ou l'outil en question. Cette méthode a le mérite de rendre possible la mise en lumière d'un effet (positif, négatif ou nul) d'une intervention sur un ensemble de variables d'intérêt (performances scolaires, compréhension, motivation, attitude…) et de mesurer la taille de cet effet. Les études expérimentales s'appuient sur un protocole rigoureux qui a été déployé dans les classes et, ce protocole étant théoriquement reproductible à l'identique, il est possible de répliquer une étude afin de confirmer l'existence d'un effet. C'est donc par la multiplication des études centrées sur la même question de recherche que les chercheurs et le monde éducatif dans son ensemble peuvent tirer des conclusions fiables quant à l'existence d'un effet particulier, lié à une approche ou à un outil censé apporter une plus-value pour les élèves.

La psychologie est une discipline qui prend de plus en plus d'importance dans la recherche en éducation. Il existe d'ailleurs depuis peu une branche de la psychologie, la psychologie de l'éducation, entièrement consacrée aux questions liées aux apprentissages scolaires. Par sa capacité à décrire, à comprendre et à expliquer les comportements humains, la psychologie semble toute indiquée pour mesurer et comprendre en quoi certaines pratiques peuvent impacter positivement ou non l'apprentissage et le vécu des élèves à l'école. Cette tendance globale d'un accroissement de l'influence des chercheurs en psychologie dans le domaine éducatif s'est récemment concrétisée avec la création du Conseil scientifique de l'Éducation nationale (CSEN) le 10 janvier 2018, qui a pour but d'émettre des recommandations fondées sur les résultats de la recherche scientifique internationale. Parmi les 25 membres du CSEN, neuf sont des chercheurs en psychologie. Ceci appuie encore davantage l'importance de la contribution de la psychologie pour faire face aux défis de l'enseignement scolaire, au même titre que d'autres disciplines telles que la sociologie ou les sciences de l'éducation.

Cette thèse de psychologie cognitive s'inscrit donc parfaitement dans la dynamique actuelle de la recherche en éducation qui s'intéresse aux mécanismes liés aux apprentissages scolaires, notamment aux mécanismes cognitifs impliqués dans les processus de mémorisation, de compréhension ou de motivation qui sont à l'œuvre en classe.

Ce travail de thèse

Cette thèse porte spécifiquement sur l'apprentissage de la programmation et de la pensée informatique et propose de comparer expérimentalement différents outils qui sont utilisés pour enseigner cette discipline. Trois approches en particulier ont été choisies pour leur capacité

supposée à améliorer les apprentissages pour diverses raisons qui seront évoquées plus loin : les activités débranchées qui n'exploitent aucun outil numérique, les logiciels de programmation à vocation éducative (tels que le logiciel Scratch) et les robots pédagogiques, qui sont très souvent associés à des logiciels éducatifs. Tout au long de cette thèse, nous désignerons par les mots « outils », « outils d'apprentissage » ou « outils d'enseignement », ces trois approches. Bien que les activités débranchées ne constituent pas un outil à proprement parler, celles-ci se basent généralement sur la manipulation d'objets en tous genres pour représenter l'exécution d'un algorithme, et exploitent donc des outils concrets, non numériques. L'objectif est de questionner l'existence de différences significatives entre ces trois outils concernant leur efficacité, c'est-à-dire de déterminer leur capacité à faciliter la transmission des connaissances et compétences fondamentales en programmation ainsi que leur capacité à stimuler la motivation, le sentiment d'auto-efficacité et l'attitude des élèves vis-à-vis de la science. Pour ce faire, deux nouveaux outils d'évaluation de la pensée informatique en contexte d'apprentissage de la programmation ont été conçus et ont suivi un processus de validation psychométrique. L'impact du feedback renvoyé par les outils numériques et de la charge cognitive associée à ces outils sera également étudié.

Ce travail de recherche se focalise plus particulièrement sur l'impact des trois outils évoqués précédemment sur l'initiation à la programmation. Pour cette raison, les élèves ayant participé à nos protocoles expérimentaux sont des élèves de cycle 3, en fin d'enseignement primaire (principalement de niveau CM2). Par ailleurs, ceci est en accord avec les programmes scolaires français de cycle 3 en « espace et géométrie » qui mentionnent la capacité à « accomplir, décrire, coder des déplacements dans des espaces familiers » et à « programmer les déplacements d'un robot ou ceux d'un personnage sur un écran en utilisant un logiciel de programmation » (Ministère de l'Éducation nationale, de l'enseignement supérieur et de la recherche, 2023).

Premièrement, nous aborderons la question de l'apprentissage de la programmation en contexte scolaire. Il s'agira tout d'abord de définir ce qu'est la programmation et de s'interroger sur les difficultés liées à cet apprentissage. Nous questionnerons également l'intérêt et les bénéfices attendus d'un apprentissage de la programmation à l'école ainsi que les différents objectifs d'apprentissage qui peuvent être associés à cette discipline.

Dans un second temps, nous essaierons de définir précisément ce que l'on appelle la « pensée informatique », dont le développement constitue l'un des enjeux majeurs de

l'enseignement de la programmation à l'école. Nous discuterons en détail de cette définition pour laquelle aucun véritable consensus n'a pu émerger à ce jour. Différents types de définitions proposées dans la littérature seront abordés. Les liens qui unissent le développement de la pensée informatique, l'apprentissage de la programmation et leur intégration fréquente en cours de mathématiques seront précisés.

Ensuite, nous présenterons les trois outils que nous avons étudiés dans le cadre de cette thèse et qui sont utilisés pour enseigner la programmation et la pensée informatique à l'école. Dans ce chapitre, nous aborderons l'enseignement de la programmation qui se distingue de l'apprentissage de cette discipline. Puis, nous décrirons chaque outil et examinerons les preuves empiriques en faveur de leur utilisation.

Enfin, nous introduirons les différentes variables qui nous semblent pertinentes pour évaluer l'efficacité relative des trois outils étudiés. Nous distinguerons les connaissances notionnelles des connaissances procédurales et des compétences qu'il est possible de développer par l'apprentissage de la programmation et de la pensée informatique. Face à la multitude et à la diversité des avis concernant ce qu'est réellement la pensée informatique, nous prendrons position en faveur d'un modèle précis. Le rôle potentiel joué par la charge cognitive des différentes situations d'apprentissage et par l'implication du corps et de la motricité dans les apprentissages sera également discuté. D'autres variables liées au vécu subjectif des élèves, à savoir la motivation, le sentiment d'auto-efficacité et l'attitude envers les disciplines scientifiques, seront considérées. Les liens qu'entretiennent ces différentes variables seront également discutés, tout comme l'impact du genre des élèves sur l'ensemble des facteurs étudiés.

Une dernière partie sera consacrée à la présentation de la problématique au cœur de ce travail de thèse et des hypothèses que nous avons formulées pour tenter d'y répondre.

I) PARTIE THEORIQUE

Chapitre 1. L'apprentissage de la programmation

1.1) Qu'est-ce que la programmation ?

Définitions

La programmation désigne « l'ensemble des activités liées à la définition, l'écriture, la mise au point et l'exécution de programmes informatiques » ainsi que la « séquence des ordres auxquels doit obéir un dispositif » (Dictionnaire Larousse en ligne, s.d.). La programmation peut également être définie comme le « processus de développement et de mise en œuvre d'un ensemble d'instructions qui permettent à un ordinateur d'exécuter une certaine tâche, de résoudre des problèmes et de fournir une interaction avec un humain » (Balanskat & Engelhart, 2015, p. 7, cités par Scherer et al., 2020). Elle est au cœur de l'informatique au sens large : un dispositif numérique (par exemple un ordinateur) exécute des programmes, la mémoire et le processeur permettent de stocker et manipuler des données et les utilisateurs exploitent ces programmes par des logiciels (Webb et al., 2017).

La programmation est considérée comme un moyen de communiquer, partager, explorer des idées et serait un outil pour apprendre et réaliser des tâches (Guzdial, 2019, cité par Chevalier, 2022, p. 41). Elle implique généralement l'utilisation d'un logiciel ou d'un langage de programmation. Programmer est une des 24 fonctions pédagogiques pouvant bénéficier des apports du numérique, identifiées par Tricot (2020). D'après lui, la programmation peut être à la fois un moyen utilisé pour enseigner quelque chose ou le but même de l'apprentissage.

Programmer à l'école

En contexte scolaire, [le terme « programmation » regroupe généralement un ensemble d'activités relativement disparates qui convergent vers un but identique : développer la capacité de l'élève à produire des algorithmes, c'est-à-dire des successions d'actions qui constituent autant d'étapes vers la résolution d'un problème algorithmique. La programmation est considérée depuis ses débuts comme un processus très complexe impliquant un grand nombre

d'activités cognitives et de représentations mentales (Rogalski & Samurçay, 1990). Généralement, les activités de programmation nécessitent l'analyse d'un problème donné afin de déterminer un algorithme adéquat qu'il faudra par la suite retranscrire dans un langage de programmation (Rogalski et al., 1988). D'un point de vue psychologique, programmer c'est élaborer une conception fonctionnelle de la relation entre la solution d'un problème et la représentation opératoire de cette solution dans un langage de programmation (Samurçay & Rouchier, 1985). Cette nécessaire prise en compte du dispositif d'exécution d'un programme distingue la programmation des activités de résolution de problèmes classiques et induit une modification de la planification des actions. Samurçay et Rouchier (1985) ont soumis des lycéens (âgés de 15-16 ans) à un problème mathématique simple dont la solution était connue de tous et ont montré qu'un grand nombre d'entre eux ne sont pas parvenus à programmer cette solution en prenant en compte les limitations de la machine qui leur étaient imposées. Les auteurs en ont conclu que les élèves avaient des difficultés à se mettre spontanément dans une situation de production de procédures plutôt que dans une situation de production de résultats.

La programmation demeure indissociable de la notion d'algorithme bien que cette notion ne soit pas propre à l'informatique et à la programmation. Dans un document rédigé à l'attention des enseignants, Tchounikine (2017, p. 11) définit l'algorithme comme « un enchaînement mécanique d'actions, dans un certain ordre, qui chacune a un effet, et dont l'exécution complète permet de résoudre ou de faire quelque chose ». Aujourd'hui, programmer et coder sont parfois considérés à tort comme synonymes (Webb et al., 2017). Le codage ne constitue cependant qu'une des phases du processus de programmation, celle où l'on écrit le programme. Ce processus inclut également d'autres phases, notamment celle qui consiste à concevoir et développer des algorithmes (Lodi & Martini, 2021 ; Parent, 2022) qui correspond davantage à ce que l'on cherche à transmettre dans l'enseignement scolaire.][1] En outre, le codage et la programmation peuvent être perçus comme une nouvelle littératie, c'est-à-dire comme une forme d'expression personnelle au même titre que l'écriture, plus que comme un ensemble de compétences techniques (Resnick & Siegel, 2015).

[1] Sigayret, K., Blanc, N., & Tricot, A. (2022). L'apprentissage de la programmation: quels outils pour évaluer le développement de la pensée informatique à l'école?. *Enfance*, *4*(4), 479-500.

Historique de la programmation en classe

La programmation prend incontestablement une importance de plus en plus grande dans notre société où la maîtrise du numérique et de ses possibilités n'est plus une option. Les premières tentatives d'introduction de la programmation en classe datent des années 1960, notamment en prenant appui sur le logiciel Logo (Papert, 1980). Baron et Boulc'h (2012) dressent un rapide historique de l'intégration progressive de la programmation dans le système scolaire français. Les auteurs identifient notamment trois périodes importantes. La première est une phase de recherche et d'innovation initiale qui a mobilisé les chercheurs autour de la programmation et des perspectives potentielles de développement cognitif que la discipline suscite.

Une seconde ère s'est ouverte, avec le « Plan informatique pour tous » de 1985, porté par une politique centrée sur un meilleur équipement des écoles primaires et une volonté de formation des enseignants aux technologies informatiques. Cette époque marque également le déclin de l'intérêt porté à la programmation au profit des outils informatiques « facilitateurs d'écrit » (traitement de texte) qui montrent des résultats intéressants.

Enfin, la diffusion rapide d'internet et des outils de communication à l'échelle mondiale à partir du début des années 2000 marque l'avènement d'une troisième période dans laquelle nous sommes toujours engagés. L'intérêt se focalise désormais sur le numérique, terme qui acquiert un sens nouveau et qui regroupe un ensemble d'outils relevant de la production de documents multimédias, de la communication, de la recherche d'information ou des technologies éducatives et inclut les activités de programmation.

1.2) Apprendre à programmer : un véritable défi en contexte scolaire ?

La programmation : une discipline complexe…

La programmation informatique serait intrinsèquement complexe et coûteuse sur le plan cognitif, car elle exigerait la maîtrise de plusieurs domaines et son apprentissage impliquerait un haut niveau d'interactivité entre les éléments (Berssanette & de Francisco, 2021). La programmation exige généralement des programmeurs qu'ils aient une connaissance des langages de programmation, une expertise relative à la conception d'algorithmes spécifiques et à la logique, la capacité d'analyser, de comprendre et de résoudre des problèmes et d'évaluer

l’exactitude et la mise en œuvre de l’algorithme dans un langage de programmation particulier (Forsström & Kaufmann, 2018).

La programmation est devenue au fil du temps une compétence dont la maîtrise est un atout dans le monde économique et sur le marché du travail (Bers et al., 2022). Elle permet en outre une familiarisation avec la « pensée informatique », la conception d'algorithmes et plus généralement la capacité à raisonner pour résoudre des problèmes. La connaissance de la programmation comprend les connaissances conceptuelles et procédurales nécessaires pour résoudre des problèmes par une forme de « calcul » (*« computation »* en anglais), c’est-à-dire des connaissances syntaxiques, sémantiques, schématiques et stratégiques (Scherer et al., 2020). Être compétent en programmation désignerait plutôt la capacité à créer, modifier ou évaluer du code informatique (Scherer et al., 2020).

La revue de littérature de Robins et al. (2003) sur l'enseignement de la programmation est plutôt pessimiste et fait état d'un grand nombre de difficultés liées à la discipline. En effet, la programmation à l'école s'adresse à des novices et, selon cette étude, ceux-ci éprouvent de grandes difficultés à maîtriser et à distinguer d'un côté les connaissances fondamentales et concepts, de l'autre, les stratégies et les compétences pratiques propres à cet apprentissage. De plus, lorsque ces connaissances et stratégies sont apprises, elles restent souvent assez fragiles, pas toujours appliquées et sources de confusion.

Dans une autre revue de littérature, centrée sur les difficultés liées à l’initiation à la programmation, Quian et Lehman (2017) détaillent les idées fausses que peuvent entretenir les élèves lorsqu’ils sont confrontés à cette pratique pour la première fois et interrogent l’impact de celles-ci sur leurs apprentissages. Pour les auteurs, les connaissances préalables des élèves et leur expérience antérieure sont à l’origine d’idées reçues concernant la programmation. Il serait donc essentiel pour les chercheurs et enseignants d’analyser ces connaissances préalables, dans un contexte indépendant de toute activité de programmation, pour empêcher la survenue d’erreurs et de difficultés évitables concernant les connaissances syntaxiques, conceptuelles ou stratégiques nécessaires pour apprendre à programmer. Les auteurs insistent également sur le rôle primordial des enseignants qui ont tendance à faire preuve d’une faible compréhension des idées fausses de leurs élèves ou échouent à appréhender les défis que peut représenter l’initiation à la programmation pour des novices.

… rendue plus accessible grâce aux nouveaux langages visuels

Denning et al. (2017) présentent quant à eux une vision plus nuancée de l'apprentissage de la programmation. La programmation et le codage seraient des compétences faciles à apprendre, mais beaucoup de pratique et d'expérience sont nécessaires pour devenir véritablement compétent et être capable de programmer des applications utiles dans le monde réel. En outre, nous savons aujourd'hui avec un haut niveau de certitude que la programmation informatique peut être enseignée en contexte scolaire. Scherer et al. (2020) ont rapporté dans leur méta-analyse que les interventions en programmation ont un effet fort sur les connaissances et compétences dans ce domaine (g = 0.81). Bien que ce résultat puisse sembler évident, il a le mérite de permettre d'affirmer empiriquement que l'apprentissage de la programmation à l'école est un objectif réalisable. Il peut également constituer un point de référence pertinent pour toute intervention future visant à évaluer l'impact d'un dispositif ou d'une approche éducative en particulier. D'ailleurs, les auteurs concluent cette analyse en indiquant que les tailles d'effets varient significativement selon les études et le contexte d'implémentation des activités de programmation. Contrairement à leur méta-analyse précédente que nous détaillerons un peu plus loin (Scherer et al., 2019), celle de 2020 ne permet pas de conclure sur l'effet de transfert associé à l'apprentissage de la programmation, les articles référencés ici ne concernant que ce qu'ils désignent comme un « transfert proche » dont l'évaluation est fondée sur des tâches de programmation similaires à celles réalisées lors de la phase d'instruction. L'apprentissage de la programmation nécessite toutefois un développement graduel poursuivi sur de nombreuses années pour mener à une réelle maîtrise (Webb et al., 2017).

Au-delà de la complexité, réelle ou supposée, de la programmation, il est important de s'intéresser à la perception de cette discipline par les élèves eux-mêmes. Certaines études indiquent que la programmation peut être perçue comme nécessitant des compétences complexes (Tsai, 2019). L'auteur en question précise par ailleurs que le sentiment d'auto-efficacité des élèves peut moduler leurs performances et que, globalement, les élèves ayant un sentiment d'auto-efficacité élevé manifestent une meilleure compréhension des concepts basiques en programmation. Toutefois, les langages visuels de programmation (tels que le logiciel Scratch, Resnick et al., 2009) pourraient réduire l'inquiétude et la méfiance des élèves vis-à-vis de cette discipline en éliminant les erreurs de syntaxe et en mettant l'accent sur la simplicité d'utilisation (Maloney et al., 2010) ce qui aurait également en retour un impact positif sur le sentiment d'auto-efficacité (Tsai, 2019).

Malgré les interrogations évoquées précédemment quant aux difficultés de l'apprentissage de la programmation à l'école, celui-ci [connaît un essor indéniable depuis vingt ans. Ceci peut s'expliquer en partie par le développement des logiciels de programmation visuels qui rendent plus accessible cette discipline parfois jugée trop complexe, ainsi que par la nécessité pour le futur citoyen de pouvoir appréhender et démystifier le numérique et ses enjeux par une réflexion critique (Romero et al., 2018)][2]. L'apparition de Scratch au cours des années 2000 vient donc rebattre les cartes en permettant aux enfants de bénéficier d'un langage de programmation adapté. Ce langage est un outil au service de la créativité des élèves qui utilisent la programmation pour travailler sur des projets qui ont du sens pour eux (Tchounikine, 2017). Au-delà des compétences en algorithmique, c'est aussi d'autres compétences fondamentales qui sont potentiellement développées par cette pratique, telles que le raisonnement et la résolution de problèmes. Apprendre à programmer exige des compétences en matière de stratégies, de planification et de pensée logique ce qui incite à penser que les activités de programmation constituent un terrain propice au développement des capacités de résolution de problèmes ou des compétences métacognitives (Lavonen et al., 2003). L'effet réel sur les apprentissages scolaires reste cependant encore à démontrer (Tricot, 2020).

1.3) Pourquoi apprendre la programmation ?

La programmation : une discipline scolaire à part entière ?

[De nos jours, selon le rapport Eurydice (Bourgeois et al., 2019), qui donne un aperçu de l'état de l'éducation numérique en Europe, les compétences en programmation et en codage sont des objectifs d'apprentissage pour l'enseignement primaire dans une vingtaine de systèmes éducatifs européens et dans une trentaine de pays en ce qui concerne l'enseignement secondaire. En France, la compétence numérique est scindée en deux axes dont le premier la définit comme une langue : les langages de programmation et les algorithmes. Le rapport Eurydice classe la programmation comme une compétence issue d'un domaine plus large qui est celui de la création de contenus numériques.

Le « socle commun de connaissances, de compétences et de culture » (Ministère de l'Éducation nationale, de l'enseignement supérieur et de la recherche, 2015) indique notamment

[2] Sigayret, K., Blanc, N., & Tricot, A. (2022). L'apprentissage de la programmation: quels outils pour évaluer le développement de la pensée informatique à l'école?. *Enfance*, *4*(4), 479-500.

que l'élève « connaît les principes de base de l'algorithmique et de la conception des programmes informatiques. Il les met en œuvre pour créer des applications simples » (p. 4). Ce socle mentionne également les « langages informatiques [qui] sont utilisés pour programmer des outils numériques et réaliser des traitements automatiques de données » (p. 4)][3].

Bien que la programmation puisse constituer une fonction pédagogique à part entière, apprendre à programmer pourrait contribuer à remplir d'autres fonctions pédagogiques parmi les 24 fonctions pour lesquelles l'apport du numérique est potentiellement intéressant (Tricot, 2020). La programmation peut être utilisée pour « résoudre des problèmes et calculer », « motiver » ou « jouer » (en particulier lorsqu'on fait appel à des logiciels éducatifs ludiques ou à des robots pédagogiques, censés renforcer la motivation des élèves), « coopérer » (certaines études apportent des éléments empiriques en faveur d'un apprentissage de la programmation par groupe de deux élèves plutôt que seul, voir Umapathy & Ritzhaupt, 2017), « découvrir des concepts abstraits », « créer un objet technique, une œuvre picturale ou sonore » ou encore « faire émerger des idées, développer sa créativité ». Il est clair que ces différentes fonctions pédagogiques associées à l'apprentissage de la programmation constituent un argument solide en faveur de l'introduction de cette discipline dans les cursus scolaires. À noter cependant que pour les deux dernières fonctions évoquées, l'impact de la programmation sur la créativité est encore trop peu étudié pour qu'on puisse véritablement se prononcer sur cet effet.

Un impact positif sur les facultés cognitives ?

Au-delà de la nécessité de faire entrer la programmation dans un contexte scolaire, son apprentissage stimulerait également certaines facultés cognitives. Les processus impliqués dans la programmation sont souvent considérés comme favorisant des compétences telles que la créativité, la pensée innovante et la communication ce qui ferait de la programmation une activité améliorant les performances cognitives (Psycharis & Kallia, 2017). Les compétences nécessaires à sa maîtrise impliquent différents aspects du raisonnement et, plus généralement, de la cognition humaine (Grover & Pea, 2013). Il est donc tout à fait logique de chercher à explorer les bénéfices que l'on peut retirer de l'apprentissage de la programmation sur le plan cognitif. Par ailleurs, la question du transfert possible des connaissances assimilées en programmation peut être posée. Les interventions visant à favoriser les compétences en

[3] Sigayret, K., Blanc, N., & Tricot, A. (2022). L'apprentissage de la programmation: quels outils pour évaluer le développement de la pensée informatique à l'école?. *Enfance*, *4*(4), 479-500.

programmation entraînent-elles des effets indirects positifs dans d'autres disciplines ou d'autres situations ?

Scherer et al. (2019) tentent de répondre à cette vaste interrogation par le biais d'une méta-analyse reposant sur 105 études distinctes. Les auteurs définissent les compétences en programmation comme la capacité de créer, modifier et évaluer un code informatique ainsi que d'utiliser les compétences conceptuelles et procédurales nécessaires pour appliquer ces connaissances dans l'optique de résoudre un problème. Ils essaient alors de mesurer ce qu'ils nomment l'« effet de transfert », lié à ces compétences en programmation, c'est-à-dire l'impact de celles-ci sur les aptitudes cognitives des individus. Pour ces auteurs, les compétences en programmation et d'autres facultés cognitives partagent des sous-compétences importantes, ce qui les conduit à penser qu'un effet de transfert doit exister. Ils distinguent d'ailleurs, en s'appuyant sur les travaux de Perkins et Salomon (1992), l'effet de transfert proche entre deux contextes similaires et étroitement liés, exigeant l'exécution de compétences et de stratégies similaires, de l'effet de transfert lointain qui concerne deux contextes dissimilaires, par essence différents, et pouvant nécessiter des compétences et stratégies différentes. Dans la méta-analyse de Scherer et al. (2019), l'effet de transfert proche désigne l'effet de l'apprentissage du code sur les facultés cognitives mobilisées par la pratique de la programmation et l'effet de transfert lointain l'effet sur les facultés cognitives sans lien direct avec la programmation. Il est important de noter que l'existence de l'effet de transfert lointain a été fréquemment remise en question, que ce soit en général ou dans le contexte de l'apprentissage de la programmation (Denning, 2017).

Les résultats de cette méta-analyse (Scherer et al., 2019) indiquent un effet positif de l'apprentissage de la programmation. Ces bénéfices sont observés à la fois sur les compétences en programmation elles-mêmes et sur la maîtrise de concepts en informatique, qui sont améliorés par la pratique, mais aussi dans d'autres domaines de la cognition, ce qui témoigne du transfert possible des compétences acquises en programmation dans d'autres situations. L'effet de transfert global est modéré (g = 0.49), mais l'effet de transfert proche est fort (g = 0.75). L'effet de transfert lointain semble exister également (g = 0.47) bien qu'une grande variabilité soit observée suivant les facultés cognitives étudiées. Parmi celles-ci, la pensée créative (g = 0.73) et les compétences en mathématiques (g = 0.57) paraissent les plus positivement influencées par l'apprentissage de la programmation. Un effet plus marginal est aussi observé sur la métacognition (g = 0.44), le raisonnement (g = 0.37), les compétences spatiales (g = 0.37) ou la réussite scolaire (g = 0.28) alors qu'aucun effet ne semble exister sur

la littératie (évaluée par la mesure de la compréhension écrite et des compétences rédactionnelles). Les auteurs évoquent cependant certaines limites de leur analyse, liées aux lacunes de certaines études sélectionnées : absence de mesure de référence (*baseline*) permettant d'isoler l'effet de l'intervention, absence de statistiques pertinentes pour calculer précisément des tailles d'effet, échantillons réduits, grande variabilité des approches pédagogiques mises en œuvre…

Toujours dans l'optique de mesurer les bénéfices secondaires de la programmation, Popat et Starkey (2019) proposent une revue de littérature, regroupant une dizaine d'articles, focalisée sur l'influence du codage sur les résultats scolaires des enfants. Les chercheurs mettent en exergue une influence de la programmation sur certaines compétences dites « de haut niveau ». La capacité à raisonner semble ainsi améliorée par la pratique du code, comme en témoignage l'étude de Psycharis et Kallia (2017), citée dans cette revue de littérature.

Dans cette dernière étude, 66 élèves de lycée étaient répartis en deux groupes de 33 élèves et un seul de ces deux groupes participait à des activités de programmation. La pratique de la programmation semble responsable d'un accroissement significatif des capacités de raisonnement des élèves qui y ont pris part en comparaison avec le groupe témoin. Il en va de même pour le sentiment d'auto-efficacité en mathématiques qui augmente significativement. En revanche, l'amélioration de l'aptitude à résoudre des problèmes mathématiques n'est pas significative pour les élèves qui ont participé aux activités de programmation. Popat et Starkey (2019) constatent qu'il est plus efficace d'entraîner cette capacité directement plutôt que par le biais de la programmation. En ce qui concerne la pensée critique, trop peu d'études solides investiguent l'influence de la programmation sur cette compétence, bien qu'elles présentent des résultats encourageants. Popat et Starkey (2019) remarquent par ailleurs que les résultats éducatifs de l'apprentissage de la programmation dépendent principalement de son intégration dans le programme scolaire et de la conception pédagogique des activités proposées qui seront susceptibles de favoriser ou non le développement de compétences secondaires.

Précédemment, Messer et al. (2017) ont voulu tester l'influence de l'apprentissage de la programmation sur les compétences des enfants en mathématiques, sur leur habileté visuospatiale et sur leur mémoire de travail. Pour ce faire, les chercheurs ont réparti aléatoirement 41 enfants de 5 ou 6 ans dans trois conditions expérimentales. Le premier groupe s'exerçait à la programmation via l'utilisation d'une tablette, un second groupe s'entraînait de manière plus classique sur papier alors que le dernier groupe (contrôle) ne participait à aucune

activité de programmation, celles-ci étant remplacées par des exercices d'addition et de soustraction. Pour l'ensemble des enfants, ces activités se déroulaient deux fois par semaine, à raison de 10 minutes par session, pendant une durée totale de six semaines.

Les trois groupes de participants manifestent globalement une amélioration de leurs compétences spatiales et de leur niveau en mathématiques, les résultats post-test étant supérieurs à ceux obtenus pré-test. Selon les auteurs, cette amélioration ne pourrait probablement pas être imputable à un simple effet de maturation, indépendant des activités réalisées, étant donnée la durée relativement faible de l'étude. Aucun effet significatif n'est en revanche observé sur la mémoire de travail. Plus important encore, ce protocole expérimental ne montre aucune différence significative entre les trois groupes. L'apprentissage de la programmation ne semble donc pas plus efficace que des activités plus directes pour développer les compétences étudiées. Cependant, les auteurs remarquent que ces résultats sont tout de même encourageants, car ils font état d'une réelle capacité de la programmation à développer des compétences transférables dans d'autres domaines plus ou moins reliés.

Un transfert des compétences acquises en programmation encore à démontrer

A contrario, Denning et al. (2017) estiment que l' « hypothèse du transfert » qui suppose qu'une compétence de pensée acquise dans un contexte précis – celui de la programmation - est automatiquement transférée dans d'autres domaines est très insuffisamment étayée. Les auteurs recommandent plutôt d'introduire directement la pensée informatique dans d'autres disciplines pour démontrer explicitement en quoi la programmation peut être pertinente dans ce domaine. En effet, le transfert a davantage de probabilités de succès lorsque les domaines concernés sont étroitement liés et lorsque celui-ci est effectué de manière consciente et explicite (Mayer, 1988 ; Popat & Starkey, 2019 ; Laurent et al., 2022). En outre, la programmation a généralement un impact plus fort et plus évident sur des compétences spécifiques plutôt que sur des compétences plus générales, telles que la capacité à résoudre des problèmes dans différents contextes indépendants des activités de programmation (Galand, 2020).

Pour Scherer et al. (2019), la programmation est une activité similaire aux activités de résolution de problèmes dans d'autres domaines, tels que les sciences ou les mathématiques. On peut observer notamment que la pensée informatique et la programmation peuvent partager un vocabulaire commun avec les mathématiques : par exemple la notion d'algorithme et de variable (Guo & Ottenbreit-Leftwich, 2020). En outre, le lien avec cette discipline ne se limite pas qu'à des questions de vocabulaire, l'élève peut facilement être amené à trouver une solution

à un problème mathématique avant de réfléchir aux moyens d'implémenter cette solution dans un ordinateur, en respectant la syntaxe et la grammaire propre au langage ou au logiciel de programmation utilisé (Psycharis & Kallia, 2017). Par ailleurs, dans de nombreux pays, la pensée informatique est introduite par le biais d'activités de programmation en cours de mathématiques en raison des liens étroits qu'entretiendraient ces différents domaines (Bocconi et al., 2016).

Récemment, Laurent et al. (2022) ont conduit des travaux visant à étudier l'impact des activités de programmation avec le logiciel Scratch sur l'apprentissage de trois concepts en mathématiques avec un échantillon large et randomisé d'élèves de niveau CM1-CM2. Malgré la rigueur du protocole expérimental et les efforts investis pour favoriser le transfert des compétences de programmation en mathématiques (haut niveau de guidage de la part des enseignants, mise en évidence des éléments communs entre les deux situations d'apprentissage), les résultats se sont avérés décevants. Les élèves du groupe ayant participé à des activités de programmation montrent une progression plus faible de leur maîtrise des trois concepts mathématiques (différences pré-test / post-test) par rapport au groupe contrôle, non impliqué dans ces activités. Cette étude confirme les doutes légitimes quant à la capacité des élèves à profiter des activités de programmation pour améliorer leurs performances dans une discipline précise, supposée proche de la programmation.

Il semblerait donc que la programmation gagne à être enseignée pour elle-même et non pas nécessairement comme un moyen d'optimiser les apprentissages d'autres disciplines scientifiques. Cependant, l'apprentissage de la programmation, récemment rendu plus accessible par les langages visuels et les logiciels éducatifs, pourrait bien avoir un impact positif sur un certain nombre de facultés cognitives. Il est également communément admis que la programmation contribuerait à l'acquisition de la « pensée informatique », qui regrouperait un ensemble de compétences, et dont le développement est parfois considéré comme l'objectif d'apprentissage fondamental de toute pratique de la programmation à l'école.

SYNTHÈSE

- Programmer englobe plusieurs activités telles que **définir, écrire, mettre au point et exécuter des programmes**.
- Depuis son introduction dans les années 1960, la programmation est considérée comme **une discipline complexe et coûteuse sur le plan cognitif**.
- **L'arrivée des nouveaux langages visuels de programmation** au début des années 2000 a permis de rendre cette discipline plus accessible.
- La programmation a d'ores et déjà commencé à **intégrer les cursus scolaires européens** (Conrads et al., 2017).
- L'apprentissage de la programmation aurait un **impact bénéfique sur certaines habiletés cognitives** telles que le raisonnement, la pensée créative, la métacognition ou les compétences spatiales.
- La **capacité des élèves à transférer ce qui a été acquis en programmation** dans d'autres domaines plus ou moins éloignés (par exemple en mathématiques) fait encore l'objet de débats.

Chapitre 2. Le développement de la pensée informatique

2.1) Plusieurs conceptions de la pensée informatique

Apprendre à programmer pour développer une pensée informatique

[De nombreuses études ont tenté de mesurer l'impact des activités de programmation sur un large panel de facultés cognitives très générales. Généralement, l'ensemble des compétences cognitives spécifiques et des processus de résolution de problèmes qui interviennent notamment dans les activités de programmation sont regroupés sous le terme de « pensée informatique » qui désigne une capacité bien au-delà de la simple capacité à programmer. En français, « pensée informatique » est une traduction imparfaite de l'expression anglaise originale *computational thinking*. Cette dernière comprend le mot *thinking* qui désigne davantage le « processus de penser » que le produit de ce processus, qui est la pensée elle-même. Même si les termes « pensée computationnelle » ou « processus de pensée informatique » seraient plus appropriés, il semble que l'expression « pensée informatique » se soit imposée comme la traduction la plus répandue de *computational thinking*. C'est ainsi que nous désignerons la pensée computationnelle/les processus de pensée informatique tout au long de cette thèse.][4]

La pensée informatique intéresse particulièrement les décideurs politiques (Bocconi et al., 2016). L'enseignement de la pensée informatique a d'ores et déjà intégré les programmes scolaires de nombreux systèmes éducatifs en Europe (Conrads et al., 2017), ce qui se traduit généralement par des activités de programmation censées être utiles pour la développer (Scherer, 2016 ; Scherer et al., 2019). [Wing (2006) estime ainsi que la pensée informatique est une compétence fondamentale et devrait faire partie des capacités analytiques de tous les enfants, au même titre que la lecture, l'écriture ou l'arithmétique. La nécessité pour tous les individus de maîtriser cette pensée informatique d'ici le milieu du XXIe siècle est un constat partagé par de nombreux chercheurs et experts du domaine (Bocconi et al., 2016 ; Mohaghegh & McCauley, 2016 ; Ioannou & Makridou, 2018 ; Nordby et al., 2022…). De même, Nogry (2018) estime que la transmission d'une culture informatique est devenue un enjeu sociétal.

[4] Sigayret, K., Blanc, N., & Tricot, A. (2022). L'apprentissage de la programmation: quels outils pour évaluer le développement de la pensée informatique à l'école?. *Enfance*, *4*(4), 479-500.

L'initiation à la pensée informatique permettrait ainsi au futur cybercitoyen d'acquérir une certaine maîtrise et donc de dépasser le statut de simple consommateur du numérique.

Bien des années plus tôt, Papert (1980) fut le premier à utiliser l'expression « pensée informatique » en faisant référence à une aptitude mentale que les enfants développent en programmant avec le langage Logo. Il suggérait notamment que l'informatique pouvait offrir aux enfants de nouvelles possibilités d'apprentissage, de réflexion, de développement émotionnel et cognitif et favoriser la pensée procédurale par le biais des activités de programmation.][5] La vision de la pensée informatique de Papert (1980) et celle de Wing (2006) partagent des similitudes, mais se distinguent par certaines différences notables. Lodi et Martini (2021) résument ces différences : Wing (2006) a mis en exergue le besoin urgent d'enseigner certains concepts fondamentaux en informatique pour comprendre et participer activement à une société où le numérique devient omniprésent, là où Papert (1980) a affirmé que la programmation peut être un formidable outil pour la réflexion et pour bénéficier d'apprentissages plus riches sur le plan cognitif et affectif. Lodi et Martini (2021) mettent en garde contre les affirmations, insuffisamment étayées et souvent soutenues par des chercheurs non informaticiens, qui ont découlé de ces deux visions : les stratégies de résolutions de problèmes informatiques ne peuvent pas nécessairement être facilement appliquées à toutes les activités humaines et l'apprentissage de la programmation n'entraîne pas automatiquement une amélioration des capacités de réflexions.

Diverses manières de définir la pensée informatique

Cependant, la définition précise de la pensée informatique ne fait pas l'objet d'un consensus entre les chercheurs et demeure aujourd'hui ouverte aux interprétations (Brennan & Resnick, 2012 ; Mohaghegh & McCauley, 2016 ; Román-González et al., 2017 ; Moreno-León et al., 2018 ; Tang et al., 2020). Dans une méta-analyse comprenant 101 études, Tikva et Tambouris (2021) ont recensé 60 éléments différents dans la littérature qui sont associés à un cadre conceptuel ou à une définition de la pensée informatique ou simplement évalués dans des études empiriques. L'utilisation de termes synonymes et certaines divergences de point de vue entre la littérature francophone et anglophone contribuent par ailleurs à complexifier notre compréhension de ce qu'est la pensée informatique (Parent, 2022). Les malentendus et les

[5] Sigayret, K., Blanc, N., & Tricot, A. (2022). L'apprentissage de la programmation: quels outils pour évaluer le développement de la pensée informatique à l'école?. *Enfance*, *4*(4), 479-500.

approximations sont courants lorsqu'il s'agit de distinguer pensée informatique, informatique, programmation et codage (Tsarava et al., 2019).

La pensée informatique est en outre liée à d'autres termes, tels que la culture/littératie informatique (capacité à utiliser les ordinateurs et les technologies apparentées) ou la culture/littératie numérique qui étend la précédente en y ajoutant des aspects associés au réseautage et accorde une place particulière aux compétences applicables (y compris les compétences sociales), mais la plupart des définitions de la pensée informatique s'appuie sur la pensée algorithmique, entendue comme la capacité à formuler des problèmes qui transforment une entrée en sortie souhaitée à l'aide d'algorithmes (Moreno-León et al., 2018).

[Román-González et al. (2017) distinguent trois types de définitions de la pensée informatique. Il existe les définitions générales plutôt focalisées sur l'ensemble des aptitudes impliquées dans la résolution de problèmes (Wing, 2006 ; Wing, 2011 ; Aho, 2012) ; les définitions opérationnelles qui fournissent un cadre en catégorisant la pensée informatique en différents sous-domaines ou processus cognitifs (Barr & Stephenson, 2011 ; CSTA & ISTE, 2011 ; Grover & Pea, 2013 ; Selby & Woollard, 2013 ; Kalelioglu et al., 2016 ; Weintrop et al., 2016 ; Shute et al., 2017) ; les définitions pédagogiques qui listent un ensemble de concepts et de compétences (savoirs et savoir-faire) à enseigner dans un contexte éducatif (Brennan & Resnick, 2012 ; Zhang & Nouri, 2019).][6]

Définitions générales

[Wing (2006) décrit la pensée informatique, ou pensée computationnelle (*computational thinking*), comme une manière de résoudre des problèmes ou de concevoir des systèmes qui s'appuie sur des concepts fondamentaux de la science informatique. Cette pensée informatique partage de nombreuses similarités avec la programmation, les mathématiques, l'ingénierie et, plus globalement, la pensée scientifique. C'est plus précisément « l'ensemble des processus de pensée impliqués dans la formulation d'un problème et l'expression de sa solution de manière à ce qu'un ordinateur - humain ou machine - puisse effectivement l'exécuter » (Wing, 2011) ou de manière à ce que « sa solution puisse être représentée sous forme d'étapes de traitement ou de calcul (*computational steps*) et d'algorithmes » (Aho, 2012, p. 1).][7] Pour Bers et al. (2019),

[6] Sigayret, K., Blanc, N., & Tricot, A. (2022). L'apprentissage de la programmation: quels outils pour évaluer le développement de la pensée informatique à l'école?. *Enfance*, *4*(4), 479-500.

[7] Sigayret, K., Blanc, N., & Tricot, A. (2022). L'apprentissage de la programmation: quels outils pour évaluer le développement de la pensée informatique à l'école?. *Enfance*, *4*(4), 479-500.

la pensée informatique n'est pas seulement une capacité à résoudre des problèmes, mais également un processus expressif qui permet de trouver de nouvelles façons de communiquer des idées. Les paragraphes suivants détaillent les différents modèles conceptuels qui sont fréquemment utilisés pour définir la pensée informatique dans les études empiriques qui la concerne. L'objectif ici ne sera pas de décrire comment les auteurs en sont venus à proposer leur modèle (souvent, il est issu d'une revue de littérature), mais plutôt de permettre au lecteur de distinguer les similitudes et les disparités entre ces modèles.

Définitions opérationnelles

[Selby et Woollard (2013) ont tenté d'apporter une définition plus précise afin notamment de faciliter l'élaboration d'outils permettant d'évaluer les compétences en pensée informatique, dans un cadre scolaire par exemple. Pour ce faire, les auteurs ont compilé et analysé les articles relatifs à cette pensée informatique, dans l'optique d'en dégager les éléments récurrents qui pourraient contribuer à la définir de manière plus rigoureuse. Il en résulte que trois expressions ou termes apparaissent systématiquement dans la littérature. La pensée informatique semble ainsi inclure inévitablement la notion de « processus de pensée » ainsi que les termes « abstraction » (prise en compte simultanée de plusieurs niveaux d'abstraction) et « décomposition » (capacité à décomposer un problème complexe en plusieurs problèmes plus simples). D'autres termes sont aussi largement utilisés, mais de manière moins consensuelle. C'est le cas d'un ensemble de termes relatifs à des formes de pensée (« pensée logique », « pensée mathématique », « pensée heuristique », etc.) ou aux compétences de « résolution de problèmes ». En particulier, le terme « généralisation », qui désigne la capacité à transférer la résolution d'un problème particulier à tout un ensemble de problèmes plus généraux, et le terme « évaluation », qui correspond à la capacité à reconnaître et évaluer des résultats, sont des termes très souvent usités.

Selby et Woollard (2013) proposent une définition qui inclut ces mots qui sont les plus fréquemment utilisés. La pensée informatique est ainsi définie comme une activité, souvent orientée vers la production de quelque chose, associée (mais pas limitée) à la résolution de problèmes. Il s'agit, plus globalement, d'un processus cognitif qui reflète la capacité à penser en termes d'abstraction, de décomposition, d'évaluation, de généralisation et de manière algorithmique (capacité à écrire des instructions spécifiques et explicites pour chaque étape d'un processus). Cependant, ces processus identifiés par Selby et Woollard (2013) comme composantes principales de la pensée informatique ne sont pas nécessairement spécifiques à ce

domaine et peuvent être retrouvés dans un grand nombre d'activités cognitives (liées à la lecture ou aux mathématiques par exemple). Ceci traduit plus généralement la difficulté des chercheurs à définir la pensée informatique en des termes suffisamment précis et spécifiques.

La *Computer Science Teachers Association* et *l'International Society for Technology in Education* ont également tenté d'apporter une définition opérationnelle de la pensée informatique (CSTA & ISTE, 2011). La pensée informatique y est ainsi décrite comme un « processus de résolution de problèmes qui comprend les caractéristiques suivantes : formuler les problèmes d'une manière qui permette d'utiliser un ordinateur et d'autres outils pour aider à les résoudre ; organiser logiquement et analyser des données ; représenter des données par des abstractions telles que des modèles et des simulations ; automatiser des solutions à travers la pensée algorithmique ; identifier, analyser et mettre en œuvre des solutions possibles dans le but d'obtenir la combinaison la plus efficiente et efficace d'étapes et de ressources ; généraliser et transférer ce processus de résolution de problèmes à une grande variété de problèmes ».

Plus tard, l'ISTE a également affirmé que ce sont d'autres compétences bien connues (la créativité, la pensée algorithmique, la pensée critique, la résolution de problèmes, la pensée coopérative et les compétences communicationnelles) qui, considérées conjointement, constituent cette nouvelle compétence que l'on nomme pensée informatique (ISTE, 2015, citée dans Korkmaz et al., 2017).][8] Mohaghegh et McCauley (2016) estiment également que certaines compétences étudiées depuis de nombreuses années composent la pensée informatique : la pensée logique, la pensée algorithmique, l'efficacité ou l'efficience (un algorithme « efficace » est ici considéré comme nécessitant le moins d'étapes que possible pour résoudre un problème) et la pensée innovante.

Barr et Stephenson (2011) identifient également plusieurs concepts et capacités rattachés à la pensée informatique : la collecte, l'analyse et la représentation des données ; l'abstraction ; l'analyse et la validation de modèles ; l'automatisation ; les essais et vérifications (débogage, qui décrit la capacité à détecter et identifier des erreurs puis à les corriger) ; les algorithmes et procédures ; la décomposition de problèmes ; les structures de contrôle (boucles de répétition, instructions conditionnelles…) ; le parallélisme (réaliser un certain nombre d'étapes en même temps) ; la simulation. Pour eux, les capacités incluent : concevoir des solutions à des problèmes par l'abstraction, l'automatisation (capacité à automatiser l'exécution

[8] Sigayret, K., Blanc, N., & Tricot, A. (2022). L'apprentissage de la programmation: quels outils pour évaluer le développement de la pensée informatique à l'école?. *Enfance*, *4*(4), 479-500.

de la procédure lorsque cela est nécessaire pour résoudre des problèmes similaires), la création d'algorithmes, la collecte et l'analyse de données ; mettre en œuvre des conceptions (par la programmation quand cela est nécessaire) ; tester et déboguer ; modéliser, exécuter des simulations et effectuer des analyses de systèmes ; réfléchir sur la pratique et communiquer ; utiliser un certain vocabulaire ; reconnaître les abstractions et passer d'un niveau d'abstraction à l'autre ; innover, explorer et faire preuve de créativité dans toutes les disciplines ; résoudre des problèmes en groupe ; employer diverses stratégies d'apprentissage. Barr et Stephenson (2011) identifient par ailleurs certaines valeurs, motivations et attitudes et certains sentiments et stéréotypes, applicables à la pensée informatique, qui incluent : la confiance en soi face à la complexité ; la persistance face aux problèmes difficiles ; la capacité à gérer l'ambigüité ; la capacité à traiter des problèmes ouverts ; la mise de côté des différences pour travailler avec d'autres personnes dans l'optique d'atteindre un objectif commun ; la connaissance des forces et faiblesses de chacun dans le travail en équipe.

Kalelioglu et al. (2016) ont quant à eux construit un cadre conceptuel composé d'une combinaison d'éléments provenant d'une revue de littérature sur la pensée informatique et la résolution de problèmes. Leur proposition, proche de celle de Barr et Stephenson (2011), est intéressante, car elle définit différentes étapes chronologiques composées d'un ensemble de capacités à maîtriser. La pensée informatique est ainsi perçue comme un processus de résolution de problèmes qui consiste à : (1) identifier le problème (abstraction, décomposition) ; (2) rassembler, représenter et analyser des données, conceptualiser et reconnaître des formes (capacité d'identifier les règles ou modèles sous-jacents à la structure des données ou du problème) ; (3) générer, sélectionner et planifier des solutions (raisonnement mathématique, construction d'algorithmes et de procédures, parallélisme) ; (4) implémenter des solutions (automatisation, modélisation et simulation) ; (5) évaluer des solutions et continuer à les améliorer (test et débogage, généralisation).

Weintrop et al. (2016) ont aussi proposé une taxonomie pour l'intégration de la pensée informatique dans les apprentissages désignés sous l'acronyme anglais STEM (sciences, technologie, ingénierie, mathématiques). Pour les auteurs, quatre catégories de pratiques composent la pensée informatique : les pratiques en matière de données (collecte, création, manipulation, analyse et visualisation des données) ; la modélisation et la simulation (compréhension conceptuelle, test de solutions, conception, construction et évaluation de modèles) ; la résolution de problèmes computationnels (préparation de solutions, programmation, sélection d'outils, développement et évaluation de solutions, abstraction,

débogage) ; la pensée systémique (compréhension des fonctions d'un système perçu comme un tout, compréhension des relations entre les éléments d'un système, réflexion à plusieurs niveaux et perspectives, communication efficace et efficiente d'informations, définition du champ d'application d'un système et gestion de sa complexité).

Enfin, Shute et al. (2017) estiment qu'il existe six facettes principales de la pensée informatique : la décomposition ; l'abstraction (collecte et analyse de données, reconnaissance de formes, modélisation) ; l'algorithmique (conception d'algorithmes, parallélisme, efficience, automatisation) ; le débogage ; l'itération (répéter les processus de conception pour affiner les solutions, jusqu'à ce qu'un résultat « idéal » soit atteint) ; la généralisation. Pour Grover et Pea (2013), l'abstraction, qui consiste à « définir des modèles et à généraliser à partir d'exemples spécifiques » (p. 41), est la clé de voûte de la pensée informatique, ainsi qu'un moyen efficace de faire face à des problèmes complexes (Wing, 2011). Les auteurs listent les éléments largement reconnus selon eux comme faisant partie de la pensée informatique : l'abstraction et la généralisation de modèles ; le traitement systématique de l'information ; les systèmes de symboles et de représentation ; les notions algorithmiques ; la décomposition structurée de problèmes ; la pensée itérative, récursive et parallèle ; la logique conditionnelle ; l'efficacité et les contraintes de performance ; le débogage et la détection systématique d'erreurs.

Définitions pédagogiques

[On peut cependant remarquer que les définitions opérationnelles présentées ci-dessus ne sont pas spécifiques à l'apprentissage de la programmation. *A contrario*, Brennan et Resnick (2012) établissent un cadre théorique, comprenant 3 dimensions essentielles, qui permet de définir et d'enseigner la pensée informatique à l'école en s'appuyant sur le logiciel Scratch. Ils y distinguent :

– les concepts computationnels qui regroupent les notions à connaître en programmation : séquences (série d'étapes ou d'instructions), événements (action en provoquant une autre), boucles (répétition d'une partie du programme), opérateurs (expression mathématique ou logique), parallélisme (séquences d'instructions se produisant en même temps), conditions (exécution d'une partie du programme uniquement lorsqu'une certaine condition est remplie), données (stocker, retrouver et actualiser des valeurs).

– les pratiques computationnelles : être incrémental et itératif (planifier la conception d'un programme avant de l'implémenter), tester et déboguer (développer des stratégies pour faire

face aux problèmes et les anticiper), réutiliser et remixer (s'appuyer sur des programmes antérieurs ou réalisés par d'autres), abstraire et modulariser (construire un programme complexe en assemblant des parts plus petites).

– les perspectives computationnelles : s'exprimer, se connecter, questionner.

Dans leur revue systématique de la littérature consacrée à l'enseignement de la pensée informatique via Scratch, Zhang et Nouri (2019) identifient un ensemble de compétences qui parachèvent les travaux de Brennan et Resnick (2012) : la capacité à lire, interpréter et communiquer du code ; la pensée prédictive (prédire l'exécution d'un programme avant de le concevoir), les notions d'entrées et de sorties, l'utilisation de supports multimodaux (capacité à synchroniser des images, des sons, des actions…) ou encore les interactions humain-machine (interactivité du programme créé).][9] Les auteurs observent que le modèle proposé par Brennan et Resnick (2012), bien que pertinent, présente certaines difficultés en ce qui concerne l'évaluation de ces trois dimensions fondamentales. Les études sélectionnées évaluent fréquemment la maîtrise des concepts computationnels et très rarement les perspectives computationnelles. Les concepts sont en effet aisément évalués, que ce soit par des tests ou par l'analyse automatique des artéfacts d'apprentissage (code), là où les perspectives sont nettement plus difficiles à mesurer par le biais des moyens traditionnels d'évaluation.

Les modèles présentés précédemment se recoupent, du moins partiellement. Toutefois, la multiplication de ces propositions théoriques traduit l'absence de consensus de la communauté sur cette question, malgré une relative proximité entre celles-ci. Par ailleurs, la question de savoir si la pensée informatique est suffisamment distincte d'autres formes de pensée que les enfants développent peut légitimement être posée (Grover & Pea, 2013). À ce sujet, Román-González et al. (2017) ont mis en exergue des corrélations entre les résultats d'élèves du secondaire à un test censé mesurer la pensée informatique et un autre test évaluant quatre capacités mentales primaires (capacités de raisonnement, visuelles, verbales et numériques), qui constituent les fondements de l'intelligence selon un modèle assez ancien de Thurstone (1938). Mais les auteurs remarquent que seulement 27% de la variance des résultats au test de pensée informatique est expliqué par les capacités mentales primaires. Dans une autre étude de Román-González et al. (2018a), ils démontrent que 24% de la variance peut être expliquée par les facteurs non cognitifs de la personnalité (ouverture, extraversion…). Ceci les

[9] Sigayret, K., Blanc, N., & Tricot, A. (2022). L'apprentissage de la programmation: quels outils pour évaluer le développement de la pensée informatique à l'école?. *Enfance*, *4*(4), 479-500.

incite à conclure que la pensée informatique ne peut être réduite à ces capacités mentales primaires et qu'elle est un nouveau construit psychologique émergent qui doit être développé et évalué spécifiquement. Une étude similaire a obtenu des résultats sensiblement équivalents avec un échantillon plus réduit d'élèves d'école élémentaire en Allemagne (Tsarava et al., 2019).

[D'une manière plus générale, certains chercheurs considèrent la pensée informatique comme un ensemble de compétences, exigeant des apprenants qu'ils développent à la fois des connaissances spécifiques à un domaine (par exemple, la programmation) et des aptitudes à la résolution de problèmes (Tang et al., 2020). Ceux-ci suggèrent que ces compétences pourraient être utilisées pour résoudre des problèmes de la vie quotidienne, dans différents domaines et à tous les niveaux scolaires.][10] En outre, dans leur revue systématique consacrée, entre autres, aux différentes définitions de la pensée informatique, Taslibeyaz et al. (2020) constatent que la capacité à analyser et résoudre un problème demeure la compétence la plus fréquemment mentionnée dans la littérature pour décrire la pensée informatique, devant l'aptitude à programmer, même si les activités de programmation constituent le contexte d'apprentissage le plus souvent mis en œuvre pour développer la pensée informatique.

2.2) Pensée informatique, programmation et mathématiques

Des liens importants entre programmation et pensée informatique

La pensée informatique peut être considérée comme l'ensemble des capacités cognitives de résolution de problèmes qui déterminent notamment la capacité à programmer (Denning, 2017). En théorie, la programmation est la meilleure approche pour enseigner la pensée informatique (Buitrago Flórez et al., 2017) et une manière d'exposer les élèves à la pensée informatique en leur permettant de créer des « artéfacts computationnels » sous la forme de code ou de programmes informatiques (Lye & Koh, 2014). Une méta-analyse récente de Sun, Guo et Zhou (2022) confirme cet état de fait en révélant un effet modéré en faveur d'un impact positif de l'apprentissage de la programmation sur les compétences en pensée informatique (g = 0.61). De même, Nouri et al. (2020) ont réalisé des entretiens avec 19 enseignants rompus à l'enseignement de la programmation et leur ont demandé quelles compétences les élèves

[10] Sigayret, K., Blanc, N., & Tricot, A. (2022). L'apprentissage de la programmation: quels outils pour évaluer le développement de la pensée informatique à l'école?. *Enfance*, *4*(4), 479-500.

acquièrent lorsqu'ils participent à des activités de programmation. Les auteurs constatent que leurs réponses s'accordent plutôt bien avec le cadre conceptuel défini par Brennan et Resnick (2012) qui décompose la pensée informatique en un ensemble de concepts, pratiques et perspectives computationnelles. Les enseignants estiment également que ces activités permettent de développer des compétences plus générales (compétences et attitudes cognitives, compétences linguistiques, compétences collaboratives, résolution créative de problèmes).

Pour Lye et Koh (2014), la programmation représente davantage qu'une simple activité de codage, en particulier car elle implique une maîtrise de la pensée informatique de la part des apprenants. Ceci fait écho aux propos bien plus anciens de Dijkstra (1974) qui estimait avoir développé de nouvelles compétences mentales en programmant. Pour Webb et al. (2017), la différence entre programmation et pensée informatique est subtile. La pensée informatique ne nécessite pas de programmation, mais, en pratique, représenter la solution à un problème sous la forme d'un programme est pertinent pour évaluer et optimiser cette solution, car l'ordinateur se contentera d'exécuter à la lettre les instructions qui lui sont transmises. L'expression « pensée informatique » a d'ailleurs été l'objet de critiques, estimant qu'elle est trop ambigüe ou floue et n'est qu'une redéfinition de ce que l'on désignait autrefois simplement comme de la programmation (Denning, 2009). Les algorithmes sont ainsi des outils permettant de développer et d'exprimer des solutions à des problèmes computationnels et la programmation est un processus créatif qui produit des « artefacts computationnels » (*computational problems* et *computational artefacts* en anglais, deux expressions liées à la pensée informatique ou *computational thinking*), ce qui fait de la programmation une façon pertinente d'évaluer l'acquisition de la pensée informatique (Grover & Pea, 2013). Il est généralement admis que programmation, pensée informatique et résolution de problèmes sont étroitement liées (Barr et Stephenson, 2011).

Bien que certaines compétences de base en pensée informatique soient nécessaires pour développer des compétences plus pratiques en programmation et en codage, la pensée informatique représente généralement une capacité cognitive plus large (Tsarava et al., 2019). Pour Román-González et al. (2019), la pensée informatique fournit le cadre qui permet de se concentrer non pas sur la syntaxe de la programmation informatique, mais sur les processus mentaux sous-jacents et soutient l'idée que les élèves programment pour apprendre plutôt qu'ils apprennent pour programmer. Néanmoins, les processus impliqués dans les activités de programmation requièrent des compétences en résolution de problèmes (décomposition de

problèmes, utilisation d'algorithmes, automatisation, abstraction…) et participent en fin de compte à l'acquisition de la pensée informatique (Yadav et al., 2017).

Pensée informatique et programmation semblent liées. Cependant, il faut préciser que la programmation peut également conduire à l'acquisition de connaissances distinctes de cette pensée informatique, notamment concernant la maîtrise technique des outils utilisés. Pour Denning et al. (2017) , un élève obtenant de bons résultats aux tests censés évaluer les capacités d'abstraction ou de décomposition peut se révéler un piètre concepteur d'algorithmes, car apprendre à programmer nécessite, au-delà des compétences associées à la pensée informatique, de nombreuses heures de pratiques avant d'atteindre un certain niveau de maîtrise. Par ailleurs, la capacité à penser en termes d'étapes et d'actions dans la vie quotidienne ne correspond pas tout à fait à ce que peut représenter l'algorithme au sens informatique du terme, car, dans le cadre de la programmation, l'algorithme doit généralement être exécutable par une machine qui obéit à certaines limites et certaines exigences (Denning et al., 2017).

De même, la pensée informatique ne se développe pas uniquement par les activités de programmation (Parent, 2022) et constitue, selon Wing (2006), une attitude et une compétence utiles pour tout le monde et pas exclusivement les informaticiens. Pour Romero et al. (2018), il s'agit de passer au-delà du simple apprentissage de la programmation pour envisager un apprentissage par le biais de la programmation qui puisse permettre le développement de la pensée informatique. Psycharis et Kallia (2017) estiment même que les activités d'apprentissage devraient être conçues de manière à inclure les dimensions de la pensée informatique avant que les élèves ne commencent à programmer.

Un enseignement à préconiser dès le plus jeune âge ?

En outre, au vu de l'importance centrale que les outils numériques ont acquise dans nos vies quotidiennes, de nombreux auteurs affirment que l'introduction précoce d'activités visant à développer la littératie numérique, telles que la programmation et la pensée informatique, devrait être fortement encouragée, et ce dès l'école maternelle, y compris pour les élèves âgés de seulement 3 ans (Buitrago Flórez et al., 2017 ; Webb et al., 2017 ; Bers et al., 2019). L'objectif est de s'assurer que les jeunes enfants apprennent et se développent dans un environnement où les compétences en littératie numérique sont pleinement intégrées à l'enseignement plus traditionnel. Avec des outils et un cursus appropriés aux élèves les plus jeunes (basé sur le cadre de Zeng et al., 2023, qui adapte le cadre de Brennan & Resnick, 2012), les enseignants pourraient réduire la surcharge cognitive et permettre aux élèves d'acquérir les

compétences de base en pensée informatique (Su & Yang, 2023). Les concepts abordés doivent également être pensés pour ce jeune public, certains concepts (itérations, conditions) ou pratiques (réutiliser et remixer) étant trop complexes à appréhender à cet âge (Su & Yang, 2023 ; Zeng et al., 2023) d'autant qu'il y a actuellement un manque d'instruments d'évaluation valides et fiables pour eux (Su & Yang, 2023). Globalement, il serait potentiellement souhaitable que les élèves commencent à programmer avant l'âge de 12 ans en raison de la plus grande capacité à apprendre les langages naturels avant l'adolescence, des similitudes entre langages naturels et langages de programmation ou encore de la disponibilité des environnements d'apprentissage pensés pour le jeune public (Webb et al., 2017). Il ne faut cependant pas oublier que l'âge des élèves est logiquement positivement corrélé à leur niveau de compréhension et de maîtrise de la pensée informatique (Zhang & Nouri, 2019).

Intégrer la pensée informatique à l'enseignement des mathématiques ?

La pensée informatique est souvent considérée comme étant au cœur des disciplines désignées sous l'acronyme STEM dans la communauté éducative anglophone (Weintrop et al., 2016 ; Tang et al., 2020). La taille d'effet globale de l'intégration de la pensée informatique dans ces disciplines est importante (g = 0.85, Cheng et al., 2022). De nombreux exemples témoignent de son intégration en cours de mathématiques dans les cursus scolaires du monde entier. Il n'y a cependant pas de véritable consensus relatif aux relations qui unissent mathématiques et pensée informatique ainsi que sur les points de chevauchement entre les deux (Ye et al., 2023). Il est admis que lorsque les élèves s'engagent dans des activités mathématiques fondées sur la pensée informatique, ceux-ci ne se contentent pas de produire des programmes, mais construisent également des concepts en lien avec les mathématiques et la pensée informatique (Ye et al., 2023). Parent (2022) estime que la programmation se situe au croisement entre la pensée informatique, la pensée algorithmique et la pensée mathématique, toutes trois nécessaires pour apprendre à programmer. En d'autres termes, ces trois types de pensée existent distinctement, mais se rejoignent dans les activités de programmation.

Wu et Yang (2022) ont notamment réalisé une revue de littérature sur les liens entre pensée informatique et pensée mathématique. Pour eux, la pensée informatique aide les élèves à développer et à appliquer des concepts et des compétences mathématiques en utilisant des logiciels ou des programmes, la pensée mathématique aide les élèves à résoudre des problèmes en lien avec la pensée informatique, sans nécessairement inclure la programmation, et il existe bien une relation réciproque entre ces deux types de pensées ce qui facilite l'intégration de la

pensée informatique en mathématiques. Lv et al. (2022) remarquent cependant que, pour le moment, cette intégration des deux domaines est observée plus fréquemment à l'école élémentaire, pour des contenus limités à la géométrie, les nombres et les opérations, alors que les auteurs estiment que la pensée informatique doit être considérée comme partie intégrante de l'apprentissage des mathématiques. Ye et al. (2023) concluent leur revue systématique sur l'intégration de la pensée informatique en mathématiques en affirmant que l'enseignement des mathématiques fondé sur la pensée informatique implique un processus interactif et cyclique de raisonnement mathématique et de raisonnement computationnel. L'intégration totale des différents aspects de la pensée informatique dans l'enseignement des mathématiques reste complexe, en raison notamment d'un manque de formation des enseignants qui peinent à voir les connexions qui existent entre les deux domaines (Hickmott et al., 2018 ; Nordby et al., 2022). Cependant, plusieurs revues systématiques récentes concluent qu'il est bel et bien possible (et même bénéfique) d'intégrer la pensée informatique dans l'apprentissage des mathématiques, en particulier en s'appuyant sur le modèle de Brennan et Resnick (2012), présenté précédemment (Refvik & Bjerke, 2022). Cheng et al. (2022) nuancent toutefois ce constat en affirmant que l'intégration de la pensée informatique en mathématiques n'a pas d'effet significatif, en tout cas pour les études sélectionnées dans leur méta-analyse.

SYNTHÈSE

- Le **développement de la pensée informatique** constitue l'objectif majeur de l'enseignement de la programmation.
- Actuellement, il n'existe **pas de consensus concernant la définition de la pensée informatique**.
- Les définitions générales considèrent que la pensée informatique est une **façon particulière de résoudre des problèmes** qui s'appuie sur des **concepts issus de l'informatique**.
- Les définitions opérationnelles listent un ensemble de sous-compétences qui ne sont pas spécifiques à la programmation, en particulier les compétences d'**algorithmique**, d'**abstraction**, de **décomposition**, de **généralisation** ou d'**évaluation**.
- Les définitions pédagogiques estiment que l'apprentissage de la pensée informatique implique l'enseignement de **concepts computationnels** et le développement de **pratiques computationnelles** permettant l'ouverture de nouvelles **perspectives computationnelles**.
- **Programmation et pensée informatique entretiennent des liens forts,** même si apprendre à programmer permet de développer d'autres compétences que celles liées à la pensée informatique et que la pensée informatique ne s'exprime pas uniquement dans des activités de programmation.
- **Tous les enfants en âge d'être scolarisés** pourraient commencer à apprendre la pensée informatique.
- L'enseignement de la pensée informatique est souvent **intégré à l'enseignement des mathématiques**, avec des effets plus ou moins forts selon les études.

Chapitre 3. Différents outils pour enseigner la programmation

3.1) Enseigner la programmation à l'école

L'importance de la conception et de l'approche pédagogique

Pour Buitrago Flórez et al. (2017), un cours de programmation réussi doit à la fois définir spécifiquement les objectifs d'enseignement et d'apprentissage (en incluant la résolution de problèmes comme objectif), introduire des outils et des stratégies éducatives appropriés par rapport au contexte et à l'environnement d'apprentissage et considérer ou prendre en compte les différences qui existent entre les novices et les experts en programmation (en particulier les différences liées à leur niveau d'abstraction). Certains résultats expérimentaux font état d'une amélioration des performances d'apprentissage lorsque les élèves collaborent entre eux. Même si ceux-ci peuvent éprouver des difficultés dans la programmation collaborative, les élèves assistent généralement leurs pairs et s'entraident ce qui les rend plus à l'aise face au défi que représente l'apprentissage de cette discipline (Stewart & Baek, 2023). Umapathy et Ritzhaupt (2017) ont réalisé une méta-analyse, comprenant 18 articles, centrée sur la comparaison entre apprentissage de la programmation en solo et en duo. Leurs résultats témoignent d'un effet positif du travail collaboratif par paires d'élèves en ce qui concerne leurs résultats aux évaluations et leur taux de succès dans ce domaine, bien que les variables affectives demeurent inchangées, quel que soit le contexte (seul ou à deux).

En outre, Sun, Guo et Zhou (2022) ont réalisé une méta-analyse sur l'influence de divers facteurs de conception pédagogique d'activités de programmation sur l'amélioration des compétences en pensée informatique. Les auteurs ont montré plusieurs résultats intéressants :

- L'enseignement de la programmation avec un échantillon d'apprenants réduit présente un effet plus important et la taille d'effet diminue graduellement lorsque la taille de l'échantillon augmente.
- La taille de l'effet diminue progressivement lorsque la durée d'intervention augmente (en particulier après un mois) ce qui indique qu'une durée trop longue peut atténuer l'efficacité et l'impact des enseignements en programmation. Les auteurs recommandent une durée située entre une semaine et un mois, les interventions ayant suivi cette voie présentent les tailles d'effet les plus importantes.

- Les résultats ne confirment que partiellement les meilleures performances d'apprentissage en duo plutôt qu'en solo, car cet effet est ici marginal (non significatif).
- Parmi les différents outils utilisés pour enseigner la programmation, les langages de programmation visuels par blocs (que nous décrirons avec plus de précisions ultérieurement dans la section 3.3) de cette « Partie théorique ») semblent les plus efficaces.
- Les chercheurs et enseignants devraient considérer employer plusieurs méthodes pour évaluer les compétences en pensée informatique, au vu de sa multidimensionnalité.

Barr et Stephenson (2011) ont également identifié des stratégies et caractéristiques qui pourraient être considérées comme les plus propices au développement de la pensée informatique et les plus bénéfiques pour les apprentissages au sens large :

- L'augmentation de l'utilisation d'un vocabulaire « computationnel » lorsque celui-ci est approprié pour décrire des problèmes et des solutions.
- L'acceptation par les enseignants et élèves des tentatives qui échouent, en reconnaissant qu'un échec précoce peut, à terme, mener à des résultats positifs.
- Le travail d'équipe entre élèves avec l'usage explicite de la décomposition, l'abstraction (définie ici comme la capacité à simplifier du concret vers le général au fur et à mesure que des solutions sont développées), la négociation (travailler ensemble pour fusionner les parties de solution) et la construction du consensus (travailler pour construire la solidarité du groupe autour d'une idée ou d'une solution).

L'approche pédagogique mise en œuvre par l'enseignant et le contexte d'apprentissage (matériel utilisé inclus) impactent significativement l'acquisition des concepts et des compétences liés à la pensée informatique et à la programmation. Strawhacker et al. (2018) ont observé que les enseignants qui font preuve de flexibilité dans la planification des séances, de réactivité face aux besoins des élèves, d'un certain niveau d'expertise associé aux technologies employées et d'intérêt pour le développement d'une pensée indépendante et critique chez leurs élèves sont ceux qui génèrent une plus grande réussite en cours de programmation dans leur classe. Ceci révèle l'importance d'une approche pédagogique solide et éprouvée, en plus de l'utilisation d'outils adaptés, pour permettre le développement de la pensée informatique à l'école (Stewart & Baek, 2023).

Le défi que représente la maîtrise de la pensée informatique à travers les activités de programmation est bien connu. Il est ainsi nécessaire de s'adapter au niveau scolaire des élèves puisque, concernant ce qui peut être appris (et donc efficacement enseigné) à un certain niveau, les enseignants doivent impérativement prendre en compte les capacités mentales de leurs élèves (Zhang & Nouri, 2019), car le développement de la pensée informatique suit le développement cognitif global des enfants (Román-González et al., 2017). Pour Xu et al. (2019), les novices en programmation ont besoin de projets représentant un challenge et de problèmes authentiques pour faciliter leurs apprentissages. Il n'est donc pas suffisant d'initier les élèves à la programmation sans qu'il y ait d'enjeu pertinent pour eux. Les auteurs estiment que les méthodes d'enseignement qui consistent à « faire la leçon » sont à bannir et insistent sur la nécessité de permettre aux élèves d'apprendre par la pratique, l'enseignant étant plutôt dans la position d'un guide censé apporter une aide pour faire face aux difficultés que rencontrent les apprenants. Ceci se rapproche des méthodes constructionnistes qui affirment que la programmation et en particulier l'action de construire un système, la recherche d'obstacles et de problèmes à résoudre est la façon la plus efficace d'apprendre pour les enfants (Olabe et al., 2011). Ce processus serait également indiqué pour stimuler les capacités métacognitives des élèves et pour leur « apprendre à apprendre ».

Scherer et al. (2020) remarquent cependant qu'il n'y a pas de preuve satisfaisante quant à la supériorité d'une approche pédagogique par rapport à une autre dans l'enseignement de la programmation, que ce soient les activités collaboratives, l'instruction fondée sur le feedback, l'apprentissage fondé sur le jeu (*game-based learning*) ou toute autre approche identifiée dans leur méta-analyse. Les auteurs recommandent aux enseignants de ne pas se restreindre à une approche spécifique, mais plutôt de varier les expériences d'apprentissage et les pratiques d'enseignement tout en s'adaptant au contexte de l'environnement d'apprentissage. Cette recommandation est d'autant plus pertinente qu'il en existerait un assez grand nombre, sachant que Spolaôr et Benitti (2017) en recensent une quinzaine en ce qui concerne la seule robotique pédagogique.

D'ailleurs, d'autres interrogations demeurent concernant les approches pédagogiques qui seraient appropriées ou non. Des recherches empiriques sont requises pour déterminer quels outils soutiennent quelles stratégies d'apprentissage et quelles sont les stratégies qui permettent l'acquisition de la pensée informatique (Tikva & Tambouris, 2021). Cependant, l'instruction fondée sur des problèmes à résoudre dans des contextes pratiques (tels que la conception de

jeux), la robotique et la modélisation informatique sont les façons les plus populaires d'intégrer la pensée informatique dans les disciplines STEM (Wang et al., 2022).

Le rôle des outils utilisés pour enseigner la programmation

[L'enseignement de la programmation à l'école s'appuie sur différents outils censés faciliter la transmission des connaissances et compétences fondamentales dans cette discipline auprès d'un public jeune, en particulier les élèves d'école primaire. Les robots pédagogiques et les interfaces visuels basés sur des blocs à glisser-déposer (*drag & drop*) constituent l'essentiel des outils qui sont utilisés aujourd'hui pour rendre la programmation accessible et attrayante (Bocconi et al., 2016). Plusieurs auteurs ont également mis en évidence l'intérêt des activités débranchées comme moyen d'enseigner la programmation en éliminant complètement la charge cognitive liée à l'utilisation des machines et des outils numériques (Brackmann et al., 2017 ; Romero et al., 2018).][11] D'autres outils existent (micromondes, cartes programmables etc.), mais les tâches d'apprentissage débranchées de la programmation, les logiciels de programmation par blocs et la robotique éducative constituent les trois stratégies les plus couramment employées dans ce domaine à la fois dans la recherche et dans les pratiques des enseignants (Piedade & Dorotea, 2022). Pour cette raison majeure, cette thèse se focalisera uniquement sur ces trois outils.

Cependant, il ne faut pas oublier que l'enseignement de la programmation depuis la perspective d'un outil spécifique (apprendre Scratch, apprendre Python…) ne permet pas de développer une compréhension profonde et des aptitudes à la résolution de problèmes complexes (Caeli & Yadav, 2020). Des discussions sur la manière avec laquelle l'informatique doit être enseignée ont cours depuis les années 1960. Pour certains chercheurs, la programmation informatique est une pratique qui devrait être enseignée à tout le monde alors que d'autres estiment que la compréhension des concepts essentiels et la capacité à s'exprimer par des algorithmes sont plus fondamentales (Caeli & Yadav, 2020).

De même, Wing (2008) met en garde contre les pratiques qui laisseraient l'outil technologique (ordinateur, tablette, logiciel de programmation…) entraver la bonne compréhension des concepts mis en jeu. Il est essentiel de prévenir la situation dans laquelle les élèves croient avoir compris les concepts parce qu'ils ont appris à maîtriser un outil. De la

[11] Sigayret, K., Blanc, N., & Tricot, A. (2022). L'apprentissage de la programmation: quels outils pour évaluer le développement de la pensée informatique à l'école?. *Enfance*, *4*(4), 479-500.

même manière, il semble illusoire de penser maîtriser l'arithmétique parce qu'on a appris à utiliser une calculatrice. Autrement dit, l'élève qui a compris comment créer et exécuter un programme sur un logiciel n'a pas nécessairement compris comment décomposer un problème en sous-problèmes plus simples ni les règles de base de l'algorithmique. L'intégration efficace des outils et des concepts demeure un enjeu majeur de l'apprentissage de la programmation et de la pensée informatique.

3.2) Les activités de programmation débranchées

Apprendre à programmer sans outil numérique

[L'apprentissage débranché de la programmation fait appel au crayon et au papier ainsi qu'aux cartes, aux jeux de logique ou à des mouvements corporels simples pour représenter des concepts informatiques tels que les algorithmes ou les boucles. Il n'utilise pas d'outil numérique, c'est-à-dire de logiciels de programmation ni d'objets numériques (ordinateurs, tablettes, etc.)][12]. Des jeux de société tridimensionnels peuvent également être utilisés en guise d'activités débranchées pour initier les élèves les plus jeunes à la programmation (Bosgoed et al., 2022). Ces jeux constituent généralement des problèmes à résoudre en suivant manuellement des étapes séquentielles (Brackmann et al., 2017). Par exemple, dans le jeu « *Robot Turtle* », les joueurs disposent des cartes (fournies avec le jeu) dans le bon ordre pour réaliser un parcours. Ces cartes comportent des commandes écrites ou pictographiques (« gauche », « droite », « avancer », « reculer »…) et ce n'est qu'après les avoir organisées pour constituer un algorithme que les élèves sont autorisés à déplacer leur tortue manuellement, conformément à cet algorithme. D'autres jeux similaires ne nécessitent d'ailleurs aucun matériel particulier. Le « *jeu du robot idiot* » est ainsi une activité largement utilisée dans les classes françaises en raison de la facilité de sa mise en place. Un des élèves y joue le rôle d'un robot « idiot » (dans le sens où celui-ci ne peut qu'obéir aux ordres qui lui sont donnés) et d'autres élèves sont chargés de constituer un algorithme avec des instructions sur papier pour lui permettre, là encore, de réaliser un parcours dans la classe en atteignant un point d'arrivée défini à l'avance et en évitant les différents obstacles sur son chemin.

[12] Sigayret, K., Tricot, A., & Blanc, N. (2022). Unplugged or plugged-in programming learning: A comparative experimental study. *Computers & Education*, *184*, 104505.

De nombreuses activités débranchées peuvent également être déployées en extérieur, ce qui peut être utile pour joindre l'activité physique aux activités de résolution de problèmes (Bell et al., 2009). Cette approche présente aussi l'avantage de ne pas être sensible à la technophilie ou à la technophobie parfois constatées chez les élèves comme les enseignants (Romero et al., 2018). Par ailleurs, la pensée informatique prend racine dans des approches humaines non numériques (donc débranchées) de résolution de problèmes même si le courant dominant l'a progressivement amené vers des approches informatiques numériques branchées, très souvent axées sur la programmation (Caeli & Yadav, 2020).

Pour Bell et al. (2009), réaliser des activités à l'écart des ordinateurs pour enseigner l'informatique est efficace car les élèves associent généralement l'ordinateur à un outil (voire à un jouet) plutôt qu'à un objet d'étude en soi. En s'éloignant de l'ordinateur, ils seraient capables de réfléchir à des questions qu'ils ne se seraient pas posées par de simples activités de programmation, concernant notamment la complexité des algorithmes ou la conception d'interfaces sans nécessité d'acquérir des compétences techniques au préalable.

Bénéfices des activités débranchées

La programmation débranchée présenterait également certains avantages majeurs, résumés ci-après par Romero et al. (2018) :

- Réduire la charge cognitive. La charge cognitive liée à l'utilisation d'une machine, qui nécessite certaines compétences techniques essentielles, serait un premier obstacle à l'apprentissage réalisé directement sur logiciel. Il serait également plus aisé de distinguer d'une part la maîtrise des outils technologiques et d'autre part la compréhension des concepts essentiels pour apprendre la pensée informatique ou la programmation.
- Faciliter les interactions sensori-motrices. L'informatique débranchée s'accorderait davantage avec les théories de la cognition incarnée qui soulignent l'importance des actions sensori-motrices concrètes dans le processus d'apprentissage.
- Construire des analogies concrètes. Les activités débranchées permettraient de construire des analogies tangibles et concrètes en relation avec les notions abstraites rencontrées en algorithmique, ce qui en faciliterait l'appropriation.

Il existe différents types d'activités débranchées qui peuvent être mises en place : des activités procédurales où les élèves doivent suivre des instructions préétablies jusqu'aux

activités de type « tours de magie » qui reposent sur un algorithme impossible à deviner sans fournir l'explication aux élèves (Romero et al., 2018). Mais Romero et al. (2018) recommandent davantage les activités pour lesquelles les élèves sont directement mis en situation, avec des instructions minimales pour commencer, avant d'enrichir l'activité d'enjeux de plus en plus complexes.

Une étude exploratoire qualitative faisant suite à l'introduction de 6 séances de 90 minutes de programmation débranchée dans 26 écoles du nord des Pays-Bas estime que les leçons d'informatique débranchée constituent une alternative précieuse aux leçons de programmation plus classiques sur ordinateur (Bell & Vahrenhold, 2018). Par ailleurs, une approche débranchée de la programmation a pour particularité de mettre plus souvent l'accent sur l'algorithmique et la notion d'algorithme, concept essentiel pour acquérir les bases de la pensée informatique et prérequis à toute pratique avancée de la programmation (Bell & Vahrenhold, 2018 ; Lim & Chen, 2021).

Dans leur revue de littérature centrée sur les outils débranchés au service de l'enseignement de la pensée informatique, de Souza Oliveira et al. (2022) constatent que la vaste majorité des artefacts utilisés au cours d'activités débranchées étaient directement conçus par les chercheurs eux-mêmes, parfois à partir de matériel issu de ressources en ligne ou de manuels dédiés à l'enseignement de l'informatique débranché (en particulier le *Computer Science Unplugged Book*). Les auteurs remarquent également que plusieurs études font état de résultats très significatifs concernant le développement de la pensée informatique pour des classes d'âge très diverses, ce qui témoigne d'une absence de restriction liée à l'âge pour l'utilisation des activités débranchées. Cependant, la plupart des études a été conduite avec des échantillons réduits (maximum 40 élèves par groupe) dans des conditions quasi expérimentales qui manquent de contrôle et de randomisation dans la répartition des participants en différents groupes. Par ailleurs, le temps alloué à l'intervention, lorsqu'il était renseigné par les chercheurs, était généralement faible (jusqu'à 2 heures). Seule un peu plus de la moitié des études sélectionnées ont utilisé des méthodes quantitatives pour analyser les données recueillies.

Bell et Vahrenhold (2018) identifient également plusieurs bénéfices pédagogiques à l'usage des activités débranchées :

- La barrière de l'apprentissage de la programmation peut sembler un obstacle insurmontable. Cette barrière est supprimée en tant que condition préalable à l'apprentissage des grandes idées liées à l'informatique.
- Les élèves peuvent s'engager davantage dans des questions plus larges concernant l'informatique en général et éviter de développer l'idée fausse que ce sujet ne concerne que la programmation.
- Les activités débranchées peuvent être utilisées lorsque l'équipement informatique n'est pas disponible dans la classe ou, s'il l'est, d'éviter le pouvoir distracteur des ordinateurs ainsi que les problèmes techniques et logistiques liés à leur usage.
- Si le créneau horaire est court ou si les apprenants sont nombreux, les activités débranchées sont plus simples à mettre en place par rapport aux activités de programmation branchées.

Les activités débranchées pour développer la pensée informatique

[Récemment, plusieurs études expérimentales (ou quasi expérimentales) ont confirmé les avantages potentiels de ces activités débranchées pour accroître l'acquisition de compétences en informatique et en pensée informatique. Par exemple, Brackmann et al. (2017) ont fourni des preuves empiriques d'une augmentation des compétences en pensée informatique chez des élèves de 10 à 12 ans après 10 heures de pratique débranchée, par rapport à un groupe témoin inactif.][13] Leur étude inclut 73 élèves qui ont été testés avant et après une intervention de 5 semaines (à raison d'une séance de 2 heures par semaine) impliquant des activités débranchées pour le groupe expérimental. Pour les auteurs, cette étude confirme que la pensée informatique consiste principalement à maîtriser des compétences et des processus de résolution de problèmes dont le développement peut être déconnecté de la programmation informatique. Ils infèrent cependant que cette approche totalement débranchée peut présenter certaines limites et qu'il serait essentiel d'identifier le point à partir duquel cette approche perd son efficacité et nécessite une transition vers une approche branchée pour poursuivre le développement des compétences en pensée informatique.

[13] Sigayret, K., Tricot, A., & Blanc, N. (2022). Unplugged or plugged-in programming learning: A comparative experimental study. *Computers & Education*, *184*, 104505.

Ces résultats ont été confirmés par une étude similaire (Delal & Oner, 2020) qui impliquait une cinquantaine d'élèves de niveau 6e. Dix tâches issues du concours international Bebras (dont l'objectif est de promouvoir les compétences en pensée informatique des élèves de tous âges) ont été sélectionnées avec trois niveaux de difficulté (facile, moyen, difficile). Ces tâches nécessitaient des compétences généralement associées à la pensée informatique pour être résolues (abstraction, décomposition, raisonnement algorithmique et généralisation). La réalisation de ces tâches constituait une intervention de trois heures visant à améliorer les compétences en pensée informatique des élèves par le biais d'activités débranchées. L'évaluation des performances des élèves se base sur une comparaison pré-test / post-test à deux tests équivalents (composés d'une quinzaine de questions, issues également d'autres tâches Bebras) réalisée avant et après l'intervention. Une amélioration significative des compétences en pensée informatique est constatée après leur participation à l'intervention débranchée de trois heures. Aucune différence significative globale n'a pu être établie selon le genre des participants. Cependant, les auteurs constatent que, si les garçons s'améliorent davantage sur les questions faciles et moyennes, ce sont les filles qui présentent le plus gros gain de performance concernant les questions les plus difficiles.

De même, Threekunprapa et Yasri (2020) ont montré que la compréhension des concepts liés à la programmation et à la pensée informatique ainsi que le sentiment d'auto-efficacité des élèves avaient augmenté consécutivement à des activités débranchées. Leur étude portait sur 160 élèves dans le secondaire et les auteurs ont veillé à ce que les instructions papier utilisées par ces élèves reproduisent les avantages des blocs disponibles sur les logiciels éducatifs de programmation et à ce que les activités proposées imitent l'environnement ludique de ces logiciels. Dans ces conditions, ils recommandent aux enseignants d'utiliser cette forme de programmation débranchée pour favoriser la compréhension et l'engagement émotionnel des élèves.

D'autres études plus récentes encore confirment le potentiel intéressant des activités débranchées. Une amélioration des compétences liées à la pensée informatique a été observée chez des élèves de niveau 5e après six séances de programmation débranchée, en comparaison avec un groupe contrôle inactif n'ayant pas participé à ces séances (Sun, Hu & Zhou, 2021). Cet effet positif des activités de programmation débranchée a également été observé avec des élèves plus jeunes, âgés de 6 à 8 ans (Zhan, He, Yi et al, 2022).

Suárez et al. (2022) ont cependant conduit une étude présentant des résultats plus mitigés avec 48 élèves de niveau lycée visant à analyser le développement de la pensée informatique suite à 14 sessions débranchées de 2 heures. Les résultats, obtenus grâce à une comparaison pré-test / post-test quantitative fondée sur des tâches Bebras, montrent une légère augmentation des compétences en pensée algorithmique, décomposition et évaluation, toutes reliées à la maîtrise de la pensée informatique (selon Selby & Woollard, 2013, notamment). Les auteurs eux-mêmes considèrent que les résultats observés sont limités. Ils associent ces résultats mitigés à la difficulté pour les élèves à mettre en lien les données d'un problème et les schémas récurrents qu'ils sont censés identifier ainsi qu'à leur relative incapacité à distinguer les détails d'un problème qui sont importants de ceux qui peuvent être mis de côté. Ces deux capacités correspondent aux compétences de généralisation et d'abstraction, évaluées par les auteurs, pour lesquelles aucune amélioration significative n'a pu être observée.

De même, Caeli et Yadav (2020) considèrent qu'il n'est pas possible d'atteindre tout le potentiel de la pensée informatique par le biais d'approches uniquement débranchées. Pour comprendre l'ordinateur en tant qu'outil et les processus d'automatisation qui y sont associés, les approches branchées sont nécessaires. Les idées et solutions proposées sous forme d'algorithme doivent être testées avec une machine.

Battal et al. (2021) résument dans une revue de littérature les bénéfices observés de l'enseignement débranché de la programmation ou de l'informatique en général. De nombreuses études affirment aujourd'hui qu'il est possible d'enseigner les différentes sous-dimensions qui composent la pensée informatique ainsi que certains concepts essentiels en informatique. Des gains significatifs sont observés concernant la compréhension, la motivation, la réussite ou l'engagement des élèves. Les enseignants, en particulier ceux ne bénéficiant pas de suffisamment d'installations matérielles ou logicielles, semblent également tirer profit de cette façon d'introduire l'informatique dans leurs cours. En revanche, la recherche a également montré que certains enseignants et élèves présentent plus de réticence à adopter cette méthode plutôt que celle fondée sur les outils technologiques et les logiciels, en raison d'un manque d'efficacité supposé.

Plus critiques, Huang et Looi (2021) considèrent dans leur revue de littérature que les résultats sont limités concernant le développement de la pensée informatique par des activités débranchées. Certaines études suggèrent que les approches débranchées peuvent tout au plus être aussi efficaces que les approches branchées. Pour Huang et Looi (2021), la plupart d'entre

elles ne se fonde pas sur des évaluations appropriées et ne sont pas en mesure de séparer certaines variables confondues, car la programmation fait partie de l'intervention. Les résultats les plus convaincants, mettant en évidence la capacité des activités débranchées à développer certaines compétences telles que la capacité d'abstraction, sont limités à des contextes très spécifiques.

Accessibilité et potentiel des activités débranchées

[Aujourd'hui, une vaste collection d'activités débranchées est disponible en ligne et utilisée dans de nombreux pays, à des fins diverses, pour explorer l'enseignement de l'informatique sans avoir à apprendre la programmation informatique (Bell et al., 2009). Il convient également de rappeler qu'une approche débranchée de la programmation est la seule approche possible pour de nombreuses écoles dans le monde. C'est notamment le cas en Afrique ou en Asie, où de nombreuses écoles ne disposent pas de l'équipement informatique de base, mais aussi dans la plupart des pays européens où certaines zones rurales et reculées manquent encore de ressources (Brackmann et al., 2017). De plus, certains élèves rencontrent des difficultés parce qu'ils ont une attitude négative envers l'enseignement de l'informatique (Bell et al., 2009 ; Delal & Oner, 2020). Au contraire, les activités débranchées pourraient améliorer l'attitude positive et l'engagement émotionnel dans l'apprentissage de l'informatique (Threekunprapa & Yasri, 2020), rendant le processus d'apprentissage plus agréable, et diminuant les difficultés des élèves (Delal & Oner, 2020).][14]

Selon Lee (2020), l'initiation à la pensée informatique par le biais de moyens débranchés dépasse le cadre des activités spécifiquement dédiées à cet objectif. Elle décrit notamment comment l'utilisation de mots directionnels (« tourner à gauche », « avancer »…) et des représentations picturales qui y sont associées (sous forme de flèches par exemple) dans le vocabulaire quotidien de la classe dès le plus jeune âge (maternelle) peuvent contribuer à renforcer la compréhension de ces notions directionnelles. L'autrice propose également aux enseignants d'utiliser régulièrement des concepts séquentiels (aujourd'hui, demain – premier, deuxième, troisième…) ce qui favoriserait la compréhension de ce qu'est une progression étape par étape, concept central dans les tâches impliquant la programmation et la pensée informatique.

---

[14] Sigayret, K., Tricot, A., & Blanc, N. (2022). Unplugged or plugged-in programming learning: A comparative experimental study. *Computers & Education*, *184*, 104505.

Cependant, les activités débranchées, généralement appropriées pour initier les élèves à la pensée informatique, ont le plus souvent pour vocation de permettre à terme une transition vers des activités branchées (à l'aide de logiciels éducatifs ou de robots pédagogiques notamment). Pour Akiba (2022), il est possible d'affirmer que des activités hybrides branchées / débranchées constituent une méthode idéale pour promouvoir la littératie numérique en permettant aux enfants d'alterner facilement entre les aspects branchés et débranchés d'un ensemble de tâches à réaliser.

Romero et al. (2018) comparent de manière quasi expérimentale l'apprentissage de la programmation par des activités débranchées dans une condition avec un apprentissage plus classique via un logiciel à vocation éducative dans une seconde condition. L'échantillon d'élèves de CM2 mis à contribution reste assez faible (13 enfants dans la première condition, 10 dans la seconde). Ce protocole ne fait état d'aucune différence significative entre les deux groupes concernant leurs performances à un test de pensée informatique adapté aux élèves de ce niveau. Cette étude s'inscrit dans la continuité de celle réalisée par Messer et al. (2017), centrée sur des objectifs d'apprentissage différents, qui montre également qu'en ce qui concerne les activités branchées et débranchées, la question de la supériorité d'une approche sur une autre est encore loin d'être tranchée. Dans cette étude, l'usage ou non d'une tablette numérique pour réaliser des activités de programmation semble n'avoir aucun effet sur les performances. D'autres études plus rigoureuses sont nécessaires pour trancher sur l'efficacité des activités débranchées au service de l'apprentissage de la programmation et de la pensée informatique.

3.3) Logiciels de programmation éducatifs et langages visuels par blocs

Caractéristiques des langages visuels par blocs

La programmation par blocs est devenue au fil du temps de plus en plus utilisée pour initier les élèves à la programmation et à l'informatique en général (Weintrop, 2019). La programmation par blocs propose un langage visuel de programmation et se distingue des langages plus traditionnels fondés sur du texte en représentant les instructions sous la forme de pièces de puzzle à emboîter qui fournissent à l'utilisateur des indications visuelles sur la manière et l'endroit où les commandes peuvent être utilisées (Weintrop, 2019). Les blocs sont facilement glissés-déposés (*dragged & dropped)* les uns à côté des autres pour constituer le programme. Les environnements de programmation par blocs permettent également d'éviter les erreurs de syntaxe. En effet, si deux blocs ne peuvent logiquement être assemblés pour

fournir un programme valide, l'environnement empêche l'utilisateur de le faire (Weintrop, 2019). En outre, les zones de saisie de texte dans les blocs sont préremplies, ce qui permet à l'utilisateur de savoir quel type d'information est attendu (nombre, chaîne de caractères…). Il n'est d'ailleurs pas possible d'indiquer une information qui n'est pas acceptée par le logiciel (un nombre à la place d'un texte par exemple). Pour faciliter encore l'apprentissage des novices, les langages visuels par blocs présentent généralement l'ensemble des instructions disponibles sous la forme d'une palette de blocs, à faire glisser vers l'espace où le programme est conçu, organisés par catégories ou concepts et agrémentés de codes couleurs simplifiant la navigation et la recherche. Plus important encore, les langages de programmation par blocs rendent possible l'utilisation d'un langage naturel pour décrire simplement et précisément le résultat obtenu lorsque cette instruction ou ce bloc est utilisé (Weintrop, 2019).

Apports et limites des langages visuels par blocs

Weintrop et Wilensky (2015) ont tenté de mesurer l'impact des langages de programmation basés sur des blocs par rapport aux langages plus traditionnels basés sur du texte. Des élèves de lycée (n = 90) ont pris part aux expérimentations lors d'une séquence d'initiation à la programmation qui a duré 10 semaines. Ceux-ci ont commencé à pratiquer à l'aide d'un langage de programmation par blocs pendant 5 semaines, et sont ensuite passés sur un langage basé sur du texte pendant les 5 semaines suivantes. Les élèves ont été évalués par le biais d'un questionnaire comprenant les 28 mêmes questions au début, au milieu et à la fin de l'expérience. L'ordre et la modalité (blocs *vs.* textes) de chaque question changeaient entre les différentes passations du questionnaire. Les résultats révèlent que la modalité affecte les performances des élèves qui sont significativement plus performants avec des environnements de programmation par blocs sur les questions qui portaient sur la logique itérative (boucles), la logique conditionnelle ou les fonctions. Pour les questions relatives aux variables, les différences n'étaient pas significatives. Il est aussi intéressant de noter que les auteurs ont également proposé des questions de compréhension plus générales et plus complexes, sur lesquelles aucune différence significative n'est observée. Weintrop et Wilensky (2015) ne parviennent pas à savoir si cela est dû à la durée de l'intervention, trop courte pour influencer la compréhension profonde, ou à une absence réelle d'effet.

Les outils visuels de programmation sont susceptibles de réduire la charge cognitive associée à la lecture, la compréhension et la construction de programmes, ce qui aurait pour effet de les rendre plus accessibles pour des élèves novices en comparaison avec les langages

textuels (Sengupta et al., 2015, cités par Scherer et al., 2020). Les langages de programmation basés sur des blocs sont donc censés être mieux adaptés à l'apprentissage scolaire. Xu et al. (2019) ont cherché à vérifier cette affirmation en réalisant une méta-analyse comparant ces langages visuels à base de blocs avec des langages plus traditionnels à base de texte pour l'enseignement de la programmation aux élèves novices. Leur étude n'a mis en évidence qu'un faible effet (g = 0.25) en faveur des langages basés sur des blocs sur les variables cognitives (réussite, résolution de problèmes…) et un effet faible (g = 0.20) sur les variables affectives (satisfaction, confiance, motivation, etc.). Plus important encore, les auteurs ont déploré un manque de données empiriques suffisantes pour pouvoir se prononcer définitivement sur l'efficacité de ces langages visuels.

Toutefois, plus récemment, Scherer et al. (2020) ont constaté un effet modéré en faveur des outils de visualisation (g = 0.44), en particulier les logiciels de programmation visuelle tels que Scratch. Dans leur méta-analyse, les auteurs remarquent notamment que les tailles d'effet sont supérieures lorsque Scratch est utilisé, en comparaison avec d'autres langages plus anciens notamment (Logo par exemple). De leur point de vue, il manque cependant des preuves empiriques permettant d'expliquer les mécanismes qui sous-tendent l'efficacité accrue de ces outils de visualisation. De nouvelles études sont requises pour étudier les fondements cognitifs de la programmation selon l'utilisation des différents langages et pour évaluer la capacité des élèves à alterner entre ceux-ci. De même, Hu et al. (2021) rapportent dans leur méta-analyse que l'utilisation des langages de programmation par blocs a un effet positif modéré sur la réussite scolaire des élèves (g = 0.37 ou g = 0.47, selon si un modèle statistique à effets fixes ou à effets aléatoires est utilisé) en comparaison avec les langages basés sur du texte, probablement en raison de leur interface accessible et intuitive. La réussite scolaire est ici définie comme l'ensemble des performances qui se réfèrent aux connaissances déclaratives ou procédurales acquises à travers l'apprentissage de la programmation. Cet effet est plus marqué pour l'enseignement primaire ou au collège et décroît progressivement du lycée jusqu'à l'université. En effet, les langages visuels sont principalement conçus à destination des élèves jeunes, même si les auteurs remarquent paradoxalement que la plupart des études porte sur des étudiants à l'université, potentiellement en raison de l'accessibilité plus aisée de ce public pour les chercheurs et parce que l'enseignement obligatoire de la programmation ou de la pensée informatique est récent dans les cursus scolaires. Cet effet dépend également du logiciel de programmation utilisé, le logiciel Scratch bénéficiant d'un impact plus favorable par rapport aux deux autres langages étudiés (Alice et MIT App Inventor). Enfin, l'effet est plus marqué

lorsque le langage visuel est utilisé en complément d'un langage textuel classique, par rapport aux expériences dans lesquelles le langage visuel est une alternative au langage textuel (qui constitue alors un groupe contrôle).

Les langages visuels de programmation sont régulièrement utilisés pour initier les élèves à la programmation et sont souvent considérés comme un moyen de préparer efficacement ces élèves à une transition future vers des langages basés sur du texte (Xu et al., 2019), notamment dans le secondaire. Weintrop et Wilensky (2019) ont conduit une étude quasi expérimentale avec 60 élèves de lycée pour tester cette supposition et mesurer des différences concernant leurs apprentissages conceptuels, selon s'ils ont été initiés avec un langage visuel ou un langage basé sur du texte. Les élèves étaient répartis en deux groupes : un groupe expérimental qui a suivi 5 semaines d'apprentissage sur un logiciel visuel de programmation par blocs et un groupe contrôle directement impliqué dans des activités de programmation avec un langage basé sur du texte (Java). Suite à ces 5 semaines, l'ensemble des participants ont suivi 10 semaines d'enseignement sur Java. Les principaux résultats de cette étude révèlent qu'après avoir vu émerger des différences entre les deux groupes, ces différences s'estompent au fil du temps lorsque tous les élèves ont effectué une transition vers les langages textuels pour devenir finalement non significatives. Les auteurs interprètent cette observation en concluant que l'usage d'un langage visuel de programmation dans l'optique d'effectuer plus tard une transition vers un langage textuel plus professionnel n'est pas nécessairement plus efficace qu'un enseignement directement focalisé sur les langages textuels, bien que certains facteurs, tels que l'approche pédagogique utilisée, puissent également jouer un rôle modérateur.

L'exemple de Scratch, le langage visuel par blocs le plus populaire

De nombreux logiciels de programmation éducatifs basés sur des blocs sont disponibles gratuitement en ligne : Scratch, ScratchJr, Alice, Kodu, Construct, StarLogo, App Inventor, Greenfoot, Blockly, etc. Certains langages, comme Alice, sont très populaires, mais plutôt centrés sur l'apprentissage de la programmation classique et se sont particulièrement développés dans le supérieur (Weintrop & Wilensky, 2015). Dans cette thèse, nous nous concentrerons plus particulièrement sur Scratch qui est le logiciel de programmation éducatif le plus utilisé dans le monde (Lye & Koh, 2014 ; Tang et al., 2020). Il s'agit d'un langage de programmation visuel et d'un logiciel gratuit conçu par le MIT (Massachussetts Institute of Technology) permettant aux enfants de facilement créer leur propre contenu interactif sous la forme de jeux, de vidéos ou d'histoires multimédias (Olabe et al., 2011). Ce langage visuel par

blocs autorise le traitement en parallèle, c'est-à-dire que chaque projet peut être constitué de plusieurs objets (ou *sprites*) et chacun de ces objets peut avoir plusieurs programmes ou scripts qui s'exécutent en parallèle. Scratch a également une fonction sociale puisqu'il permet le partage de contenu entre les utilisateurs et favorise donc le travail collaboratif. Scratch est considéré comme *low floor, high ceilings* (Olabe et al., 2011 ; Weintrop, 2019), c'est-à-dire capable d'offrir des fonctionnalités facilement accessibles et rapidement maîtrisées (*low floor*) tout en garantissant la possibilité de concevoir des programmes complexes et sophistiqués (*high ceilings*). Le logiciel est également considéré comme *wide walls*, c'est-à-dire qu'il prend en charge et soutient une grande variété de types de programmes et est donc susceptible d'intéresser différents « profils » de programmeurs (Resnick et al., 2009 ; Weintrop, 2019). Enfin, Scratch est compatible avec certains robots pédagogiques, ce qui permet à l'utilisateur de connecter un robot à son ordinateur puis de créer un programme sur Scratch pour contrôler les mouvements et actions du robot (Olabe et al., 2011).

Scratch a été conçu pour les enfants à partir de 8 ans jusqu'à l'âge de 16 ans (avec un pic d'utilisation observé aux alentours de 12 ans), même s'il existe également une communauté non négligeable d'utilisateurs adultes, et s'adresse principalement à un public qui ne s'imaginait pas pratiquer un jour la programmation (Resnick et al., 2009). Il a été développé dans l'optique de permettre aux « Scratchers » d'apprendre des concepts importants en mathématiques et en informatique, tout en apprenant à penser de manière créative, à raisonner de façon systématique et à travailler en collaboration, des compétences jugées essentielles pour le XXIe siècle (Resnick et al., 2009). Les auteurs ont été guidés par trois principes fondamentaux lors de la phase de conception du logiciel : celui-ci devait être plus facile à utiliser, porteur de projets plus riches de sens pour les utilisateurs (à travers la diversité et les possibilités de personnalisation) et plus social (par la possibilité d'échanger et de partager des projets) que les autres environnements de programmation disponibles. À noter cependant que certains éléments graphiques présents dans Scratch pourraient également être considérés comme distracteurs vis-à-vis de la tâche à réaliser et donc nécessiter de la part des élèves certaines capacités d'inhibition cognitive (Çakiroğlu et al., 2018).

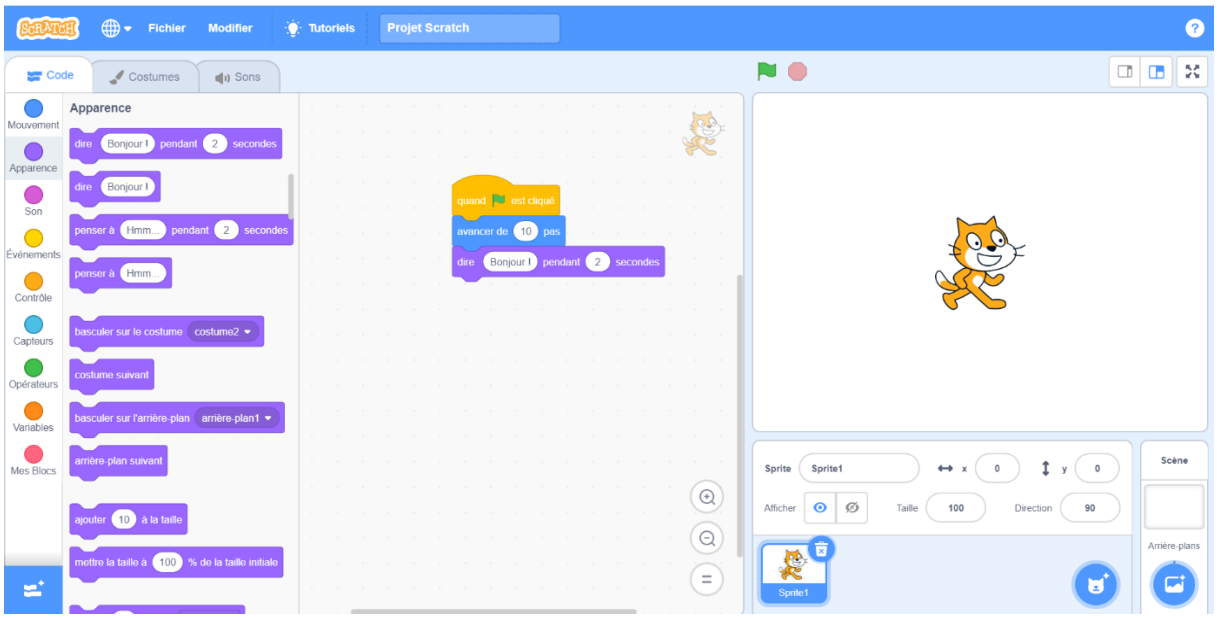
L'interface de Scratch est particulièrement simple à prendre en main. Maloney et al. (2010) décrivent dans le détail les particularités de cette interface (Figure 1). Il présente l'ensemble de ses fonctionnalités sur une fenêtre unique composée de plusieurs espaces distincts permettant d'assurer à l'utilisateur que les composants clés du logiciel sont toujours visibles. La fenêtre de Scratch comporte quatre volets principaux. Le volet de gauche est

constitué de la palette de blocs (ou d'instructions) disponibles divisée en 8 catégories qui se distinguent par des codes couleur (« Mouvement », « Apparence », « Son », « Contrôle », etc.). Le volet central affiche le programme (« script ») de l'objet (ou « *sprite* ») sélectionné.

En haut à droite se situe la « scène », un volet permettant de visualiser en temps réel l'exécution du programme et fournissant donc à l'utilisateur une rétroaction visuelle (ou feedback) sur le programme qu'il a réalisé. Cet aspect de l'interface est particulièrement intéressant, car, lorsque l'élève exécute son programme, les actions qui se produisent à l'écran lui permettent de potentiellement constater un écart entre le résultat attendu et celui que son programme a produit. L'inadéquation entre ses attentes et le résultat objectif visualisé dans la « scène » peut conduire l'élève à constater son erreur et l'inciter à poursuivre ses efforts pour obtenir le résultat souhaité. L'élève entre donc en interaction avec la machine qui lui fournit le feedback nécessaire, sans aucun jugement en cas d'erreurs, pour valider ou invalider le raisonnement qui l'a conduit à générer son programme. Enfin, le dernier volet en bas à droite affiche l'ensemble des *sprites* utilisés, celui actuellement sélectionné étant mis en évidence.

Figure 1

Interface de Scratch



Impact de Scratch sur la programmation et la pensée informatique

[Depuis 2009, Scratch est utilisé dans la plupart des études expérimentales portant sur la contribution des logiciels de programmation éducatifs. Sáez-López et al. (2016) ont montré

que l'utilisation de Scratch pouvait favoriser l'apprentissage des concepts de programmation, de la logique et des pratiques computationnelles chez les élèves de niveau CM2 ou 6e. Ces élèves ont trouvé le logiciel amusant, motivant et engageant. Les auteurs recommandent l'utilisation de Scratch à la fin de l'école primaire et au début du secondaire et estiment que la conception pédagogique mise en œuvre dans leur étude (apprentissage par projet) a favorisé une approche active qui n'est pas étrangère aux bons résultats observés. Certaines recherches menées auprès d'élèves de l'enseignement secondaire ont également montré la capacité de Scratch à les motiver à continuer à pratiquer la programmation, ce qui pourrait se traduire par une augmentation des inscriptions aux cours d'informatique (Armoni et al., 2015 ; Ouahbi et al., 2015).][15] Les élèves, interrogés sur leur pratique de la programmation avec des langages par blocs, considèrent que la forme et la disposition visuelle des blocs, la possibilité de parcourir les commandes disponibles, la facilité d'interaction par le « glisser-déposer » et le langage naturel des instructions sont utiles pour leurs apprentissages (Weintrop, 2019).

Ouahbi et al. (2015) ont constaté, à la lumière des recherches précédentes, que beaucoup d'élèves de niveau lycée jugent la programmation trop fastidieuse en raison des concepts et techniques complexes qui composent son enseignement. Les auteurs estiment qu'un changement d'environnement de programmation est souhaitable. Ils ont comparé deux groupes d'élèves, ayant participé à des activités de programmation sur Scratch ou à l'aide d'un langage plus traditionnel : Pascal. Les résultats sont très encourageants puisque près des deux tiers des élèves ayant utilisé Scratch souhaitent continuer à étudier la programmation contre seulement un peu plus de 10% dans le groupe « Pascal ». De même, 15% des élèves seulement ont trouvé les activités sur Scratch ennuyeuses contre presque 80% dans l'autre groupe.

Dans une étude impliquant 120 élèves au début de leur cursus au lycée, Armoni et al. (2015) ont montré que l'utilisation de Scratch permettait de mieux comprendre et maîtriser certains concepts jugés complexes (notamment la notion d'exécution répétée, ou boucle), ce qui semble en soi suffisant pour justifier son usage. Indépendamment des performances d'apprentissage, les enseignants ont noté un doublement des inscriptions en cours (optionnels) d'informatique dans les établissements concernés suite à l'introduction de Scratch. Ceux-ci ont également fait état d'une efficacité accrue et estiment qu'un concept qui nécessitait six leçons pour être appris en nécessite seulement trois avec Scratch. Cependant, ces résultats qualitatifs

[15] Sigayret, K., Tricot, A., & Blanc, N. (2022). Unplugged or plugged-in programming learning: A comparative experimental study. *Computers & Education*, *184*, 104505.

intéressants ne s'accordent pas parfaitement avec les résultats quantitatifs plus décevants qui ne montrent que peu de différences significatives que le logiciel soit utilisé ou non.

De même, Sukirman et al. (2022) ont initié des élèves de collège à la programmation à l'aide du logiciel Scratch *Jr.* (initialement pensé pour des élèves plus jeunes que la cible visée par le logiciel Scratch, y compris dès l'école maternelle) au cours d'une séquence pédagogique basée sur la conception d'un jeu. L'analyse des projets réalisés par les élèves révèle, selon les auteurs, que les environnements de programmation par blocs sont bel et bien susceptibles de développer la maîtrise de plusieurs concepts fondamentaux en programmation (séquences, boucles, instructions conditionnelles, variables…). Les participants de cette étude ont jugé que le logiciel permettait de facilement apprendre et se sont montrés confiants dans leurs apprentissages.

En outre, Tsai (2019) a conduit une étude auprès de 180 étudiants à l'université qui ont suivi 17 semaines de cours en programmation. Les étudiants étaient répartis en deux groupes dans lesquels ils utilisaient un langage de programmation visuelle pour la condition expérimentale et suivaient une introduction à l'informatique plus classique pour le groupe contrôle. Dans les deux groupes, l'enseignement portait sur trois concepts de base de programmation (séquences, conditions, boucles). L'ensemble des étudiants a été soumis à un questionnaire mesurant les compétences de base en programmation pré-test et post-test ainsi qu'à un test évaluant leur sentiment d'auto-efficacité. Les résultats démontrent un effet significatif de taille moyenne sur l'apprentissage des trois concepts (d = 0.45) en faveur du groupe expérimental (programmation visuelle), par rapport au groupe contrôle. En fonction des résultats obtenus au pré-test mesurant le sentiment d'auto-efficacité, les élèves ont été classés en trois catégories : sentiment d'auto-efficacité faible, modéré ou fort. L'analyse des résultats relatifs à cette variable indique que les étudiants ayant un sentiment d'auto-efficacité faible ou modéré ont davantage progressé dans leur compréhension des trois concepts étudiés en comparaison avec les étudiants présentant un fort sentiment d'auto-efficacité. Ce résultat est observable quel que soit le groupe (expérimental ou contrôle), mais est nettement plus prononcé pour le groupe expérimental qui a utilisé un langage de programmation visuel. Cette étude nous indique que les langages de programmation visuels par blocs sont particulièrement adaptés auprès d'un public de novices susceptibles de présenter un faible niveau de confiance dans leur capacité à apprendre et à maîtriser la programmation.

Par ailleurs, Cheng (2019) a mené une étude auprès de 431 élèves d'école primaire pour essayer d'évaluer l'acceptation des logiciels de programmation visuels par ce public. L'impact de quatre variables externes (sentiment d'auto-efficacité en informatique, influence sociale, assistance externe et encouragement externe) sur l'utilité perçue et la facilité d'utilisation perçue des langages visuels a été étudié. Le résultat majeur de cette recherche met en lumière l'effet positif significatif du sentiment d'auto-efficacité en informatique à la fois sur l'utilité perçue et la facilité d'utilisation perçue. Par ailleurs, des différences liées au genre sont observées : l'influence sociale influence davantage l'utilité perçue des langages visuels chez les garçons et les encouragements externes influencent davantage cette même utilité perçue chez les filles.

Récemment, l'étude de Piedade et Dorotea (2022) a montré que les activités de programmation via Scratch ont un impact positif sur les compétences et connaissances en lien avec la pensée informatique. Les auteurs ont comparé un groupe expérimental d'élèves de niveau CM1 ayant participé à une heure de programmation par semaine durant toute une année scolaire avec un groupe contrôle composé d'élèves de même niveau n'ayant pas été impliqué dans ces activités (n = 186 au total). Les résultats au test final montrent une différence significative en faveur du groupe expérimental avec un effet fort (d = 1.16). Aucune différence significative n'a pu être observée en fonction de l'âge, du genre ou du niveau des élèves dans différentes disciplines (mathématiques, science, portugais) bien que les auteurs remarquent que les garçons âgés de 9 ans sont les plus performants. De surcroit, Fagerlund et al. (2022) ont analysé 325 projets Scratch issus de 57 élèves de niveau CM1 ayant suivi une leçon de programmation par semaine pendant environ 4 mois. Leurs résultats mettent en exergue les différents schémas et les différentes constructions qui sous-tendent les programmes générés par les élèves ainsi que leurs liens avec certains concepts ou compétences généralement associés à la pensée informatique (abstraction, décomposition de problèmes…). Les auteurs observent que, parmi la multitude de projets analysés, tous ne mettent pas en évidence les mêmes sous-dimensions de la pensée informatique, ce qui les incite à préconiser l'introduction d'activités de programmation nombreuses et variées afin de cibler l'acquisition de la pensée informatique dans son ensemble.

Stewart et Baek (2023) ont depuis réalisé une revue de littérature centrée sur le développement de la pensée informatique à travers des activités de programmation sur Scratch. Ils estiment que la capacité de Scratch à favoriser les compétences liées à la pensée informatique ne fait plus aucun doute aujourd'hui, en particulier à travers la programmation de mini-jeux.

Les auteurs recommandent d'ailleurs d'intégrer la programmation sur Scratch à différentes disciplines, par exemple pour améliorer le raisonnement mathématique ou la capacité à raconter des histoires, afin d'optimiser les gains d'apprentissage.

Cependant, Jiang et Li (2021) présentent des conclusions plus mitigées concernant l'impact positif de Scratch. Dans leur étude, 336 élèves de niveau CM2 ont suivi des activités de programmation sur Scratch pendant 5 semaines (à raison de 45 minutes par semaine). Leur progression a été évaluée par une comparaison pré-test / post-test à l'aide d'un questionnaire visant à mesurer les compétences de haut niveau associées à la pensée informatique (selon le cadre défini par CSTA & ISTE, 2011, décrit précédemment). L'analyse des résultats témoigne d'une amélioration de la créativité, de la pensée critique et de la capacité à coopérer des élèves, mais aucune différence significative n'est observée concernant la pensée algorithmique et les capacités de résolution de problèmes, deux compétences très largement considérées comme constitutives de la pensée informatique. Les tailles d'effet sont relativement petites, mais les auteurs l'imputent à la courte durée de l'intervention et à la simplicité des tâches demandées qui n'ont pu influencer que modestement des compétences dites de « haut niveau ». Il est également important d'observer que les garçons ont globalement mieux réussi que les filles à la fois sur le pré-test et le post-test. *A contrario*, Gökçe et Yenmez (2022) ont mis en évidence une augmentation des compétences en pensée informatique auprès de 524 élèves de niveau CM2 / 6e après une intervention plus longue impliquant des activités sur Scratch (17 semaines d'instruction, deux séances de 40 minutes par semaine) et ce, y compris en ce qui concerne les capacités de résolution de problèmes (dont l'évaluation est ici fondée sur des scores à une échelle de pensée réflexive). Malgré les résultats mitigés de Jiang et Li (2021), il est raisonnable de penser que [les possibilités offertes par les logiciels de programmation éducatifs et le feedback immédiat de l'ordinateur pourraient faciliter l'acquisition des connaissances et compétences pratiques nécessaires à la résolution de problèmes algorithmiques, bien que la charge cognitive liée à l'utilisation d'une machine puisse également constituer un obstacle à un apprentissage adéquat.

Certaines limites ont également été identifiées par Aivaloglou et Hermans (2016) qui ont analysé près de 250 000 projets Scratch. Dans leurs conclusions, ils signalent qu'une grande proportion de mauvaises pratiques de programmation (*dead script,* c'est-à-dire un morceau de code qui n'est pas nécessaire et qui surcharge donc inutilement le programme) dans plus d'un quart des projets. Certaines fonctionnalités et certains concepts étaient rarement utilisés. Seuls

quelques projets présentaient une réelle complexité][16] et la plupart des programmes réalisés sont petits et limités. Leurs résultats confirment que Scratch est principalement utilisé comme un environnement dédié à une première exposition à la programmation. Par ailleurs, même si Scratch s'est souvent montré efficace pour permettre d'acquérir les concepts basiques de la pensée informatique (selon le modèle de Brennan & Resnick, 2012), certains concepts plus complexes constituent un véritable défi pour les élèves (boucles imbriquées, boucles conditionnelles, boucles et instructions conditionnelles incluant une ou plusieurs variables), malgré l'utilisation de ce langage visuel de programmation (Zhang & Nouri, 2019).

3.4) La robotique éducative

Origines et principes de la robotique éducative

D'une manière générale, la robotique éducative peut être définie comme l'interaction entre un dispositif robotique impliquant une structure mécanique, un système électronique comprenant généralement un logiciel de programmation et un étudiant, dans le but de promouvoir les processus cognitifs (Dos Reis et al., 2015). Elle est devenue récemment une pratique attrayante pour encourager l'intérêt et l'engagement des élèves envers l'étude des sciences et des technologies et développer certaines compétences. L'apparition récente de nouveaux robots pédagogiques sur le marché de l'éducation pose la question de l'efficacité de cet outil pour l'apprentissage de la programmation et, plus généralement, pour la maîtrise des compétences numériques qui font progressivement leur entrée dans les programmes scolaires.

Seymour Papert est le créateur du langage de programmation Logo, et aussi le pionnier des études sur la robotique éducative qui ont débuté il y a plus d'un demi-siècle. Papert (1980) suggère que l'apprentissage est plus efficace lorsque les élèves font des expériences et découvrent des choses par eux-mêmes. Il affirme également que les activités robotiques ont un potentiel immense d'amélioration des activités en classe (Papert, 1980). Les connaissances activement construites à l'appui de la réalisation d'un objet concret permettent de créer des liens vers d'autres connaissances dans d'autres domaines. La robotique éducative s'appuie sur les théories constructivistes de l'apprentissage qui soutiennent que l'élève construit ses connaissances par la manipulation. L'éducation fondée sur des principes constructivistes permet

[16] Sigayret, K., Tricot, A., & Blanc, N. (2022). Unplugged or plugged-in programming learning: A comparative experimental study. *Computers & Education*, *184*, 104505.

de sortir de la relation de subordination entre l'enseignant qui détient le savoir et l'élève auquel il le transmet. L'enseignant devient alors plutôt un accompagnateur, un conseiller et un organisateur de l'activité (Tocháček et al., 2016).

Le « constructionnisme » de Papert propose de fournir un contexte réel pour guider la génération de nouvelles connaissances (Papert 1980). Alors que le constructivisme se réfère principalement à des processus mentaux d'apprentissage, le constructionnisme suppose des processus qui sont bien physiques et qui impliquent la construction réelle d'objets tangibles (Ackermann, 2001). Cependant, il est utile de préciser que la construction de robots dans un but pédagogique bénéficie peut-être d'un potentiel relativement réduit. En effet, les apprenants semblent atteindre assez rapidement une limite quant aux types de robots qu'ils sont capables de créer et de programmer. Il devient vite très difficile pour un non-spécialiste de fabriquer un robot techniquement évolué et susceptible d'être programmé pour réaliser des tâches et actions complexes présentant un véritable intérêt (Alimsis & Kynigos, 2009).

De plus, les objections aux principes mêmes de l'éducation constructiviste sont nombreuses et bien étayées par la recherche. Cette méthode d'apprentissage induit le plus souvent un accompagnement minimal de la part du professeur. Or, après des décennies de débat, il semblerait que cette conception de l'enseignement ne soit pas si efficace en comparaison avec d'autres situations où le soutien pédagogique de l'élève par le professeur est plus marqué (Kirschner et al., 2006). Les auteurs cités ont réalisé une revue minutieuse des données empiriques disponibles sur ce sujet et concluent que les études les plus solides et les plus contrôlées sont presque unanimement en faveur d'un étayage pédagogique fort et direct. Seuls les étudiants bénéficiant de connaissances préalables considérables parviennent à obtenir des résultats similaires dans les deux conditions (étayage minimal et fort). Certaines études, cités par Kirschner et al. (2006) nous incitent même à penser qu'un soutien pédagogique insuffisant pourrait avoir des effets néfastes en favorisant l'acquisition d'idées fausses ou de connaissances incomplètes et désorganisées. De même, Tobias et Duffy (2009) regroupent dans leur ouvrage les contributions de nombreux chercheurs et confirment l'ensemble de ces observations tout en mettant en garde le lecteur sur le caractère intuitivement attrayant des techniques d'instruction constructivistes. Les arguments en faveur de cette méthode d'apprentissage relèvent, selon les auteurs, davantage de la rhétorique que du discours scientifique appuyé sur des recherches empiriques.

Le robot se situe entre l'objet technologique et l'outil éducationnel. Il se distingue d'un logiciel éducatif par sa nature d'objet réel qui s'oppose à celle d'artefact virtuel et intégré propre au logiciel (Gaudiello & Zibetti, 2013). Pour Gaudiello et Zibetti (2019), les robots sont des « artefacts de comportement » qui permettent de faire l'expérience de notions scientifiques « incarnées » par ce support physique, grâce au feedback immédiat renvoyé par le robot. Le robot peut être tour à tour un outil de perception et d'action, suscitant l'intérêt de l'élève sur les caractéristiques de l'environnement et introduisant le concept de boucle de perception / action, un outil de questionnement (sur l'informatique, sur la notion d'intelligence...), un outil de résolution de problème, un outil de modélisation et de simulation ou même un outil social, favorisant la communication et l'aptitude à résoudre les conflits. Les auteurs affirment que les robots sont attractifs de par leur nature de « jouets sérieux » et la manipulation des robots pourrait permettre une compréhension approfondie, au-delà de la simple mémorisation de notions, transférable dans d'autres domaines d'apprentissage. Il est aussi important de noter que le robot présente la particularité de pouvoir être considéré à la fois comme un objet d'étude en soi, un outil cognitif utilisé pour réaliser des expérimentations ainsi que comme un moyen pédagogique d'enseigner ou de développer certaines connaissances et compétences (Ospennikova et al., 2015).

Impact de la robotique éducative sur les apprentissages et la cognition

La thèse de Spach (2017) conclut sur la réelle possibilité de scénariser des apprentissages autour des robots programmables sans apporter de grandes modifications à l'organisation de la classe. L'approche de la robotique éducative, qui met en place une situation d'apprentissage, un problème à résoudre et une organisation de type projet, paraît favorable aux apprentissages même si elle est limitée actuellement par le manque de conceptualisation de ces apprentissages. L'utilisation de la robotique comme outil d'enseignement, fondée sur les théories de l'apprentissage, peut favoriser l'apprentissage de sujets non directement liés à la robotique, dont la programmation (Spolaôr & Benitti, 2017). Utilisée dans le cadre d'un apprentissage de la programmation, la robotique éducative permettrait une « mise à l'échelle » et une concrétisation des apprentissages tout en favorisant la mobilisation et la réutilisation de connaissances déjà acquises (Nijimbere et al., 2013). La programmation peut ainsi être utilisée à la fois pour former les élèves au raisonnement conditionnel (« Si... Alors... ») et pour donner aux enseignants l'occasion d'observer les étapes logiques du développement de la pensée de l'élève (Gaudiello & Zibetti, 2019). Pour Romero et Sanabria (2017), la robotique éducative serait particulièrement indiquée pour développer les « compétences du XXIe siècle » que sont

la pensée critique (autour des technologies et des enjeux éthiques de la robotique notamment), la collaboration, la capacité d'identifier un problème et de proposer des solutions créatives pour le résoudre ou la pensée informatique.

L'état de l'art proposé par Gaudiello et Zibetti (2013) fait émerger l'impact positif de la robotique éducative sur l'acquisition et le transfert de compétences grâce à la démarche de résolution de problèmes. Les auteurs regrettent cependant qu'il y ait encore peu d'études expérimentales et que la plupart d'entre elles mesure les différences observées avant et après les activités de robotique, sans comparaison avec un groupe contrôle. De même, aucun consensus n'a pu émerger jusqu'à présent concernant la manière d'implémenter des activités de robotique liées à la pensée informatique en classe, que ce soit au niveau de la durée de l'intervention, de la structure des apprentissages ou du contenu à enseigner aux élèves (Bakala et al., 2021).

Dans sa revue de littérature, Benitti (2012) recense 10 articles des années 2000 portant sur l'efficacité de la robotique éducative comme outil au service de l'enseignement, dont les deux études citées précédemment. Des élèves de 6 à 16 ans sont testés. Malheureusement, la majorité des études mentionnées proposent des protocoles qui ont lieu en dehors de la classe, à l'occasion de programmes extrascolaires ou de camps d'été, et qui impliquent des « tuteurs » chargés de guider les enfants dans leurs activités. Ces conditions diffèrent largement du contexte traditionnel de la classe. Les auteurs remarquent que la plupart des utilisations pédagogiques du robot est effectuée pour aider à la compréhension des disciplines STEM. Ils observent par ailleurs qu'il y a un déficit de données concernant les élèves âgés de 11-12 ans. Sur les études recensées, seules trois sont réalisées dans des conditions expérimentales avec échantillons randomisés et présence d'un groupe contrôle. Les résultats globaux sont considérés comme plutôt positifs et témoignent d'un certain potentiel pour la robotique éducative, même si les preuves empiriques en faveur de son efficacité demeurent insuffisantes.

La robotique éducative semble avoir un impact positif sur un ensemble de capacités cognitives, telles que l'habileté visuospatiale et les connaissances spatiales (Julià & Antolí, 2016 ; Mari, 2022), les capacités métacognitives, en particulier la régulation cognitive et l'autocontrôle (Socratous & Ioannou, 2019), les fonctions exécutives (Di Lieto et al., 2017) ou encore la créativité (Park et al., 2015 ; Zawieska & Duffy, 2015 ; Noh & Lee, 2020). Un effet positif, bien que faible et très hétérogène selon les études, est également observé sur les performances académiques (Athanasiou et al., 2018). Dans une revue de littérature plus récente,

Talan (2021) confirme cet effet positif, mais faible sur les résultats académiques dans différentes disciplines. L'auteur estime que la robotique éducative peut être plus efficace que d'autres méthodes traditionnelles, car elle facilite l'apprentissage en concrétisant les notions à apprendre et en faisant jouer aux élèves un rôle actif dans leurs apprentissages qui stimule leur intérêt, leur attention et leur motivation.

Kim et Lee (2016) ont comparé une classe de géométrie avec et sans robot pédagogique. Ils observent que les élèves manifestent plus d'intérêt et une participation plus importante dans le groupe qui fait usage des robots et ce, y compris trois mois après l'introduction de ceux-ci. Ils en concluent que les robots peuvent provoquer un impact affectif sur les élèves (et pas seulement cognitif au niveau des apprentissages). Ces robots auraient donc un certain potentiel en classe de géométrie, notamment pour stimuler l'intérêt et la motivation à apprendre. En outre, la robotique éducative présenterait l'avantage de permettre de maintenir une motivation intrinsèque élevée au cours des apprentissages comme en témoignent Kaloti-Hallak et al. (2015) qui ont suivi des élèves âgés de 13 à 15 ans et ont mesuré leur niveau de motivation avant et après la participation à une compétition de robotique, celle-ci étant restée stable au cours du temps.

Anwar et al. (2019) présentent une revue systématique de la littérature sur la robotique éducative prenant en compte 147 études publiées entre 2000 et 2018. Tous ces protocoles incluent des élèves dont le niveau scolaire varie de l'école primaire aux dernières années du lycée (*K-12 students*). Les études sont unanimes sur la capacité de la robotique éducative à soutenir une pédagogie centrée sur les apprentissages actifs, supposée améliorer l'expérience d'apprentissage et la motivation des élèves (Valls Pou et al., 2022). D'une manière générale, l'incorporation de la robotique éducative présenterait un avantage pour augmenter la réussite académique des étudiants ainsi que certaines compétences professionnelles (travail collaboratif, engagement, résolution de problèmes...). D'autres études explorent le potentiel de la robotique éducative pour faciliter la construction de nouveaux savoirs par le biais d'expériences pratiques qui suscitent des associations cognitives avec des connaissances et expériences préalables. La pratique de la robotique semble permettre aux élèves de mieux comprendre des concepts abstraits et de faciliter le transfert des connaissances acquises dans un environnement, dans un cadre nouveau ou face à un problème inédit, conformément à ce que d'autres études ont avancé (voir notamment Chiazzese et al., 2019). Les auteurs concluent cette revue de littérature en soulignant que des études supplémentaires sont nécessaires pour évaluer précisément l'efficacité relative de la robotique en comparaison avec d'autres méthodes d'instruction.

Une méta-analyse de Scherer et al. (2020) propose d'interroger cette fois l'usage des outils visuels et physiques disponibles pour enseigner la programmation aux élèves. L'utilisation de robots physiques programmables est supposée renforcer la compréhension de concepts informatiques ainsi que l'engagement et la motivation des étudiants. D'ailleurs, la plupart des outils robotiques est associée aux logiciels de programmation visuelle par blocs en raison de la simplicité de la syntaxe employée et de leur accessibilité qui encouragent les utilisateurs, notamment les jeunes élèves à s'engager rapidement et simplement dans la programmation (Moraiti et al., 2022 ; Stewart & Baek, 2023). L'analyse de Scherer et al. (2020) révèle une augmentation de la taille de l'effet lorsque les langages de programmation visuels ou la robotique sont utilisés, bien que l'influence des robots pédagogiques, et des outils physiques en général, soit encore trop peu étudiée expérimentalement.

Robotique éducative et développement de la pensée informatique

Selon Catlin et Woollard (2014), il y aurait une forte corrélation entre les principes de la robotique éducative et la pensée informatique qui auraient ainsi naturellement une relation symbiotique. Atmatzidou et Demetriadis (2016) ont conduit plusieurs séminaires d'entraînement à la robotique en Grèce auprès de 89 élèves de 15 ans et 75 élèves de 18 ans (de niveau équivalent aux élèves de 3e et terminale en France) pour en tester l'impact sur l'acquisition de la pensée informatique en fonction des différences d'âge et de genre. Deux heures par semaine étaient consacrées à des activités d'apprentissage, sans groupe témoin. Ils identifient cinq compétences majeures, proches de celles définies par Selby et Woollard (2013) et détaillées précédemment, pour maîtriser la pensée informatique : l'abstraction, la généralisation, l'algorithmique, la modularité (développement de processus autonomes et réutilisation de certaines commandes pour différents problèmes) et enfin la décomposition. Les principaux résultats de cette étude montrent que les élèves peuvent surmonter leurs difficultés initiales et développer avec succès leurs compétences en pensée informatique, à condition que le contexte pédagogique soit favorable et le temps consacré aux activités d'apprentissage suffisant. Les chercheurs insistent sur l'importance de la variable temporelle puisque les compétences évaluées lors des sessions finales se sont avérées, dans la plupart des cas, nettement meilleures que lors de la session initiale. En ce qui concerne les différences de genre, les filles semblent généralement avoir besoin de plus de temps pour atteindre le même niveau de compétence que les garçons, même si on observe que, quels que soient l'âge et le genre des élèves, ceux-ci finissent par atteindre un niveau de compétence plutôt équivalent. Une étude plus récente montre également que, dans le cadre d'une comparaison pré-test / post-test, aucune

différence significative n'a pu être observée entre les filles et les garçons en ce qui concerne les performances d'apprentissage de la pensée informatique par la robotique (Noh & Lee, 2020). Les résultats de Atmatzidou et Demetriadis (2016) confirment et poursuivent les recherches menées précédemment par les mêmes auteurs (Atmatzidou et Demetriadis, 2014) qui montraient déjà une amélioration de la compréhension des principaux concepts de la pensée informatique au fil de la pratique de la robotique. Les concepts d'abstraction et de généralisation semblaient cependant être plus complexes à assimiler que les trois autres.

Plus récemment, Ioannou et Makridou (2018) reprennent l'étude de Atmatzidou et Demetradis (2016) dans une revue de littérature comprenant neuf études sur la robotique éducative et son impact sur l'acquisition de la pensée informatique. Les études sélectionnées montrent globalement un effet positif avec des gains significatifs observés sur la résolution de problèmes, les compétences en collaboration et communication, et l'intérêt vis-à-vis de la programmation et des disciplines STEM en général. Les auteurs minimisent néanmoins ces résultats en soulignant certains problèmes à la base de la recherche en robotique éducative tels que l'absence de consensus sur la définition de la notion de pensée informatique parmi les chercheurs et surtout l'absence d'évaluation des outils censés mesurer les compétences des élèves en pensée informatique. Ils en concluent que la recherche sur la pensée informatique et son développement par le biais des activités de robotique en est encore à ses balbutiements. Des études expérimentales solides sont requises.

Angeli et Valanides (2020) se sont intéressées à l'utilisation de robots Bee-Bot pour initier les jeunes enfants à la pensée informatique. Les auteurs ont étudié plus particulièrement le rôle de l'étayage pédagogique (*scaffolding*), c'est-à-dire du soutien apporté par l'enseignant tout au long du processus d'apprentissage. Les limitations de la mémoire de travail chez les jeunes enfants imposent une charge cognitive externe élevée et impliquent donc le besoin de trouver des moyens efficaces d'étayer l'apprentissage de manière appropriée. Des élèves de 5-6 ans (fin de maternelle, n = 50) ont participé à leur étude et ont été testés pré-test / post-test sans comparaison avec un groupe contrôle. Un accroissement significatif des compétences en pensée informatique des enfants est observé. La compréhension des relations spatiales entre les objets est aussi significativement améliorée par la pratique de la robotique, notamment en ce qui concerne les référents spatiaux « gauche » et « droite » qui sont moins utilisés dans le vocabulaire quotidien des enfants que les référents « aller vers l'avant » ou « aller vers l'arrière ». Quelques années plus tôt, Bers et al. (2014) avaient déjà démontré qu'il était tout à fait possible pour les plus jeunes enfants (en maternelle) de participer à des activités de

résolution de problèmes et d'acquérir des concepts fondamentaux de la pensée informatique par la manipulation de robots éducatifs. Il est aussi intéressant de noter, à la lumière des résultats obtenus par Angeli et Valanides (2020), que les enfants d'un très jeune âge sont capables de faire face à une tâche complexe d'apprentissage en la décomposant en plusieurs sous-tâches plus simples, ce qui est un des éléments de la pensée informatique (i.e., décomposition). D'autres études mettent en avant la possibilité de développer la pensée informatique par des activités de robotique dès le plus jeune âge, avec des élèves de 3 à 6 ans (García-Valcárcel-Muñoz-Repiso & Caballero-González, 2019) ou à des âges plus avancés, avec des élèves de 8 à 10 ans (Chiazzese et al., 2019 ; Noh & Lee, 2020) ou des élèves au début du secondaire (Valls Pou et al., 2022).

Ching et al. (2018) considèrent, après avoir fait le tour de la littérature, qu'il existe des preuves solides de la possibilité et des résultats positifs du développement de la pensée informatique par la programmation chez les jeunes apprenants en utilisant diverses technologies éducatives, dont les robots pédagogiques. Ils remarquent d'ailleurs que la plupart des dispositifs en question est susceptible d'englober à la fois un agent d'exécution (le robot par exemple) et un outil de programmation doté d'une interface graphique (de style glisser-déposer) pour le commander. L'utilisation de ce type d'interface permet aux apprenants de se concentrer prioritairement sur les concepts plutôt que sur la syntaxe des langages de programmation. Enfin, Zhang et al. (2021) ont réalisé une méta-analyse sur l'impact de la robotique éducative sur la pensée informatique auprès des élèves de la maternelle jusqu'à la fin du lycée (*K-12 students*) sur la période 2010-2021. Ils concluent à un effet positif modéré des robots sur l'acquisition de compétences liées à la pensée informatique (*Standardized Mean Differences* ou SMD = 0.46). Cependant, la taille de l'effet diminue lorsque le temps de l'expérimentation augmente. En outre, aucun effet significatif n'a pu être observé concernant l'attitude des élèves envers les disciplines STEM qui tend à décroître avec l'âge, mais qui ne semble pas impactée par la pratique de la robotique éducative.

Dans le même registre, Ching et Hsu (2023) ont mené une revue de littérature sur l'enseignement de la pensée informatique par le biais d'activités de robotique éducative auprès d'élèves dont les âges varient de 3 à 11 ans. Les auteurs observent que la robotique est efficace pour enseigner certains concepts computationnels simples (séquences) aux élèves de niveau maternelle au niveau CE1 ainsi que pour leur transmettre certaines compétences de débogage. D'autres concepts plus complexes (boucles, conditions…) peuvent également être enseignés pour des élèves de niveau plus élevé (du CE2 à la 6e). La pensée algorithmique se développerait

plutôt aux alentours du niveau CM2. Les auteurs insistent sur la nécessité d'adapter le matériel robotique au niveau des élèves. Les robots les plus simples, présentant peu de fonctionnalités et fournissant un environnement de programmation sans écran, sont à préconiser pour les élèves les plus jeunes (de la maternelle au CE1) alors que les élèves plus âgés deviennent progressivement capables d'utiliser des kits de robotique complexes aux fonctionnalités nombreuses et plus avancées, en s'appuyant sur des langages de programmation visuels par blocs. Globalement, il est cependant regrettable d'observer un manque d'évaluation systématique et de données expérimentales fiables dans le domaine de la robotique éducative (souligné par Alimsis, 2013 ; Anwar et al., 2019 ; Bakala et al., 2021 ; Ioannou & Makridou, 2018 ; Scherer et al., 2020 ; Zhang et al., 2021).

SYNTHÈSE

- **La conception pédagogique des séquences peut jouer un rôle majeur** dans l'apprentissage de la programmation et de la pensée informatique. Activités collaboratives, nombre réduit d'élèves, durée moyenne de l'intervention, langages visuels, réactivité, expertise et intérêt du professeur sont des facteurs qui influencent positivement l'expérience d'apprentissage.
- Différents outils peuvent être utilisés, mais il est essentiel de **ne pas centrer l'apprentissage sur la maîtrise de ces outils** en particulier.
- **Les activités débranchées** ne s'appuient sur **aucun outil numérique.** Elles sont donc plus **accessibles** et présentent des **avantages potentiels** (diminution de la charge cognitive, implication du corps et de la motricité, concrétisation des concepts abstraits…).
- **Les logiciels de programmation visuels par blocs** proposent une **interface plus simple, intuitive et adaptée aux jeunes élèves** en comparaison aux langages de programmation traditionnels. Le **feedback** renvoyé par la machine pourrait faciliter les apprentissages.
- **Les robots pédagogiques** bénéficient de certains atouts des approches branchées et débranchées : **manipulation d'un objet concret** pour représenter les processus abstraits en programmation, **feedback**…

Chapitre 4. Psychologie et apprentissage de la programmation

Dans le chapitre précédent, nous avons abordé les trois thématiques principales de cette thèse : l'apprentissage de la programmation, le développement de la pensée informatique et les caractéristiques des outils utilisés pour enseigner la programmation et la pensée informatique. Dans le présent chapitre, il s'agira de s'intéresser à ce qui constitue l'apprentissage de cette discipline d'un point de vue psychologique, et aux différents facteurs qui pourraient l'influencer. Nous verrons qu'apprendre la programmation et la pensée informatique implique à la fois l'acquisition de certaines connaissances conceptuelles et le développement de certaines compétences. La charge cognitive associée aux outils utilisés pourrait agir sur cet apprentissage, tout comme la faculté de ces outils à susciter des interactions motrices. L'impact potentiel des outils sur certaines variables conatives (motivation, sentiment d'auto-efficacité, attitude envers les sciences) sera exploré, ainsi que les liens entre ces variables conatives et les performances d'apprentissage des élèves. Enfin, nous étudierons en quoi le genre des élèves peut être un facteur qui influence significativement l'apprentissage de cette discipline, et en quoi le choix des outils pourrait favoriser une plus grande égalité entre filles et garçons.

4.1) Des connaissances et des processus cognitifs à maîtriser

[Dans leur revue de la littérature, Robins et al. (2003) distinguent deux composantes de l'apprentissage de la programmation qui sont d'un côté les connaissances et de l'autre les stratégies employées pour programmer et résoudre un problème. La question est alors de déterminer pourquoi et comment différentes stratégies peuvent être apprises, et comment celles-ci sont liées aux connaissances sous-jacentes. De même, Robins et al. (2003) différencient la compréhension d'un programme et la capacité à en générer un. Pour les auteurs, il est clair que ces deux derniers aspects sont liés, la compréhension jouant un rôle important dans la création d'un programme, mais ils suggèrent par ailleurs que les capacités des individus en ce qui concerne ces deux tâches ne sont pas forcément bien corrélées. Les tâches d'évaluation fondées sur la mesure de la compréhension d'un programme ne sont pas nécessairement de bons indicateurs de la capacité à concevoir et rédiger des programmes.][17]

[17] Sigayret, K., Blanc, N., & Tricot, A. (2022). L'apprentissage de la programmation: quels outils pour évaluer le développement de la pensée informatique à l'école?. *Enfance*, *4*(4), 479-500.

De nombreux modèles ont émergé au cours des deux dernières décennies pour essayer de décrire précisément les connaissances et compétences associées à la pensée informatique (Barr & Stephenson, 2011 ; Brennan & Resnick, 2012 ; CSTA & ISTE, 2011 ; Grover & Pea, 2013 ; Selby & Woollard, 2013 ; Kalelioglu et al., 2016 ; Weintrop et al., 2016 ; Shute et al., 2017 ; Zhang & Nouri, 2019), dont le développement est perçu comme l'objectif majeur des activités de programmation (Barr & Stephenson, 2011 ; Grover & Pea, 2013 ; Lye & Koh, 2014 ; Yadav et al., 2017), bien qu'aucune définition unique ne fasse actuellement consensus sur cette pensée informatique (Brennan & Resnick, 2012 ; Mohaghegh & McCauley, 2016 ; Román-González et al., 2017 ; Moreno-León et al., 2018 ; Tang et al., 2020). Pour les besoins de cette thèse, nous essaierons ici d'expliciter notre conception de la pensée informatique, afin de préciser le cadre dans lequel nos recherches s'inscrivent. Il ne sera pas question de proposer un nouveau modèle, mais plutôt de s'inspirer des différents modèles existants pour proposer un contexte adapté à nos questions de recherche. L'objectif de cette thèse est de mesurer l'impact de plusieurs outils d'enseignement sur le développement de la pensée informatique, dans le cadre d'un apprentissage de la programmation, et se distingue donc des nombreuses études qui répondent aujourd'hui aux demandes des décideurs politiques internationaux concernant les recherches portant sur l'acquisition de la pensée informatique en général, indépendamment des disciplines concernées par l'intégration de cette pensée informatique (Cheng, 2019 ; Moreno-León & Robles, 2016 ; Shute et al., 2017).

Pour la plupart des auteurs, la pensée informatique regroupe un ensemble de compétences (*computational thinking skills*). [Selon Van Lint (2016), la compétence se distingue de la connaissance qui regroupe un ensemble de concepts, d'énoncés, de définitions, de règles, de formules ou de faits à comprendre et à mémoriser, et de la procédure qui désigne un enchaînement organisé d'actions automatisées qui restent les mêmes dans toutes les situations. La compétence est plutôt une aptitude qui se révèle dans des situations complexes nouvelles qui nécessitent une certaine adaptation. Cette aptitude repose sur la mise en œuvre de connaissances et de procédures adéquates dans une situation donnée (Van Lint, 2016). Pour évaluer une compétence, il convient ainsi de placer l'élève face à une situation nouvelle et complexe (c'est-à-dire mobilisant plusieurs ressources) qui exige, de la part de l'apprenant, l'accomplissement d'une tâche.

L'identification des concepts et des compétences de résolution de problèmes, qui constituent la maîtrise de la pensée informatique, est une première étape vers notre capacité à les évaluer. L'apprentissage peut être considéré comme un changement relativement permanent

de nos connaissances (Mayer, 2011). Or, certains de ces changements ne sont pas directement observables : ils sont invisibles. Pour résoudre ce problème, Tricot et Musial (2020) proposent de considérer que l'apprentissage concerne un couple {tâche ; connaissance} dans lequel la connaissance est ce qui permet de réaliser une tâche et, réciproquement, la réalisation d'une tâche permet l'acquisition d'une connaissance. La réalisation d'une tâche au moment de l'évaluation peut donc être révélatrice d'une connaissance intrinsèquement invisible. Les mêmes auteurs définissent la compétence comme un ensemble {tâche ; connaissances théoriquement nécessaires à la tâche ; connaissances issues de la tâche}. La compétence correspond ainsi à la « capacité d'un individu à réaliser une certaine tâche et pour laquelle on peut décrire l'ensemble des connaissances nécessaires à la réalisation de cette tâche », certaines de ces connaissances étant spécifiquement issues de la pratique de la tâche.

En ce qui concerne l'apprentissage de la programmation dans l'enseignement scolaire français, Tchounikine (2017) identifie 5 notions fondamentales qui constituent les bases que les élèves doivent connaître et comprendre : la notion d'algorithme (ou de programme), la notion d'instruction (ou d'action), la notion de condition (« Si… alors… »), la notion de boucle (« répéter … ») et, éventuellement, la notion de variable.][18] Même si le vocabulaire employé n'est pas tout à fait identique, les notions proposées regroupent en partie les concepts computationnels de Brennan et Resnick (2012), notamment les boucles, les conditions, les séquences (proches des notions d'algorithme et d'instruction) et les données (qui incluent la manipulation de variables). Certains concepts présents chez Brennan et Resnick (2012) sont absents, mais Nouri et al. (2020) constatent dans une étude concernant les enseignants que ceux-ci ne mettent pas en évidence les concepts d'événements, de parallélisme et d'opérateurs comme acquis de la pensée informatique, probablement parce qu'ils manquent de connaissances à ce sujet. En l'état, les 5 notions fondamentales proposées par Tchounikine (2017) nous semblent suffisantes dans le cadre d'un enseignement de la programmation au cycle 3 auprès d'élèves novices.

En outre, Tchounikine (2017) présente les compétences sous-jacentes à la pensée informatique de la manière suivante : « savoir décomposer un problème en sous-problèmes plus simples ; savoir réfléchir aux tâches à accomplir pour résoudre un problème en termes d'étapes et d'actions ; savoir décrire les problèmes et les solutions à différents niveaux d'abstraction, ce

[18] Sigayret, K., Blanc, N., & Tricot, A. (2022). L'apprentissage de la programmation: quels outils pour évaluer le développement de la pensée informatique à l'école?. *Enfance*, *4*(4), 479-500.

qui permet d'identifier des similitudes entre problèmes et, par la suite, de pouvoir réutiliser des éléments de solution » (p. 11). Là encore, les compétences proposées par Tchounikine (2017) recoupent un certain nombre de compétences assez largement représentées dans les définitions opérationnelles de la pensée informatique (Barr & Stephenson, 2011 ; Brennan & Resnick, 2012 ; CSTA & ISTE, 2011 ; Kalelioglu et al, 2016 ; Selby & Woollard, 2013 ; Shute et al, 2017 ; Weintrop et al., 2016) : la décomposition, la pensée algorithmique, l'abstraction, la reconnaissance de formes et la généralisation. La proposition de Tchounikine (2017) a le mérite de recentrer les débats sur un nombre réduit de compétences (là où Tikva et Tambouris, 2021 dénombrent 60 éléments présents dans les différents cadres conceptuels de la pensée informatique), présentées de manière simple et explicite et pouvant être décrites dans une seule phrase, qui constitue un enchaînement logique. Cependant, il nous semble important d'ajouter à ces compétences essentielles celle de l'évaluation, présente sous différentes dénominations, dans un certain nombre de modèles : capacité à reconnaître et évaluer des résultats (Selby & Woollard, 2013), identifier, analyser et mettre en œuvre des solutions possibles dans le but d'obtenir la combinaison la plus efficiente et efficace d'étapes et de ressources (CSTA & ISTE, 2011), débogage ou capacité à détecter et identifier des erreurs puis à les corriger (Barr & Stephenson, 2011), capacité à évaluer des solutions et à continuer de les améliorer (Kalelioglu et al., 2016), développement, évaluation de solutions et débogage (Weintrop et al., 2016), efficience et débogage (Shute et al., 2017), tester et déboguer (Brennan & Resnick, 2012).

[En suivant le modèle de Tricot et Musial (2020), dans le cadre d'un apprentissage de la programmation en contexte scolaire tel que défini par Tchounikine (2017), et en considérant que la pensée informatique est la compétence majeure que l'enseignement de la programmation cherche à développer, nous pouvons en tirer la conclusion suivante : la pensée informatique peut être considérée comme un ensemble regroupant une tâche (la résolution de problèmes algorithmiques), des concepts nécessaires à la réalisation de cette tâche (les notions fondamentales en programmation) et des connaissances pratiques issues de cette tâche qui permettent au sujet d'utiliser les stratégies adéquates pour y faire face (décomposition, généralisation, abstraction…). Dans ce modèle, les connaissances pratiques issues des activités de programmation permettent le développement des compétences identifiées précédemment comme relevant de la maîtrise de la pensée informatique et de son utilisation. Ce modèle est également conforme à l'idée que la pensée informatique s'appuie sur des concepts issus de l'informatique (Wing, 2006) en considérant que les notions fondamentales en programmation constituent les connaissances théoriques nécessaires à la résolution de problèmes

algorithmiques, du moins dans le cadre de l'apprentissage de la programmation.][19] Comme pour tout apprentissage, le processus de développement de la pensée informatique implique nécessairement la mobilisation de ressources cognitives de la part de l'apprenant. C'est d'autant plus vrai que nous avons constaté qu'il s'agit ici d'un apprentissage complexe, faisant appel à des connaissances et compétences multiples. La charge cognitive associée à une situation d'apprentissage est l'un des facteurs déterminants des performances des élèves.

4.2) La charge cognitive d'une situation d'apprentissage

La théorie de la charge cognitive « vise à expliquer comment la charge de traitement de l'information induite par les tâches d'apprentissage peut affecter la capacité des étudiants à traiter de nouvelles informations et à construire des stratégies d'apprentissage » (Sweller et al., 2019, p. 1-2). Elle se base sur le principe selon lequel le traitement cognitif est fortement contraint par les limites de notre mémoire de travail qui ne peut traiter qu'un nombre réduit d'éléments simultanément. Lorsque la charge cognitive est trop élevée, elle entrave l'apprentissage et le transfert vers la mémoire à long terme. Il convient donc d'adapter les méthodes et les supports d'enseignement afin d'éviter toute distraction qui viendrait surcharger le traitement cognitif inutilement. La théorie de la charge cognitive permet ainsi en pratique de trouver un juste équilibre en supprimant les informations non pertinentes et en mettant l'accent sur celles qui le sont, dans l'optique d'optimiser les situations d'apprentissage. Cette théorie a vu le jour dans les années 1980 et une description complète de celle-ci a été réalisée par Sweller (1988), ce qui a suscité un intérêt croissant et toujours très actuel de l'ensemble des chercheurs dans le domaine de la psychologie de l'éducation. Sweller et al. (1998) proposent de distinguer trois catégories de charge cognitive qu'il est essentiel de prendre en compte lors d'une conception pédagogique efficace (voir aussi la discussion de Sweller et al., 2019).

La charge cognitive intrinsèque désigne la complexité des informations traitées et fait référence aux interactions entre les différents éléments à prendre en compte. Il est difficile d'estimer la complexité d'un ensemble d'informations. Celle-ci va dépendre notamment des connaissances antérieures de la personne censée comprendre et mémoriser ces informations. Plus une information implique pour l'apprenant un degré élevé d'interactions entre plusieurs éléments, plus il y a de risque de surcharge cognitive. Il n'existe donc que deux façons de

[19] Sigayret, K., Blanc, N., & Tricot, A. (2022). L'apprentissage de la programmation: quels outils pour évaluer le développement de la pensée informatique à l'école?. *Enfance*, *4*(4), 479-500.

diminuer la charge cognitive intrinsèque : en changeant ce qui doit être appris ou en changeant le niveau d'expertise de l'apprenant.

La charge cognitive extrinsèque est déterminée par la manière dont l'information est présentée et correspond à l'ensemble des informations présentes dans la situation d'apprentissage, mais inutiles à cet apprentissage. Contrairement à la charge cognitive intrinsèque, elle peut être modifiée (et diminuée) par la conception pédagogique des séquences d'enseignement. L'interactivité des éléments est ici également à prendre en compte. Une conception pédagogique efficace réduit au minimum les interactions entre les éléments à apprendre afin d'éviter la surcharge cognitive.

Enfin, la charge cognitive essentielle (ou charge cognitive utile) désigne la charge cognitive nécessaire à l'apprentissage et fait donc référence aux ressources en mémoire de travail qui sont consacrées à la gestion de la charge cognitive intrinsèque plutôt qu'à celle de la charge cognitive extrinsèque. La charge cognitive intrinsèque et la charge cognitive essentielle entretiennent donc des liens étroits. La caractérisation exacte de cette charge cognitive essentielle fait encore l'objet de débats et de suppositions (Sweller et al., 2019). À l'heure actuelle, la charge cognitive essentielle est considérée comme permettant la redistribution des ressources de la mémoire de travail depuis les activités extrinsèques à la tâche d'apprentissage vers les activités intrinsèques à cette tâche d'apprentissage. En d'autres termes, la charge cognitive essentielle a une fonction de redistribution des aspects extrinsèques vers les aspects intrinsèques de la tâche.

À notre connaissance, aucune étude n'a véritablement mesuré empiriquement la charge cognitive d'une situation d'apprentissage de la programmation et de la pensée informatique, en fonction notamment des outils employés pour l'enseigner. Certains arguments théoriques ont pu cependant être avancés concernant l'impact de cette charge cognitive. Par exemple, Romero et al. (2018) supposent que les activités débranchées, parce qu'elles n'impliquent aucun outil numérique, permettraient logiquement de supprimer la charge cognitive extrinsèque liée à la manipulation de ces outils. Il serait toutefois intéressant d'évaluer et de comparer plusieurs situations d'apprentissage impliquant des outils différents afin de constater d'éventuelles différences significatives et d'en observer l'impact sur les performances d'apprentissage des élèves.

Par leur syntaxe exigeante et très restrictive, les langages de programmation classiques imposent une charge cognitive extrinsèque importante qui explique probablement pourquoi

cette discipline a été longtemps considérée comme trop complexe pour être enseignée à l'école, du moins auprès d'un public jeune. Mais, comme nous l'avons vu précédemment, l'arrivée des nouveaux langages de programmation visuels par blocs permet justement de réduire cette charge cognitive extrinsèque. Ceux-ci ne nécessitent aucun apprentissage d'une syntaxe particulière, les instructions à utiliser sont écrites dans un langage naturel et connu des élèves. La possibilité de glisser-déposer ces instructions n'implique pas non plus la nécessité de maîtriser parfaitement l'écriture au clavier. Ainsi, nous pouvons supposer que ces langages de programmation visuels par blocs, même s'ils supposent la maîtrise de certaines compétences en informatique, ne génèrent pas une situation d'apprentissage pour laquelle la charge cognitive extrinsèque serait trop élevée, y compris pour des élèves d'école élémentaire.

L'utilisation des robots pédagogiques est souvent préconisée en raison du caractère attrayant et motivant de ces robots, ainsi que de leur capacité à concrétiser les notions souvent abstraites mises en œuvre dans l'apprentissage de la programmation. Cependant, peu de chercheurs s'interrogent sur la charge cognitive que peut représenter une situation d'apprentissage pour laquelle des robots seraient mobilisés, qui plus est si ceux-ci sont utilisés conjointement avec un logiciel de programmation visuel par blocs. La gestion simultanée d'un logiciel et du robot qui y est associé pourrait impliquer une charge cognitive extrinsèque élevée, de par les différents aspects techniques indépendants des objectifs d'apprentissage à prendre en considération. Néanmoins, cette capacité à concrétiser les éléments abstraits de la programmation (par exemple les notions de variable ou de condition) pourrait potentiellement réduire les interactions entre les éléments essentiels aux apprentissages. Il convient donc de reconnaître que l'impact de la charge cognitive, associée aux différents outils utilisés pour enseigner la programmation et la pensée informatique qui constituent le cœur de cette thèse, demeure une question en suspens, bien qu'il semble évident que la charge cognitive est une variable fondamentale qui doit très certainement jouer un rôle dans la capacité de ces outils à transmettre efficacement les notions et compétences essentielles dans ce domaine. D'autres facteurs pourraient également jouer un rôle prépondérant dans l'optimisation des apprentissages. En effet, selon la perspective de la cognition incarnée et située, les interactions motrices sont supposées faciliter l'acquisition des connaissances et compétences à enseigner aux élèves.

4.3) Différentes implications du corps et de la motricité

Les trois outils que nous avons choisis d'étudier dans le cadre de cette thèse impliquent nécessairement des usages différents du corps et des compétences motrices. Les activités débranchées s'appuient très souvent sur la manipulation d'objets ou les mouvements du corps pour représenter l'exécution d'un algorithme. De même, la robotique éducative présente la particularité de concilier l'usage d'un outil numérique avec la manipulation d'un objet physique et concret : le robot. Pour les activités réalisées exclusivement sur logiciel (sans association avec un robot), le rôle de la motricité est de fait réduit.

L'utilisation d'objets concrets pour favoriser l'enseignement de concepts abstraits n'est pas nouvelle. Bara et al. (2004) ont pu mettre en évidence le rôle positif joué par la manipulation des objets concrets sur les capacités de lecture des enfants de maternelle. Les principaux auteurs de cette étude ont poursuivi leurs investigations (Bara & Gentaz, 2011) en s'interrogeant sur l'impact de la modalité haptique sur la capacité de 44 enfants, âgés de 5 ou 6 ans, à maîtriser l'écriture manuscrite. L'exploration visuohaptique des lettres (plutôt que seulement visuelle) entraînait de meilleures performances en ce qui concerne la reconnaissance des lettres et la capacité à les recopier.

Dans d'autres domaines, et notamment en mathématiques, Kaminsky et al. (2006) considèrent qu'il y n'y a pas suffisamment de preuves empiriques pour affirmer la supériorité de ces objets physiques et matériels sur d'autres modèles de représentation plus abstraits ou symboliques. Les preuves recueillies seraient souvent limitées à des démonstrations de compréhension des élèves dans une situation particulière. Or, les auteurs jugent que le but de l'apprentissage de concepts abstraits n'est pas la compréhension d'une situation, mais bien le transfert de cette compréhension à toutes les situations qui impliquent ce concept. Les élèves doivent aussi apprendre à construire une représentation duelle d'un objet, à la fois en tant qu'objet physique et concret et en tant que représentation symbolique de quelque chose d'autre. Kaminsky et al. (2006) ont donc mis en place une étude avec 19 élèves de niveau 6e répartis aléatoirement en deux groupes. Les résultats de cette étude montrent que les enfants n'ont pas nécessairement besoin d'une situation concrète pour acquérir un concept abstrait.

Cependant, pour McNeil et al. (2009), il est largement admis que les élèves obtiennent de meilleurs résultats lorsque les problèmes de mathématiques sont présentés dans le cadre de situations pratiques et réelles plutôt que lorsqu'ils sont présentés dans des contextes scolaires

traditionnels. Le contexte pratique permettrait par exemple de réduire la charge cognitive en facilitant la répartition des tâches en éléments maniables, de simuler mentalement des concepts qui seraient trop complexes à aborder de manière seulement abstraite ou encore de rendre concrets des problèmes, de les mettre en relation avec le monde « réel », extérieur à l'école.

Carbonneau et al. (2013) ont mené une méta-analyse sur les preuves empiriques en faveur de l'usage des objets matériels pour l'apprentissage des mathématiques. Ce sont 55 études distinctes qui ont été sélectionnées et chacune d'entre elles proposait une comparaison avec un groupe contrôle dans lequel l'apprentissage était réalisé uniquement à l'aide de symboles abstraits. Au total, ce sont les résultats de plus de 7000 élèves, tous âges confondus (de la primaire à l'université), qui sont évalués et analysés. Globalement, cette analyse met en exergue un effet faible à modéré en faveur de l'apprentissage des mathématiques par le biais des objets concrets. Cependant, les données obtenues semblent indiquer que l'usage d'objets matériels a un impact différent selon les objectifs d'apprentissage. Un effet de taille importante peut notamment être observé en ce qui concerne la rétention de l'information, mais, lorsqu'il s'agît de la résolution de problèmes, du transfert des connaissances acquises à des situations nouvelles ou encore de la capacité à justifier ses réponses en fournissant une explication sur la méthode employée, l'effet mesuré est plus faible.

Dans un domaine qui nous intéresse davantage, Sullivan et Heffernan (2016) définissent le matériel de manipulation éducatif comme un référent concret à des concepts abstraits. Ils s'intéressent plus particulièrement au matériel de manipulation numérique qui possède une capacité informatique embarquée, tels que les kits de construction robotique par exemple. Ces kits peuvent avoir deux fonctions distinctes, à la fois comme matériel d'initiation à la robotique, mais aussi comme outils susceptibles de favoriser l'acquisition de concepts abstraits dans des domaines proches de la pensée informatique. Pour les auteurs, la relation directe entre le programme qui est exécuté et le mouvement du robot facilite la compréhension de la nature duelle de l'objet, en tant qu'objet physique et concret se déplaçant dans l'espace d'une part, mais aussi en tant que représentation d'un programme, qui n'est rien d'autre qu'une suite de symboles et de processus abstraits. Le feedback immédiat permis par la robotique induit l'observation de divergences entre le mouvement attendu du robot et son mouvement réel, ce qui favorise l'association entre le robot concret et le programme abstrait dans l'esprit de l'enfant, tout en incitant l'élève à générer des inférences sur les raisons qui sous-tendent ces divergences, et donc à potentiellement améliorer ses aptitudes en matière de raisonnement causal. Dans le domaine de la robotique éducative, la quasi-transparence entre les mouvements et déplacements du robot

et le contenu du programme qui les détermine pourrait faciliter la capacité des apprenants à intégrer la dimension duelle de l'objet qui agit comme incarnation concrète d'un processus abstrait sous-jacent. Cette compréhension pourrait avoir un impact sur l'acquisition des notions et des compétences en programmation et, plus généralement, des compétences liées à la pensée informatique. En outre, les performances d'apprentissage ne sont pas les seules variables qui peuvent présenter un intérêt pour nos études. L'amélioration du vécu subjectif des élèves, notamment en termes de motivation, pourrait également représenter un objectif souhaitable.

4.4) Quel impact sur le vécu subjectif des élèves ?

Motivation

La définition de la motivation est complexe, car celle-ci prend racine dans une grande variété de théories distinctes (McGill, 2012). L'objectif ne sera pas ici de les décrire ou de prendre position en faveur de l'une d'entre elles. Un distinction est généralement opérée entre la motivation intrinsèque qui désigne la satisfaction inhérente associée à l'apprentissage en lui-même de la motivation extrinsèque qui considère l'apprentissage comme un moyen d'obtenir une satisfaction autre (une bonne note à un examen, une carrière réussie, etc.). La motivation intrinsèque serait au cœur de l'éducation (Kaloti-Hallak et al., 2015) et permettrait d'apprécier davantage l'activité d'apprendre en comparaison avec la motivation extrinsèque (Deci & Ryan, 2000). La motivation a tendance à diminuer tout au long du parcours scolaire, pouvant mener jusqu'au décrochage et à l'échec scolaire (Galand, 2006). Par ailleurs, la motivation n'est pas binaire, car un élève n'est pas simplement motivé ou non. Il est plus ou moins motivé par une préoccupation particulière, un aspect particulier d'une tâche et par plusieurs choses qui peuvent être contradictoires (Galand, 2006).

Très souvent, les outils numériques sont considérés comme vecteurs d'une plus grande motivation pour les élèves (Tricot, 2020). Ils sont réputés amusants ou adaptés à la génération des élèves, mais également vecteurs de certaines qualités intrinsèques censées améliorer la motivation des élèves (interactivité, multimodalité, etc.) sans que ces affirmations naïves soient étayées par des données empiriques solides et fiables. Pour Tricot (2020), il existe différentes stratégies qui sont employées pour tenter d'améliorer la motivation des élèves par le biais des outils numériques. Parfois, la tâche proposée aux élèves est exactement la même, mais réalisée sur un support qui est différent du support papier (par exemple, lire un texte sur tablette numérique plutôt que sur une feuille polycopiée). En d'autres occasions, la tâche peut être

profondément modifiée par l'usage de l'outil numérique. Écrire un texte sur support numérique implique d'autres processus que ceux qui sont associés à l'écriture manuscrite et certaines fonctions des logiciels peuvent faciliter la réalisation de la tâche (telles que les correcteurs orthographiques ou la mise en gras et en italique). Enfin, certaines tâches ne peuvent exister que par le biais d'un support numérique. C'est le cas notamment des tâches de programmation qui consistent à créer un jeu vidéo interactif sur un ordinateur.

Sentiment d'auto-efficacité

Le sentiment d'auto-efficacité désigne la croyance que l'on peut exécuter avec succès les comportements nécessaires pour obtenir le résultat désiré (Bandura, 1977), c'est-à-dire une certaine forme de confiance en sa capacité à réaliser une tâche (Bandura, 1986). Ce sentiment d'auto-efficacité détermine la quantité d'efforts que l'on peut déployer pour faire face aux difficultés (Bandura, 1977). Appliqué au domaine de la programmation en contexte scolaire, le sentiment d'auto-efficacité désigne la confiance d'un élève en sa capacité à réussir à accomplir les tâches demandées dans ce domaine. Pour Cheng (2019), plus le sentiment d'auto-efficacité en informatique est élevé, plus la motivation et les efforts mis en œuvre seront élevés pour s'engager dans une tâche en lien avec l'informatique. Le sentiment de compétence, proche du sentiment d'auto-efficacité, est un élément essentiel de la dynamique motivationnelle des élèves et un atout indéniable pour leur réussite (Galand, 2006) et serait positivement corrélé au plaisir d'apprentissage (Stewart & Baek, 2023).

En éducation, le sentiment d'auto-efficacité des enseignants comme des élèves aurait une influence directe sur l'efficacité d'une intervention pédagogique (Kadiran et al., 2018 ; Tsai, 2019). Les élèves qui ont un fort sentiment d'auto-efficacité perçoivent les difficultés comme un défi, sont capables de s'investir dans leurs apprentissages et redoublent d'efforts lorsqu'ils sont confrontés à un échec (Tsai, 2019). À l'inverse, les élèves présentant un faible sentiment d'auto-efficacité surestiment les difficultés et interprètent l'échec comme un signe de malchance ou d'une incapacité à réussir. Lin et al. (2016), cités par Tsai (2019), ont montré qu'il existe un lien entre le sentiment d'auto-efficacité et la maîtrise des concepts basiques en programmation. De même, la motivation à programmer et le sentiment d'auto-efficacité seraient également liés (Law et al., 2010). Lishinski et al. (2016) observent dans une étude empirique que le sentiment d'auto-efficacité est le meilleur prédicteur des performances d'apprentissage en programmation et démontrent une certaine réciprocité entre ces deux

variables puisque, à l'inverse, la réussite en programmation affecte positivement le sentiment d'auto-efficacité.

Attitude vis-à-vis des sciences

Selon Weinburgh et Steele (2000), inciter les élèves à apprécier la science représente une difficulté majeure et un enjeu important en raison de liens supposés entre l'attitude, la motivation, l'intérêt et la réussite en sciences. Pour Alsancak (2020), la science et la pensée informatique sont indéniablement liées, ce qui le conduit à penser qu'il existerait un lien significatif et prédictif entre l'attitude vis-à-vis des sciences et les compétences associées à la pensée informatique. Son étude, qui concerne 722 élèves du secondaire, met en évidence ce lien puisque les élèves qui ont une attitude positive à propos des sciences présentent également des scores plus importants en ce qui concerne la maîtrise de la pensée informatique. Ce serait même la variable dont l'effet prédictif est le plus fort, en comparaison avec le genre des élèves ou leurs usages et compétences vis-à-vis des technologies. Sun, Hu et Zhou (2021) ont également observé que l'attitude des élèves d'école élémentaire vis-à-vis des disciplines STEM influence significativement leurs compétences en pensée informatique. De même, l'attitude des élèves vis-à-vis des disciplines scientifiques (STEM) serait un excellent prédicteur de l'engagement professionnel futur dans ces disciplines (Zhang et al., 2021). La recherche suggère que les expériences d'apprentissage tangibles (avec des robots notamment) améliorent la motivation et l'intérêt général pour les sujets en lien avec les disciplines STEM (Bers et al., 2014). Sáez-López et al. (2019) soulignent également l'efficacité de l'introduction de la robotique et de la programmation visuelle fondée sur des méthodologies actives dans l'enseignement primaire pour obtenir un degré élevé de participation et d'intérêt chez les élèves. Cependant, pour Kind et al. (2007), l'attitude en général, et l'attitude par rapport aux sciences en particulier, est une notion qui peut inclure selon les définitions, les sentiments, la motivation, le plaisir, les affects ou même l'estime de soi.

Kandlhofer et Steinbauer (2016) notent un désintérêt croissant vis-à-vis des sciences et technologies qui a pour conséquence un manque d'ingénieurs, de techniciens et de chercheurs dans certains pays. Lapierre (2017) recense dans son mémoire de recherche un ensemble d'études qui attestent de cette diminution de l'intérêt des élèves pour les sciences. La robotique éducative est alors perçue par Kandlhofer et Steinbauer (2016) comme un moyen à la fois d'accroître l'intérêt des élèves pour ces disciplines, mais aussi d'améliorer leurs compétences techniques et sociales. Ces chercheurs ont mené une étude de 8 mois auprès de 148 étudiants

d'une quinzaine d'années pour mesurer les bénéfices de la robotique en comparant les performances pré-test et post-test et en incorporant un groupe contrôle dans lequel les élèves ne participaient à aucune activité de robotique. L'évaluation des performances prenait la forme d'un long questionnaire, sous forme de QCM ou d'échelles de Likert, regroupant trois grandes catégories (compétences techniques, compétences sociales et relationnelles, intérêt et attitudes vis-à-vis de la science), elles-mêmes subdivisées en plusieurs sous-catégories. Il est intéressant de noter que les élèves ayant participé aux activités de robotique ont choisi volontairement de faire partie de ce groupe (absence de randomisation de l'échantillon), ce qui se traduit par un intérêt pré-test pour les sciences déjà initialement supérieur à celui du groupe contrôle. Globalement, l'effet de la robotique sur l'intérêt pour les sciences n'est ici pas significatif.

Recherches relatives à la programmation et à la pensée informatique

Certaines recherches empiriques se sont intéressées aux variables conatives décrites précédemment, à savoir la motivation, le sentiment d'auto-efficacité ou l'attitude envers les disciplines scientifiques. Cependant, lorsque ces études s'interrogent sur les liens entre les deux premières variables citées et l'apprentissage de la programmation ou de la pensée informatique, ce sont des liens spécifiques qui sont recherchés, c'est-à-dire l'impact sur la motivation à programmer ou le sentiment d'auto-efficacité relatif à la programmation par exemple. Globalement, ces variables ne sont que peu étudiées, les chercheurs se concentrant davantage sur les performances d'apprentissage.

Les activités débranchées auraient le potentiel de diminuer les sentiments négatifs des élèves vis-à-vis de l'informatique et de susciter une attitude plus positive, un engagement plus important et une motivation plus élevée (Battal et al., 2021 ; Delal & Oner, 2020 ; Threekunprapa & Yasri, 2020). Un impact positif sur le sentiment d'auto-efficacité a également été observé (Threekunprapa & Yasri, 2020). Cependant, les langages visuels de programmation peuvent également être perçus comme motivants, amusants et engageants (Armoni et al., 2015 ; Sáez-López et al., 2016), notamment par rapport aux langages plus traditionnels fondés sur du texte (Ouahbi et al., 2015). La facilité d'utilisation associée à ces langages pourrait en outre améliorer le sentiment de confiance et d'auto-efficacité des élèves (Sukirman et al., 2022), en particulier chez ceux pour qui ce sentiment est faible (Tsai, 2019). Enfin, la robotique pédagogique serait également indiquée pour permettre un haut niveau d'intérêt et de motivation chez les élèves (Kaloti-Hallak et al., 2015 ; Kim & Lee, 2016) bien que la revue de littérature

de Zhang et al. (2021) ne montre aucun effet positif des robots sur l'attitude envers les disciplines STEM.

4.5) Le paradoxe préférences / performances

Amadieu et Tricot (2014) constatent qu'il n'y a pas de lien évident entre une technologie, la motivation que celle-ci est censée apporter et les performances scolaires d'un élève, ce qui constitue ce que les auteurs désignent comme le paradoxe préférences / performances. En revanche, la liberté de prise de décision peut être un facteur de motivation, mais cette liberté est bien souvent relativement restreinte. La capacité du numérique à motiver les élèves est une idée reçue particulièrement présente dans le milieu de l'éducation. Mais de quelle motivation parle-t-on ? Celle relative au dispositif d'apprentissage, à la tâche à apprendre ou au contexte dans lequel s'effectue cet apprentissage ? Pour aborder le thème de la motivation chez l'apprenant, il est nécessaire de savoir si les technologies sont bien perçues et jugées utiles par celui-ci (Amadieu & Tricot, 2014). Les deux auteurs précédemment cités insistent sur la nécessité de prendre en compte la distinction entre utilisabilité (facilité d'utilisation) et utilité (perception d'apprendre grâce à l'outil). Ces deux paramètres couplés à l'attitude plus ou moins favorable vis-à-vis de la technologie vont déterminer l'intention d'usage.

En outre, l'adéquation entre la tâche à réaliser et l'outil utilisé participe à la motivation de l'apprenant (Amadieu & Tricot, 2014). La motivation dépend d'un contexte d'utilisation. Senkbeil et Ihme (2017) montrent que la motivation des élèves est corrélée à la fois à leur maîtrise des outils informatiques et à certaines caractéristiques personnelles (i.e., origine sociale, besoin de cognition). Le manque de motivation peut être dû à la croyance de l'apprenant qui estime qu'il n'est pas compétent pour utiliser tel ou tel outil. Certaines études montrent les effets réels et intéressants des nouvelles technologies sur la motivation. C'est le cas notamment pour Morris et al. (2012) qui observent que l'attitude vis-à-vis de l'utilisation des tablettes comme contribution possible aux apprentissages est généralement positive. L'outil est perçu comme engageant pour les tâches d'apprentissage. Cependant, certains de ces résultats sont critiquables sur le plan méthodologique et il est essentiel de mentionner à nouveau qu'il existe une distinction entre les préférences des apprenants et ce qui va objectivement améliorer leurs performances.

Tricot (2020) insiste sur la nécessité de distinguer d'une part la motivation des élèves et d'autre part la satisfaction engendrée par l'utilisation d'un outil ou d'un dispositif. Par ailleurs,

les effets des technologies sur la motivation sont trop disparates pour pouvoir en tirer des conclusions générales et les élèves évaluent souvent assez mal la capacité d'un outil à améliorer leurs apprentissages (Galand, 2020). En effet, lorsque l'usage de certains outils (tablettes graphiques par exemple) est évalué comme moins efficace qu'un apprentissage traditionnel (papier et crayon) dans certaines conditions, les apprenants pensent le contraire (Oviatt & Cohen, 2010). Les élèves n'ont donc pas forcément conscience de ce qui est le plus favorable à leurs performances. L'effet des technologies numériques dépend d'un certain nombre d'éléments, en particulier l'approche pédagogique mise en œuvre par l'enseignant, mais également des particularités des élèves ou des contenus enseignés, bien plus que de l'outil technologique à proprement parler (Tricot, 2020). De surcroît, une confusion est souvent possible entre l'effet du numérique et l'effet de nouveauté. Cheung et Slavin (2013) observent que l'introduction d'un nouvel outil ou d'une nouvelle approche suscite généralement un intérêt et une motivation accrue de la part des élèves. Cependant, cet impact positif n'est que transitoire et tend à décroître avec le temps. Il est donc possible que l'effet des technologies numériques sur les apprentissages ne soit que temporaire, ce qui remet en cause les études pour lesquelles les durées d'intervention sont trop courtes pour prendre en compte le déclin de cet effet de nouveauté.

En outre, lorsque l'introduction d'ordinateurs portables ne montre aucun bénéfice pour la classe, les élèves comme les enseignants perçoivent un impact positif fictif sur la réussite scolaire (Zucker & Light, 2009). Dans leur étude, Sung et Mayer (2013) comparent deux outils d'apprentissages (ordinateur ou tablette) avec pour chacun d'entre eux deux méthodes pédagogiques distinctes. Ils ne constatent aucun effet particulier. Cependant, les élèves sont plus motivés à poursuivre les apprentissages sur tablette, et ce, alors qu'aucun effet positif de cet outil sur les performances n'a pu être établi. De même, il n'existe aucun lien entre l'évaluation donnée par les étudiants aux enseignants d'université et les performances d'apprentissage (Uttl et al., 2017). Par conséquent, les enseignants et cours les plus appréciés, qui devraient donc susciter le plus de motivation chez l'apprenant, ne sont pas nécessairement vecteurs d'enseignements plus efficaces. Enfin, Hessler et al. (2018) montrent que les étudiants en médecine sont plus satisfaits d'un enseignant et de son support pédagogique lorsqu'ils ont reçu des biscuits au chocolat pendant le cours. Ce résultat illustre bien la distinction à opérer entre préférences de l'apprenant et efficacité des apprentissages.

4.6) L'impact du genre des élèves sur les apprentissages

Des différences liées au genre souvent défavorables aux filles

Historiquement, l'enseignement de la programmation attire généralement plus de garçons que de filles (Weintrop & Wilensky, 2019). Les garçons sont souvent plus familiers avec la technologie (Noh & Lee, 2020) et l'intérêt des filles pour l'informatique et les disciplines techniques tend à être plus faible (Gökçe & Yenmez, 2020). Globalement, les femmes sont souvent sous-représentées dans les disciplines scientifiques à partir du collège et certains chercheurs estiment qu'enseigner la pensée informatique, plutôt que l'informatique au sens large ou la programmation, pourrait davantage motiver les filles (Shute et al., 2017). Attirer celles qui sont peu présentes dans ce domaine représente donc un enjeu considérable (Bell et al., 2009 ; Bers et al., 2022). Les différences de genre en relation avec la science informatique sont maintenant bien documentées et montrent que les hommes présentent un sentiment d'auto-efficacité plus élevé et de plus grandes probabilités de succès dans les disciplines STEM, même si cet écart entre les genres tend à diminuer depuis quelques décennies (Atmatzidou & Demetriadis, 2016 ; Buitrago Flórez et al., 2017 ; Cheng, 2019 ; Wang et al., 2022). En Europe, dès le collège, les garçons sont plus nombreux à participer aux activités de programmation et cette tendance est encore plus marquée au lycée où 85 % des filles n'ont jamais (ou presque jamais) participé à ces activités contre 66% des garçons (Bourgeois et al., 2019).

Malgré l'utilisation croissante des technologies numériques dans nos sociétés, cet écart entre hommes et femmes persiste. Cependant, Master et al. (2017) nous montrent qu'il n'est pas irrémédiable, mais malléable et susceptible d'être influencé par des expériences spécifiques visant à le réduire. Dans leur étude, des enfants de 6 ans étaient aléatoirement répartis en deux groupes : un groupe expérimental qui participait à des activités de programmation avec des robots et un groupe contrôle. Les élèves en situation expérimentale ne montrent aucune différence liée au genre (contrairement au groupe contrôle) en ce qui concerne l'intérêt pour la programmation, l'intérêt pour les robots ou le sentiment d'auto-efficacité avec les robots. Par conséquent, les filles ayant participé à ces activités spécifiquement conçues pour réduire cet écart hommes-femmes font preuve d'un intérêt et d'un sentiment d'auto-efficacité plus élevés par rapport à celles n'y ayant pas pris part. Cette recherche montre également que les stéréotypes de genre dans ce domaine sont déjà présents dès l'âge de 6 ans, notamment celui selon lequel les garçons seraient meilleurs que les filles avec les robots, qui a été observé à la

fois chez les filles et chez les garçons. De même, Rubio et al. (2015) constatent que dans un cours d'introduction à la programmation à l'université, la facilité perçue de la programmation et l'intention de continuer à programmer sont plus élevées pour les hommes, bien que l'usage d'outils de programmation physiques (cartes programmables) semble réduire cet écart.

Cependant, Espino et González (2015, cités par del Olmo-Muñoz et al., 2020) ont observé qu'il existe une plus grande homogénéité entre garçons et filles dans l'enseignement primaire, ce que confirme Román-González et al. (2017) qui ne rapportent aucune différence de genre jusqu'au début du collège, avant que les garçons ne se mettent progressivement à mieux performer dans ce domaine à partir de la 5e. Il est d'ailleurs souvent recommandé d'initier les élèves à l'informatique ou à la programmation dès le plus jeune âge, notamment pour réduire cet écart entre hommes et femmes tant qu'il est n'est pas encore totalement présent (Sullivan & Bers, 2013 ; Webb et al., 2017). Cette initiation précoce serait non seulement possible, mais essentielle pour permettre aux filles de développer rapidement le sentiment d'auto-efficacité qui leur fait défaut (Webb et al., 2017) et pour combattre efficacement les stéréotypes qui supposent que l'informatique est réservée aux garçons (Armoni et al, 2015 ; Bers et al., 2022). Ceci aurait un impact positif sur les futurs choix de carrière en réduisant les obstacles souvent rencontrés par les filles (Ching et al., 2018 ; Bers et al., 2022).

L'étude d'Atmatzidou et Demetriadis (2016) montre que les filles ont besoin de plus de temps pour atteindre le même niveau de compétence que les garçons, même si les performances atteignent rapidement un niveau équivalent. D'autres études font état de meilleures performances d'apprentissage, notamment concernant la pensée informatique, pour les garçons (Korkmaz & Bai, 2019 ; Kong & Lai, 2021). Cependant, les études centrées sur les liens entre le genre et l'apprentissage de la pensée informatique sont souvent contradictoires et nombre d'entre elles ne témoignent d'aucune différence significative (Noh & Lee, 2020 ; Relkin et al., 2020 ; Li et al., 2021). Dans une revue de littérature sur l'intégration de la pensée informatique dans les disciplines STEM, Wang et al. (2022) recensent 11 études qui analysent les différences de genre. Cinq d'entre elles ne montrent aucune différence significative avec des échantillons supérieurs à 40 sujets alors que seulement deux études sur les six restantes font état de meilleures performances pour les filles (mais avec des échantillons plus réduits).

L'impact potentiel des outils d'enseignement sur les effets de genre

Pour lutter contre ces stéréotypes de genre, les activités débranchées, plus neutres, ont souvent été préconisées (Bell et al., 2009). L'étude de Delal et Oner (2020) ne montre en effet

aucune différence significative selon le genre des participants. Les auteurs observent même que, concernant les questions les plus difficiles associées à la pensée informatique, les filles progressent davantage que les garçons. Sun, Hu et Zhou (2021) ont en outre analysé les facteurs qui pourraient avoir une influence sur l'apprentissage débranché de la programmation et ne trouvent aucun effet lié au genre. Cependant, certaines études mettent en exergue un déclin de l'intérêt des filles pour l'informatique après une intervention composée d'activités débranchées (Huang & Looi, 2021). Battal et al. (2021) s'interrogent ainsi : « Quelle approche débranchée devrait être préconisée selon le genre des élèves ? ». Par ailleurs, Scherer et al. (2020) remarquent que les effets des outils de visualisation (tels que Scratch) sont plus prononcés lorsque l'échantillon est composé d'un grand nombre de filles. Scratch pourrait ainsi motiver davantage les filles en tenant compte des différences de genre (Tikva & Tambouris, 2019). Cependant, Jiang et Li (2021) observent que les garçons ont globalement mieux réussi que les filles à la fois sur le pré-test et le post-test avant et après 5 semaines de Scratch.

Noh et Lee (2020) ont réalisé une expérience de robotique qui ne révèle aucune différence au niveau des compétences en pensée informatique entre les garçons et les filles. La robotique éducationnelle pourrait d'ailleurs permettre d'accroître la motivation des filles et leur volonté de réussir dans les sciences de l'informatique ou de l'ingénierie (Gunes & Kucuk, 2022). Toutefois, si Angeli et Valanides (2020) ont trouvé que les garçons comme les filles peuvent améliorer leur maîtrise de la pensée informatique, les auteurs remarquent qu'il existe un effet d'interaction entre les stratégies d'étayage pédagogique et le genre. Les garçons bénéficient davantage des étayages individuels, kinesthésiques, orientés dans l'espace et fondés sur des activités de manipulation avec des cartes alors que les filles profitent plutôt des activités d'écriture collaborative. Par ailleurs, la robotique éducative semble impacter plus significativement les garçons que les filles (Zhang et al., 2021). Su et Yang (2023) mettent également en garde contre l'effet de la robotique dans les activités de programmation qui pourrait accroître les différences de genre, les filles ayant tendance à être démotivées par ce type d'objet ou de jouet à prédominance masculine.

Concernant l'étude de l'impact du genre, Dagienė et al. (2015, cités par del Olmo-Muñoz et al., 2020) soulignent que trop peu de recherches explorent spécifiquement les différences de genre relatives aux capacités des jeunes élèves à résoudre des problèmes et aux compétences en programmation. Ce constat est partagé plus récemment par Stewart et Baek (2023) pour qui davantage d'études concernant les effets de genre sur les compétences en pensée informatique ou en programmation sont requises.

SYNTHÈSE

- L'impact des activités débranchées, des logiciels de programmation éducatifs et de la robotique pédagogique sur l'apprentissage de la programmation constitue le cœur de ce travail, mais il est nécessaire de s'interroger sur les **différentes variables psychologiques** qui peuvent nous intéresser dans le cadre de cette thèse.
- Nous distinguons deux variables en lien direct avec l'apprentissage de la programmation et de la pensée informatique : la **maîtrise des notions fondamentales en programmation** et les **compétences en lien avec la pensée informatique**.
- Nous nous intéressons également au vécu subjectif des élèves, à travers l'impact de cet enseignement sur leur **niveau de motivation**, leur **sentiment d'auto-efficacité** et leur **attitude vis-à-vis des sciences**.
- Le **paradoxe préférences/performances** nous incite à penser que l'impact sur les performances des élèves et l'impact sur leur vécu subjectif ne seront pas nécessairement bien corrélés.
- L'efficacité des différents outils pourrait être déterminée notamment par la **charge cognitive** imposée par chacun de ces outils et par leur capacité à susciter des **interactions motrices**, potentiellement favorables aux apprentissages.
- Le genre des élèves pourrait également jouer un rôle : les filles sont souvent moins attirées par la programmation et l'informatique en général, d'où la nécessité d'évaluer si l'un de nos outils favoriserait une **réduction de cet écart entre les filles et les garçons**.

Chapitre 5. Problématique et hypothèses

5.1) Problématique

D'une manière générale, l'efficacité des technologies éducatives dans l'enseignement scolaire dépend d'un grand nombre de facteurs. Leur impact sur les apprentissages est limité, mitigé et dépend du contexte d'utilisation. La programmation est une discipline complexe, exigeant la mobilisation de multiples ressources cognitives. Son apprentissage pourrait toutefois influencer positivement les facultés cognitives des enfants, telles que leur capacité à résoudre des problèmes ou leur capacité de raisonnement. Certaines avancées récentes dans les outils d'enseignement, en particulier les langages visuels de programmation, facilitent aujourd'hui l'intégration de la programmation en contexte scolaire, y compris auprès des élèves les plus jeunes. En outre, apprendre à programmer serait particulièrement indiqué pour développer la pensée informatique.

La programmation et la pensée informatique peuvent être enseignés par le biais de différents outils, dont les activités débranchées, les logiciels de programmation visuels basés sur des blocs et les robots pédagogiques, qui constituent les trois outils les plus utilisés de nos jours. L'apprentissage débranché de la programmation ne s'appuie sur aucun outil numérique, ce qui permettrait d'éliminer la charge cognitive associée à l'utilisation des machines et de faciliter les apprentissages en construisant des analogies concrètes en relation avec les concepts abstraits de la programmation et en favorisant les actions sensori-motrices, vectrices de meilleures performances d'apprentissage, selon une perspective incarnée et située de la cognition. Les logiciels de programmation éducatifs s'appuient sur un langage visuel de programmation, basé sur des blocs à glisser-déposer, qui serait nettement plus simple à utiliser que les langages traditionnels basés sur du texte tout en permettant à l'élève de bénéficier d'un feedback basique, mais immédiat, en rendant visible l'exécution du programme créé. Les robots pédagogiques permettraient d'incarner les notions abstraites en programmation tout en permettant également d'obtenir un feedback et sont généralement associés à un langage visuel de programmation. La question générale qui est au cœur de ce travail de recherche est la suivante :

Q : Quel outil (activités débranchées, logiciels visuels de programmation, logiciels visuels associés à des robots pédagogiques) est le plus efficace pour enseigner la programmation et la pensée informatique ?

De cette question générale découlent un ensemble de questions plus spécifiques que nous détaillerons ci-après. La pensée informatique demeure aujourd'hui encore une expression aussi largement utilisée que mal définie, de par l'absence de consensus sur ce qu'elle est réellement en tant que construit psychologique. Par conséquent, l'évaluation de la maîtrise de cette pensée informatique dans le cadre des activités de programmation est un enjeu majeur de la recherche actuelle. La nécessité de distinguer les connaissances notionnelles en programmation et les stratégies ou compétences employées pour résoudre des problèmes algorithmiques nous semble essentielle. Ainsi, la question suivante se pose :

Q1 : Comment évaluer l'apprentissage de la programmation et de la pensée informatique à l'école ?

Plusieurs études empiriques ou revues de littérature ont mis en lumière un impact bénéfique des trois outils décrits précédemment sur les performances d'apprentissage des élèves, c'est-à-dire sur leur compréhension des notions essentielles en programmation, leur capacité à résoudre des problèmes algorithmiques ou leur maîtrise des compétences associées à la pensée informatique. Nous pouvons alors nous interroger :

Q2 : Quel outil a le plus d'impact sur la transmission des notions et compétences fondamentales en lien avec l'apprentissage de la programmation et de la pensée informatique ?

Par ailleurs, ces outils pourraient influencer significativement et positivement certaines variables conatives, telles que la motivation, le sentiment d'auto-efficacité et l'attitude vis-à-vis des sciences. De ce constat émerge la question suivante :

Q3 : Quel outil a le plus d'impact sur le vécu subjectif des élèves et les variables conatives en lien avec l'apprentissage de la programmation et de la pensée informatique ?

De surcroît, un impact positif de ces outils sur les stéréotypes de genre, plutôt défavorables aux filles, a parfois été observé ou supposé et pourrait contribuer à réduire les inégalités hommes-femmes dans ce domaine. De ce fait, la question de recherche suivante nous semble importante :

Q4 : Quel outil est le plus susceptible de réduire l'écart entre les genres dans l'apprentissage de la programmation et de la pensée informatique ?

Très peu d'études expérimentales proposent d'étudier les liens qui existent entre les différentes variables qui nous intéressent dans cette thèse. Par conséquent :

Q5 : Peut-on mesurer des corrélations significatives entre les variables (cognitives ou conatives) que nous avons choisi d'étudier ?

Enfin, au-delà des comparaisons expérimentales qui nous permettront peut-être de distinguer différents niveaux d'efficacité selon les outils utilisés, la question se pose de savoir pour quelle raison un outil a un impact plus prononcé qu'un autre. Les arguments avancés pour défendre l'utilisation de tel ou tel outil reposent bien souvent sur la diminution de la charge cognitive et le feedback immédiat rendu possible par les outils numériques. Notre dernière question sera donc :

Q6 : Quel est l'impact du feedback renvoyé par un outil numérique et de la charge cognitive propre à chaque outil sur les apprentissages des élèves ?

5.2) Hypothèses

Concernant notre question générale Q, notre objectif est de mesurer des différences significatives entre les trois outils mobilisés sur les variables cognitives et conatives ainsi que sur leur capacité à réduire l'écart entre les genres, afin de déterminer lequel de ces outils serait le plus efficace dans l'enseignement de la programmation et de la pensée informatique. Néanmoins, il nous semble peu probable qu'un seul de ces outils puisse influencer plus positivement l'ensemble des variables en comparaison avec les deux autres. Pour cette raison, nous ne nous risquerons pas à proposer une hypothèse générale H qui n'aurait que peu de sens. Une réponse argumentée et nuancée sera cependant apportée à cette question de recherche générale dans la partie « Discussion générale ».

De même, il serait excessif de prétendre apporter une réponse simple à la question Q1 qui concerne la manière avec laquelle l'enseignement de la programmation et de la pensée informatique peut être évalué. Cette évaluation dépend d'un grand nombre de facteurs et, au vu de l'absence de consensus sur la notion même de pensée informatique, toute réponse ne peut être que contextuelle. Dans l'introduction du premier chapitre de la « Partie expérimentale », nous réaliserons une analyse de l'existant en termes d'outils d'évaluation. Cette analyse nous

permettra de mettre en lumière les manques actuels concernant ces outils d'évaluation et de proposer nous-mêmes deux nouveaux outils, pour lesquels un processus de validation psychométrique a été mené à bien. Ce premier chapitre constituera notre réponse à la question Q1 et ne repose sur la validation d'aucune hypothèse préalable.

À propos des questions Q2, Q3 et Q4, l'introduction du deuxième chapitre de la « Partie expérimentale » mettra en lumière le manque de données empiriques relatif à l'approche comparative que nous avons mise en place dans cette thèse. Trop peu d'études proposent de comparer les différents outils d'enseignement de la programmation dans la littérature actuelle. Par conséquent, les expériences menées dans cette thèse présentent une dimension exploratoire indéniable. Cependant, nous avons pu constater que les arguments théoriques les plus répandus pour défendre l'utilisation d'un outil plutôt qu'un autre concernent la charge cognitive et la présence d'un feedback renvoyé par une machine. Les activités débranchées ne permettent pas ce type de feedback, mais suppriment totalement la charge cognitive associée à l'usage des outils numériques. À l'inverse, les robots pédagogiques associés à un langage visuel de programmation bénéficient d'un double feedback (du robot et du logiciel), mais pourraient surcharger cognitivement les élèves par la présence de ces deux éléments utilisés conjointement. Les logiciels de programmation éducatifs, prenant appui sur un langage visuel basé sur des blocs, pourraient se révéler le meilleur compromis en permettant à l'élève de bénéficier d'un feedback potentiellement précieux pour ses apprentissages tout en maintenant une charge cognitive acceptable par leur facilité d'utilisation :

H1 : Les langages visuels de programmation, non associés à des robots pédagogiques, constituent l'outil le plus efficace pour transmettre les notions et compétences fondamentales en lien avec la programmation et la pensée informatique.

Au sujet de l'impact de nos outils sur les variables conatives et les stéréotypes de genre, la formulation d'hypothèses parait d'autant plus périlleuse que les recherches à ce propos sont encore plus rares. En nous basant sur notre revue de littérature, nous faisons le pari que les approches branchées de la programmation (avec ou sans robot) sont les plus susceptibles d'impacter favorablement les variables conatives par la facilité d'utilisation des langages visuels de programmation et par le caractère nouveau et attrayant de ces outils. Concernant l'attitude envers les disciplines scientifiques, l'impact de la programmation sur cette variable est si peu étudié que nous proposerons à son sujet une hypothèse plus générale selon laquelle les activités de programmation ont bel et bien un effet positif sur celle-ci. Quant à l'impact du

genre, nous avons pu constater que les stéréotypes défavorables aux filles peuvent déjà être bien installés dans l'esprit des élèves de cycle 3 et concernent l'informatique en général. Par leurs capacités à s'éloigner drastiquement du matériel informatique, les activités débranchées nous semblent les plus indiquées pour réduire l'écart entre les garçons et les filles. Ainsi, nous sommes en mesure de formuler les hypothèses suivantes :

H2 : Les approches branchées de la programmation sont les plus efficaces pour maintenir ou susciter une plus grande motivation et un plus grand sentiment d'auto-efficacité chez les élèves.

H3 : Les activités de programmation peuvent influencer positivement l'attitude des élèves vis-à-vis des disciplines scientifiques.

H4 : Les activités débranchées permettent de réduire ou d'éliminer l'écart entre les genres, concernant l'apprentissage de la programmation et de la pensée informatique, en comparaison avec les autres outils.

Il nous semble probable que les variables cognitives (connaissances des notions et maîtrise des compétences) soient reliées d'une part, tout comme les variables conatives (motivation, auto-efficacité, attitude envers les sciences) d'autre part. Cependant, en raison du paradoxe préférences / performances, il est loin d'être certain que les variables cognitives et conatives présentent des relations de corrélation entre elles. Par conséquent, nous formulerons cette hypothèse :

H5 : Il existe un lien de corrélation relativement fort entre les variables cognitives d'une part, et les variables conatives d'autre part, mais le lien de corrélation entre les variables cognitives et conatives est faible ou inexistant.

Enfin, il est fort possible que la présence ou non d'un feedback permis par les outils numériques et la charge cognitive associée à une situation d'apprentissage aient un impact sur les apprentissages. L'introduction du troisième chapitre de la « Partie expérimentale » reviendra sur la littérature qui abonde en faveur de l'importance de ces deux facteurs dans l'enseignement. Par conséquent, il est raisonnable de penser que :

H6 : La charge cognitive associée à un outil et la présence ou l'absence d'un feedback issu des outils numériques ont un impact significatif sur les performances d'apprentissage.

II) PARTIE EXPÉRIMENTALE

Chapitre 1. Nouveaux outils d'évaluation de la pensée informatique

1.1) Introduction

Cette thèse porte sur l'apprentissage de la programmation et de la pensée informatique et mesure l'impact de trois outils d'enseignement sur un ensemble de variables. Au-delà des aspects motivationnels et liés au vécu subjectif des élèves, il est nécessaire de s'interroger sur la manière d'évaluer les acquis qui font suite à une intervention visant à développer la pensée informatique par le biais des activités de programmation. Cette présente section constitue une revue de la littérature concernant les différents outils et approches employés pour évaluer ce construit. Les apports et les limites de l'existant seront commentés et donneront lieu à une synthèse relative aux manques actuels de la recherche dans ce domaine. Cette synthèse nous permettra par la suite de proposer deux nouveaux outils que nous avons validés et qui nous semblent apporter une plus-value non négligeable pour évaluer les compétences liées à la pensée informatique dans le contexte d'un apprentissage de la programmation.

Des échelles autoévaluatives

[Korkmaz et al. (2017) proposent le test *Computational Thinking Scales (CTS)* qui regroupe des items issus de plusieurs tests distincts. Les auteurs s'appuient sur un document de l'ISTE (2015, citée dans Korkmaz et al., 2017) qui définit la pensée informatique comme un ensemble de compétences. Le *CTS* vise ainsi à mesurer les capacités créatives, la pensée algorithmique, la pensée critique, la capacité à résoudre des problèmes et les capacités coopératives. Tous les items qui les composent ont été récupérés à partir de tests préexistants (*Problem Solving Scale, Cooperative Learning Attitude Scale, Scale of California Critical Thinking tendency*, etc.). Le questionnaire est autoadministré, ce qui pose des problèmes liés à la subjectivité des réponses fournies par les élèves, et a été validé auprès de plus de 700 étudiants à l'université qui ont répondu aux 74 affirmations du test à l'aide d'une échelle de Likert en cinq points (*jamais, rarement, parfois, souvent, toujours*). Le test a donc suivi un processus rigoureux de validation, mais la question de sa généralisation à des niveaux inférieurs

peut légitimement être posée. Pour autant, cette échelle a été testée avec succès auprès d'un millier d'étudiants chinois âgés de 15-16 ans (Korkmaz & Bai, 2019), ce qui nous laisse envisager la possibilité de l'utiliser également en France à la fin de l'enseignement secondaire. Cet outil d'évaluation est intéressant, bien que la pensée informatique soit ici considérée comme la simple agrégation d'un ensemble de compétences bien connues et bien identifiées. S'il semble pertinent d'affirmer que la pensée créative ou la pensée critique sont liées à la pensée informatique, la proposition de Korkmaz et al. (2017) de considérer la seconde comme la combinaison des deux premières, ainsi que d'autres facultés, est aussi potentiellement critiquable, car relativement réductrice.

D'autres échelles du même type, visant à évaluer les perceptions et l'attitude des élèves vis-à-vis de la pensée informatique, ont été validées récemment, en particulier en ce qui concerne le sentiment d'auto-efficacité des élèves en relation avec la maîtrise de la pensée informatique (Weese & Feldhausen, 2017 ; Kukul & Karatas, 2019). Ces échelles présentent les mêmes limites que celle de Korkmaz et al. (2017) et s'adressent à un public plus jeune (plutôt les 10-14 ans). Globalement, les échelles autoévaluatives peuvent être préconisées pour établir un autodiagnostic, faisant suite à une intervention ayant eu pour objectif de développer la pensée informatique.

Des outils d'analyse objectifs du code produit par l'élève

Brennan et Resnick (2012) proposent différentes façons d'évaluer la pensée informatique. Premièrement, en associant certains blocs du logiciel de programmation Scratch à certains concepts de la pensée informatique, il serait possible d'analyser de manière objective les projets Scratch réalisés par les élèves. Cependant, les chercheurs reconnaissent eux-mêmes que cette manière de procéder est limitée. L'utilisation de certaines instructions ne garantit pas que celles-ci soient parfaitement comprises et que les compétences associées soient bien maîtrisées. De même, Alves et al. (2019) font état de différents outils qui se fondent exclusivement sur l'analyse du code produit pour analyser les programmes réalisés sur des logiciels de programmation visuels, en particulier sur Scratch. Globalement, ce type d'évaluation est critiquable sur au moins un aspect majeur. Il admet que les compétences en pensée informatique sont visibles directement à travers l'analyse du code qui est généré par l'élève, ce qui reste une supposition. Par ailleurs, cette approche limite l'évaluation de la maîtrise de la pensée informatique aux activités de programmation.

Dans ce registre, Dr. Scratch est un outil capable d'analyser automatiquement des fichiers Scratch, notamment pour évaluer différents aspects de la pensée informatique (Moreno-León & Robles, 2015). Un feedback est ensuite renvoyé à l'utilisateur avec trois niveaux de compétence (« basique », « développé », « maîtrisé »). Les auteurs ont réussi à montrer que ce feedback peut être bénéfique pour améliorer les performances de codage des élèves, et donc potentiellement la maîtrise sous-jacente des compétences liées à la pensée informatique (Moreno-León & Robles, 2015). Il existe par ailleurs de fortes corrélations entre les évaluations réalisées automatiquement via Dr. Scratch et les évaluations réalisées par des experts humains (Moreno-León et al., 2017). L'abstraction, la synchronisation, la représentation des données, l'interactivité, la pensée logique, le contrôle des flux (lié ici à l'utilisation pertinente de boucles « répéter ») et le parallélisme sont les compétences évaluées par Dr. Scratch. Ce sont des compétences très techniques, qui semblent parfois plus liées à la bonne utilisation du logiciel qu'à la pensée informatique elle-même. Certaines ne paraissent pas toujours pertinentes. Par exemple, l'interactivité du programme est un critère d'évaluation pris en compte par Dr. Scratch bien que ce critère ne semble pas directement relié à la maîtrise de la pensée informatique, selon la grande majorité des auteurs ayant tenté de la définir.

Ce type d'outils est le plus souvent utilisé dans le cadre d'une évaluation formative / itérative qui consiste à évaluer les élèves pendant le processus d'apprentissage afin de leur permettre de développer et d'améliorer leurs compétences en pensée informatique en améliorant les programmes qui sont créés. Ils sont conçus pour fonctionner avec un environnement de programmation particulier (et sont donc spécifiques à un langage ou à un logiciel), ce qui constitue également une limite à leur utilisation. Ils peuvent toutefois être utilisés à tous les âges, puisqu'ils se basent uniquement sur les programmes réalisés par les élèves. Par exemple, Dr. Scratch peut être adapté à tous les élèves qui ont appris à programmer sur Scratch, quel que soit leur niveau ou leur âge.

Des tâches de résolution de problèmes

Il existe d'autres démarches intéressantes qui diffèrent de la simple analyse du code produit par un élève et qui consistent en différentes tâches que l'élève doit accomplir. La réussite dans l'accomplissement de ces tâches est ainsi censée traduire la maîtrise de certains aspects de la pensée informatique. Grover et al. (2014) ont utilisé des captures d'écran de Scratch accompagnées de questions pour évaluer les apprentissages (« pourquoi ce programme ne fonctionne-t-il pas ? » ; « Quand ce programme est exécuté, quelle valeur nous est renvoyée

? », etc.). Brennan et Resnick (2012) ont demandé aux élèves de partir d'un programme existant pour l'améliorer ou rajouter de nouvelles fonctionnalités définies par l'expérimentateur. De leur côté, Atmatzidou et Demetriadis (2016) ont mis en place des exercices de résolution de problème qui impliqueraient l'utilisation de compétences identifiées comme faisant partie de la pensée informatique. Il était par exemple demandé d'identifier les structures communes qui guidaient le comportement d'un robot dans deux tâches différentes (abstraction), de proposer une solution plus générale à un problème (généralisation), etc. La pensée informatique étant très souvent associée au processus de résolution de problèmes, ce type d'outil d'évaluation semble pertinent, notamment dans le cadre d'activités de programmation pour lesquelles il est fréquent de considérer que les connaissances sont construites et évaluées par la résolution de problèmes (Rogalski & Samurçay, 1990).

L'une des tentatives les plus abouties en ce sens reste probablement le *Computational Thinking Test* ou CTt (Román-González, 2015), dont la pertinence des items a été mesurée par un jury d'experts. Le CTt se compose de 28 items qui visent à estimer la maîtrise de certaines notions d'informatique : les instructions basiques, les boucles, les conditions et les fonctions. Le test est centré sur des exercices nécessitant la réalisation d'un parcours pour lesquels les participants ont un choix de réponse limité et prédéterminé. Ces réponses sont présentées sous forme de texte, de flèches ou de blocs similaires à ceux utilisés dans les logiciels de programmation visuels. Dans ces exercices, il est alternativement demandé au participant de trouver la séquence qui permet de résoudre le problème, de compléter l'instruction manquante dans un programme qui est proposé ou encore d'y trouver l'erreur qui empêche ce programme de répondre au problème posé.

Le test semble un excellent moyen de mesurer la compréhension de certaines notions fondamentales en programmation (*computational concepts*, Brennan & Resnick, 2012) telles que les boucles ou les structures conditionnelles et la compréhension de leur fonctionnement. Son utilisation est généralement préconisée pour établir un diagnostic du niveau de l'élève à un moment donné et permet de comparer facilement les performances obtenues avant et après une intervention éducative (comparaison pré-test / post-test). Il présente par ailleurs une validité convergente, puisque les performances au CTt sont corrélées aux performances à d'autres tests mesurant les capacités spatiales, de raisonnement ou de résolution de problèmes (Román-González et al., 2017), ainsi que d'une validité prédictive vis-à-vis des performances académiques et de la réussite en programmation au collège (Román-González et al., 2018b).

Enfin, il a été testé (et semble adapté) auprès des élèves dont le niveau se situe de la fin de l'école élémentaire jusqu'au début du lycée (de 9 à 16 ans).

Cependant, ce qui relève des notions et des savoirs semble confondu avec ce qui relève plus de la compétence et du savoir-faire dans ce CTt. Román-González (2015) reconnaît que la structure du test, qui n'est composé que de questions fermées à choix multiples, ne permet de mesurer le degré de maîtrise de la pensée informatique qu'en ce qui concerne ses aspects les plus simples, c'est-à-dire la capacité à comprendre les concepts mis en jeu et à reconnaître la solution à un problème posé parmi plusieurs possibilités données. Un instrument qui aurait pour objectif de mesurer également les aspects les plus complexes de la pensée informatique devrait alors demander à l'élève de montrer qu'il est capable d'assimiler et d'appliquer ces concepts pour résoudre des problèmes, en proposant lui-même les solutions adéquates.

Certains outils proposent par ailleurs une évaluation de la pensée informatique fondée sur la résolution de problèmes indépendants des activités de programmation. Ces outils visent à évaluer dans quelle mesure les élèves sont capables de transférer leurs compétences en pensée informatique à une grande variété de problèmes, situations ou contextes nouveaux. C'est le cas notamment des épreuves du *Bebras International Contest* en Lituanie. La capacité à transférer ces compétences dans d'autres contextes, relatifs à des problèmes de la vie quotidienne, peut y être évaluée, y compris pour des élèves n'ayant jamais été initiés à la programmation. Román-González et al. (2019) ont montré que le CTt, Dr. Scratch et les tâches Bebras peuvent être considérés comme des outils complémentaires, présentant des liens de corrélations et permettant d'évaluer différents niveaux d'acquisition de la pensée informatique.

Très récemment, plusieurs études ont proposé et validé de nouveaux outils visant à évaluer la capacité des élèves à résoudre des problèmes de programmation, en utilisant des langages naturels, permettant à tous les élèves de réaliser ces tests, quels que soient les logiciels ou langages qu'ils ont appris à utiliser (Basu et al., 2021 ; El-Hamamsy et al., 2022 ; Kong & Wang, 2021 ; Relkin et al., 2020 ; Sung, 2022). Ces nouveaux outils ont été validés auprès d'élèves relativement jeunes, âgés de 5 à 11 ans. La résolution des problèmes présentés est ainsi censée révéler, non pas la maîtrise de certains concepts en informatique, comme pour le CTt, mais plutôt la maîtrise de certains processus cognitifs (abstraction, pensée algorithmique…), proches de ceux définis par Brennan et Resnick (2012) ou Selby et Woollard (2013) comme faisant partie de la pensée informatique.

À ce titre, le *Computational Thinking Assessment for Chinese Elementary Students* (CTA-CES, pas encore adapté dans les pays occidentaux) semble particulièrement prometteur (Li et al., 2021). Là encore, le cadre défini par Selby et Woolard (2013) pour décrire la pensée informatique constituait la base sur laquelle se sont appuyés les auteurs. Ceux-ci ont proposé un test comportant 25 items, similaires aux tâches Bebras dans le sens où les problèmes à résoudre ne sont pas des problèmes de programmation, mais plutôt des problèmes de la vie quotidienne, à destination d'élèves âgés de 8 à 12 ans. En plus des méthodes psychométriques classiques utilisées pour valider cet outil, les auteurs ont réalisé des images par résonance magnétique (IRMf). Celles-ci ont montré que les activités neuronales des participants durant la réalisation de ce test et durant la réalisation d'activités de programmation sont corrélées. Ceci suggère que programmer et effectuer le CTA-CES sont deux activités qui impliquent les mêmes processus cérébraux, ce qui constitue un indice supplémentaire en faveur de la validité de ce test, selon les auteurs.][20]

Lai et Ellefson (2022) ont également proposé (et validé) un test, le *Computational Thinking Challenge* (CTC), basé sur des exercices de résolution de problèmes, qui propose à la fois des problèmes en lien avec des activités de programmation et des problèmes indépendants de toute activité de codage. Cependant, comme les autres tests évoqués précédemment, le CTA-CES et le CTC ne proposent que des questions à choix multiples (ou des lignes de code à remettre dans l'ordre pour le CTC) et n'évaluent donc pas la capacité des élèves à générer eux-mêmes leur propre solution à un problème. [À notre connaissance, seuls Chen et al. (2017) ont élaboré (et validé) un outil d'évaluation de la pensée informatique (selon la définition opérationnelle proposée par CSTA & ISTE, 2011) centré sur la résolution de problèmes et partiellement composé de questions ouvertes pour lesquelles les élèves (âgés de 9-10 ans) ne pouvaient pas choisir entre plusieurs solutions préétablies. Cependant, cet outil s'intègre dans le cadre d'un apprentissage de la programmation réalisé à l'aide de robots pédagogiques, nécessite la maîtrise d'une syntaxe spécifique, et semble difficile à adapter dans un autre contexte. Les tâches de résolution de problèmes font généralement appel à des compétences qui dépassent le strict apprentissage de la programmation informatique. Pour cette raison, elles peuvent être utilisées auprès d'élèves n'ayant pas (encore) appris à programmer et se prêtent donc particulièrement bien aux comparaisons pré-test / post-test (évaluations avant et après une

[20] Sigayret, K., Blanc, N., & Tricot, A. (2022). L'apprentissage de la programmation: quels outils pour évaluer le développement de la pensée informatique à l'école?. *Enfance*, *4*(4), 479-500.

séquence pédagogique) qui permettent de facilement mesurer les bénéfices d'une intervention éducative.

Évaluer l'apprentissage « implicite » de la pensée informatique

Rowe et al. (2021) ont récemment proposé une approche innovante qui consiste à évaluer l'apprentissage implicite des élèves confrontés à une tâche faisant appel à certaines compétences liées à la pensée informatique. L'apprentissage implicite fait ici référence à un apprentissage dont la manifestation peut être faite lors de l'activité d'apprentissage elle-même (et non de manière différée avec des tests). L'outil présenté par Rowe et al. (2021) peut être considéré comme un outil de *data-mining* (exploration des données, en français) qui récupère et enregistre l'activité de l'élève en temps réel. Les auteurs utilisent le jeu vidéo éducatif Zoombinis, composé d'épreuves basées sur la logique et la déduction. L'activité du joueur/élève pendant la résolution des problèmes posés est enregistrée sous la forme de données numériques qui peuvent ainsi être analysées et répliquées. Ici, ce n'est pas la capacité à résoudre les problèmes qui est évaluée, mais bien les processus qui ont mené à cette résolution. À travers les données recueillies, les auteurs ont construit manuellement des outils permettant de détecter la maîtrise implicite de la pensée informatique des élèves âgés de 8 à 14 ans.

Les actions réalisées par l'élève lors de la résolution du problème permettent ainsi de mettre en évidence sa capacité à décomposer le problème en sous-problèmes plus simples, à identifier une règle sous-jacente au problème, à tester de manière systématique différentes hypothèses ou encore à réutiliser certaines stratégies qui avaient fonctionné précédemment. Ces comportements (abstraction, décomposition…) sont caractéristiques de la pensée informatique selon de nombreux auteurs (Brennan & Resnick, 2012 ; Selby & Woollard, 2013 ; Shute et al., 2017). Cette approche présente l'originalité de proposer une évaluation formative en temps réel fondée sur des activités ludiques, de mettre en évidence des comportements révélateurs d'une maîtrise émergente de la pensée informatique et de se placer aux antipodes des évaluations plus classiques sous forme de tests, en ne s'appuyant sur aucune représentation visuelle ou verbale de la pensée informatique. Cependant, les connaissances techniques et le temps nécessaires pour récupérer et analyser les données du jeu et en déduire certaines compétences sont un véritable frein à la généralisation de cette méthode d'évaluation.][21]

[21] Sigayret, K., Blanc, N., & Tricot, A. (2022). L'apprentissage de la programmation: quels outils pour évaluer le développement de la pensée informatique à l'école?. *Enfance*, *4*(4), 479-500.

Limites des outils existants

L'absence de consensus concernant une définition conceptuelle de la pensée informatique se traduit logiquement par une absence de consensus quant à la manière la plus appropriée de la mesurer ou de l'évaluer. Dans une revue de la littérature comprenant 26 études concernant l'évaluation de la pensée informatique, Babazadeh et Negrini (2022) recensent 59 dimensions associées à la pensée informatique évaluées par les différents outils proposés, bien que seulement 21 d'entre elles ne soient pas spécifiques à un outil particulier. [De nombreux chercheurs recommandent d'ailleurs de pallier les difficultés rencontrées pour évaluer la pensée informatique en associant plusieurs outils complémentaires pour permettre d'avoir une idée plus précise du niveau des élèves (Román-González et al., 2019).

Globalement, les outils de mesure de la pensée informatique que nous avons détaillés précédemment sont des tentatives louables et pertinentes d'évaluer cette compétence. Cependant, ils ne proposent actuellement aucune distinction claire entre d'un côté les connaissances des élèves et d'un autre leur capacité à appliquer ces connaissances pour résoudre des problèmes complexes. Généralement, ces deux aspects fondamentaux sont confondus, si bien qu'on ne sait pas toujours si les outils d'évaluation proposés permettent de mesurer le niveau de compréhension des élèves vis-à-vis de certaines notions ou leur niveau de maîtrise d'une compétence qui leur permet de faire face à des situations inédites pour proposer des solutions en s'appuyant sur leurs connaissances. Or, nous considérons que cette distinction est capitale.

Si l'objectif est de développer des outils d'évaluation permettant de discriminer différents niveaux de maîtrise de la pensée informatique, alors il faut pouvoir en distinguer les aspects les plus fondamentaux, la connaissance et la compréhension des notions et concepts, et les aspects les plus complexes, l'assimilation et l'application pratique de ces concepts et le développement de stratégies permettant la résolution de problèmes. Une évaluation distincte de ces deux aspects de l'apprentissage de la programmation permettrait à l'enseignant d'affiner son diagnostic quant aux éventuelles difficultés rencontrées par ses élèves. Ceux-ci maîtrisent-ils suffisamment les concepts fondamentaux utilisés en programmation ? Si c'est le cas, sont-ils capables de les appliquer dans un contexte pratique en utilisant les stratégies adéquates ? Cette distinction entre la maîtrise de concepts et la compétence de résolution de problèmes s'accorde d'ailleurs particulièrement bien avec les attendus de fin de cycle 4 (en France) en relation avec l'informatique et la programmation, bien que ceci ne constitue pas un argument

scientifique en faveur de notre modèle. La capacité à « écrire, mettre au point et exécuter un programme » y figure ainsi que la maîtrise des « notions d'algorithme et de programme, notion de variable informatique, déclenchement d'une action par un événement, séquences d'instructions, boucles, instructions conditionnelles ».][22]

Par ailleurs, nous avons pu constater certaines limites qui rendent difficile l'application de l'un ou de plusieurs de ces tests dans les classes françaises. Les outils d'évaluation fondés sur la résolution de problèmes semblent particulièrement indiqués pour mesurer la pensée informatique. Cependant, aucun de ces outils n'a été validé en langue française et il n'est pas évident qu'ils s'adaptent suffisamment à notre système scolaire français. De plus, les tests proposés, bien que relativement complets, sont généralement longs et donc peu pratiques à appliquer dans un contexte scolaire, particulièrement dans le second degré où les heures de cours n'excèdent pas 55 minutes. Ils sont généralement composés de plusieurs dizaines d'items pour une durée supérieure à 1 heure. Enfin, ceux-ci sont quasi exclusivement constitués de questions à choix multiples qui ne permettent donc pas d'évaluer la capacité de l'élève à produire par lui-même une solution à un problème sous la forme d'un algorithme ou d'un programme.

Pour ces différentes raisons, il nous a semblé pertinent de proposer nous-mêmes deux nouveaux outils d'évaluation en français de la pensée informatique, en situation d'apprentissage de la programmation, qui respectent cette distinction entre connaissances et compétences liées à la pensée informatique, essentielle à nos yeux, qui répondent aux impératifs écologiques relatifs à l'organisation des séances en classes et qui ne se basent pas uniquement sur des questions à choix multiples. La conception et la validation de ces deux tests feront l'objet des parties suivantes.

1.2) Test de Maîtrise des Concepts Computationnels

Le premier test que nous avons conçu propose de mesurer la connaissance et la compréhension des notions fondamentales en programmation, telles que définies par Tchounikine (2017), à savoir les notions d'algorithme, d'instruction, de boucle, d'instruction conditionnelle et de variable. Afin de définir un cadre clair et explicite, nous définirons ces cinq

[22] Sigayret, K., Blanc, N., & Tricot, A. (2022). L'apprentissage de la programmation: quels outils pour évaluer le développement de la pensée informatique à l'école?. *Enfance*, *4*(4), 479-500.

concepts de la manière suivante : un algorithme est une succession d'actions (ou d'instructions), présentées dans un certain ordre, qui permet de résoudre un problème (au sens large) ; une instruction est un ordre (ou une action) à exécuter ; une boucle est une instruction qui permet de répéter plusieurs fois certaines instructions ; une instruction conditionnelle ne s'exécute que lorsqu'une certaine condition est remplie ; une variable permet de stocker et de modifier une valeur. Nous avons nommé ce test : Test de Maîtrise des Concepts Computationnels. En effet, au vu des similitudes entre les notions identifiées par Tchounikine (2017) et les *computational concepts* identifiés par Brennan et Resnick (2012), nous avons jugé plus opportun d'opter pour une traduction littérale en « concepts computationnels », ce qui permet de faire le lien entre ces notions de programmation et la pensée informatique (*computational thinking*). L'objectif est de valider un test en français, relativement court comparé aux autres outils d'évaluation des connaissances et compétences en pensée informatique ou en programmation, qui pourra nous servir pour nos expériences futures. Pour ce test, les items retenus correspondent tous à des questions à choix multiples.

1.2.1) Méthode

Participants

La validation de ce test a mené au recrutement de 624 participants (320 filles) dont 3 élèves de CM1, 27 de CM2, 98 de 6e, 116 de 5e, 183 de 4e et 197 de 3e. Nous avons conçu deux versions de ce test, une version papier et une version en ligne, strictement similaires. Pour la version en ligne, nous avions 449 élèves (238 filles) dont 3 élèves de CM1, 19 de CM2, 98 de 6e, 16 de 5e, 183 de 4e et 130 de 3e. Pour la version papier, nous avions 175 élèves (82 filles), dont 8 élèves de CM2, 100 de 5e et 67 de 3e. Ce test a vocation à être utilisé ultérieurement dans nos expériences, qui seront réalisées avec un public globalement plus jeune (niveau CM2) que la plupart des élèves recrutés ici. Cependant, après discussion avec des enseignants du premier degré, nous avons pu constater que nos expériences prévoient un enseignement plus approfondi que celui généralement proposé au cycle 3. Pour cette raison, nous avons validé notre outil en faisant prioritairement appel à des élèves de cycle 4. Nous avons bénéficié du réseau de la DRANE de l'Académie de Montpellier pour recruter des classes de différents niveaux dans cette région. Pour la version en ligne du test, nous avons élargi notre périmètre de recherche et nous avons fait appel à des enseignants issus d'autres régions françaises ainsi qu'à des associations ou à des écoles d'apprentissage du code informatique pour les enfants.

Matériel

Nous avons tout d'abord conçu un questionnaire visant à évaluer le niveau de pratique de la programmation des élèves. L'objectif est de permettre de mesurer un lien de corrélation entre ce niveau de pratique de la programmation et les performances des élèves au Test de Maîtrise des Concepts Computationnels. Après avoir recueilli le genre, l'âge et le niveau scolaire, nous demandons aux élèves s'ils ont déjà pratiqué la programmation (ou le codage). Si leur réponse est positive, ceux-ci doivent remplir un questionnaire comprenant 8 questions. Les quatre premières questions nous permettent de savoir approximativement à quelle période ils ont commencé la programmation, quand ils ont programmé pour la dernière fois, la fréquence de leurs activités de programmation depuis le début de l'année scolaire et enfin, une estimation du nombre d'heures passées à programmer. Les réponses à ces quatre questions devaient être apportées sur une échelle de Likert en trois, quatre ou cinq options de réponse (et donc codés de 0 à 2, 3 ou 4 points). Les quatre dernières questions proposaient aux élèves de répondre par oui ou par non (0 à 1 point), selon s'ils ont déjà programmé en dehors de l'école, s'ils ont déjà réalisé des activités de programmation débranchées (sans ordinateur ni outil numérique), s'ils ont déjà utilisé des logiciels de programmation tels que Scratch, Blockly etc., et s'ils ont déjà programmé un robot ou un objet connecté. Au total, le questionnaire de niveau de pratique en programmation était évalué sur 16 points.

Nous avons conçu deux versions de notre Test de Maîtrise des Concepts Computationnels : une version papier et une version en ligne (réalisée à l'aide du logiciel Google forms). Pour ce faire, nous nous sommes appuyés sur la taxonomie révisée de Bloom (Krathwohl, 2002). Cette taxonomie des objectifs éducatifs se présente sous la forme d'un système de classification des buts, des objectifs et, plus récemment, des normes en matière d'éducation. Elle distingue deux composantes essentielles. D'une part, la dimension des connaissances regroupe les connaissances factuelles, conceptuelles, procédurales ou métacognitives. D'autre part, la dimension des processus cognitifs regroupe la capacité à se remémorer, comprendre, appliquer, analyser, évaluer et créer. Il est important de remarquer que ces processus sont hiérarchisés (et souvent représentés sous la forme d'une « pyramide ») en fonction de leur niveau de complexité, du plus simple (« se remémorer ») au plus complexe (« créer »).

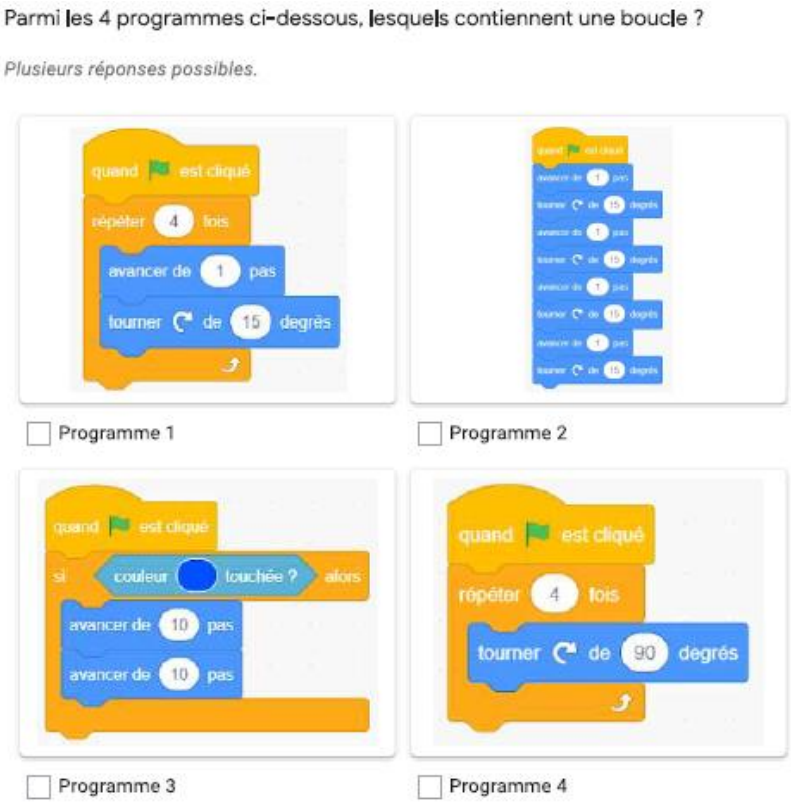
Notre outil de mesure propose plutôt de tester les connaissances factuelles, à savoir les éléments de base en programmation et la terminologie qui leur est associée, ainsi que les

connaissances conceptuelles des élèves, à savoir les relations entre ces éléments de base (les concepts de programmation) qui peuvent être catégorisés, classifiés ou structurés. Certaines connaissances procédurales peuvent aussi être mises en jeu, comme nous le verrons un peu plus loin. De même, notre test est pensé pour évaluer les quatre types de processus cognitifs les plus simples, c'est-à-dire la capacité à se remémorer les concepts fondamentaux en programmation (retrouver des connaissances pertinentes), à comprendre ces concepts (en déterminer le sens), à appliquer les connaissances liées à ces concepts (réaliser ou utiliser une procédure dans une situation donnée) et enfin à analyser (décomposer quelque chose en ses éléments constitutifs et détecter les relations entre ces éléments).

Nous avons ainsi conçu 22 items qui constituent ce Test de Maîtrise des Concepts Computationnels. Les cinq premiers items sont des items de catégorisation (chacun est relatif à un des cinq concepts évalués par ce test), pour lesquels l'élève doit distinguer parmi plusieurs propositions ce qui relève ou non du concept en question. Par exemple, l'item 1 propose quatre textes différents et l'élève doit sélectionner ceux qui sont des algorithmes. La Figure 2 est un exemple d'item de catégorisation. Les quatre items suivants sont des items d'application, directement issus et traduits du *Computational Thinking Test* de Román-González (2015). Les élèves sont confrontés à une illustration représentant un personnage (Pac-Man) devant être programmé pour atteindre un autre personnage (un fantôme). Un ou plusieurs programmes leur sont proposés et ceux-ci doivent, selon l'item concerné, indiquer quelle est l'instruction fausse, quelle est l'instruction manquante ou tout simplement quel programme permet d'atteindre l'objectif donné. Ceci implique que l'élève maîtrise les connaissances procédurales qui consistent à exécuter mentalement un programme pour en déterminer le résultat. Deux de ces items concernent le concept de boucle et les deux autres le concept d'instruction conditionnelle, car ce type d'items est mieux adapté à l'évaluation de ces deux concepts.

Figure 2

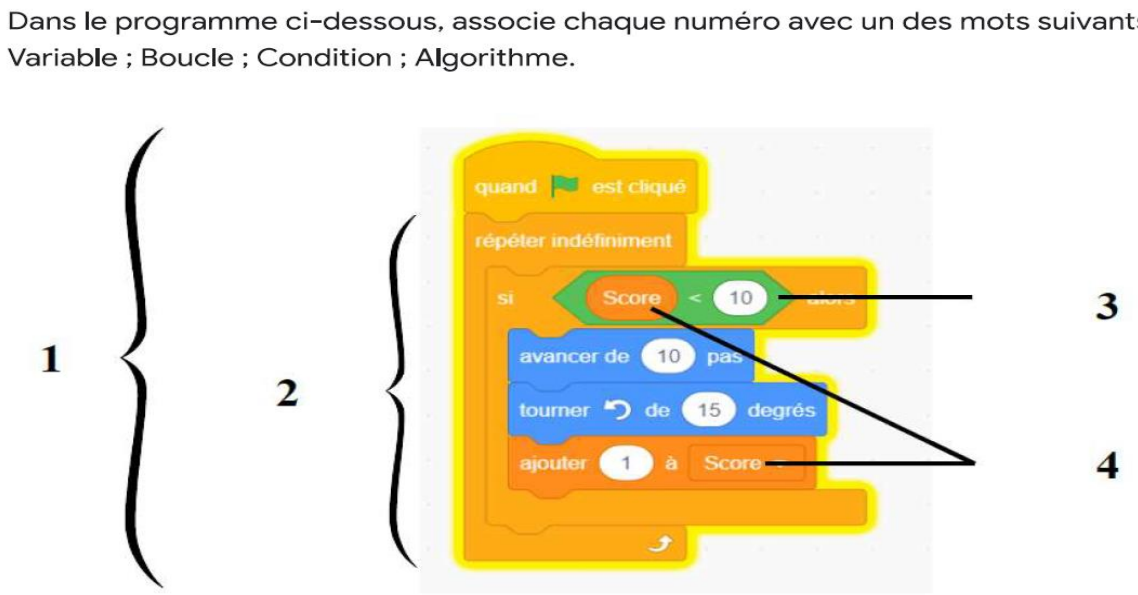
Exemple d'item de catégorisation relatif au concept de boucle



Les items 10 à 14 sont des affirmations pour lesquelles les élèves doivent indiquer si elles sont vraies ou fausses. Ces affirmations mettent systématiquement en relation plusieurs de nos concepts. Pour les items 15 à 18, un programme est proposé aux élèves. Certaines parties de ce programme sont désignées par des flèches ou des accolades numérotées. Les élèves doivent associer ce qui est indiqué par les flèches ou accolades aux concepts d'algorithme, de boucle, de condition ou de variable, comme illustré sur la Figure 3. Ceci permet donc d'évaluer la capacité à différencier les constituants d'un programme et à les organiser les uns à l'intérieur des autres (par exemple, une boucle est intégrée à l'intérieur d'un algorithme). Aucune de ces flèches ou accolades ne correspond au concept d'instruction, afin de ne pas introduire une confusion. En effet, une boucle ou une instruction conditionnelle sont des instructions. Enfin, les quatre derniers items sont des textes à trous qui proposent aux élèves de retrouver les définitions des différents concepts (chaque trou doit être complété par un des concepts).

Figure 3

Items 15 à 18 du Test de Maîtrise des Concepts Computationnels (ici les élèves doivent associer chaque flèche ou accolade à un concept computationnel)



Le Tableau 1 associe chaque catégorie d'items à une ou plusieurs dimensions des connaissances et à une ou plusieurs dimensions des processus cognitifs (entre parenthèses la sous-dimension correspondante), selon la taxonomie révisée de Bloom (Krathwohl, 2002).

Tableau 1

Répartition des items en fonction des dimensions des connaissances et des dimensions des processus cognitifs qui y sont associés, selon la taxonomie révisée de Bloom (Krathwohl, 2002)

Dimensions	**5 items catégorisation**	**4 items application**	**5 items vrai / faux**	**4 items reconnaissance**	**4 items textes à trous**
Connaissances	Factuelles	Procédurales	Factuelles Conceptuelles	Conceptuelles	Factuelles Conceptuelles
Processus cognitifs	Comprendre (classifier)	Appliquer (exécuter)	Analyser (différentier)	Se remémorer (reconnaître) Analyser (différentier – organiser)	Se remémorer (rappeler) Analyser (différentier)

Pour chaque question, les élèves étaient informés si celle-ci exigeait une seule réponse correcte ou si plusieurs réponses correctes étaient possibles. Lorsqu'une seule réponse était attendue, une réponse correcte valait 1 point et toute autre réponse valait 0 point. Lorsque

plusieurs réponses étaient correctes, cet item valait autant de points qu'il y avait de réponses correctes. Chaque bonne réponse valait 1 point et chaque mauvaise réponse valait -1 point. Si le nombre de points obtenus pour un item était négatif, cette valeur était ramenée à 0. Les items peuvent être regroupés en cinq catégories différentes correspondant aux cinq concepts fondamentaux : algorithme (quatre items, 7 points), instruction (deux items, 5 points), boucle (six items, 8 points), condition (six items, 7 points) et variable (quatre items, 4 points). Ce test est conçu pour évaluer une performance (sur 31 points). Ainsi, les résultats totaux ont été pris en compte même si certains items sont restés sans réponse. Le Test de Maîtrise des Concepts Computationnels est disponible en annexes (Annexe C). Les cinq items vrais / faux ayant été supprimés sont disponibles en Annexe D.

Procédure

Avant de passer le test, les élèves devaient tout d'abord remplir le questionnaire évaluant leur niveau de pratique en programmation. Pour la version papier du test, les élèves ont passé consécutivement le Test de Maîtrise des Concepts Computationnels (en premier) et le Test d'Algorithmique (version 1, voir plus loin). Pour la version en ligne, les élèves n'ont passé que le Test de Maîtrise des Concepts Computationnels. Les interlocuteurs à qui nous avons présenté ce test (professeurs, membres d'associations…) ont été informés que les élèves devaient travailler seuls et ne pouvaient bénéficier de l'aide d'un adulte, à l'exception de certaines précisions ponctuelles qui pouvaient être apportées concernant le vocabulaire employé ou la formulation des questions, lorsque celles-ci étaient jugées nécessaires. La réalisation des deux tests (version papier) ne pouvait pas dépasser une heure. Lorsque le test de Maîtrise des Concepts Computationnels a été réalisé seul (version en ligne), un temps maximum de 20 minutes avait été fixé. L'ensemble des questionnaires était anonymisé.

1.2.2) Résultats

Afin d'analyser les résultats obtenus, nous allons mesurer un indice de fiabilité et un indice de validité du test que nous proposons. La fiabilité est entendue comme la capacité d'un test à reproduire un résultat de manière cohérente dans le temps et l'espace alors que la validité se réfère à la capacité d'un test à mesurer exactement ce qu'il est censé mesurer (El-Hamamsy et al., 2022). L'impact du niveau scolaire et du genre des élèves sur leur niveau de pratique et leurs performances au test a également été mesuré. Les analyses que nous avons menées ont été réalisées sur le logiciel Jamovi, version 2.3.26.0.

Analyse de fiabilité – version en ligne

Pour nos analyses de fiabilité, nous utiliserons le coefficient ω de Mcdonald, reconnu comme étant un indicateur plus pertinent et fiable que le très utilisé coefficient α de Cronbach, en particulier en ce qui concerne la recherche appliquée (Dunn et al., 2014 ; Trizano-Hermosilla & Alvarado, 2016). La fiabilité globale du test peut être considérée comme élevée, selon Taber (2018), car ω = .834. Le Tableau 2 présente, pour certains items, la valeur du coefficient ω si cet item était supprimé du test ainsi que le lien de corrélation entre l'item en question et le reste des items, mesuré à l'aide du coefficient r de Pearson. Nous n'avons reporté dans ce Tableau que les items pour lesquels une suppression entraîne une augmentation de la fiabilité globale du test. Les items sont notés A, I, B, C ou V selon s'ils permettent d'évaluer, respectivement, la maîtrise du concept d'algorithme, d'instruction, de boucle, de condition ou de variable. Par exemple, l'item I2 correspond au deuxième item portant sur le concept d'instruction.

Tableau 2

Valeur de ω si l'item correspondant est supprimé et lien de corrélation entre chacun de ces items et les autres items du Test de Maîtrise des Concepts Computationnels (version en ligne)

Item	Valeur de ω si l'item est retiré	Corrélation item – reste du test
I2	.838	.13
B4	.836	.21
C4	.837	.17
V2	.843	-.03

Nous pouvons observer dans le Tableau 2 que la fiabilité du test augmente lorsque les items I2, B4, C4 et V2 sont retirés. Ce sont aussi ces quatre items qui corrèlent le moins avec le reste du test (avec une valeur de *r* comprise entre -.03 et .21). Ces items correspondent à quatre des cinq items vrai / faux de notre test. L'item A2, également un item vrai / faux, ne semble pas poser problème. Cependant, conserver un seul des items vrai / faux aurait peu de sens, d'autant plus que celui-ci n'est pas non plus l'item qui corrèle le mieux avec les autres items du test. Ainsi, nous avons pris la décision de supprimer l'ensemble des items vrai / faux de notre test, en raison de leur manque de fiabilité. Ceci a pour conséquence d'augmenter la fiabilité globale de notre test (ω = .85).

Il peut parfois être intéressant de mesurer la fiabilité de chaque facteur mesuré, c'est-à-dire, dans notre contexte, de chaque concept dont on évalue la maîtrise. Cependant, et ce

d'autant plus après le retrait des cinq items vrai / faux, le nombre d'items par concept demeure faible (entre trois et cinq items pour les concepts d'algorithme, de boucle, de condition et de variable, un seul item pour le concept d'instruction). Pour cette raison, il est peu probable que la cohérence interne de chaque facteur mesuré soit élevée, puisque la fiabilité tend à augmenter avec le nombre d'items (Taber, 2018). Pour le concept d'instruction, basé sur un seul item après la suppression des items vrai / faux, cette mesure ne peut pas être effectuée. Pour les concepts d'algorithme, de boucle, de condition et de variable, les mesures de fiabilité sont respectivement : $\omega = .59$, $\omega = .56$, $\omega = .58$ et $\omega = .50$. Notons que, malgré le faible nombre d'items, ces valeurs sont toutes supérieures ou égales à .50, ce qui correspond à une fiabilité modérée selon Hinton et al. (2014), cités par El-Hamamsy et al. (2022), et peut être considéré comme acceptable selon certains auteurs (Taber, 2018).

Validité – version en ligne

Nous avons mesuré les liens de corrélation qui existent entre d'une part, le niveau de pratique de la programmation des élèves et d'autre part, leurs performances à ce test. Le niveau de pratique et les performances globales au test sont liés par une corrélation positive d'intensité modérée ($r = .61$). Concernant les différents concepts dont la maîtrise est mesurée dans ce test, il existe également des liens de corrélations positifs d'intensité modérée avec le niveau de pratique en programmation, bien que ces coefficients soient plus faibles, que ce soit pour le concept d'algorithme ($r = .48$), d'instruction ($r = .52$), de boucle ($r = .50$), de condition ($r = .49$) ou de variable ($r = .45$). Toutes ces corrélations sont très significatives ($p < .001$). Par ailleurs, nous avons pu observer que les performances au test varient significativement selon le niveau de pratique en programmation des élèves, avec un effet fort ($F(14, 434) = 21.9$, $p < .001$, $\eta^2 = .414$).

Différences en fonction du genre et du niveau scolaire – version en ligne

Pour l'ensemble des 449 élèves ayant passé la version en ligne du test de Maîtrise des Concepts Computationnels, la moyenne, après suppression des items vrai / faux, est de 14.6/26 ($ET = 6.09$). Les moyennes et écarts-types pour la maîtrise des cinq concepts évalués sont les suivantes : algorithme ($M = 3.33/6$, $ET = 1.61$), instruction ($M = 2.86/4$, $ET = 1.14$), boucle ($M = 3.64/7$, $ET = 2.01$), condition ($M = 3.33/6$, $ET = 1.72$), variable ($M = 1.47/3$, $ET = 1.05$). Les résultats obtenus démontrent un effet significatif fort du niveau scolaire sur les performances au test ($F(5, 443) = 27.6$, $p < .001$, $\eta^2 = .237$). Au vu du faible échantillon d'élèves de niveau CM1, CM2 et 5e (respectivement, $N = 3$, $N = 19$ et $N = 16$), nous avons comparé deux à deux

les élèves de niveau 6e, 4e et 3e (respectivement, $N = 98$, $N = 183$ et $N = 130$). Le Tableau 3 rend compte des statistiques descriptives relatives au niveau de pratique en programmation et aux performances au test de ces élèves, en fonction de leur niveau scolaire.

Tableau 3

Moyennes (et écarts-types entre parenthèses) des scores au questionnaire de niveau de pratique en programmation et au Test de Maîtrise des Concepts Computationnels (version en ligne), selon le niveau scolaire

Niveau scolaire	Niveau de pratique en programmation	Score total – test de Maîtrise des Concepts Computationnels
6e	3.66 (3.78)	9.33 (4.70)
4e	6.75 (4.62)	15.7 (5.65)
3e	7.68 (4.00)	17.3 (5.16)

La distribution du niveau de pratique en programmation ne suit pas la loi normale, ce qui nous a conduits à utiliser le test non paramétrique de Kruskal-Wallis et à mener des comparaisons post-hoc de Dwass, Steel, Critchlow et Fligner, concernant cette variable. Les élèves de 4e présentent un niveau de pratique de la programmation et un niveau de performance significativement plus élevés que les élèves de 6e, avec une grande taille d'effet (respectivement, $W(280) = 7.77$, $p < .001$ et $t(280) = 9.52$, $p < .001$, $d = 1.19$). De même, le niveau de pratique et les performances des élèves de 3e sont plus élevés que ceux des élèves de 6e, encore une fois avec une grande taille d'effet (respectivement, $W(227) = 10.07$, $p < .001$ et $t(227) = 11.13$, $p < .001$, $d = 1.49$). En revanche, même si les élèves de 3e présentent des niveaux de pratique et de performances plus élevés que ceux des élèves de 4e, ces différences ne sont pas significatives (respectivement, $W(312) = 2.24$, $p = .61$ et $t(312) = 2.59$, $p = .10$). Le Tableau 4 rend compte des statistiques descriptives relatives au niveau de pratique en programmation et aux performances au test des élèves, en fonction de leur genre.

Tableau 4

Moyennes (et écarts-types entre parenthèses) des scores au questionnaire de niveau de pratique en programmation et au Test de Maîtrise des Concepts Computationnels (version en ligne), selon le genre

Genre	Niveau de pratique en programmation	Score total – test de Maîtrise des Concepts Computationnels
Féminin	6.16 (4.40)	15.0 (6.21)
Masculin	6.33 (4.58)	14.2 (5.93)

Les différences observées ici ne sont pas significatives, qu'elles concernent l'impact du genre sur le niveau de pratique de la programmation, ou sur les performances au test (respectivement, $\chi^2 = 0.12$, $p = .73$ et $F(1, 447) = 1.78$, $p = .18$).

Analyse de fiabilité – version papier

La fiabilité globale du test en version papier peut être considérée comme élevée ($\omega = .826$). Le Tableau 5 présente la valeur du coefficient ω si les items correspondants étaient supprimés du test ainsi que le lien de corrélation entre l'item en question et le reste des items, mesuré à l'aide du coefficient *r* de Pearson.

Tableau 5

Valeur de ω si l'item correspondant est supprimé et lien de corrélation entre chacun de ces items et les autres items du Test de Maîtrise des Concepts Computationnels (version papier)

Item	Valeur de ω si l'item est retiré	Corrélation item – reste du test
I2	.830	.11
B4	.827	.19
C4	.835	- .03
V2	.832	.06

Comme pour la version en ligne, nous pouvons observer que la fiabilité du test augmente lorsque les items I2, B4, C4 et V2 sont retirés. Cette version papier confirme la nécessaire suppression des items vrai / faux. Ceci a pour conséquence d'augmenter la fiabilité globale de notre test ($\omega = .85$). Pour les concepts d'algorithme, de boucle, de condition et de variable, les mesures de fiabilité sont respectivement de $\omega = .52$, $\omega = .57$, $\omega = .57$ et $\omega = .58$.

Validité – version papier

Le niveau de pratique et les performances globales au test sont liés par une corrélation positive d'intensité modérée (r = .54). Concernant les différents concepts dont la maîtrise est mesurée dans ce test, il existe également des liens de corrélations positifs d'intensité modérée avec le niveau de pratique en programmation, bien que ces coefficients soient plus faibles, que ce soit pour le concept d'algorithme (r = .44), d'instruction (r = .36), de boucle (r = .44), de condition (r = .49) ou de variable (r = .40). Toutes ces corrélations sont significatives à p < .001. Par ailleurs, nous avons pu observer que les performances au test varient significativement selon le niveau de pratique en programmation des élèves, avec un effet de grande taille (F(14, 160) = 6.18, p < .001, η^2 = .351).

Différences en fonction du genre et du niveau scolaire – version papier

Pour l'ensemble des 175 élèves ayant passé la version en ligne du test de Maîtrise des Concepts Computationnels, la moyenne, après suppression des items vrai / faux, est de 14.6/26 (ET = 5.94). Les moyennes et écarts-types pour la maîtrise des cinq concepts évalués sont les suivants : algorithme (M = 3.63/6, ET = 1.58), instruction (M = 2.85/4, ET = 1.11), boucle (M = 3.40/7, ET = 2.06), condition (M = 3.33/6, ET = 1.62), variable (M = 1.37/3, ET = 1.09). Les résultats obtenus indiquent un effet significatif de grande taille du niveau scolaire sur les performances au test (F(2, 172) = 40.1, p < .001, η^2 = .318). Au vu du faible échantillon d'élèves de niveau CM2 (N = 8), nous avons comparé deux à deux les élèves de niveau 5e et 3e (respectivement, N = 100 et N = 67). Le Tableau 6 rend compte des statistiques descriptives relatives au niveau de pratique en programmation et aux performances au test de ces élèves, en fonction de leur niveau scolaire.

Tableau 6

Moyennes (et écarts-types entre parenthèses) des scores au questionnaire de niveau de pratique en programmation et au Test de Maîtrise des Concepts Computationnels (version papier), selon le niveau scolaire

Niveau scolaire	Niveau de pratique en programmation	Score total – test de Maîtrise des Concepts Computationnels
5e	6.27 (3.45)	12.2 (5.09)
3e	7.48 (2.79)	18.8 (4.86)

La distribution du niveau de pratique en programmation ne suit pas la loi normale, ce qui nous a conduits à utiliser le test non paramétrique de Kruskal-Wallis, concernant cette variable. Les élèves de 3e présentent un niveau de pratique de la programmation et un niveau de performance significativement plus élevés que les élèves de 5e, avec un effet de grande taille en ce qui concerne les performances (respectivement $\chi^2 = 15.2$, $p < .001$, $\varepsilon^2 = .087$ et $t(166) = 8.45$, $p < .001$, $d = 1.33$). Le Tableau 7 rend compte des statistiques descriptives relatives au niveau de pratique en programmation et aux performances au test des élèves, en fonction de leur genre.

Tableau 7

Moyennes (et écarts-types entre parenthèses) des scores au questionnaire de niveau de pratique en programmation et au Test de Maîtrise des Concepts Computationnels (version en ligne), selon le genre

Genre	Niveau de pratique en programmation	Score total – test de Maîtrise des Concepts Computationnels
Féminin	6.74 (3.07)	15.3 (5.87)
Masculin	6.58 (3.34)	13.9 (5.96)

Les différences observées ici ne sont pas significatives, qu'elles concernent l'impact du genre sur le niveau de pratique de la programmation, ou sur les performances au test (respectivement, $\chi^2 = 0.17$, $p = .68$ et $F(1, 173) = 2.41$, $p = .12$).

1.3) Test d'Algorithmique

Le deuxième test que nous avons conçu propose d'évaluer la capacité des élèves à résoudre des problèmes algorithmiques, c'est-à-dire des problèmes dont la solution peut être formulée sous la forme d'un algorithme. Plus spécifiquement, ce test vise à mesure la capacité à mettre en pratique les concepts fondamentaux en programmation (dont nous avons évalué la maîtrise dans notre Test de Maîtrise des Concepts Computationnels) afin de résoudre un problème concret. Les problèmes proposés impliquent nécessairement, pour parvenir à leur résolution, d'user de certaines compétences, souvent associées à la pensée informatique, telles que la pensée algorithmique, l'abstraction, la généralisation, etc. Nous avons nommé ce test : Test d'Algorithmique. L'objectif est de valider un test en français, relativement court comparé aux autres outils d'évaluation des connaissances et compétences en pensée informatique ou en

programmation, qui pourra nous servir pour nos expériences futures. Contrairement à la quasi-totalité des outils de mesure existants, notre Test d'Algorithmique implique la production d'un algorithme (écrit avec un langage naturel) pour résoudre les problèmes posés et ne repose donc pas sur des questions à choix multiples. De même, nous avons fait le choix de concevoir un test proposant des problèmes de programmation et non des problèmes de la vie quotidienne, ou *a minima* non reliés à des activités de programmation, et dont la résolution est censée être révélatrice de certaines compétences en pensée informatique, comme c'est très souvent le cas dans ce domaine.

1.3.1) Méthode (version 1 du test)

Participants

La validation de ce test a mené au recrutement de 175 participants (82 filles) dont 8 élèves de CM2, 100 de 5e et 67 de 3e. Ces élèves sont ceux qui ont été recrutés pour la version papier du Test de Maîtrise des Concepts Computationnels. Cependant, seuls 136 élèves (66 filles) ont tenté de réaliser ce test et ont fourni des réponses exploitables (dont 86 élèves de 5e et 50 élèves de 3e). Initialement, nous avions également prévu une version en ligne de ce test, pour laquelle nous avions déjà recruté plusieurs dizaines de participants. Cependant, cette version, réalisée sur le logiciel Qualtrics, ne permettait pas d'exploiter correctement les résultats. En effet, la zone de texte utilisée pour fournir une réponse aux problèmes posés, sous forme d'algorithmes, autorisait la possibilité d'aller à la ligne ou de sauter une ligne pour bien séparer les instructions et rendre compréhensible la structure proposée, ce qui est nécessaire (en particulier lorsque des boucles ou des instructions conditionnelles sont utilisées). Malheureusement, les données que nous avons recueillies à l'aide du logiciel ne rendaient pas compte de cette mise en forme, ce qui rendait l'interprétation des résultats périlleuse et incertaine. Cette version a finalement été abandonnée. Pour la même raison que celle évoquée pour le Test de Maîtrise des Concepts Computationnels, nous avons validé notre Test d'Algorithmique en faisant prioritairement appel à des élèves de cycle 4, même si celui-ci a vocation à être utilisé dans nos expériences avec des élèves de CM2. Nous avons bénéficié du réseau de la DRANE de l'Académie de Montpellier pour recruter des classes de différents niveaux dans cette région.

Matériel

Nous avons utilisé le même questionnaire visant à évaluer le niveau de pratique de la programmation des élèves que celui décrit dans la partie « Méthode – Matériel » du Test de Maîtrise des Concepts computationnels avec le même objectif : permettre de mesurer un lien de corrélation entre ce niveau de pratique de la programmation et les performances des élèves au Test d'Algorithmique. En se référant toujours à la taxonomie révisée de Bloom (Krathwohl, 2002), ce nouvel outil de mesure propose cette fois-ci de tester principalement les connaissances procédurales des élèves, c'est-à-dire des connaissances relatives à comment faire quelque chose (ici, produire un algorithme pour résoudre un problème). Par ailleurs, on peut également considérer que des connaissances métacognitives peuvent être mises en jeu, en particulier les connaissances stratégiques qui permettent de résoudre des problèmes de programmation. De plus, notre test est pensé pour évaluer essentiellement le processus cognitif le plus complexe, selon la taxonomie révisée de Bloom (Krathwohl, 2002), qui est la capacité à créer, c'est-à-dire à assembler des éléments pour former un ensemble nouveau et cohérent ou à réaliser un produit original. Cette dimension se décompose en trois sous-dimensions essentielles dans le processus de conception d'un algorithme : « générer », « planifier » et « produire ». Le processus d'évaluation (« évaluer »), décrit par cette même taxonomie, est également une dimension des processus cognitifs impliquée dans la réalisation de ce test. Il consiste à porter un jugement sur la base de critères et de normes et se décompose en deux sous-dimensions : « vérifier » et « critiquer ». L'évaluation et la vérification de la pertinence des instructions utilisées font partie du processus naturel de la conception d'un algorithme.

Nous avons ainsi conçu 4 exercices pratiques qui constituent ce Test d'Algorithmique, nécessitant la création d'un algorithme comme solution aux problèmes décrits. En introduction, un personnage de robot est présenté (nommé « BOT »). Il est expliqué que ce robot comprend le français et peut être programmé pour réaliser certaines tâches à condition de lui donner des instructions claires et précises. Deux exemples d'algorithme, permettant à « BOT » de se déplacer sur une grille, sont fournis aux élèves pour les aider à comprendre comment ils doivent écrire leur algorithme. Le langage utilisé est un langage naturel (en français) et deux formulations différentes sont proposées aux élèves pour les inciter à comprendre que l'important n'est pas le vocabulaire et la syntaxe utilisés, mais la précision des instructions données.

Pour chaque exercice, les élèves devaient programmer plusieurs actions et chaque étape à programmer était numérotée pour faciliter leur travail. Les deux premiers exercices sont des exercices concrets demandant aux élèves de programmer les mouvements d'un personnage sur une grille. Les deux derniers exercices sont plus abstraits, car ils n'impliquent pas la programmation de mouvements. Les élèves devaient programmer « BOT » pour qu'il compte indéfiniment (exercice 3) ou pour qu'il compte le temps qui passe, comme un chronomètre (exercice 4). La Figure 4 illustre l'exercice 3, plus abstrait et plus complexe que les précédents. Dans chaque exercice, la capacité à produire un algorithme avec des instructions pertinentes a été évaluée, ainsi que la capacité à utiliser des boucles (uniquement dans les exercices 1, 3 et 4), des instructions conditionnelles (exercices 2 et 4) ou des variables (exercices 1, 2 et 4). Dans les deux premiers exercices, les élèves étaient explicitement informés qu'ils devaient utiliser une boucle, une instruction conditionnelle ou une variable, alors que dans les deux derniers exercices, ils ont parfois dû trouver cette solution par eux-mêmes, ce qui rend ces exercices beaucoup plus difficiles.

Figure 4

Exercice 3 du Test d'Algorithmique (version 1)

Exercice 3 : Compter à voix haute

1) Réalise un algorithme pour que **BOT** compte de 1 à 5 à voix haute. Tu peux utiliser l'instruction « Dire » par exemple.
2) Rajoute une instruction pour que **BOT** recommence à compter à partir de 1 quand il arrive à 5.
3) Crée une variable que tu appelles « Nombre ». Mets cette variable à 1 au début de ton programme.
4) Dans l'instruction « Dire 1 », remplace le 1 par la variable « Nombre ».
5) Trouve un moyen pour que **BOT** compte jusqu'à l'infini sans s'arrêter et supprime les instructions qui sont maintenant inutiles.

Dans chaque exercice, la capacité à formuler une réponse sous forme d'algorithme (c'est-à-dire une suite d'instructions ordonnées) valait 1 point et la pertinence des instructions (précision et cohérence des termes utilisés) est également notée sur 1 point. Lorsque des mouvements devaient être programmés pour compléter un chemin, la capacité à programmer correctement ce chemin permettait l'obtention d'1 point supplémentaire. Lorsque l'utilisation d'une boucle ou d'une condition était requise, 1 point était attribué pour l'utilisation de l'énoncé correct (« répéter » ou « si ... alors ... »). La capacité à utiliser correctement les boucles et les

conditions pour obtenir le résultat requis a également été notée sur 1 point. Enfin, la capacité à utiliser des variables a été notée de 1 à 3 points en fonction de l'exercice. L'initialisation d'une variable, son incrémentation et la capacité à remplacer une valeur par une variable ont été évaluées sur 1 point chacune.

Ainsi, même si les compétences associées à la pensée informatique (capacité à appréhender un problème et sa solution à différents niveaux d'abstraction, capacité à décomposer un problème complexe en plusieurs sous-problèmes plus simples, etc.) sous-tendent la résolution de ces problèmes, celles-ci ne sont pas directement évaluées. Les items peuvent être regroupés en quatre catégories différentes correspondant à la mise en pratique des concepts computationnels pour résoudre des problèmes algorithmiques : la capacité à produire un algorithme composé d'instructions pertinentes pour le problème à résoudre (quatre items, 10 points) et la capacité à utiliser des boucles (trois items, 6 points), des instructions conditionnelles (deux items, 4 points) et des variables (trois items, 6 points). Ce test est conçu pour évaluer une performance (sur 26 points). Ainsi, les résultats totaux ont été pris en compte même si certains items sont restés sans réponse. Le Test d'Algorithmique (version 1) est disponible en Annexe E.

Procédure

Avant de passer le test, les élèves devaient tout d'abord remplir le questionnaire évaluant leur niveau de pratique en programmation. Les élèves ont passé ce Test d'Algorithmique après avoir passé le Test de Maîtrise des Concepts Computationnels, comme nous avions prévu de le faire pour l'expérience 1 que nous avons réalisée par la suite (voir « Chapitre 2 » de cette « Partie expérimentale »). Les professeurs qui ont fait passer ce test à leurs élèves ont été informés que ceux-ci devaient travailler seuls et ne pouvaient bénéficier de leur aide, à l'exception de certaines précisions ponctuelles qui pouvaient être apportées concernant le vocabulaire employé ou la formulation des questions, lorsque celles-ci étaient jugées nécessaires. La réalisation des deux tests ne pouvait pas dépasser une heure. L'ensemble des questionnaires était anonymisé.

1.3.2) Résultats (version 1 du test)

Comme pour le Test de Maîtrise des Concepts Computationnels, nous avons mesuré un indice de fiabilité et un indice de validité du Test d'Algorithmique. L'impact du niveau scolaire et du genre des élèves sur leur niveau de pratique et leurs performances au test a également été

mesuré. Ce test impliquait, pour les élèves, la création de plusieurs algorithmes pour proposer une solution aux problèmes qui leur étaient posés. À la différence du Test de Maîtrise des Concepts Computationnels, celui-ci ne repose donc pas sur des questions à choix multiples et peut ainsi être considéré comme plus complexe. Par conséquent, le taux de complétion de ce test était nettement inférieur. Nous avons donc également compté le nombre et la proportion d'élèves ayant *a minima* essayé de produire un algorithme, pour chaque exercice du test. Les analyses que nous avons menées ont été réalisées sur le logiciel Jamovi, version 2.3.26.0.

Analyse de fiabilité

La fiabilité globale du test peut être considérée comme élevée (ω = .899). Deux items, relatifs à la production de boucles, produiraient une augmentation de la fiabilité globale du test, s'ils étaient supprimés (ω = .902 et ω = .903). Cependant, ces deux items sont issus de deux exercices distincts et sont intégrés de manière cohérente avec le reste de chacun de ces exercices. Il ne nous semble donc pas pertinent de les supprimer, en particulier au vu du peu d'impact que cette décision aurait sur l'indice de fiabilité globale. La fiabilité des items relatifs à la capacité à produire un algorithme composé d'instructions pertinentes est très élevée (ω = 0.91). Ce résultat n'est pas surprenant, car un élève capable de créer un algorithme pertinent pour un exercice devrait normalement en être capable pour les autres exercices. Les autres indices de fiabilité relatifs à la capacité à produire des boucles (ω = 0.58), à la capacité à utiliser des instructions conditionnelles (ω = 75) et à la capacité à utiliser des variables (ω = .85) sont logiquement inférieurs.

Validité

Les résultats au Test d'Algorithmique ne suivent pas une distribution normale. Par conséquent, nous avons utilisé le coefficient ρ de Spearman, plutôt que le coefficient *r* de Pearson. Le niveau de pratique et les performances globales au test sont liés par une corrélation positive d'intensité modérée (ρ = .54). Concernant les différentes capacités qui sont évaluées dans ce test, il existe également des liens de corrélations positifs d'intensité modérée avec le niveau de pratique en programmation, bien que ces coefficients soient plus faibles, que ce soit pour la capacité à produire des algorithmes composés d'instructions pertinentes (ρ = .51), la capacité à produire des boucles (ρ = .47), des instructions conditionnelles (ρ = .35) et la capacité à utiliser des variables (ρ = .44). Toutes ces corrélations sont très significatives ($p < .001$). Le niveau de pratique de la programmation est également différemment corrélé aux exercices proposés dans ce test (exercice 1, ρ = .56 ; exercice 2, ρ = .44 ; exercice 3, ρ = .57 ; exercice 4,

$\rho = .76$). Par ailleurs, nous avons pu observer, à l'aide d'un test non paramétrique de Kruskal-Wallis, que les performances au test varient significativement selon le niveau de pratique en programmation des élèves, avec un effet fort ($\chi^2 = 52.5$, $p < .001$, $\varepsilon^2 = .389$).

Taux de complétion par exercice

Comme nous l'avons déjà évoqué dans la partie consacrée à la méthode de ce test, sur les 175 élèves qui ont été recrutés, seuls 136 se sont essayés à réaliser au moins un de ces exercices. De plus, seule une petite minorité de ces 136 élèves a pu aller au bout du test. 135 d'entre eux ont réalisé le premier exercice (99.3%), 100 le deuxième (73.5%), 37 le troisième (27.2%) et seulement 14 le dernier (10.3%).

Différences en fonction du genre et du niveau

Pour l'ensemble des 136 élèves ayant passé ce Test d'Algorithmique, la moyenne est de 6.69/26 (*ET* = 4.89). On remarque que cette moyenne est faible, même si cela s'explique par l'absence de réponses de la majorité des participants aux exercices 3 et 4. Les moyennes et écarts-types pour les sous-variables mesurées sont les suivants : capacité à produire un algorithme composé d'instructions pertinentes (*M* = 4.80/10, *ET* = 2.42), capacité à produire des boucles (*M* = 0.97/6, *ET* = 1.26), et des instructions conditionnelles (*M* = 0.47/4, *ET* = 0.93), capacité à utiliser des variables (*M* = 0.45/6, *ET* = 1.15). Le Tableau 8 rend compte des statistiques descriptives relatives au niveau de pratique en programmation et aux performances au test de ces élèves, en fonction de leur niveau scolaire.

Tableau 8

Moyennes (et écarts-types entre parenthèses) des scores au questionnaire de niveau de pratique en programmation et au Test d'Algorithmique (version 1), selon le niveau scolaire

Niveau scolaire	Niveau de pratique en programmation	Score total – Test d'Algorithmique (version 1)
5e	6.05 (3.37)	5.65 (4.52)
3e	7.44 (2.97)	8.47 (5.04)

Le niveau de pratique en programmation ne suit pas une distribution normale, ce qui nous a conduits à utiliser le test non paramétrique de Kruskal-Wallis. Les élèves de 3e présentent un niveau de pratique de la programmation et un niveau de performance

significativement plus élevés que les élèves de 5e, avec un effet de grande taille concernant les performances (respectivement, $\chi^2 = 8.12$, $p < .01$, $\varepsilon^2 = .060$ et $\chi^2 = 16.6$, $p < .001$, $\varepsilon^2 = .123$). Le Tableau 9 rend compte des statistiques descriptives relatives au niveau de pratique en programmation et aux performances au test des élèves, en fonction de leur genre.

Tableau 9

Moyennes (et écarts-types entre parenthèses) des scores au questionnaire de niveau de pratique en programmation et au Test d'Algorithmique (version 1), selon le genre

Genre	Niveau de pratique en programmation	Score total – Test d'Algorithmique (version 1)
Féminin	6.80 (3.12)	6.77 (4.29)
Masculin	6.33 (3.45)	6.61 (5.42)

Les différences observées ici ne sont pas significatives, qu'elles concernent l'impact du genre sur le niveau de pratique de la programmation, ou sur les performances au test (respectivement, $\chi^2 = 0.50$, $p = .48$ et $\chi^2 = 0.69$, $p = .41$).

1.3.3) Méthode (version 2 du test)

Au vu des faiblesses de notre premier Test d'Algorithmique (analysées de manière plus détaillée dans la partie « Discussion » de ce chapitre expérimental), nous avons conçu une deuxième version que nous espérons mieux adaptée. En effet, la validation de la première version a révélé un taux d'abandon conséquent et une absence de réponse massive pour les deux derniers exercices du test (les plus complexes). Par conséquent, la moyenne des performances observées à ce test est faible. La deuxième version du Test d'Algorithmique poursuit les mêmes objectifs que la première version, en essayant de diminuer la difficulté des problèmes posés et la longueur du test tout en améliorant la formulation des consignes afin de réduire les confusions et incompréhensions.

Participants

La validation de cette deuxième version du Test d'Algorithmique a mené au recrutement de 180 participants (85 filles) dont 131 élèves de 5e et 49 de 3e. Contrairement à ce qui avait été observé pour la première version, tous les élèves recrutés ont tenté de réaliser ce test et ont fourni des réponses exploitables. Nous avons validé notre Test d'Algorithmique (version 2) en

faisant toujours appel à des élèves de cycle 4, même si celui-ci a vocation à être utilisé dans nos expériences avec des élèves de CM2. Nous avons bénéficié du réseau de la DRANE de l'Académie de Montpellier pour recruter des classes de différents niveaux dans cette région.

Matériel

Nous avons utilisé le même questionnaire visant à évaluer le niveau de pratique de la programmation des élèves pour permettre, à nouveau, de mesurer un lien de corrélation entre ce niveau de pratique de la programmation et les performances des élèves au Test d'Algorithmique (version 2). Les connaissances procédurales et métacognitives ainsi que les processus cognitifs d'évaluation et de création constituent, comme pour la première version, les dimensions que nous évaluons, décrites par la taxonomie révisée de Bloom (Krathwohl, 2002). Nous avons ainsi conçu trois exercices pratiques, réadaptés de la première version du test, qui constituent ce Test d'Algorithmique (version 2), nécessitant la création de quatre algorithmes différents. L'introduction est similaire à la première version (présentation de « BOT » et exemples d'algorithmes). Cependant, nous avons introduit une différence majeure en fournissant aux élèves une liste d'instructions compréhensibles par le robot. Ainsi, les élèves ne sont plus censés produire eux-mêmes leurs instructions, mais simplement utiliser les instructions pertinentes, parmi une sélection.

L'exercice 1 demande aux élèves de produire deux algorithmes distincts qui doivent permettre à « BOT » de rejoindre sa maison en suivant un parcours représenté sur une grille, tout en évitant les obstacles qui l'empêchent de passer. Le premier impose pour consigne de choisir le chemin le plus court (c'est-à-dire nécessitant de traverser le moins de cases que possible). Le second implique de produire un algorithme composé du moins d'instructions que possible pour aboutir à ce résultat. La consigne pour ce deuxième algorithme suggérait aux élèves d'utiliser une ou plusieurs boucles pour répéter certaines instructions. La capacité à choisir la meilleure solution parmi celles qui étaient possible (ici, celle qui correspond à la consigne donnée), souvent associée à la maîtrise de la pensée informatique, est ici évaluée. L'exercice 1 est illustré par la Figure 5. L'exercice 2 demande également aux élèves de produire un algorithme permettant à « BOT » de rejoindre sa maison, mais cette fois-ci, ils doivent uniquement réutiliser des instructions issues de deux algorithmes incorrects (proposés par deux élèves fictifs), ne prenant pas en compte les obstacles sur le chemin. Enfin, l'exercice 3 demande aux élèves, à partir d'un algorithme proposé par un élève fictif permettant à « BOT »

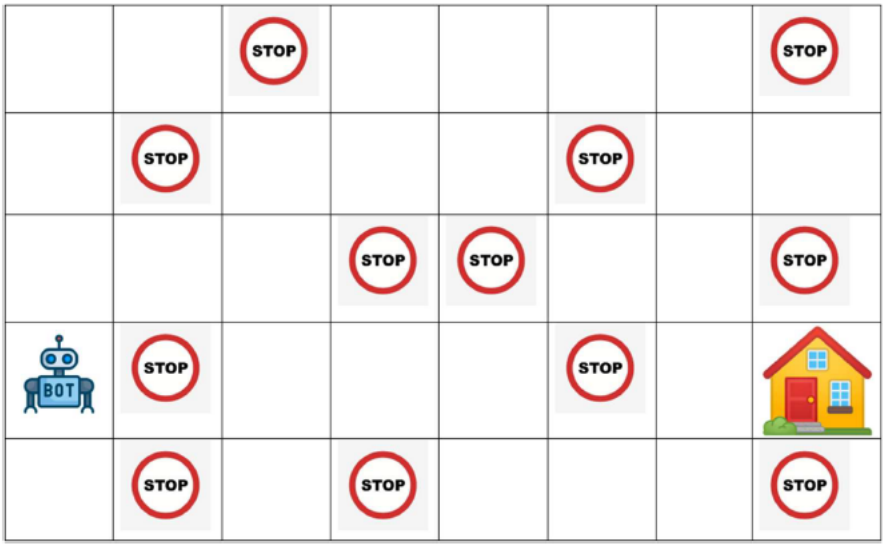
de compter jusqu'à 5, de généraliser cet algorithme pour permettre à ce robot de compter indéfiniment les secondes qui passent, puis les minutes, en utilisant des variables.

Figure 5

Exercice 1 du Test d'Algorithmique (version 2)

Exercice 1 : Plusieurs chemins

BOT doit rejoindre sa maison en évitant les obstacles (**STOP**) qui l'empêchent de passer.
Il y a plusieurs chemins possibles.



Ecris l'algorithme qui permet à BOT d'atteindre l'arrivée par le chemin le plus court, c'est-à-dire en passant par le moins de cases possible (Algorithme 1).
Puis, écris l'algorithme qui permet à BOT d'atteindre l'arrivée avec le moins d'instructions possibles (le moins de lignes dans l'algorithme). Tu dois utiliser au moins une boucle (« Répéter ») pour répéter certaines instructions (Algorithme 2).

Dans chaque exercice, la capacité à formuler une réponse sous forme d'algorithme, composé d'instructions pertinentes pour la résolution du problème, valait 1 point. Lorsque des mouvements devaient être programmés pour compléter un chemin, la capacité à programmer correctement ce chemin permettait l'obtention d'1 point supplémentaire. Lorsque l'utilisation d'une boucle ou d'une condition était requise, 1 point était attribué pour la capacité à utiliser correctement les boucles et les conditions pour obtenir le résultat requis. La capacité à utiliser correctement une variable dans l'exercice 3 permettait également l'obtention d'1 point. Dans l'exercice 1, la capacité à choisir le chemin correct et la capacité à produire un algorithme composé du moins d'instruction possible rapportaient chacune 1 point. Dans l'exercice 2, la

capacité à n'utiliser que les instructions issues des deux algorithmes incorrects était également évaluée sur 1 point. Enfin, dans l'exercice 3, la capacité à généraliser l'algorithme permettant de compter les secondes pour compter cette fois-ci les minutes rapportait également 1 point.

Pour cette deuxième version du Test d'Algorithmique, nous avons tenté de mesurer directement certaines compétences généralement associées à la pensée informatique (évaluation, réutilisation, généralisation), en plus d'évaluer la capacité des élèves à mettre en pratique les concepts computationnels fondamentaux pour résoudre des problèmes algorithmiques. Les items peuvent être regroupés en quatre catégories différentes correspondant à la fois à l'utilisation pratique des concepts computationnels et certaines compétences associées à la pensée informatique : la capacité à produire des algorithmes composés d'instructions pertinentes (quatre items, 4 points), la capacité à produire un algorithme qui constitue une solution correcte permettant au robot d'atteindre sa maison (trois items, 3 points), la capacité à évaluer différentes solutions pour sélectionner et mettre en œuvre celle qui est demandée (deux items, 2 points), la capacité à utiliser des boucles (deux items, 2 points), des instructions conditionnelles (deux items, 2 points) et des variables (un item, 1 point), la capacité à réutiliser des éléments de solution déjà fournis (un item, 1 point) et la capacité à généraliser un élément de solution à un autre problème similaire (un item, 1 point). Ce test est conçu pour évaluer une performance (sur 16 points). Ainsi, les résultats totaux ont été pris en compte même si certains items sont restés sans réponse. Le Test d'Algorithmique (version 2) est disponible en Annexe F.

Procédure

Avant de passer le test, les élèves devaient tout d'abord remplir le questionnaire évaluant leur niveau de pratique en programmation. Contrairement à la première version, les élèves ont passé ce Test d'Algorithmique (version 2) sans avoir passé le Test de Maîtrise des Concepts Computationnels. Les professeurs qui ont fait passer ce test à leurs élèves ont été informés que ceux-ci devaient travailler seuls et ne pouvaient bénéficier de leur aide, à l'exception de certaines précisions ponctuelles qui pouvaient être apportées concernant le vocabulaire employé ou la formulation des questions, lorsque celles-ci étaient jugées nécessaires. Si aucune limite de temps n'a été imposée, nous avons cependant demandé à ce que le test soit réalisé dans son intégralité durant une seule et même heure de cours et avons précisé qu'une demi-heure était suffisante pour la passation. L'ensemble des questionnaires était anonymisé.

1.3.4) Résultats (version 2 du test)

Comme pour la première version du Test d'Algorithmique, nous avons mesuré un indice de fiabilité et un indice de validité de cette deuxième version, ainsi que le nombre et la proportion d'élèves ayant *a minima* essayé de produire un algorithme, à chaque fois que cette tâche était demandée. L'impact du niveau scolaire et du genre des élèves sur leur niveau de pratique et leurs performances au test a également été mesuré. Les analyses que nous avons menées ont été réalisées sur le logiciel Jamovi, version 2.3.26.0.

Analyse de fiabilité

La fiabilité globale du test peut être considérée comme élevée (ω = .852). Un item, nommé Eval2 relatif à la capacité des élèves à évaluer quelle solution parmi toutes celles qui sont possibles correspond à celle demandée par la consigne, produirait une augmentation de la fiabilité globale du test, s'il était supprimé (ω = .857). Nous avons mesuré les indices de fiabilité concernant chaque sous-catégorie d'items, lorsque cette sous-catégorie comprenait au moins deux items. La capacité à produire des algorithmes composés d'instructions pertinentes (ω = .71), la capacité à produire un algorithme correct permettant au robot d'atteindre sa maison (ω = .68) ainsi que la capacité à produire des boucles (ω = .71) et des instructions conditionnelles (ω = .60) présentent des indices de fiabilité acceptables, au vu du très faible nombre d'items qui les compose (deux à quatre items).

Cependant, la capacité à évaluer correctement différentes solutions et à sélectionner la plus adaptée (composée de deux items, nommés Eval1 et Eval2) présente un indice de fiabilité anormalement faible (ω = .28). Ce résultat peut cependant s'expliquer. Le premier item de cette sous-catégorie demande aux élèves de choisir le chemin permettant au robot d'atteindre son objectif en passant par le moins de cases que possible. Cet item est assez facilement réussi par les élèves (il n'y a que deux chemins qui permettent d'atteindre l'objectif directement sans déplacements inutiles). Le deuxième item leur demande de produire un algorithme permettant également au robot d'atteindre son objectif, mais cette fois-ci, avec le moins d'instructions que possible (ce qui implique l'utilisation d'une boucle pour répéter certaines instructions). Ainsi, ce deuxième item suppose que les élèves doivent réussir à repérer un chemin plus long (en termes de distance, qui est représentée par le nombre de cases), mais qui peut être programmé en moins d'instructions, car une portion de ce trajet se répète trois fois. Cet item est donc nettement plus complexe que le premier et implique à la fois une capacité d'évaluation, mais également des capacités procédurales de mise en œuvre de cette solution. Il nous semble donc

logique que la fiabilité liée à ces deux items ne soit pas très élevée. Ce résultat est à prendre en considération, car il traduit l'incapacité de ce deuxième item à évaluer une seule capacité. Cependant, nous pensons que, de par sa complexité, il est utile pour discriminer différents niveaux de performance chez les élèves et ne doit pas nécessairement être supprimé du test.

Validité

Le niveau de pratique et les performances globales au test sont liés par une corrélation positive d'intensité assez forte (r = .67). Ce coefficient est assez nettement supérieur à celui mesuré pour la première version du Test d'Algorithmique (pour rappel, r = .54). Concernant les différentes capacités qui sont évaluées dans ce test, il existe également des liens de corrélations positifs d'intensité plus modérée avec le niveau de pratique en programmation, bien que ces coefficients soient plus faibles, que ce soit pour la capacité à produire des algorithmes composés d'instructions pertinentes (r = .54), la capacité à produire un algorithme qui constitue une solution correcte permettant au robot d'atteindre sa maison (r = .56), la capacité à évaluer différentes solutions pour sélectionner et mettre en œuvre celle qui est demandée (r = .49), la capacité à utiliser des boucles (r = .54), des instructions conditionnelles (r = .53) et des variables (r = .38), la capacité à réutiliser des éléments de solution déjà fournis (r = .24) et la capacité à généraliser un élément de solution à un autre problème similaire (r = .37). Toutes ces corrélations sont significatives à $p < .001$. Le niveau de pratique de la programmation est également différemment corrélé suivant l'algorithme que les élèves devaient produire (algorithme 1, r = .61 ; algorithme 2, r = .51 ; algorithme 3, r = .52 ; algorithme 4, r = .71). Par ailleurs, nous avons pu observer que les performances au test varient significativement selon le niveau de pratique en programmation des élèves, avec un effet de grande taille ($F(13, 166) = 12.7$, $p < .001$, $\eta^2 = .498$).

Taux de complétion par algorithme

Contrairement à notre première version du Test d'Algorithmique, tous les élèves recrutés ont ici essayé de produire au moins un des quatre algorithmes demandés. Cependant, tous n'ont pas terminé le test. Sur les 180 élèves, 179 ont produit le premier algorithme (99.4%), 170 le deuxième (94.4%), 155 le troisième (86.1%) et seulement 63 le quatrième (35.0%). Même si le quatrième algorithme n'a pas été produit par la majorité des élèves, ce taux de complétion est nettement plus élevé que pour la première version du Test d'Algorithmique.

Différences en fonction du genre et du niveau

Pour l'ensemble des 180 élèves ayant passé ce Test d'Algorithmique (version 2), la moyenne est de 7.70/16 (*ET* = 3.45), ce qui est largement supérieur à la moyenne obtenue pour la première version de ce test (pour rappel, 6.69/26). Les moyennes et écarts-types pour les sous-variables mesurées sont les suivants : capacité à produire des algorithmes composés d'instructions pertinentes (*M* = 2.79/4, *ET* = 0.91), capacité à produire un algorithme qui constitue une solution correcte permettant au robot d'atteindre sa maison (*M* = 1.93/3, *ET* = 1.07), capacité à évaluer différentes solutions pour sélectionner et mettre en œuvre celle qui est demandée (*M* = 1.02/2, *ET* = 0.57), capacité à utiliser des boucles (*M* = 0.68/2, *ET* = 0.59), des instructions conditionnelles (*M* = 0.38/2, *ET* = 0.56) et des variables (*M* = .09/1, *ET* = .28), la capacité à réutiliser des éléments de solution déjà fournis (*M* = .67/1, *ET* = .46) et la capacité à généraliser un élément de solution à un autre problème similaire (*M* = .11/1, *ET* = .31). Le Tableau 10 rend compte des statistiques descriptives relatives au niveau de pratique en programmation et aux performances au test de ces élèves, en fonction de leur niveau scolaire.

Tableau 10

Moyennes (et écarts-types entre parenthèses) des scores au questionnaire de niveau de pratique en programmation et au Test d'Algorithmique (version 2), selon le niveau scolaire

Niveau scolaire	Niveau de pratique en programmation	Score total – Test d'Algorithmique (version 2)
5e	6.79 (3.53)	7.37 (3.21)
3e	8.94 (3.15)	8.57 (3.91)

Le niveau de pratique de la programmation ne suit pas une distribution normale, ce qui nous a conduits à utiliser le test non paramétrique de Kruskal-Wallis. Les élèves de 3e présentent un niveau de pratique de la programmation et un niveau de performance significativement plus élevés que les élèves de 5e (respectivement, $\chi^2 = 15.9$, $p < .001$, $\varepsilon^2 = .089$ et $t(178) = 2.09$, $p < .05$, $d = 0.35$). Le Tableau 11 rend compte des statistiques descriptives relatives au niveau de pratique en programmation et aux performances au test des élèves, en fonction de leur genre.

Tableau 11

Moyennes (et écarts-types entre parenthèses) des scores au questionnaire de niveau de pratique en programmation et au Test d'Algorithmique (version 2), selon le genre

Genre	Niveau de pratique en programmation	Score total – Test d'Algorithmique (version 2)
F	7.21 (3.51)	7.46 (3.56)
M	7.53 (3.61)	7.92 (3.35)

Les différences observées ici ne sont pas significatives, qu'elles concernent l'impact du genre sur le niveau de pratique de la programmation, ou sur les performances au test (respectivement, $\chi^2 = 0.54$, $p = .46$ et $t(178) = 0.89$, $p = .38$).

1.4) Discussion des tests

Afin de pouvoir mesurer les performances d'apprentissage des élèves dans nos expériences futures, visant à comparer différentes interventions centrées sur l'apprentissage de la programmation et le développement de la pensée informatique, nous avons réalisé une revue des outils d'évaluation existants (Sigayret, Blanc & Tricot, 2022). Cependant, nous n'avons pas considéré que ces outils étaient suffisamment adaptés à notre contexte d'expérimentations. Pour cette raison, nous avons conçu deux outils supplémentaires permettant d'évaluer les connaissances et compétences à acquérir dans cette discipline. Nos instruments de mesure constituent des outils d'évaluation sommative qui mesurent la maîtrise de concepts computationnels et incluent également des tâches de résolution de problèmes algorithmiques, relatifs à des activités de programmation. En ce sens, ils se rapprochent donc d'autres outils tels que le *Computational Thinking Test* (Román-González, 2015).

Nous avons entrepris une démarche de validation psychométrique de nos nouveaux outils d'évaluation. Pour ce faire, nous avons mesuré un indice de validité, fondé sur un coefficient de corrélation entre le niveau de pratique en programmation (évalué par un questionnaire de notre conception) et les performances à nos tests. Cette manière de procéder avait déjà été employée par Li et al. (2021), bien que leur questionnaire ne comportait que deux questions fermées relatives à la programmation (avec seulement deux réponses possibles, « oui » ou « non »). Nous avons également mesuré un indice de fiabilité interne à l'aide du

coefficient ω de McDonald. Nos outils ont principalement été validés avec des élèves de niveau collège alors que les expériences que nous avons menées par la suite impliquaient des élèves de niveau CM2. Au vu des séances prévues pour nos expériences futures, relativement approfondies comparé à ce qui se fait habituellement à l'école élémentaire, nous avons jugé plus pertinente cette validation avec des élèves de niveau plus élevé.

Test de Maîtrise des Concepts Computationnels

Nous avons conçu le Test de Maîtrise des Concepts Computationnels, constitué de questions à choix multiples, évaluant la maîtrise de certains concepts fondamentaux en programmation, identifiés par Tchounikine (2017), et similaires à ceux décrits par Brennan et Resnick (2012), à savoir les concepts d'algorithme, d'instruction, de boucle, d'instruction conditionnelle et de variable. Une version en ligne et une version papier ont été proposées. L'analyse des résultats obtenus pour ces deux versions révèle un manque de fiabilité de la plupart des items de type vrai / faux, que nous avons donc choisi de supprimer de la version définitive du test qui sera utilisée dans nos expériences.

Pour les deux versions, la fiabilité interne globale du test est satisfaisante et élevée ce qui signifie que l'ensemble du test évalue bien un seul et même construit, à savoir la maîtrise des concepts computationnels fondamentaux en programmation. La fiabilité interne relative aux différentes sous-catégories d'items est modérée, ce qui peut parfois être considéré comme trop faible selon les auteurs, mais ce résultat s'explique par le très petit nombre d'items qui constitue chaque sous-catégorie (Taber, 2018).

Il existe par ailleurs, dans les deux versions, une corrélation modérée entre le niveau de pratique de la programmation des élèves et leurs performances à ce Test de Maîtrise des Concepts Computationnels. Des corrélations modérées, bien que plus faibles, existent également entre le niveau de pratique et les performances relatives aux différentes sous-catégories d'items (maîtrise du concept d'algorithme, d'instruction, de boucle, etc.). Ceci est un indice satisfaisant de la validité de notre test, car il est logique de penser que la maîtrise de ces concepts est globalement plus élevée lorsque le niveau de pratique de la programmation augmente. On pourrait cependant s'interroger sur l'absence de lien de corrélation fort entre ces deux variables. Nous pensons toutefois qu'il serait trop optimiste de s'attendre à un lien de cette intensité. En effet, le niveau de pratique de la programmation est autoévalué par les élèves, ce qui a certainement pu mener à des approximations ou à des erreurs. De plus, suivant le contexte dans lequel les élèves ont pratiqué la programmation (à l'école ou ailleurs) et l'objectif pour

lequel ils se sont engagés dans cette pratique, il n'est pas évident que le nombre d'heures passées à programmer, la fréquence des activités de programmation, etc., devraient être parfaitement reliés à la maîtrise des concepts en question. Enfin, le Test de Maîtrise des Concepts Computationnels étant composé de questions à choix multiples, il est probable que certains élèves ayant un faible niveau réel de maîtrise des concepts computationnels aient pu, ponctuellement, répondre correctement à certains items, soit par hasard, soit par déduction au vu des autres réponses proposées. En outre, nous avons constaté des différences significatives et des effets de grande taille concernant les performances à notre test, selon le niveau de pratique de la programmation, en faveur des élèves ayant les niveaux de pratique les plus élevés, ce qui constitue également un indice de validité.

Nous avons toutefois observé que le coefficient de corrélation entre le niveau de pratique de la programmation et les performances au test est légèrement plus élevé dans la version en ligne du test, par rapport à la version papier. Ceci s'explique probablement par la taille et la variété de l'échantillon recruté pour la version en ligne, plus de deux fois supérieure à celle de l'échantillon pour la version papier et impliquant des élèves issus de toutes les classes du CM1 à la 3e. Pour les deux versions, le niveau scolaire influence significativement à la fois le niveau de pratique en programmation et les performances au test. Ceci est plutôt cohérent, puisque plus le niveau scolaire augmente, plus les élèves ont eu le temps de pratiquer des activités de programmation, ce qui pourrait confirmer que notre questionnaire évaluant le niveau de pratique en programmation est suffisamment bien construit. De même, nous savons que l'âge des élèves est positivement corrélé au niveau de compréhension et de compétence liée à la pensée informatique (Zhang & Nouri, 2019). Cependant, pour la version en ligne du test, on observe que les différences entre les élèves de 4e et de 3e, en faveur de ces derniers, ne sont pas significatives. Il est possible qu'entre deux niveaux scolaires proches, les différences soient trop faibles pour que notre questionnaire de pratique de la programmation et notre Test de Maîtrise des Concepts Computationnels puissent les détecter. Enfin, aucun effet du genre sur ces deux variables n'a été observé.

Les indices de fiabilité et de validité que nous avons recueillis nous incitent à penser que notre test mesure bien la maîtrise de nos cinq concepts computationnels. Certaines limites sont cependant à noter. En particulier, après la suppression des items vrai / faux, il ne subsiste qu'un seul item évaluant la maîtrise du concept d'instruction (sur 4 points). Ceci est probablement dû au fait que ce concept est finalement le plus simple : une instruction est un ordre à exécuter. Aucun item d'application ne nous a semblé pertinent pour évaluer ce concept.

De même, demander aux élèves de reconnaître une instruction dans un programme pourrait s'avérer source de confusion, puisque d'autres concepts (boucles, instructions conditionnelles) sont également des instructions. Nous aurions cependant pu ajouter un item « texte à trous » relatif à ce concept. D'une manière générale, d'autres types d'items auraient pu être exploités. La taxonomie révisée de Bloom (Krathwohl, 2002) suggère par exemple d'évaluer la capacité des élèves à produire des inférences, des explications ou encore des interprétations pour permettre de mesurer leur niveau de compréhension. Cependant, ce type d'items se prête moins bien à des questions à choix multiples et notre volonté de proposer un test rapide et simple à mettre en œuvre nous a conduits à écarter ces options.

Test d'Algorithmique

Nous avons également conçu un Test d'Algorithmique visant à évaluer la capacité des élèves à résoudre des problèmes algorithmiques, c'est-à-dire des problèmes dont la solution peut être présentée sous la forme d'un algorithme. Notre première version de ce test consistait à mesurer plus précisément la capacité des élèves à mettre en pratique les différents concepts computationnels (algorithme, instruction, boucle, instruction conditionnelle, variable) pour résoudre les problèmes posés. Ces problèmes impliquaient également l'utilisation de certaines compétences et stratégies régulièrement associées à la pensée informatique (pensée algorithmique, décomposition, abstraction, généralisation…), mais celles-ci n'étaient pas évaluées directement, en raison de la difficulté d'isoler chacune de ces compétences dans un item précis. Pour produire leurs algorithmes, les élèves pouvaient utiliser un langage naturel (en français) et libre, c'est-à-dire sans aucune formulation ou syntaxe imposée, à condition que les instructions utilisées soient claires et précises.

Cependant, nous avons pu observer certaines faiblesses inhérentes à cette première version du test. Celui-ci était probablement trop complexe et trop long, ce qui a produit un taux d'abandon important, la majorité des élèves n'ayant pas réalisé les deux derniers exercices. Une part non négligeable d'entre eux n'ont pas même essayé de résoudre un seul des problèmes qui étaient posés. Les exercices 3 et 4 étaient certainement trop abstraits, car non fondés sur la programmation de déplacements, impliquaient trop de concepts en même temps (en particulier des variables, souvent considérées comme le concept le plus complexe parmi les cinq qui nous intéressent) et, à la différence des deux premiers exercices, ne suggéraient pas l'utilisation d'une boucle ou d'une instruction conditionnelle lorsque c'était nécessaire. En outre, nous avons conçu nos deux tests (Test de Maîtrise des Concepts Computationnels et première version du

Test d'Algorithmique) afin qu'ils soient adaptés aux séances que nous voulions mettre en place dans notre future expérience 1. Il est donc fort probable que cette première version du Test d'Algorithmique souffre d'une trop grande dépendance à ces séances et soit donc trop spécifique et pas assez généralisable à l'ensemble de la population. Cette supposition a par la suite été confirmée par les faits puisque les résultats à cette première version du test sont largement supérieurs dans notre expérience 1 (voir le « Chapitre 2 » de cette « Partie expérimentale »).

Pour l'ensemble des limites évoquées précédemment, nous avons conçu une deuxième version de notre Test d'Algorithmique. Cette deuxième version est plus courte (trois exercices au lieu de quatre) et certainement plus simple (moins de concepts mis en jeu simultanément, du moins dans les deux premiers exercices). Contrairement à la première version, la passation de cette deuxième version n'a pas été précédée par la passation du Test de Maîtrise des Concepts Computationnels, afin d'éviter l'effet probablement délétère des deux passations consécutives, qui a pu produire un effet de fatigue cognitive ou de lassitude ayant entraîné une diminution du niveau de motivation des élèves. Cette fois-ci, la syntaxe à employer était imposée par une liste exclusive d'instructions (toujours en langage naturel) à utiliser, afin de réduire les confusions et le manque de précision, observés pour la première version. Nous avons également tenté de mesurer directement trois compétences associées à la pensée informatique : l'évaluation, la réutilisation de certaines parties d'un programme et la généralisation.

Les résultats obtenus témoignent d'une amélioration en ce qui concerne le taux de complétion du test de cette deuxième version, comparativement à la première. Tous les élèves ont essayé de produire au moins un algorithme et seul l'exercice 3 présente des faiblesses à ce niveau. Celui-ci nécessitait l'utilisation de variables, et de nombreux élèves ont indiqué sur leur feuille qu'ils ne comprenaient pas ce qu'était une variable. Ainsi, même si la notion de variable informatique fait partie des attendus de fin du cycle 4 en France, nous avons pu constater que ce concept était finalement rarement abordé, y compris en 3e. Ce résultat témoigne finalement de la difficulté à proposer un test qui implique la connaissance et la maîtrise de certains concepts, car la réalisation du test dépend de la manière dont la programmation est enseignée, qui n'est certainement pas identique entre toutes les écoles et tous les collèges français.

Les deux versions du Test d'Algorithmique présentent un niveau global de fiabilité interne élevé. La fiabilité de la première version est légèrement plus élevée que celle de la deuxième version et atteint des scores presque anormaux à la fois pour la fiabilité globale et

pour la fiabilité relative aux items mesurant la capacité des élèves à produire un algorithme composé d'instructions pertinentes. En effet, un indice de fiabilité supérieur à .90 (respectivement, .90 et .91 pour les deux scores cités précédemment) peut parfois être considéré comme problématique, celui-ci reflétant une trop grande proximité entre les items (Taber, 2018). Cependant, ces scores élevés peuvent s'expliquer par le taux d'abandon important des élèves dans cette première version qui a pour conséquence un fort taux de réussite chez les rares élèves qui étaient suffisamment à l'aise pour venir à bout du test, renforçant mécaniquement la fiabilité globale du test. En d'autres termes, au-delà du manque de complétion de certains exercices, lorsqu'un élève était suffisamment confiant (ou motivé) pour s'engager dans la production d'un algorithme, celui-ci était souvent correct. De même, la possibilité d'utiliser un langage libre (sans syntaxe et vocabulaire imposés) dans la première version nous a probablement conduits à être plus conciliants dans le codage des résultats liés à la capacité à produire un algorithme composé d'instructions pertinentes. Si les mots employés étaient assez précis, nous comptions le maximum de points. Dans la deuxième version, pour laquelle une liste d'instructions était proposée, nous avons certainement été plus sévères dans la notation, ce qui expliquerait donc le score de fiabilité très (voire trop) élevé dans la première version du test, en relation avec cette catégorie d'items. La fiabilité associée aux sous-catégories d'items est logiquement plus faible dans les deux versions, au vu du petit nombre d'items qui les compose. Dans la deuxième version, la fiabilité des deux items liés à la mesure de la capacité d'évaluation des élèves est excessivement faible, mais s'explique par la difficulté d'évaluer cette compétence de manière isolée, sans impliquer d'autres compétences.

Les liens de corrélation que nous avons mesurés entre le niveau de pratique de la programmation et les performances au Test d'Algorithmique sont significatifs, d'intensité modérée pour la première version et assez forte pour la deuxième version ce qui témoigne d'une meilleure validité pour la deuxième version que pour la première. Pour les sous-catégories d'items relatives à la mise en pratique des concepts computationnels, les coefficients sont également plus élevés dans la deuxième version. En revanche, pour les sous-catégories d'items relatives à la mesure des compétences associées à la pensée informatique, présentes uniquement dans la deuxième version du test, les liens de corrélations avec le niveau de pratique en programmation sont plus faibles et renforcent l'idée que ces compétences sont finalement relativement indépendantes des activités de programmation, ce qui est généralement admis par la plupart des chercheurs dans ce domaine. En outre, le lien de corrélation le plus important, parmi les différentes sous-catégories d'items qui composent la deuxième version de notre Test

d'Algorithmique, est celui qui met en relation le niveau de pratique et la capacité à produire un algorithme correct pour permettre à « BOT » d'atteindre sa maison. La pratique de la programmation semble donc, avant tout, permettre de résoudre efficacement un problème concret. Des liens de corrélation forts sont également observés entre le niveau de pratique et les performances à l'exercice le plus complexe de chaque version du test (exercice 4 pour la version 1, exercice 3 pour la version 2), ce qui signifie que ces exercices sont tout de même pertinents pour permettre une distinction entre différents niveaux d'expertise. Enfin, les analyses de variance mettent en lumière des différences significatives entre les performances des élèves aux deux versions du test, selon leur niveau de pratique de la programmation, en faveur des élèves ayant le plus pratiqué.

Un autre point majeur à relever concerne la moyenne des résultats, nettement plus élevée pour la deuxième version, comparativement à la première version pour laquelle la moyenne était excessivement faible, due à un taux d'abandon très important. Notre deuxième version du Test d'Algorithmique est donc probablement moins spécifique aux séances et expériences que nous souhaitons mettre en place, et plus facilement généralisable à l'ensemble d'une population. Le niveau scolaire des élèves a un effet significatif sur leurs performances. Cependant, cet effet est plus important pour la première version que pour la deuxième. Nous estimons que ce résultat est un argument favorable à la deuxième version de notre test. En effet, cette deuxième version est à la fois plus sensible au niveau de pratique de la programmation, et moins sensible au niveau scolaire, ce qui nous indique que l'augmentation naturelle des capacités cognitives qui accompagne l'adolescence a moins de prise sur les performances à cette deuxième version, plus dépendante du niveau réel en résolution de problèmes algorithmiques. Comme pour le Test de Maîtrise des Concepts Computationnels, aucune différence n'est observée entre les garçons et les filles concernant leurs performances ou leur niveau de pratique de la programmation.

Malgré certaines limites que nous avons pu identifier dans cette discussion, les deux outils que nous avons conçus présentent des indices de fiabilité et de validité suffisants pour que nous puissions exploiter ces instruments dans nos études expérimentales afin d'évaluer les performances d'apprentissage des élèves relatives à un enseignement de la programmation et de la pensée informatique.

SYNTHÈSE

- Nous avons conçu deux nouveaux outils d'évaluation de l'apprentissage de la programmation et de la pensée informatique : le **Test de Maîtrise des Concepts Computationnels** et le **Test d'Algorithmique**.
- Ces deux outils présentent des indices de **validité** et de **fiabilité** acceptables.
- La première version du Test d'Algorithmique était probablement **trop longue, trop complexe et trop spécifique** aux expériences que nous souhaitions mettre en place, ce qui a entraîné un **taux d'abandon important** et des **performances faibles** à ce test.
- Nous avons par la suite conçu une deuxième version du Test d'Algorithmique qui **corrige en partie les défauts de la première**, et qui est **plus facilement généralisable** à l'ensemble des élèves qui apprennent la programmation.
- Ces deux tests ont été **mis à contribution dans les quatre expériences** que nous avons menées dans cette thèse, qui seront décrites dans les deux chapitres suivants.

Chapitre 2. Comparaison de trois outils d'apprentissage

2.1) Introduction de l'expérience 1

[Malgré un certain nombre d'études portant sur les avantages potentiels de l'apprentissage « débranché » et « branché » de la programmation, très peu de recherches ont tenté de comparer leur efficacité relative. Quelques études ont comparé une approche mixte (souvent débranchée au début, puis branchée ensuite) et un apprentissage purement branché chez des élèves de l'école primaire (âgés de 5 à 11 ans). Celles-ci n'ont parfois signalé aucune différence significative entre les deux groupes dans l'apprentissage des concepts de programmation (Hermans & Aivaloglou, 2017 ; Romero et al., 2018) ou dans les capacités mathématiques et les capacités spatiales (Messer et al., 2018). Delal et Oner (2020) estiment d'ailleurs que la programmation informatique n'est peut-être pas indispensable pour enseigner les compétences en pensée informatique.][23]

Apprentissage branché ou débranché ?

Wohl et al. (2015) ont comparé trois différentes modalités d'apprentissage de l'informatique et de la programmation, dont une modalité basée sur des activités débranchées et une autre basée sur l'utilisation de Scratch. Des élèves âgés de 5 à 7 ans (n = 28) ont participé à ces activités dans cette étude dont le plan expérimental était apparié, ce qui signifie que chacun d'entre eux a été confronté aux trois modalités. L'ordre de ces modalités a été contrebalancé, certains élèves ont donc réalisé des activités débranchées avant les activités sur Scratch, et inversement pour d'autres élèves. Les résultats s'appuient sur des analyses qualitatives et quantitatives et font état d'un meilleur niveau de compréhension des notions d'algorithme, de prédiction logique et de débogage dans le groupe débranché. Les auteurs remarquent toutefois que c'est après à la session réalisée sur Scratch que les élèves ont eu les idées les plus pertinentes sur la suite à donner à ces activités. Ils considèrent ainsi que l'utilisation de Scratch favoriserait une certaine créativité. En outre, ils constatent que l'ordre de passation de ces différentes modalités ne semble pas impacter très significativement les résultats bien qu'un léger effet négatif ait pu être constaté lorsque les activités débranchées étaient réalisées en dernier et

[23] Sigayret, K., Tricot, A., & Blanc, N. (2022). Unplugged or plugged-in programming learning: A comparative experimental study. *Computers & Education*, *184*, 104505.

lorsque les activités sur Scratch étaient réalisées au début. Dans ce cas de figure, les élèves ont jugé les activités de programmation moins « amusantes ». Les résultats observés dans cette étude ne constituent cependant pas des preuves très fiables, au vu de la faible taille de l'échantillon et de l'absence de validation psychométrique des outils utilisés pour évaluer les variables évoquées.

Comme nous l'avons indiqué précédemment, la plupart des études qui propose une approche comparative entre plusieurs outils d'enseignement de la programmation s'appuient sur des dispositifs mixtes qui incluent plusieurs outils ou méthodes d'enseignement utilisés consécutivement. C'est le cas notamment de l'étude de Hermans et Aivaloglou (2017) qui implique 35 élèves âgés de 8 à 12 ans, répartis en deux groupes. Un premier groupe d'élèves suivait quatre séances de programmation branchées alors que le deuxième suivait quatre séances débranchées. À l'issue de ces quatre séances, l'ensemble des élèves suivaient alors quatre nouvelles séances exclusivement branchées, sur le logiciel Scratch. Les résultats ne montrent aucune différence significative dans la compréhension des concepts de programmation, bien qu'il soit intéressant de remarquer que les programmeurs novices éprouvent des difficultés dans l'utilisation des variables. Cependant, le sentiment d'auto-efficacité rapporté par les élèves était plus élevé dans la condition où ils ont commencé par des activités débranchées. De même, ce groupe fait preuve d'une utilisation plus variée des blocs Scratch lors du projet final qui devait être réalisé au cours des deux dernières séances.

A contrario, Namli et Aybek (2022) n'ont rapporté aucune différence entre les élèves utilisant un logiciel de programmation visuel par blocs et les élèves ayant réalisé des activités débranchées, au niveau de leur sentiment d'auto-efficacité. Leur étude impliquait 82 élèves de niveau CM2 et contredit également l'étude de Hermans et Aivaloglou (2017) en ce qui concerne les performances en programmation et le développement des compétences liées à la pensée informatique. Seul le groupe qui a utilisé le logiciel s'était significativement amélioré entre le pré-test et le post-test pour cette variable.

del Olmo-Muñoz et al. (2020) utilisent quant à eux une procédure expérimentale similaire à celle de Hermans et Aivaloglou (2017), avec 84 élèves de niveau CE1. Les élèves étaient séparés en deux groupes initialement (branché et débranché) puis réunis à nouveau avec des activités branchées. L'objectif était de déterminer si un apprentissage débranché suivi d'un apprentissage branché était plus efficace qu'un apprentissage exclusivement branché. La progression la plus marquée concernant le développement de la pensée informatique est cette

fois-ci en faveur du groupe expérimental, ayant commencé par des activités débranchées. De même, si la motivation des élèves décroît au fil du temps, cette diminution est moins importante pour le groupe expérimental. Enfin, l'amélioration des performances relatives à la pensée informatique est plus marquée chez les garçons que chez les filles, bien que cet effet soit marginal. Les scores de motivation en général sont également plus élevés chez les garçons, mais dans le groupe expérimental, qui a débuté par des activités débranchées, il n'y a pratiquement aucune différence de genre, ce qui incite à penser qu'un apprentissage mixte de la programmation (initialement débranché puis branché) pourrait contribuer à réduire cet écart.

L'étude de Sun et al. (2022b) propose justement de comparer dispositifs mixtes et dispositifs uniques. Les auteurs ont réparti 158 élèves de niveau 5e en cinq groupes expérimentaux en fonction des outils utilisés pour enseigner la programmation : « branché uniquement », « débranché uniquement », « branché puis débranché », « débranché puis branché » et un groupe contrôle. Chacune des approches expérimentales proposées s'est traduite par un accroissement significatif des compétences en pensée informatique mais l'approche la plus efficace demeure une approche mixte, qui débute par des activités débranchées et qui est suivie d'activités branchées. De plus, les auteurs remarquent que l'impact de l'expérience antérieure des élèves en programmation est réel dans les conditions « branchée » et « branchée puis débranchée », ce qui n'est pas le cas dans les deux autres conditions expérimentales. On peut donc en déduire que les activités débranchées ont également le potentiel de réduire l'influence des précédentes expositions à la programmation sur le développement de la pensée informatique. Enfin, Sun et al. (2022b) remarquent que le niveau initial des garçons est plus élevé que celui des filles, bien qu'aucune différence significative n'ait pu être observée dans l'amélioration des compétences entre les filles et les garçons.

Apprentissage avec ou sans robot ?

Merkouris et Chorianopoulos (2015) ont mené une étude comparative auprès de 36 élèves âgés de 12 à 13 ans sur les différentes approches possibles pour apprendre la programmation. Ils proposent notamment une comparaison entre une approche centrée uniquement autour des activités sur ordinateur (avec des logiciels à vocation éducative tels que Scratch ou d'autres logiciels similaires) et une approche qui intègre des activités de robotique. Les résultats montrent que l'utilisation des robots provoque davantage d'émotions et de comportement positifs (excitation, satisfaction, engagement, intérêt...). Cependant, aucun bénéfice significatif n'a pu être observé sur les performances en matière de programmation entre

les différentes modalités d'apprentissage. Ici encore, ces résultats sont à prendre avec précaution au vu du faible échantillon (12 élèves par groupe). Leur étude met également en lumière une absence de différence entre les genres concernant l'intérêt vis-à-vis des différentes approches. Garçons et filles semblent préférer l'utilisation de robots à l'usage exclusif d'un logiciel de programmation.

Dans le même registre, Kert et al. (2020) ont comparé un groupe expérimental ayant suivi des activités de programmation via des robots pédagogiques avec un groupe contrôle ayant utilisé un logiciel de programmation visuel basé sur des blocs, durant 10 semaines. L'échantillon comprenait 78 élèves de niveau 6e (11-12 ans). Cette fois-ci, les auteurs remarquent que les robots éducatifs semblent responsables d'un accroissement plus important des performances d'apprentissage et du sentiment d'auto-efficacité des élèves en condition expérimentale (avec robot) en comparaison avec la condition contrôle (sans robot). Par ailleurs, Kert et al. (2020) ont utilisé un test d'association de mots pour évaluer la compréhension conceptuelle des élèves, en relation avec les notions présentées en classe. Là encore, les participants ayant suivi des activités de robotique sont davantage en mesure d'élaborer des connexions entre les différents concepts abordés.

En France, Bellegarde et al. (2019) ont mené une étude comparative visant à initier à la programmation des élèves de grande section de maternelle, âgés de 5 ou 6 ans, à l'aide de différents outils d'enseignement. Trois « supports » d'apprentissage ont été utilisés au cours de ces expérimentations : une tablette, un robot Blue Bot et le corps des élèves (activités débranchées). Plus de 200 élèves ont été mobilisés dans le cadre de cette étude qui a permis de les répartir dans six conditions expérimentales distinctes : une condition pour chacun des trois supports évoqués plus haut, une condition combinant robot et tablette, une condition combinant robot et corps et une dernière condition combinant les trois supports à la fois. Les élèves ont ainsi réalisé trois activités de programmation sur une durée totale de 30 minutes. À l'aide d'une comparaison pré-test / post-test, les chercheurs ont pu mettre en évidence la progression des performances des élèves. L'utilisation de deux des trois supports semble la plus intéressante, en particulier la combinaison tablette + robot pour laquelle le taux de progression est le plus fort. Il est intéressant de noter que lorsque ces deux supports sont utilisés individuellement, les performances des élèves sont nettement plus faibles. Les résultats de l'expérimentation, réalisée sur un temps très court auprès d'enfants très jeunes, ne peuvent cependant pas être généralisés à une population plus âgée dans le cadre d'une séquence d'apprentissage de la programmation plus fournie impliquant plusieurs séances réparties sur plusieurs semaines.

Limites de la recherche actuelle

[Webb et al. (2017) recommandent de poursuivre les recherches afin d'identifier les atouts des différentes approches, y compris « des nouveaux types de langages de programmation, des activités informatiques débranchées ainsi que des approches pédagogiques plus générales rendues possibles par les opportunités en ligne ». Afin de répondre aux questions sur les différentes manières d'enseigner la programmation, del Olmo-Muñoz et al. (2020) suggèrent que les études futures présentent des tailles d'échantillon plus grandes et des durées d'intervention plus longues et qu'elles développent des outils de mesure adaptés aux jeunes élèves, en particulier pour évaluer la motivation. Dans une revue de littérature concernant l'utilisation de Scratch pour améliorer les compétences en pensée informatique, Zhang et Nouri (2019) constatent que les résultats expérimentaux sont trop rarement répliqués, ce qui affaiblit la validité des recherches dans ce domaine. Ces auteurs affirment également que des efforts sont nécessaires pour explorer si les approches didactiques actuelles sont efficaces et appellent à des interventions pour comparer différentes façons d'apprendre la même compétence.

D'autre part, Brackmann et al. (2017) suggèrent que l'apprentissage de la programmation débranchée peut présenter certaines limites et que les ordinateurs peuvent être nécessaires pour développer davantage certaines compétences, mais ils s'interrogent sur le moment précis où les activités débranchées deviennent moins efficaces.][24] En effet, Ching et al. (2018) considèrent que le feedback permis par les logiciels de programmation pourrait faciliter la capacité des élèves à tester différentes hypothèses et à affiner leurs idées. De plus, ces derniers soutiennent que certaines technologies, et en particulier les outils robotiques, sont plus aptes à permettre des activités de programmation complexes et la maîtrise des concepts computationnels, bien que les robots nécessiteraient un investissement initial plus important à la fois de la part des apprenants et des enseignants. Pour Ching et al. (2018), les autres activités de manipulation d'objets concrets, tels que les jeux de type jeux de société, sont certes faciles à mettre en place, mais aussi bien plus limitées sur le plan pédagogique. Les auteurs insistent cependant sur la nécessité pour les élèves les plus jeunes de pouvoir toucher et sentir les agents d'exécution d'un programme, notamment car ceci rend la programmation plus attrayante pour les jeunes apprenants, plus souvent en recherche de stimulations sensorielles.

[24] Sigayret, K., Tricot, A., & Blanc, N. (2022). Unplugged or plugged-in programming learning: A comparative experimental study. *Computers & Education*, *184*, 104505.

Notre étude

Les recherches antérieures sur l'apprentissage de la programmation se concentrent souvent sur le développement de la pensée informatique, qu'il soit indépendant ou non des activités de programmation. Notre expérience 1 n'échappe pas à cette tendance puisque nous nous intéressons à l'impact de trois outils sur les performances d'apprentissage, qu'elles concernent la maîtrise de concepts fondamentaux en programmation ou le développement de certaines compétences généralement associées à la pensée informatique. Dans l'ensemble, les études précédentes sur la programmation débranchée ou branchée tendent à montrer que l'apprentissage débranché n'est pas nécessairement moins efficace que l'apprentissage branché plus traditionnel, qui s'appuie sur l'utilisation des machines. Néanmoins, les données actuelles sont encore largement insuffisantes pour pouvoir se prononcer sur l'efficacité relative de ces deux méthodes d'enseignement de la programmation. De même, si certains arguments semblent suggérer une influence positive de la robotique sur les apprentissages, il est aujourd'hui impossible de conclure en ce sens. La recherche dans ce domaine s'est jusqu'à présent limitée à de rares études portant sur de petits échantillons, souvent menées sur une courte période et impliquant des élèves d'âges et de niveaux scolaires différents, ce qui empêche la généralisation de ces résultats.

Pour cette raison, nous avons mené une étude comparative à plus grande échelle, évaluant les performances d'apprentissage en programmation (et distinguant la compréhension des concepts utilisés en programmation et la capacité à résoudre des problèmes algorithmiques) d'élèves de CM2 en France. Au moment où nous avons conçu notre expérience (2020), aucun élément théorique ou empirique ne permettait d'affirmer avec certitude que les approches mixtes, combinant plusieurs outils distincts, sont nécessairement plus adaptées que les approches centrées sur un seul outil. Par conséquent, nous avons comparé un dispositif exclusivement débranché avec deux autres dispositifs quasi exclusivement branchés, avec ou sans robot. Notons tout de même que l'étude de Sun et al. (2022b) en faveur d'une approche mixte permettrait aujourd'hui de considérer cette question différemment. Il est possible d'utiliser des robots sans logiciel de programmation visuel, mais il nous a semblé intéressant de bénéficier de deux conditions branchées, l'une dans laquelle les élèves travaillent sur Scratch, et l'autre dans laquelle ils utilisent le même logiciel associé à un robot pédagogique. Les éventuelles différences observées pourront être plus facilement directement imputées à l'usage ou non du robot. Saritepeci et Durak (2017) semblent d'ailleurs montrer que

l’intégration de Scratch et de la robotique était plus efficace que l’utilisation unique de Scratch pour améliorer les compétences de haut niveau liées à la pensée informatique.

En outre, les quelques études comparatives réalisées précédemment contrôlent peu l’équivalence entre les différentes conditions expérimentales. Or, une comparaison n’a de sens que lorsque tout chose est égale par ailleurs. Dans notre étude, nous avons veillé à ce que les activités proposées, les concepts abordés et les compétences développées soient les mêmes dans les trois conditions, dans la mesure du possible, afin de s’assurer que les différences observées ne soient dues qu’à l’influence de nos trois outils et non à d’autres variables parasites.

2.2) Méthode de l’expérience 1

L'objectif de cette expérience 1 est de comparer comment les élèves apprennent la programmation à la fin de l'école élémentaire, selon si cet apprentissage est débranché ou branché, et dans le deuxième cas de figure, avec ou sans robot. Cette étude comparative s'est d’abord déroulée en janvier et février 2021. Cependant, trois des quatre professeurs qui ont été recrutés dans la condition branchée avec robot ont été absents et remplacés durant la période au cours de laquelle a eu lieu notre expérimentation. Cette situation ayant engendré un biais évident, nous avons relancé l’expérience un an plus tard dans la condition branchée avec robot, en janvier et février 2022, avec quatre nouvelles classes. Nous avons ainsi pris soin de nous assurer que l’ensemble des élèves des trois conditions expérimentales ont suivi cet apprentissage de la programmation lors de la même période scolaire (entre les vacances de Noël et les vacances d’hiver). Les résultats présentés dans cette thèse, relatifs aux élèves ayant appris à programmer avec un robot ne concernent donc que les classes ayant participé à notre expérience en 2022, les classes de 2021 ayant été exclues.

Dix sessions d'apprentissage de 45 minutes ont été menées sur cinq semaines. Les concepts computationnels introduits étaient les suivants : la notion d’algorithme, d’instruction, de boucle, de condition et de variable. Cette expérience était compatible avec le programme scolaire français selon lequel les élèves de cycle 3 doivent savoir "programmer les déplacements d'un robot ou ceux d'un personnage sur un écran en utilisant un logiciel de programmation" (Ministère de l’Éducation nationale, de l’enseignement supérieur et de la recherche, 2023).

Participants

Nous avons recruté 306 élèves de CM2 (155 filles) issus de l'Académie de Montpellier pour participer à cette expérience. Les classes ont été réparties au hasard en trois groupes : six classes dans le groupe débranché (N = 133 élèves), quatre classes dans le groupe branché sans robot (N = 84 élèves) et quatre classes dans le groupe branché avec robot (N = 89 élèves). Les trois groupes étaient homogènes en termes d'équilibre entre les genres. Aucun élève n'avait suivi de cours de programmation avant l'expérience. Nous avons contrôlé la sélection des participants, car seules les classes d'écoles situées hors éducation prioritaire (REP et REP+) ont été acceptées pour notre étude. Deux classes parmi les 14 recrutées étaient des classes de CM1-CM2. Cependant, nous avons choisi d'exclure *a posteriori* les élèves de CM1 de l'analyse des données, car cette variable n'était pas au cœur de notre recherche, rendait plus complexe et moins explicite l'interprétation des résultats de notre étude, au vu du faible effectif d'élèves de niveau CM1.

Les enseignants qui nous ont fait part de leur volonté de participer à cette expérimentation ont été répartis aléatoirement dans chaque condition (avec un minimum de quatre classes par condition, afin d'éviter tout déséquilibre entre les populations des trois groupes). Notre étude est donc quasi expérimentale : ce sont les enseignants (et leur classe respective) qui ont été assignés aléatoirement à chaque groupe, et non les élèves. Quatorze enseignants, ayant plusieurs années d'expérience professionnelle, ont participé à cette étude. Ceux-ci ont reçu 10 séances de programmation qu'ils devaient mettre en œuvre dans leur classe. Des réunions ont été organisées avec les enseignants participants avant le début de l'expérience, afin de s'assurer qu'ils comprenaient et approuvaient l'importance de respecter au maximum le déroulement des séances pour permettre une comparaison rigoureuse entre les différentes conditions expérimentales. Les enseignants ayant participé à cette étude ont volontairement choisi de le faire, en réponse à une demande de recrutement que nous avons formulée auprès de la DRANE de l'Académie de Montpellier, qui a soutenu et fait suivre notre recherche de classes participantes.

Matériel

Dans le groupe débranché, des instructions sur papier ont été utilisées. Ces instructions reproduisaient les blocs disponibles sur Scratch : le texte et les couleurs des blocs étaient les mêmes. Pendant les séances d'apprentissage, les élèves ont été invités à assembler ces "blocs de papier" de la même manière que les blocs sont assemblés sur Scratch, afin de construire leurs

algorithmes. Dans le groupe branché sans robot, les élèves ont utilisé le logiciel de programmation éducatif Scratch 3.0 et ont appris à manipuler des instructions sous forme de blocs. Seuls des ordinateurs ont été utilisés pendant les activités. Aucune tablette, aucun téléphone portable ou autre appareil numérique n' été fourni aux élèves. Dans le groupe branché avec robot, les élèves ont manipulé un robot Thymio II (utilisable en wifi, sans connexion filaire) en plus du logiciel Scratch (voir Figure 6). Thymio est un petit robot développé dans un but éducatif qui peut être programmé à travers différents langages de programmation, en particulier via le langage Scratch. Il se présente sous la forme d'un petit véhicule motorisé, dont on peut programmer les déplacements, doté d'un ensemble de diodes électroluminescentes de différentes couleurs que l'on peut activer ou désactiver. Il est aussi susceptible d'être programmé pour produire des sons. Thymio est censé permettre à un large public de se familiariser avec la programmation et la robotique, à travers les diverses activités dont il se fait le support. Riedo et al. (2013) ont montré que ce robot est jugé attrayant pour les jeunes enfants comme les adolescents, indépendamment du genre des individus.

Figure 6

Robot pédagogique Thymio II

Nous avons contrôlé l'environnement expérimental dans chaque classe en veillant à ce que celles-ci disposent d'un équipement informatique suffisant (un ordinateur pour trois ou

quatre élèves) dans les deux groupes branchés (avec ou sans robot) et en fournissant nous-mêmes l'équipement nécessaire dans le groupe branché avec robot (six robots Thymio par classe) et dans le groupe débranché (instructions au format papier). Les 10 séances de programmation réalisées dans chaque groupe expérimental (Annexe G) ont été conçues avec l'aide d'une conseillère pédagogique de circonscription (CPC), spécialisée dans les technologies numériques et les cours de programmation à l'école primaire. Les concepts, les compétences et les activités développés au cours de cette séquence d'enseignement étaient très similaires dans les trois groupes expérimentaux. La seule différence majeure entre les trois groupes reposait sur l'utilisation (ou non) du logiciel de programmation Scratch et d'un robot pédagogique Thymio II.

Outils d'évaluation

Nous avons utilisé quatre tests différents pour évaluer l'impact de la programmation branchée ou débranchée, avec ou sans robot, sur quatre variables. Les questionnaires ont tous fait l'objet d'un processus de validation psychométrique. Nous avons cependant adapté ces tests à notre contexte en supprimant certains items lorsque cela était nécessaire.

Pour mesurer la maîtrise des concepts computationnels et la capacité des élèves à résoudre des problèmes algorithmiques, nous avons exploité les deux outils d'évaluation que nous avons conçus et qui ont été présentés dans le « Chapitre 1 » de la « Partie expérimentale » (Annexes C et E). Les items Vrai / Faux n'ont pas été maintenus dans le Test de Maîtrise des Concepts Computationnels, conformément à ce qui a été décidé dans ce chapitre. La première version du Test d'Algorithmique a été utilisée dans cette expérience. À ce stade, la deuxième version n'avait pas encore été conçue ni validée. Nous avons vu que la première version présentait certaines limites importantes, notamment une trop grande complexité ayant mené à un taux d'abandon élevé. Cependant, lorsque nous avons discuté ces résultats, nous avons précisé que la première version du Test d'Algorithmique avait été développée pour servir d'outil d'évaluation dans notre première expérience. Ainsi, cette version était tellement spécifique aux séances que nous étions en train de concevoir pour cette expérience qu'elle s'est révélée inadaptée aux élèves qui n'étaient pas issus de ce contexte. En résumé, la première version du Test d'Algorithmique semble difficilement généralisable à tous les élèves, mais semble adaptée aux élèves ayant participé à notre première expérience, comme en témoignent des résultats moyens plus élevés et un plus faible taux d'abandon constaté lors de l'analyse des données (voir la partie « Résultats » suivante). Le codage des résultats obtenus pour le Test de

Maîtrise des Concepts Computationnels et pour le Test d'Algorithmique a suivi la même logique que celle présentée dans le « Chapitre 1 » de la « Partie expérimentale ». Par conséquent, le Test de Maîtrise des Concepts Computationnels, tout comme le Test d'Algorithmique, était noté sur 26 points.

Afin de mesurer la motivation et le sentiment d'auto-efficacité des élèves, nous avons utilisé l'échelle SAL (*Students' Approaches to Learning*, Marsh, 2006) que nous avons adaptée à notre contexte (Annexe H). Cet instrument mesure 14 facteurs qui évaluent les stratégies d'apprentissage autorégulé, le sentiment d'auto-efficacité (ici compris dans la catégorie « *self-belief* »), la motivation et les préférences d'apprentissage. Sur ces quatre dimensions, seuls les items relatifs au sentiment d'auto-efficacité et à la motivation ont été retenus. Pour la dimension de la motivation, les items appartenant à la sous-catégorie *Instrumental Motivation Factor* ont également été exclus, car ils n'étaient pas pertinents dans notre contexte. L'échelle SAL est généralement utilisée pour évaluer le sentiment d'auto-efficacité et l'intérêt pour les mathématiques et la lecture. Nous avons adapté ces items pour qu'ils correspondent au sentiment d'auto-efficacité et à l'intérêt pour la programmation. Pour évaluer la motivation et le sentiment d'auto-efficacité dans notre expérience, nous nous sommes appuyés sur une comparaison pré-test / post-test. Dans le pré-test, certaines questions ont été reformulées pour s'adresser aux élèves qui n'ont jamais participé à des activités de programmation, afin d'évaluer leur motivation a priori, en prévision des séances à venir. Le pré-test est composé de 18 items alors que le post-test en contient 20. L'item 19 (*i.e.,* « *Quand je fais de la programmation, il m'arrive d'être totalement absorbé par ce que je fais.* ») et l'item 20 (*i.e.,* « *J'aimerais faire de la programmation pendant mon temps libre.* ») ont été exclus du pré-test car les élèves novices que nous avons recrutés n'étaient pas en mesure d'y répondre. Les questions sont regroupées en trois catégories distinctes : le sentiment d'auto-efficacité général (neuf items), le sentiment d'auto-efficacité relatif aux activités de programmation (six items) et la motivation (trois items dans le pré-test, cinq items dans le post-test).

Les réponses aux questions devaient être fournies sur une échelle en quatre points. Pour les questions 1 à 6, toutes liées à l'auto-efficacité générale, les élèves ont utilisé une échelle de fréquence (« *quasiment jamais* », « *parfois* », « *souvent* », « *presque toujours* »). Pour les autres items, les élèves ont utilisé une échelle d'accord (« *pas du tout d'accord* », « *pas vraiment d'accord* », « *plutôt d'accord* », « *totalement d'accord* »). Zéro à 3 points ont été attribués à chaque réponse pour un total de 54 points pour le pré-test et de 60 points pour le post-test. En général, les réponses « *presque toujours* » et « *totalement d'accord* » ont rapporté le plus de

points, sauf pour l'item 11, formulé négativement, pour lequel la réponse *« pas du tout d'accord* » a rapporté le maximum de points (3 points). Pour chaque élève, si aucune réponse n'a été donnée pour deux ou plusieurs items dans le pré-test ou dans le post-test, les résultats totaux n'ont pas été pris en considération. En cas d'absence de réponse à un seul item dans l'un ou les deux tests, les résultats totaux de l'élève ont été exploités, mais pas les résultats relatifs à la catégorie à laquelle appartenait l'item laissé sans réponse.

Pour mesurer l'attitude des élèves à l'égard des sciences, nous avons utilisé deux tests préexistants : le *Modified Attitudes toward Science Inventory* (Weinburgh & Steele, 2000) et un autre test évaluant l'attitude envers les sciences (Kind et al., 2007). Nous avons combiné des éléments de ces deux instruments pour créer notre propre test (Annexe I). Certains items n'ont pas été retenus en raison de leur manque de pertinence pour notre étude, comme les items liés à la perception de l'enseignant par l'élève (présents dans le test de Weinburgh et Steel, 2000) ou sur les travaux pratiques en sciences (dans le test de Kind et al., 2007). Les deux tests présentent des items mesurant exactement le même concept, qui ne diffèrent que par leur formulation. Pour ce type d'items, nous avons sélectionné celui dont la formulation semblait la plus appropriée aux élèves de niveau CM2. Pour évaluer l'attitude vis-à-vis des sciences, nous nous sommes appuyés sur une comparaison pré-test / post-test. Le pré-test et le post-test sont identiques et sont tous deux composés de 20 items. Les questions sont regroupées en cinq catégories distinctes : apprentissage des sciences à l'école (six items), sentiment d'auto-efficacité en sciences (cinq items), désir de faire des sciences à l'avenir (quatre items), valeur perçue des sciences pour la société (trois items) et anxiété à l'égard des sciences (deux items).

À l'origine, les questions du test de Weinburgh et Steele (2000) devaient être traitées sur une échelle d'accord en 6 points, tandis que les items du test de Kind et al. (2007) devaient l'être sur une échelle d'accord en 5 points. Afin d'éviter que les élèves n'aient à répondre aux questions sur deux échelles différentes et afin de faciliter la cohérence de l'évaluation, nous avons choisi d'utiliser une échelle d'accord en 4 points, identique à celle utilisée dans le test évaluant la motivation et le sentiment d'auto-efficacité (« *pas du tout d'accord* », « *pas vraiment d'accord* », « *plutôt d'accord* », « *totalement d'accord* »). Zéro à 3 points ont été attribués à chaque réponse pour un total de 60 points. En général, les réponses « *totalement d'accord* » ont rapporté le plus grand nombre de points, sauf pour les items formulés négativement (items 5 à 7 et 11 à 13) pour lesquels les réponses « *pas du tout d'accord* » ont rapporté le maximum de points (3 points). Pour chaque élève, si aucune réponse n'a été donnée pour deux ou plusieurs items dans le pré-test ou dans le post-test, les résultats totaux n'ont pas été pris en considération.

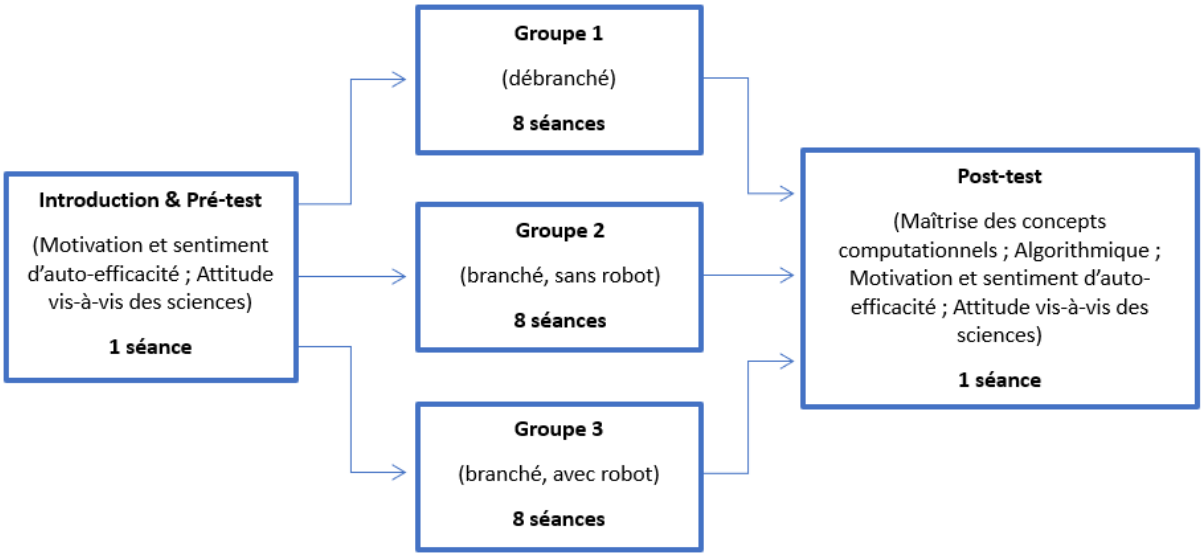
En cas d'absence de réponse à un seul item dans l'un ou les deux tests, les résultats totaux de l'élève ont été exploités, mais pas les résultats relatifs à la catégorie à laquelle appartenait l'item laissé sans réponse.

Procédure

Le pré-test et le post-test ont été administrés lors de la première et de la dernière séance. Entre ces deux étapes, huit séances ont été consacrées à l'enseignement de la programmation. Les élèves ont été initiés aux concepts computationnels fondamentaux, nécessaires à la pratique de la programmation, et ont appris à manipuler des instructions pour construire des algorithmes afin de programmer des mouvements et des interactions avec des objets (réels ou virtuels, selon la condition expérimentale), pendant les quatre premières séances d'enseignement. Les élèves devaient ensuite réaliser un projet créatif, leur permettant de remobiliser les concepts vus précédemment, au cours des quatre dernières séances d'enseignement. La Figure 7 résume la conception des séances.

Figure 7

Conception des séances (expérience 1)



Dans chaque condition expérimentale (débranchée ou branchée, avec ou sans robot), les élèves ont travaillé en petits groupes (trois ou quatre élèves par groupe). À la fin des séances 2, 3 et 5, cinq minutes ont été consacrées à la rédaction d'une trace écrite résumant les principaux éléments vus en classe. Cette trace écrite était exactement la même dans les trois conditions, ce qui permet par ailleurs d'attester d'un apprentissage conceptuel commun entre les trois groupes.

2.3) Résultats de expérience 1

Les résultats décrits et analysés dans cette partie ont été partiellement publiés dans la revue *Computers & Education* (Sigayret, Tricot et Blanc, 2022). Au moment de la publication de l'article, les résultats concernant la « condition branchée avec robot » n'avaient pas encore été analysés (voir la partie « Méthode de l'expérience 1 » précédente pour plus d'explications). Ainsi, notre publication ne rendait compte que de la comparaison entre la condition « débranchée » et la condition « branchée sans robot ». Par ailleurs, nous avions réalisé des analyses de variance (ANOVA) pour mesurer des différences significatives entre nos deux groupes expérimentaux.

Dans cette section, nous avons décidé de pousser plus loin l'analyse de nos résultats, comparativement à ce qui avait été fait pour *Computers & Education*. Bien évidemment, nos résultats concerneront cette fois-ci les trois conditions expérimentales de notre étude (incluant donc la condition branchée avec robot), mais nos analyses reposent désormais sur la réalisation de modèles mixtes que nous pensons plus adaptés à notre conception expérimentale. Un modèle mixte (aussi appelé analyse multiniveaux) est un modèle statistique comprenant à la fois des effets fixes et des effets aléatoires. Les effets fixes représentent l'effet d'un prédicteur (variable indépendante) dont on suppose qu'il explique au moins une partie de la variance observée sur un résultat ou un comportement (variable dépendante). Les effets aléatoires représentent les autres potentielles sources de variation, dont on sait qu'elles peuvent avoir une influence sur nos résultats, même si cette influence n'est pas l'objet de notre étude.

Les modèles mixtes sont aujourd'hui considérés comme une référence dans l'analyse des données, en particulier lorsque celles-ci impliquent des mesures répétées ou des données regroupées par *clusters*, même si ce type d'analyse n'a que récemment commencé à être largement utilisé en psychologie (Muradoglu et al., 2023). Le terme « modèle hiérarchique » est également employé fréquemment pour désigner les modèles mixtes. La notion de hiérarchie fait référence aux jeux de données imbriqués, particulièrement propices aux modèles mixtes (par exemple, en éducation, la variable « élève » est imbriquée dans une variable « classe », elle-même imbriquée dans une variable « établissement », etc.).

Notre modèle propose de tester l'influence de deux variables indépendantes (le groupe expérimental et le genre des élèves) sur un ensemble de variables dépendantes. Nous avons donc deux effets fixes à intégrer à notre modèle mixte. Cependant, chaque classe impliquée

dans notre expérience a été associée à un seul groupe expérimental. Or, au-delà du groupe expérimental et du genre des élèves, il existe très probablement un effet dû à la classe en elle-même et au professeur (Hattie, 2003), même si cet effet n'est pas l'objet de notre étude. La variable « classe » peut donc être intégrée à notre modèle en tant qu'effet aléatoire. Chaque classe ayant été recrutée dans un établissement différent, il n'y a pas de nécessité d'intégrer un deuxième effet aléatoire lié à l'établissement.

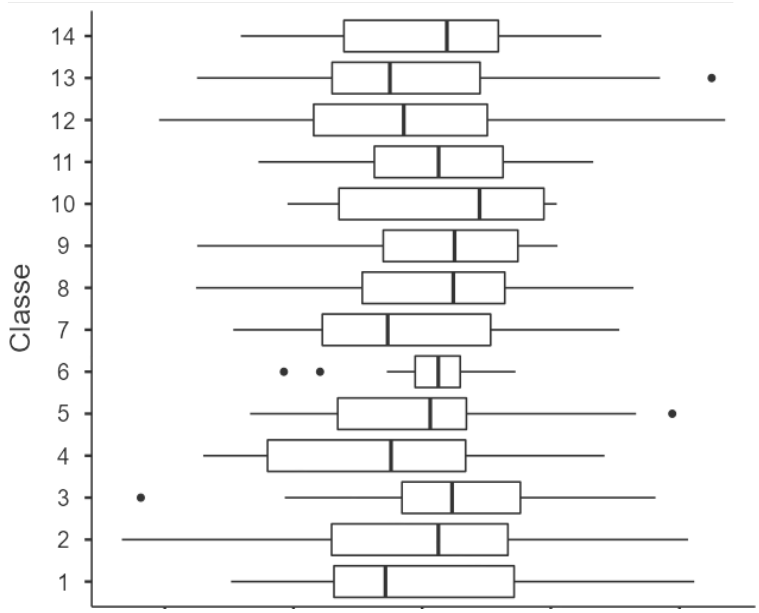
Analyses préliminaires

Nous avons procédé à l'élimination des valeurs aberrantes (*outliers*) qui peuvent avoir une influence sur l'analyse des résultats, mettre en péril la qualité des données recueillies et induire une surestimation ou une sous-estimation de la taille des effets observés. Une valeur aberrante est une valeur qui diffère significativement du reste des données (Dash et al., 2023). Pour ce faire, nous avons utilisé la méthode basée sur les écarts interquartiles (EI), qui divisent la distribution en quatre quartiles et mesurent une dispersion représentant la différence entre le troisième et le premier quartile (EI = Q3 – Q1). En bref, cette méthode considère comme valeur aberrante toute observation inférieure au premier quartile d'une fois et demie l'écart interquartile (Q1 – 1.5 x EI) et toute observation supérieure au troisième quartile d'une fois et demie l'écart interquartile (Q3 + 1.5 x EI).

Conformément à ce qui a été observé maintes fois dans la littérature, nous nous attendons à ce qu'il existe une variabilité non négligeable entre les classes, en raison notamment de « l'effet maître » ou « effet enseignant » qui a une influence significative sur les résultats des élèves, en plus d'autres facteurs indépendants du contenu à enseigner (Hattie, 2003). Par conséquent, il nous semble plus pertinent de supprimer les valeurs aberrantes classe par classe, plutôt que de les supprimer sur l'ensemble de la distribution des données. Pour ce faire, nous avons généré un graphique de type *box-plot* (voir Figure 8). Le *box-plot* est un graphique simple qui permet de visualiser la distribution des données et la présence de valeurs aberrantes.

Figure 8

Graphiques box-plots représentant la distribution des données selon la classe concernée, pour la maîtrise des concepts computationnels



Les limites du rectangle représentent les quartiles Q1 et Q3, la valeur centrale symbolisée par un segment à l'intérieur du rectangle représente la médiane et les deux segments qui partent de l'extérieur du rectangle représentent les valeurs inférieures à Q1 et supérieures à Q3 qui ne dépassent pas d'une fois et demie l'écart interquartile. Les valeurs aberrantes sont représentées par des points aux extrémités du graphique. Sur la Figure 8, chaque *box-plot* correspond à la distribution des données dans chacune des 14 classes ayant participé à notre expérience, en relation avec la maîtrise de concepts computationnels. Nous pouvons constater qu'il y a des valeurs aberrantes à supprimer dans les classes numérotées 3, 5, 6 et 13. Nous avons exécuté cette même procédure en ce qui concerne les autres variables dépendantes mesurées (résolution de problèmes algorithmiques, motivation et sentiment d'auto-efficacité, attitude vis-à-vis des sciences).

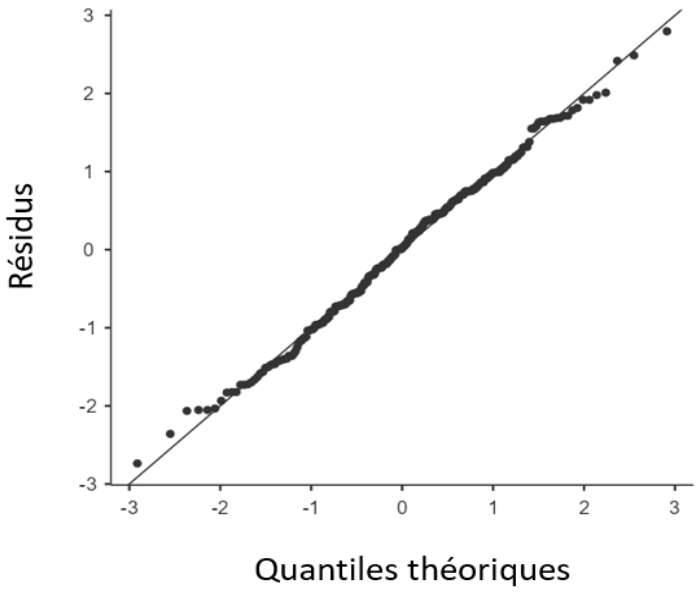
Par ailleurs, suite à la suppression des valeurs aberrantes, nous avons centré et réduit nos données, en soustrayant la moyenne à chaque valeur et en divisant cette valeur par l'écart-type, afin d'obtenir une variable pour laquelle la moyenne est nulle et l'écart-type est égal à 1. Cette transformation est fréquemment utilisée en analyses de données, en particulier en ce qui concerne les modèles mixtes, car le calcul du coefficient standardisé β est une estimation fiable de la taille d'effet lorsque les données sont centrées et réduites (Sommet et Morselli, 2021).

Afin de pouvoir mettre en œuvre des modèles mixtes linéaires, il est nécessaire de

vérifier la normalité des résidus. Les tests de Shapiro-Wilk et de Kolmogorov-Smirnov sont généralement indiqués pour tester si une distribution de données peut être considérée comme suivant la loi normale. Cependant, le test de Shapiro-Wilk est surtout adapté à de faibles tailles d'échantillon et le test de Kolmogorov-Smirnov est un test d'hypothèse utilisé pour vérifier que la distribution suit bien une loi donnée, pas nécessairement la loi normale. Dans leur ouvrage de référence, Judd et al. (2018) considèrent que l'outil le plus précieux pour détecter si la distribution des erreurs s'écarte de la distribution normale est le diagramme quantile-quantile de la loi normale (*Q-Q plot*), qui donne à voir la distribution des données en percentile au regard de la distribution en percentile à laquelle on s'attendrait si la distribution était normale. Les auteurs exposent leur méthode qui permet d'analyser graphiquement ce diagramme afin de déterminer si les déviations observées par rapport à la loi normale sont problématiques ou non, de manière plus pertinente que les tests habituellement utilisés, souvent trop restrictifs. Nous avons donc généré des *Q-Q plots* et suivi les recommandations de Judd et al. (2018) pour vérifier la normalité de nos distributions. La Figure 9 est un exemple de *Q-Q plot*.

Figure 9

Q-Q plot représentant la distribution des données relatives à la maîtrise des concepts computationnels. Ici, on peut graphiquement estimer que la distribution suit la loi normale



Les analyses que nous avons menées ont été réalisées sur le logiciel Jamovi, version 2.3.26.0, à l'aide du module Gamlj (*General Analyses for Linear Models in jamovi*).

Maîtrise des concepts computationnels

Le Tableau 12 présente les statistiques descriptives concernant les scores de maîtrise des concepts computationnels (moyennes et écarts-types), dans les groupes expérimentaux, selon le genre : le score total (sur 26 points) et les sous-scores relatifs à la maîtrise du concept d'algorithme (sur 6), d'instruction (sur 4), de boucle (sur 7), de condition (sur 6) et de variable (sur 3).

Tableau 12

Moyennes (et écarts-types entre parenthèses) des scores de maîtrise des concepts computationnels dans les trois groupes expérimentaux, selon le genre.

Groupe	Genre	Total	Algorithme	Instruction	Boucle	Condition	Variable
Débranché	**F**	14.2 (4.29)	4.08 (1.31)	2.41 (0.90)	3.20 (1.90)	3.22 (1.39)	1.31 (0.91)
	M	14.0 (5.66)	3.86 (1.55)	2.64 (1.07)	3.13 (2.17)	3.04 (1.58)	1.30 (1.00)
Branché sans robot	**F**	18.5 (4.11)	4.20 (1.27)	3.24 (0.73)	4.73 (1.82)	4.15 (1.26)	2.15 (0.76)
	M	17.2 (5.16)	4.03 (1.40)	2.95 (1.12)	4.04 (1.89)	3.99 (1.38)	2.18 (0.97)
Branché avec robot	**F**	16.3 (5.02)	4.23 (1.56)	2.87 (0.92)	3.79 (1.89)	3.72 (1.54)	1.69 (0.89)
	M	14.4 (4.66)	3.79 (1.59)	2.67 (0.66)	3.13 (1.69)	3.51 (1.41)	1.33 (1.01)

Notre modèle comprend deux effets fixes (groupe et genre) et un effet aléatoire (classe) et nous testons ici l'influence de ces effets sur le score total de maîtrise des concepts computationnels. Nous avons vérifié la pertinence de notre modèle en testant le rapport de vraisemblance pour les effets aléatoires (*Likelihood Ratio Test for random effects* ou LRT). Ceci permet de comparer notre modèle à un modèle nul, c'est-à-dire un modèle pour lequel il n'y aurait aucun effet fixe, seulement l'effet aléatoire de la variable « classe ». Le résultat obtenu (LRT = 30.0, $p < .001$) confirme la pertinence de notre modèle puisqu'il existe une différence significative entre notre modèle et le modèle nul. Par ailleurs, nous avons calculé le critère d'information d'Akaike (AIC), qui est une mesure de la qualité d'un modèle statistique. Dans notre modèle, AIC = 742, et dans le modèle nul, AIC = 770. Ce résultat nous indique que notre modèle est plus adapté que le modèle nul, car la valeur de l'AIC est inférieure à celle du

modèle nul. Ceci nous a permis de confirmer que notre analyse, basée sur un modèle mixte comprenant deux effets fixes, est pertinente.

Le coefficient de corrélation intraclasse (*Intraclass Correlation Coefficient* ou ICC), associé à l'effet aléatoire de notre variable « classe » est de .19, ce qui signifie que 19% de la variance observée concernant le score total de maîtrise des concepts computationnels s'explique par une variabilité inter-sujet (ici, une variabilité entre nos différentes classes) et 81% par une variabilité intra-sujet (ici, une variabilité au sein d'une même classe, entre les élèves).

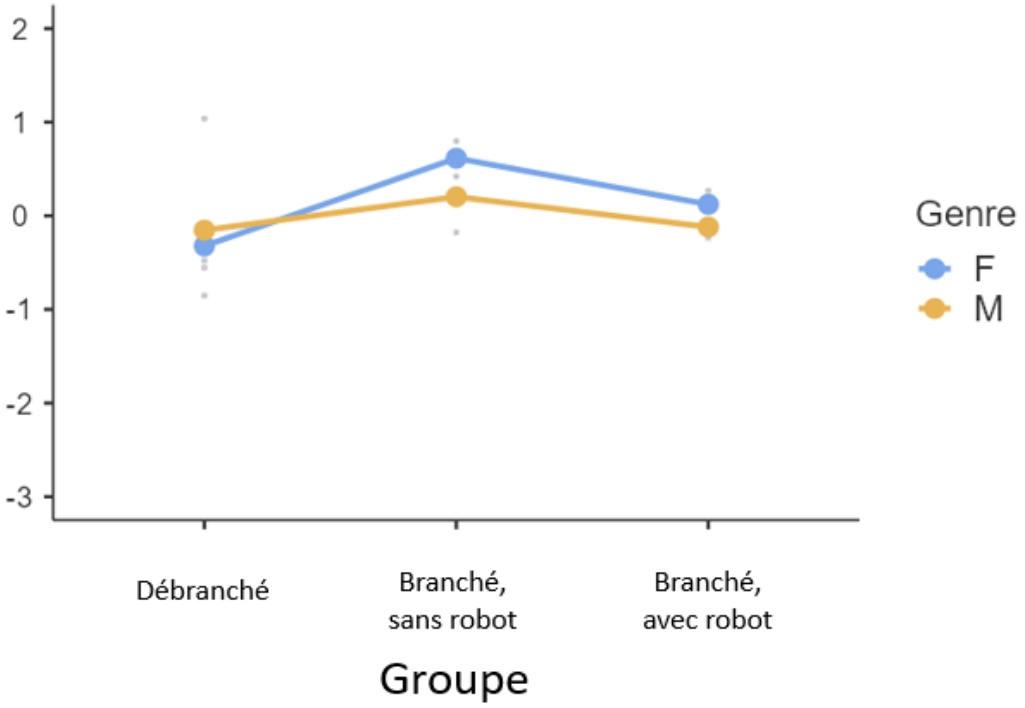
L'analyse des résultats indique qu'il existe un effet principal significatif du groupe expérimental ($F(2, 275) = 3.72$, $p = .05$) et du genre ($F(1, 276) = 6.29$, $p = .01$) sur la maîtrise des concepts computationnels. Les filles sont significativement plus performantes que les garçons. Cependant, l'effet de l'interaction entre le groupe et le genre n'est pas significatif ($F(2, 272) = 0.90$, $p = .41$). Pour analyser plus précisément ces résultats, nous avons considéré la variable « groupe expérimental » comme une variable polynomiale, ce qui nous permet de tester une relation linéaire et une relation quadratique entre nos trois groupes. La relation linéaire permet de comparer notre condition débranchée (sans outil numérique) avec notre condition branchée avec robot (avec donc deux outils numériques). La relation quadratique permet de tester si notre deuxième condition (branchée sans robot) diffère significativement des deux autres. En effet, l'hypothèse que nous avons posée dans le « Chapitre 5 » de la « Partie théorique » repose sur une supériorité de la condition branchée sans robot. L'effet linéaire n'est pas significatif ($\beta = 0.20$, $p = .41$, 95% IC [-0.21 ; 0.62], ce qui nous indique qu'il n'y a pas de différence significative entre notre condition débranchée et notre condition branchée avec robot. En revanche, l'effet quadratique nous indique que les élèves de la condition branchée sans robot semblent mieux maîtriser les concepts computationnels par rapport aux deux autres groupes, conformément à notre hypothèse ($\beta = -0.55$, $p < .05$, 95% IC [-1.00 ; -0.11]). La valeur du coefficient standardisé peut être considérée comme une estimation fiable de la taille d'effet dans notre modèle mixte, puisque nous avons centré et réduit nos données (Sommet & Morselli, 2021). Selon Cohen (1992), une taille d'effet supérieure à 0.50 (ou inférieure à -0.50 dans ce cas précis) peut être considérée comme forte. Entre 0.30 et 0.50, l'effet est moyen. Il est faible entre 0.10 et 0.30. Enfin, l'effet du genre peut ici être considéré comme faible ($\beta = -0.27$, $p = .01$, 95% IC [-0.47 ; -0.06]).

Nous avons également entrepris de mesurer l'effet de nos variables sur les sous-scores de maîtrise des concepts computationnels. Concernant la maîtrise du concept d'algorithme,

aucune différence significative n'a pu être établie selon le groupe expérimental (F(2, 275) = 0.16, p = .85) ou le genre (F(1, 276) = 3.44, p = .07). L'effet d'interaction est également inexistant (F(2, 272) = 0.19, p = .83). Pour le concept d'instruction, il n'y a toujours pas d'effet significatif du groupe (F(2, 275) = 1.78, p = .20) ni du genre (F(1, 276) = 2.60, p = .11). Cependant, il existe un effet d'interaction entre le groupe et le genre (F(2, 272) = 3.29, p < .05). Comme nous pouvons le voir sur la Figure 10, les filles semblent mieux maîtriser le concept d'instruction que les garçons dans le groupe branché sans robot alors qu'à l'inverse, ce sont les garçons qui semblent mieux maîtriser ce concept dans le groupe débranché. Cet effet d'interaction est relativement fort (β = -0.58, p < .05, 95% IC [-1.05 ; -0.11]).

Figure 10

Moyennes des scores standardisés de maîtrise du concept d'instruction selon le groupe et le genre des élèves



Pour le concept de boucle, il n'y a pas d'effet principal significatif du groupe (F(2, 275) = 2.50, p = .12). Cependant, l'effet simple quadratique est significatif et de taille moyenne (β = -0.46, p = .05, 95% IC [-0.88 ; -0.04]), ce qui indique que le groupe branché sans robot semble meilleur que les deux autres concernant la maîtrise du concept de boucle. Ici encore, les filles semblent plus performantes que les garçons, mais cet effet est faible (F(1, 276) = 6.40, p = .01, β = -0.27, 95% IC [-0.49 ; -0.06]). Aucun effet d'interaction n'est observé (F(2, 272) = 1.29, p

= .28). Pour le concept de condition, l'effet du groupe est significatif ($F(2, 275) = 8.16$, $p = .01$), en particulier l'effet quadratique qui est de taille moyenne ($\beta = -0.41$, $p = .01$, 95% IC [-0.65 ; -0.16]). Ceci signifie que c'est encore une fois le groupe branché sans robot qui présente de meilleures performances que les deux autres. L'effet du genre n'est pas significatif ici ($F(1, 276) = 1.18$, $p = .28$) tout comme l'effet d'interaction entre le groupe et le genre ($F(2, 272) = 0.01$, $p = .99$). Enfin, pour la maîtrise du concept de variable, l'effet du groupe est significatif ($F(2, 275) = 7.82$, $p < .01$). C'est à nouveau un effet quadratique fort que l'on observe ($\beta = -0.66$, $p < .01$, 95% IC [-1.01 ; -0.30]), ce qui signifie que le groupe branché sans robot semble présenter de meilleures performances pour la maîtrise du concept de variable, en comparaison avec les deux autres groupes. Aucun impact significatif du genre ($F(1, 276) = 1.88$, $p = .17$) ou de l'interaction entre le groupe et le genre ($F(2, 272) = 0.92$, $p = .40$) n'a pu être observé.

Résolution de problèmes algorithmiques

Le Tableau 13 présente les statistiques descriptives concernant les scores au Test d'Algorithmique (moyennes et écarts-types), dans les groupes expérimentaux, selon le genre : le score total (sur 26 points) et les sous-scores relatifs à la capacité à produire un algorithme constitué d'instructions pertinentes (sur 10), à utiliser des boucles (sur 6), à utiliser des instructions conditionnelles (sur 4) et à utiliser des variables (sur 6). Par ailleurs, nous avons observé dans le « Chapitre 1 » de cette « Partie expérimentale » que les deux premiers exercices du Test d'Algorithmique sont généralement nettement mieux réussis que les deux derniers. Le Tableau 13 présente donc également les scores relatifs aux deux premiers exercices (sur 11) et aux deux derniers exercices (sur 15).

Tableau 13

Moyennes (et écarts-types entre parenthèses) des scores au test d'Algorithmique dans les trois groupes expérimentaux, selon le genre

Groupe	Genre	Total	Algorithme	Boucle	Condition	Variable	Ex 1 + 2	Ex 3 + 4
Débranché	**F**	9.80 (5.02)	6.26 (2.26)	2.03 (1.79)	0.92 (1.09)	0.58 (1.03)	6.97 (2.26)	2.83 (2.26)
	M	9.16 (4.40)	5.93 (2.19)	1.69 (1.49)	1.09 (1.08)	0.44 (0.76)	6.63 (2.39)	2.53 (2.94)
Branché sans robot	**F**	14.0 (5.03)	7.65 (1.63)	2.77 (1.61)	1.52 (1.22)	2.06 (1.63)	7.20 (1.78)	6.81 (4.58)
	M	14.4 (5.70)	7.78 (1.93)	3.08 (1.67)	1.47 (1.31)	2.11 (1.56)	7.16 (2.25)	7.28 (4.73)
Branché avec robot	**F**	8.81 (5.39)	5.93 (2.21)	1.49 (1.87)	0.54 (0.93)	0.85 (1.51)	5.32 (2.73)	3.49 (3.81)
	M	8.34 (4.91)	5.80 (2.38)	1.69 (1.50)	0.45 (0.92)	0,62 (1,27)	5.43 (2.56)	2.91 (3.22)

Notre modèle mixte comprend deux effets fixes (le groupe expérimental et le genre) et un effet aléatoire (la classe). Nous avons comparé ce modèle au modèle nul (LRT = 51.9, $p < .001$, AIC = 678 pour notre modèle et AIC = 728 pour le modèle nul) et cette analyse confirme la pertinence de notre modèle. Nous observons que 25% de la variance concernant les scores au Test d'Algorithmique s'explique par une variabilité entre les classes et 75% par une variabilité au sein d'une même classe, entre les élèves (ICC = .25).

Aucun impact significatif du genre n'a pu être mis en évidence, à la fois sur les scores totaux ($F(1, 270) = 0.36$, $p = .55$) et sur les sous-scores liés à la capacité à produire un algorithme avec des instructions pertinentes ($F(1, 270) = 0.31$, $p = .58$), à utiliser des boucles ($F(1, 270) = 0.03$, $p = .87$), à utiliser des instructions conditionnelles ($F(1, 270) = 0.04$, $p = .85$) ou à utiliser des variables ($F(1,270) = 0.86$, $p = .36$). De même, sur les deux premiers exercices du test ($F(1,270) = 0.08$, $p = .78$) et sur les deux derniers ($F(1,270) = 0.40$, $p = .53$). En outre, nous n'avons trouvé aucun effet d'interaction entre le groupe expérimental et le genre concernant les scores totaux ($F(2, 265) = 0.36$, $p = .55$) et les sous-scores relatifs à la production d'algorithmes ($F(2, 265) = 0.87$, $p = .42$), de boucles ($F(2, 265) = 1.10$, $p = .33$), d'instructions conditionnelles ($F(2, 265) = 0.32$, $p = .72$) et de variables ($F(2, 265) = 0.08$, $p = .93$) ou relatifs aux exercices 1 et 2 ($F(2, 265) = 1.14$, $p = .32$) et aux exercices 3 et 4 ($F(2, 265) = 0.23$, $p = .80$).

En revanche, il existe un effet du groupe expérimental. Nous ne détaillerons ici que les effets quadratiques, les seuls à être significatifs. Les élèves du groupe branché sans robot présentent une capacité à résoudre des problèmes algorithmiques significativement supérieure à celle des deux autres groupes, avec un effet fort, que ce soit en ce qui concerne le score total (β = -0.75, p < .01, 95% IC [-1.21 ; -0.29]), le score relatif à la production d'algorithmes composés d'instructions pertinentes (β = -0.58, p < .05, 95% IC [-1.07 ; -0.10]) et les scores relatifs à l'utilisation des boucles (β = -0.60, p < .05, 95% IC [-1.05 ; -0.15]), des instructions conditionnelles (β = -0.54, p < .01, 95% IC [-1.87 ; -0.21]) et des variables (β = -0.81, p < .01, 95% IC [-1.23 ; -0.39]). Il est également intéressant de noter qu'il n'y a pas de supériorité du groupe branché sans robot sur les deux autres groupes concernant les deux premiers exercices qui sont les plus simples (β = -0.27, p = .30, 95% IC [-0.77 ; 0.22]) alors que la supériorité de ce groupe sur les deux autres est particulièrement marquée concernant les deux derniers exercices qui sont les plus complexes (β = -0.83, p < .01, 95% IC [-1.29 ; -0.37]).

Motivation et sentiment d'auto-efficacité

Le Tableau 14 présente les statistiques descriptives des scores au pré-test concernant la motivation et le sentiment d'auto-efficacité des élèves (moyennes et écarts-types), dans les groupes expérimentaux, selon le genre : le score total (sur 54) et les sous-scores relatifs au sentiment d'auto-efficacité général (sur 27), au sentiment d'auto-efficacité relatif aux activités de programmation (sur 18) et à la motivation (sur 9).

Tableau 14

Moyennes (et écarts-types entre parenthèses) des scores au pré-test relatif à la motivation et au sentiment d'auto-efficacité dans les trois groupes expérimentaux, selon le genre

Groupe	Genre	Total	Auto-efficacité générale	Auto-efficacité programmation	Motivation
Débranché	**F**	34.7 (8.95)	18.3 (4.36)	11.3 (3.54)	5.49 (2.35)
	M	35.4 (8.71)	17.9 (4.68)	12.1 (3.28)	5.54 (2.39)
Branché, sans robot	**F**	36.3 (7.36)	18.9 (5.32)	12.0 (2.67)	5.34 (2.25)
	M	35.7 (8.49)	17.3 (4.69)	12.3 (3.43)	6.57 (2.13)
Branché, avec robot	**F**	33.8 (6.77)	16.5 (3.91)	11.6 (3.83)	5.73 (2.52)
	M	33.6 (7.84)	15.2 (5.10)	11.7 (3.44)	6.81 (2.04)

Le tableau 15 présente les mêmes statistiques que le tableau 14, relatives cette fois-ci au post-test de motivation et de sentiment d'auto-efficacité. Par ailleurs, les items 19 et 20, présents dans le post-test étaient absents du pré-test pour les raisons évoquées dans la partie « Méthode de l'expérience 1 » précédente. Les statistiques descriptives liées à ces deux items sont indiquées dans le tableau (sur 6) et ne sont pas comptées dans le score total de motivation et sentiment d'auto-efficacité.

Tableau 15

Moyennes (et écarts-types entre parenthèses) des scores au post-test relatif à la motivation et au sentiment d'auto-efficacité dans les trois groupes expérimentaux, selon le genre

Groupe	Genre	Total	Auto-efficacité générale	Auto-efficacité programmation	Motivation	Motivation 19 + 20
Débranché	**F**	35.9 (9.32)	19.4 (4.95)	12.1 (4.21)	4.69 (2.14)	2.66 (1.60)
	M	36.1 (7.58)	18.7 (4.27)	12.2 (3.31)	4.86 (4.72)	2.93 (1.71)
Branché sans robot	**F**	36.3 (7.91)	19.0 (6.23)	12.6 (2.93)	4.72 (2.31)	3.20 (1.52)
	M	37.7 (9.28)	17.9 (5.09)	13.5 (4.13)	6.81 (1.75)	4.37 (1.55)
Branché avec robot	**F**	40.4 (10.4)	18.4 (4.25)	12.2 (3.92)	6.07 (3.13)	3.90 (2.13)
	M	43.7 (8.66)	19.5 (5.01)	13.5 (3.54)	6.96 (2.51)	4.27 (1.91)

Notre modèle mixte comprend toujours les deux mêmes effets fixes (le groupe expérimental et le genre) et le même effet aléatoire (la classe). Cependant, pour la variable « Motivation et sentiment d'auto-efficacité », nous avons réalisé des mesures répétées (pré-test et post-test). L'une des méthodes utilisées pour intégrer les mesures répétées dans un modèle mixte consiste à considérer le pré-test comme une covariable dont on va mesurer l'influence sur notre variable dépendante : le post-test. En effet, on suppose que le pré-test covarie avec le post-test et affecte donc indirectement les relations qu'entretiennent nos variables indépendantes avec notre variable dépendante. Nous avons comparé ce modèle au modèle nul (LRT = 0.48, p = .49, AIC = 520 pour notre modèle et AIC = 518 pour le modèle nul). On constate qu'il n'y a pas de différence significative. En effet, la valeur de l'ICC est égale à 2%, ce qui signifie que seulement 2% de la variance observée s'explique par les différences entre les classes. En d'autres termes, notre effet aléatoire « classe » sur la motivation et le sentiment d'auto-efficacité est quasi nul. Ceci ne nous empêche pas de réaliser un modèle mixte, mais nous savons désormais que notre effet aléatoire n'aura quasiment aucune influence sur nos résultats.

Très logiquement, l'effet de notre covariable (pré-test) sur notre variable dépendante (post-test) est très significatif et fort (F(1, 235) = 193, p < .001, β = -0.65, 95% IC [0.56 ; 0.74]), ce qui nous indique que les résultats au pré-test déterminent largement les résultats au post-test.

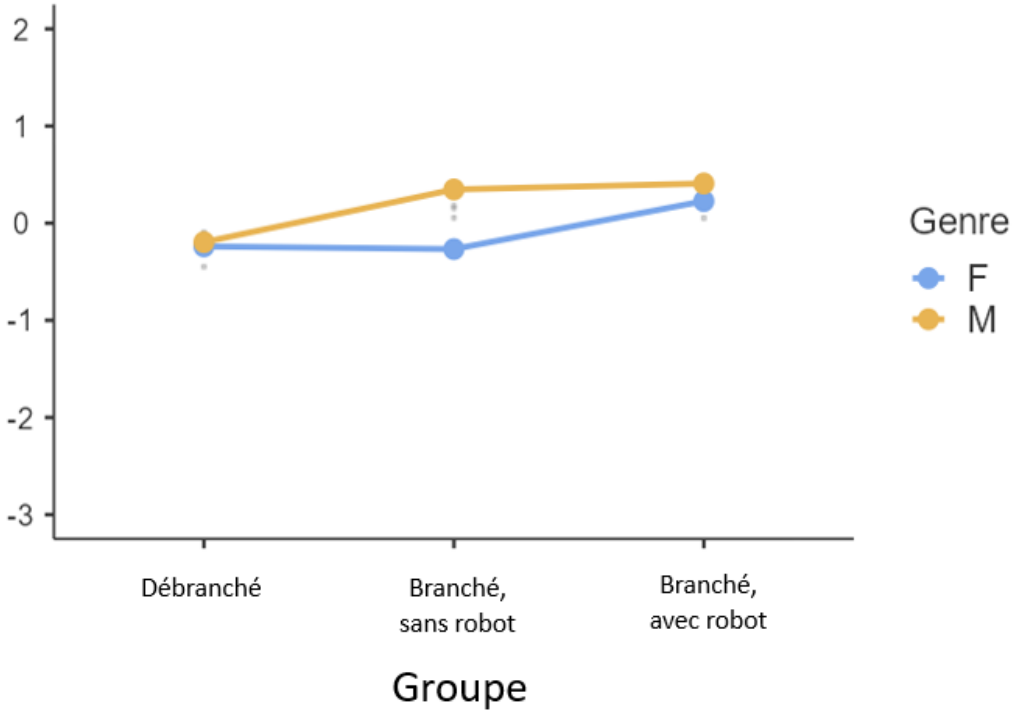
Cependant, alors que les résultats au pré-test sont très similaires entre les trois groupes, les résultats au post-test diffèrent assez largement en faveur du groupe branché avec robot (voir Tableau 14 et Tableau 15). Notre analyse confirme cette observation. La variable « groupe » semble avoir un effet significatif sur la motivation et le sentiment d'auto-efficacité global ($F(2, 234) = 16.53$, $p < .001$). Il n'y a aucune différence entre les groupes débranché et branché sans robot ($\beta = 0.03$, $p = .80$, 95% IC [-0.22 ; 0.29]). En revanche, la différence entre les groupes débranché et branché avec robot est significative avec une grande taille d'effet ($\beta = 0.75$, $p < .001$, 95% IC [0.48 ; 1.03]). Aucun effet du genre ($F(1, 235) = 3.51$, $p = .06$) ou de l'interaction entre le genre et le groupe ($F(2, 231) = 2.03$, $p = .13$) n'est observé.

Concernant le sentiment d'auto-efficacité général à l'école, on retrouve un effet significatif du groupe expérimental ($F(2, 222) = 6.56$, $p < .01$) mais toujours aucun effet du genre ($F(1, 223) = 1.18$, $p = .28$) et de l'interaction entre le groupe et le genre ($F(2, 219) = 2.40$, $p = .09$). Le groupe branché avec robot est le seul à marquer une progression de taille modérée ($\beta = 0.35$, $p < .01$, 95% IC [0.11 ; 0.59]) par rapport au pré-test. On remarque cependant que les résultats au pré-test étaient plus faibles dans ce groupe comparé aux deux autres (voir Tableau 14). Pour le sentiment d'auto-efficacité relatif aux activités de programmation, même si l'on observe une augmentation globale des résultats de cette variable entre le pré-test et le post-test dans les trois groupes, celle-ci n'est pas significative. En effet, aucun impact du groupe ($F(2, 233) = 0.88$, $p = .44$), du genre ($F(1, 236) = 1.10$, $p = .30$) ou de l'interaction entre les deux ($F(2, 230) = 1.54$, $p = .22$) ne peut être observé.

En ce qui concerne la motivation, il existe un effet significatif du groupe expérimental ($F(2, 233) = 4.10$, $p = .05$) en faveur du groupe branché avec robot ($\beta = 0.54$, $p < .05$, 95% IC [0.17 ; 0.91]). De même, un faible effet du genre peut être observé en faveur des garçons qui semblent plus motivés que les filles ($F(1, 234) = 9.18$, $p < .05$, $\beta = 0.28$, 95% IC [0.06 ; 0.50]). Cependant, l'effet d'interaction est toujours non significatif ($F(1, 230) = 2.71$, $p = .07$). Notons toutefois un effet marginal concernant cette interaction. Une analyse plus précise, centrée sur des comparaisons deux à deux, montre qu'il y a potentiellement un effet d'interaction non négligeable. Les différences de genre ne sont observables que dans la condition branchée sans robot ($\beta = 0.57$, $p < .05$, 95% IC [0.08 ; 1.06]) qui se distingue des deux autres, comme illustré en Figure 11.

Figure 11

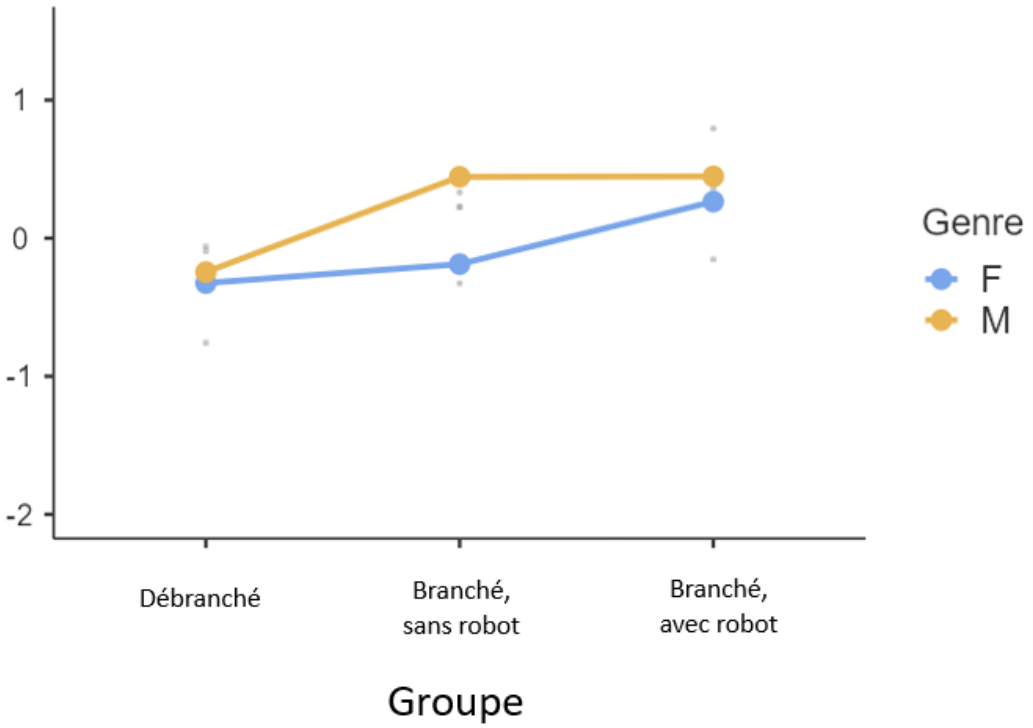
Moyennes des scores standardisés de motivation, issus du post-test relatif à la motivation et au sentiment d'auto-efficacité, selon le groupe et le genre des élèves



Les items 19 et 20 n'étaient pas présents dans le pré-test. Notre modèle mixte concernant la variable composée de l'addition de ces deux items ne comporte donc aucune covariable, mais toujours nos deux effets fixes (groupe et classe) et notre effet aléatoire (classe). L'analyse ne révèle pas d'effet significatif global du groupe ($F(2, 243) = 3.43$, $p = .07$) mais une comparaison deux à deux montre qu'il existe une différence significative en faveur du groupe branché avec robot ($\beta = 0.64$, $p < .05$, 95% IC [0.14 ; 1.14]). Par ailleurs, on retrouve à nouveau des résultats très similaires à ce que nous avons observé concernant les trois autres items de motivation présents à la fois dans le pré-test et le post-test : un faible effet du genre en faveur des garçons ($F(1, 244 = 6.42$, $p = .01$, $\beta = 0.30$, 95% IC [0.07 ; 0.53]) et un effet d'interaction non significatif ($F(2, 240) = 2.36$, $p = .10$). Cependant, comme précédemment, une comparaison plus précise montre que l'effet du genre n'est visible que dans la condition branchée sans robot ($\beta = 0.55$, $p < .05$, 95% IC [0.04 ; 1.06]). La Figure 12 illustre cette interaction.

Figure 12

Moyennes des scores standardisés aux items 19 et 20, issus du post-test relatif à la motivation et au sentiment d'auto-efficacité, selon le groupe et le genre des élèves



Attitude vis-à-vis des sciences

Le tableau 16 présente les statistiques descriptives des scores au pré-test concernant l'attitude des élèves envers les sciences (moyennes et écarts-types), dans les groupes expérimentaux, selon le genre : le score total (sur 60) et les sous-scores relatifs à l'apprentissage des sciences à l'école (sur 18), au sentiment d'auto-efficacité en sciences (sur 15), au désir de faire de la science (sur 12), à la valeur perçue des sciences pour la société (sur 9) et à l'anxiété à l'égard des sciences (sur 6).

Tableau 16

Moyennes (et écarts-types entre parenthèses) des scores au pré-test d'Attitude envers les sciences dans les trois groupes expérimentaux, selon le genre

Groupe	Genre	Total	Science à l'école	Auto-efficacité	Désir	Valeur perçue	Anxiété
Débranché	**F**	36.2 (11.4)	10.5 (4.58)	10.3 (3.45)	4.72 (2.56)	6.08 (2.20)	5.07 (1.48)
	M	37.3 (9.77)	10.9 (4.48)	10.5 (2.83)	4.71 (2.65)	6.25 (2.07)	5.06 (1.24)
Branché sans robot	**F**	36.5 (9.73)	11.3 (3.98)	10.5 (3.11)	4.00 (2.39)	5.42 (2.26)	5.42 (1.12)
	M	40.7 (10.5)	13.1 (4.01)	10.5 (3.29)	5,97 (2.99)	6.29 (2.02)	5.26 (1.24)
Branché avec robot	**F**	39.7 (10.2)	12.9 (4.00)	10.4 (3.00)	5.54 (2.79)	5.86 (2.34)	5.11 (1.35)
	M	38.8 (10.7)	11.6 (4.30)	9.58 (3.61)	5.18 (2.28)	6.33 (2.17)	5.18 (1.47)

Le Tableau 17 présente les mêmes statistiques que le Tableau 16, relatives cette fois-ci au post-test d'attitude envers les sciences.

Tableau 17

Moyennes (et écarts-types entre parenthèses) des scores au post-test d'Attitude envers les sciences dans les trois groupes expérimentaux, selon le genre

Groupe	Genre	Total	Science à l'école	Auto-efficacité	Désir	Valeur perçue	Anxiété
Débranché	**F**	37.0 (11.9)	10.0 (4.73)	10.8 (3.67)	4.78 (2.54)	6.27 (1.99)	5.25 (1.36)
	M	35.8 (9.96)	10.2 (4.62)	10.5 (327)	4.15 (2.37)	6.12 (2.21)	5.21 (1.13)
Branché sans robot	**F**	37.1 (7.17)	10.8 (4.13)	10.8 (2.52)	4.30 (1.85)	5.74 (1.80)	5.53 (0.96)
	M	41.2 (8.89)	13.1 (3.63)	10.8 (2.97)	5.79 (2.41)	6.17 (1.74)	5.41 (1.05)
Branché avec robot	**F**	37.7 (11.8)	11.7 (5.27)	10.2 (3.20)	4.97 (2.82)	6.05 (2.11)	5.05 (1.37)
	M	38.3 (12.1)	11.6 (5.42)	10.0 (3.41)	4.82 (3.07)	6.69 (1.91)	5.31 (1.38)

Notre modèle mixte comprend trois effets fixes (le groupe expérimental, le genre et le pré-test d'attitude envers les sciences en covariable) et un effet aléatoire (la classe), dont on souhaite mesurer l'influence sur le post-test d'attitude envers les sciences. Nous avons comparé ce modèle au modèle nul (LRT = 7.63, $p < .01$, AIC = 501 pour notre modèle et AIC = 506 pour le modèle nul) et cette analyse confirme la pertinence de notre modèle. Nous observons que 8% de la variance concernant les scores au post-test d'attitude envers les sciences s'explique par une variabilité entre les classes et 92% par une variabilité au sein d'une même classe, entre les élèves (ICC = .08).

Aucun impact significatif du groupe expérimental ($F(2, 252) = 0.41$, $p = .67$), du genre ($F(1, 253) = 0.03$, $p = .87$) et de l'interaction entre le groupe et le genre ($F(2, 249) = 1.53$, $p = .22$) n'a pu être observé. En ce qui concerne le groupe expérimental, l'effet linéaire ($\beta = 0.00$, $p = .99$, 95% IC [-0.21 ; 0.21] et l'effet quadratique ($\beta = -0.11$, $p = .38$, 95% IC [-0.35 ; 0.13]) ne sont pas significatifs, ce qui indique qu'aucun des trois groupes ne se distingue des deux autres. Par ailleurs, l'impact de la covariable « pré-test d'attitude envers les sciences » est très significatif et fort ($\beta = 0.75$, $p < .001$, 95% IC [0.68 ; 0.84]). Ceci signifie que le pré-test est la variable qui détermine en grande partie les résultats observés au post-test, contrairement au groupe et au genre qui n'ont aucune influence. Par souci de concision, nous ne détaillerons pas les résultats relatifs aux sous-scores d'attitude envers les sciences, qui sont similaires à ceux concernant le score total, et qui ne témoignent d'aucune influence du groupe, du genre et de l'interaction entre les deux.

Corrélations entre les différentes variables

Notre dernière analyse consiste à mettre en évidence des liens de corrélation plus ou moins forts entre les variables dépendantes que nous avons mesurées (maîtrise des concepts computationnels, capacité à résoudre des problèmes algorithmiques, pré-test et post-test relatifs à la motivation et au sentiment d'auto-efficacité, pré-test et post-test relatifs à l'attitude envers les sciences). Pour ce faire, nous avons réalisé une matrice de corrélation. Le Tableau 18 indique les liens de corrélation entre ces variables, en se basant sur le coefficient de corrélation *r* de Pearson (maîtrise des concepts computationnels = MCC ; résolution de problèmes algorithmiques = A ; motivation et sentiment d'auto-efficacité = MAE ; attitude envers les sciences = AS). Par ailleurs, nous avons signalé par des astérisques les liens significatifs : un astérisque correspond à une valeur de *p* inférieure à .05, deux astérisques à une valeur inférieure à .01 et trois astérisques à une valeur inférieure à .001.

Tableau 18

Matrice de corrélation entre les variables dépendantes, basée sur le r de Pearson

	MCC	A	Pré-test MAE	Post-test MAE	Pré-test AS	Post-test AS
MCC	//////////////////	//////////////////	//////////////////	//////////////////	//////////////////	//////////////////
A	.19 **	//////////////////	//////////////////	//////////////////	//////////////////	//////////////////
Pré-test MAE	.34 ***	.15 *	//////////////////	//////////////////	//////////////////	//////////////////
Post-test MAE	.30 ***	.09	.61 ***	//////////////////	//////////////////	//////////////////
Pré-test AS	.18 **	.02	.39 ***	.50 ***	//////////////////	//////////////////
Post-test AS	.25 ***	-.01	.33 ***	.57 ***	.76 ***	//////////////////

Au vu de la taille relativement importante de notre échantillon, un grand nombre de corrélations sont significatives, d'intensités variables. Comme le souligne Taylor (1990), il peut être difficile d'interpréter un coefficient de corrélation. Une corrélation statistiquement significative n'est pas nécessairement une forte corrélation, comme en témoigne notre Tableau 18. Certaines valeurs seuils sont parfois proposées pour interpréter ce coefficient. Un coefficient inférieur ou égal à .35 peut être considéré comme la marque d'un lien de corrélation faible. Entre .36 et .67, le lien est considéré comme modéré et, au-delà de .68, comme fort. Mais Taylor (1990) recommande de surtout se fier à la valeur de *p*, plus pertinente, car elle prend en compte la taille de l'échantillon.

Logiquement, les liens de corrélation les plus forts s'observent entre le pré-test et le post-test de motivation et sentiment d'auto-efficacité d'une part (r = .61) et le pré-test et le post-test d'attitude envers les sciences d'autre part (r = .76). On remarque également que ces deux variables conatives sont aussi liées entre elles par des liens de corrélation statistiquement très significatifs, en particulier le post-test de motivation et sentiment d'auto-efficacité qui présente un lien relativement fort avec les deux mesures d'attitude envers les sciences (r = .50 pour le pré-test, r = .57 pour le post-test). Même si une relation de cause à effet ne peut être établie en se basant sur ces résultats, on peut supposer qu'une attitude positive envers les sciences en général pourrait favoriser la motivation et le sentiment d'auto-efficacité en relation avec les

activités de programmation. *A contrario*, nos variables cognitives (maîtrise des concepts computationnels et capacité à résoudre des problèmes algorithmiques) ne sont que faiblement corrélées (r = .19), même si cette corrélation est statistiquement significative. D'ailleurs, les résultats au Test d'Algorithmique, censé mesurer la capacité à résoudre des problèmes algorithmiques, entretiennent des liens de corrélation faibles, voire inexistants, avec toutes les autres variables étudiées. En revanche, la maîtrise des concepts computationnels est corrélée à l'ensemble des variables, même si ces liens demeurent assez faibles. On notera tout de même, un coefficient plus élevé (r > .30) entre la maîtrise des concepts computationnels et les deux mesures relatives à la motivation et au sentiment d'auto-efficacité. Là encore, il n'est pas possible d'affirmer un lien de causalité entre les deux, même si un effet de la motivation et du sentiment d'auto-efficacité sur les performances d'apprentissage (probablement de faible taille) pourrait exister.

2.4) Discussion de l'expérience 1

Cette première expérience avait pour but de comparer trois outils régulièrement utilisés dans l'enseignement de la programmation et de la pensée informatique : les activités débranchées et les activités branchées via un logiciel éducatif visuel de programmation par blocs (Scratch), associées ou non à un robot pédagogique (le robot Thymio II). L'impact du genre des élèves a également été étudié. Nous avons mesuré l'influence de ces outils et du genre sur la maîtrise de concepts computationnels, la capacité à résoudre des problèmes algorithmiques, la motivation, le sentiment d'auto-efficacité et l'attitude envers les sciences. Nous avons également observé l'impact du groupe et du genre sur les sous-scores qui constituaient ces variables, mais il est important de noter que ces résultats sont nécessairement moins fiables. Les sous-scores sont composés d'un nombre d'items réduit et leur distribution suit moins bien la loi normale. Les élèves ont suivi 10 séances de programmation de 45 minutes qui ont été conçues pour être similaires entre nos trois conditions expérimentales, c'est-à-dire centrées sur les mêmes notions à apprendre, les mêmes compétences à développer et les mêmes types de tâches à réaliser (avec ou sans support numérique).

Performances d'apprentissage

Les performances d'apprentissage des élèves ont été évaluées à la fin des 10 séances de programmation. Le résultat principal de cette expérience révèle des performances significativement supérieures dans le groupe branché sans robot, comparativement aux groupes

débranché et branché avec robot qui présentent un niveau de performance sensiblement équivalent. Concernant la maîtrise des concepts computationnels, l'effet du groupe expérimental est fort, en faveur de la condition branchée sans robot. Cependant, cet effet n'est pas observable sur les concepts d'algorithme et d'instruction, que nous pouvons considérer comme les deux concepts les plus basiques et simples. En revanche, l'effet du groupe est moyen pour les concepts de boucle et de condition et fort pour le concept de variable. Cette analyse des sous-scores est intéressante, car les trois concepts évoqués sont généralement assez complexes à appréhender pour des élèves novices, même avec des langages visuels de programmation par blocs (Zhang & Nouri, 2019), en particulier le concept de variable (Hermans & Aivaloglou, 2017 ; Tchounikine, 2017). Il semble donc qu'une approche branchée sans robot permette une meilleure compréhension des concepts computationnels, et que ce gain d'apprentissage est d'autant plus marqué que le concept à apprendre est complexe.

De même, les élèves du groupe branché sans robot semblent plus à même de résoudre des problèmes algorithmiques, en comparaison avec ceux des groupes débranché et branché avec robot qui, là encore, ne se distinguent pas l'un de l'autre. L'effet observé est fort sur cette variable, ainsi que sur l'ensemble des sous-variables étudiées. Il paraît donc assez clair qu'une approche branchée sans robot favorise la capacité des élèves à produire un algorithme composé d'instructions pertinentes, et à utiliser correctement des boucles, des conditions et des variables pour résoudre des problèmes algorithmiques. La taille d'effet la plus importante concerne à nouveau l'utilisation des variables. Par ailleurs, les différences entre les groupes ne s'observent que sur les deux derniers exercices qui nécessitaient une meilleure maîtrise des variables. Les deux premiers exercices étaient plus simples et plus concrets, car ceux-ci ne demandaient aux élèves que de programmer les déplacements d'un personnage sur un chemin. Ainsi, ces résultats confirment à nouveau qu'une approche branchée sans robot est d'autant plus efficace que les concepts et compétences abordés sont relativement complexes. Ce constat avait d'ores et déjà été observé avec de jeunes lycéens (Armoni et al., 2015).

Ces résultats semblent contredire d'autres études réalisées précédemment, ayant mis en évidence une absence de différence entre les activités branchées et débranchées (Hermans et Aivaloglou, 2017 ; Romero et al., 2018), voire une supériorité des approches incluant des activités débranchées (del Olmo-Muñoz et al., 2020 ; Wohl et al., 2015). Par ailleurs, nos résultats contredisent également les quelques études mettant en exergue une absence de différence entre les conditions branchées, avec et sans robot (Merkouris & Chorianopoulos, 2015), ou une supériorité des approches avec robot (Bellegarde et al., 2019 ; Kert et al., 2020).

Cependant, nous nous devons de rappeler que ces quelques études présentent des protocoles expérimentaux bien différents du nôtre. La plupart d'entre elles met en œuvre des approches mixtes, combinant activités branchées et débranchées. Lorsqu'elles utilisent des instruments de mesure validés, ceux-ci sont souvent constitués de problèmes à résoudre indépendants des activités de programmation et peuvent donc potentiellement mettre en évidence un effet de transfert, c'est-à-dire une capacité à transférer les compétences liées à la pensée informatique, supposément développées par les activités de programmation, dans un contexte nouveau. Cet effet de transfert n'était pas l'objet de notre étude. Enfin, ces études comportaient généralement des échantillons de taille plus faible, comprenant des élèves souvent plus jeunes et donc moins habitués à l'utilisation des outils numériques, ce qui pourrait expliquer les meilleures performances parfois observées dans les conditions débranchées. En revanche, les résultats de notre expérience s'accordent avec l'étude de Namli et Aybek (2022) qui montre une supériorité des approches branchées en ce qui concerne les performances d'apprentissage. En outre, Huang et Looi (2021) se sont également montrés critiques envers les activités débranchées, qui peuvent, selon eux, être tout juste aussi efficaces que les activités branchées dans certains contextes très spécifiques.

Vécu subjectif des élèves

Nous avons intégré des échelles autoévaluatives, mesurées pré-test et post-test pour évaluer la motivation, le sentiment d'auto-efficacité et l'attitude envers les sciences des élèves dans nos trois conditions expérimentales. Le pré-test de motivation et sentiment d'auto-efficacité a été réalisé avant les activités de programmation et reflète donc l'anticipation de ces activités par les élèves, celle-ci étant probablement influencée par leurs préconceptions au sujet de la programmation. Les résultats témoignent d'un accroissement de la motivation et du sentiment d'auto-efficacité général des élèves dans la condition branchée avec robot. Deux des items relatifs à la motivation n'étaient présents que dans le post-test (Item 19 : « Quand je fais de la programmation, il m'arrive d'être totalement absorbé par ce que je fais » et Item 20 : « J'aimerais faire de la programmation sur mon temps libre »), car nous avons estimé que ceux-ci n'auraient pas de sens dans un pré-test auprès de novices n'ayant jamais programmé. Pour ces deux items, qui traduisent un certain engagement dans les activités de programmation, c'est encore une fois le groupe branché avec robot qui présente des résultats significativement supérieurs à ceux des deux autres groupes. Globalement, il existe une certaine linéarité dans les effets sur la motivation : les élèves en condition débranchée semblent moins motivés que ceux en condition branchée sans robot, eux-mêmes moins motivés que ceux en condition branchée

avec robot, même si les différences significatives concernent surtout la relation débranchée / branchée avec robot. Pour le sentiment d'auto-efficacité en lien avec les activités de programmation, celui-ci augmente marginalement dans toutes les conditions, mais jamais de manière significative.

Notre étude montre ainsi un effet positif des outils numériques sur la motivation et l'engagement des élèves en programmation, qui avait déjà été observé avec le logiciel Scratch, censé être attrayant pour les élèves (Armoni et al., 2015 ; Sáez-López et al., 2016) et donner envie de continuer les activités de programmation (Armoni et al., 2015 ; Ouahbi et al., 2015). Le caractère attrayant des robots pédagogiques est déjà étayé empiriquement (Ching et al., 2018 ; Riedo et al., 2013) et l'effet particulièrement positif de la robotique sur ces variables conatives a parfois été évoqué ou supposé précédemment (Kaloti-Hallak et al., 2015 ; Kim & Lee, 2016), notamment une préférence, indépendante du genre des élèves, pour les dispositifs associant logiciel et robot pédagogique, comparé à un apprentissage réalisé exclusivement sur logiciel (Merkouris & Chorianopoulos, 2015 ; Kert et al., 2020). L'effet positif de l'association logiciel / robot sur le sentiment d'auto-efficacité dans notre étude confirme également les résultats observés par Kert et al. (2020). Notre expérience semble donc s'opposer aux auteurs qui considéraient les activités débranchées comme potentiellement vectrices d'une attitude plus positive, d'une motivation plus importante et d'un engagement et d'un sentiment d'auto-efficacité plus grands de la part des élèves, sans se baser sur des données empiriques comparatives (Battal et al., 2021 ; Delal & Oner, 2020 ; Threekunprapa & Yasri, 2020). Les dernières séances que nous avons mises en place dans les classes proposaient aux élèves de réaliser un projet créatif, relativement libre. Nous pouvons supposer que la richesse et la variété des solutions proposées par les outils numériques ont permis le maintien ou l'augmentation de la motivation des élèves.

Enfin, aucun effet sur l'attitude envers les sciences n'a été observé, et ce, quel que soit le groupe ou le genre des élèves. Il semblerait donc que les activités de programmation ne produisent pas d'effet évident ou immédiat sur l'attitude envers les sciences, comme la revue de littérature de Zhang et al. (2021), centrée uniquement sur la robotique, le laissait présager. Nous pouvons tout de même supposer qu'une intervention plus longue pourrait produire un certain impact sur cette variable.

Effet du genre

Les activités de programmation révèlent certaines inégalités entre hommes et femmes. Les femmes sont moins attirées par cette discipline (Gökçe & Yenmez, 2020 ; Weintrop & Wilensky, 2019) même si cet effet s'observe moins à l'école primaire (Román-González et al., 2017). L'impact du genre sur les performances est très contradictoire selon les études. Dans notre expérience, les effets du genre des élèves sur les performances sont globalement faibles ou incertains et contribuent à entretenir le flou sur cette question. Celle-ci met tout de même en lumière une meilleure maîtrise des concepts computationnels chez les filles, en particulier en ce qui concerne le concept de boucle. Un effet d'interaction peut être observé, les filles semblent mieux maîtriser le concept d'instruction dans la condition branchée sans robot que les garçons et, à l'inverse, les garçons semblent légèrement mieux maîtriser ce concept dans la condition débranchée. Aucun effet lié au genre n'est observable en relation avec la capacité à résoudre des problèmes algorithmiques. Globalement, il n'existe aucun effet d'interaction important entre le groupe expérimental et le genre sur les performances, contrairement à ce qui avait pu être observé par certains auteurs : impact positif de Scratch sur les performances des filles (Scherer et al., 2020) ou sur celles des garçons (Jiang & Li, 2021) et de la robotique éducative sur les celles des garçons (Zhang et al., 2021).

L'impact du genre sur la motivation et l'engagement des élèves est cependant plus intéressant et plus contrasté. Nous avons vu précédemment que la multiplication des outils numériques produisait un effet positif sur les résultats obtenus concernant cette variable (plus de motivation avec Scratch, plus encore lorsque Scratch est associé à un robot). Cependant, une analyse plus approfondie révèle une différence significative entre garçons et filles dans la condition branchée sans robot, les garçons étant plus motivés et engagés que les filles. Dans les deux autres conditions, les différences de genre sont très faibles ou inexistantes. Nous pouvons donc en déduire qu'une approche exclusivement virtuelle, sans manipulation ou utilisation d'objets concrets, est défavorable aux filles qui présentent un niveau de motivation moins élevé que celui des garçons. Ces résultats confirment donc une absence d'écart entre les genres dans la condition débranchée (Delal & Oner, 2020), souvent considérée comme plus neutre vis-à-vis des stéréotypes de genre (Bell et al., 2009). Cependant, les activités de robotique sont également susceptibles d'accroître la motivation des filles, en comparaison avec une situation pour laquelle seul un logiciel visuel est utilisé, conformément à ce qui avait été avancé par Gunes et Kucuk (2022). Les filles semblent aussi motivées que les garçons dans la condition branchée avec robot, comme le montraient déjà Merkouris et Chorianopoulos (2015), en opposition à Su

et Yang (2023), qui considéraient que les robots seraient des jouets à prédominance masculine, défavorables aux filles. L'impact motivationnel positif de Scratch sur les filles, supposé par Tikva et Tambouris (2019), n'est ici pas retrouvé. Au contraire, ce sont les garçons qui en bénéficient.

Liens entre les variables cognitives et conatives

Cette expérience 1 a permis de mettre en lumière les liens de corrélation existant entre nos différentes variables d'intérêt. Les variables conatives (motivation et sentiment d'auto-efficacité, attitude envers les sciences) sont assez fortement corrélées. On peut finalement supposer que la motivation et l'engagement des élèves dans les activités de programmation dépendent au moins en partie de leur attitude vis-à-vis des sciences. Les variables cognitives semblent toutefois plutôt mal corrélées. Il existe un lien de corrélation, relativement faible, entre la maîtrise des concepts computationnels et la capacité à résoudre des problèmes algorithmiques. Ceci nous incite à penser, comme l'affirmaient déjà Robins et al. (2003), que la compréhension et la capacité à générer un programme sont deux aspects de la programmation qui sont liés, mais qui constituent également deux tâches bien distinctes l'une de l'autre. Ces deux variables étaient évaluées par deux tests que nous avons conçus, et pour lesquels nous avons constaté dans le « Chapitre 1 » de cette « Partie expérimentale » qu'ils n'étaient pas du même niveau de complexité.

Par ailleurs, les résultats au Test d'Algorithmique sont assez mal corrélés à l'ensemble des autres variables, ce qui peut signifier que la capacité à résoudre des problèmes dépend surtout de certains processus cognitifs, assez faiblement reliés à la connaissance et la compréhension de concepts, à la motivation, au sentiment d'auto-efficacité ou à l'attitude envers les sciences. Enfin, la maîtrise de concepts computationnels et la capacité à résoudre des problèmes algorithmiques sont significativement corrélées aux mesures de motivation et de sentiment d'auto-efficacité, en particulier pré-test. Même si les coefficients de corrélation ne permettent pas d'établir un lien de causalité, il est possible de supposer que la motivation et le sentiment d'auto-efficacité initiaux des élèves peuvent produire un effet positif sur les apprentissages.

Perspectives

Nos résultats mettent en lumière les limites des approches exclusivement débranchées et confortent les opinions de certains chercheurs qui estiment que le potentiel de ces activités

débranchées est restreint (Brackmann et al., 2017 ; Caeli & Yadav, 2020). Notre expérience ne révèle aucun apport positif de la robotique éducative sur les performances d'apprentissage, bien que l'utilisation de robots produise un gain non négligeable en termes de motivation et de sentiment d'auto-efficacité. Les effets de genre sont faibles, mais témoignent de performances légèrement supérieures pour les filles. La motivation des filles est cependant moins élevée, exclusivement dans la condition branchée sans robot. L'apport motivationnel des robots, et dans une moindre mesure de Scratch, pourrait s'expliquer par le caractère attrayant de ces outils, souvent pensés pour un jeune public, et par un effet de nouveauté (Cheung & Slavin, 2013) qui n'a pas nécessairement disparu au bout de 10 séances de programmation. En revanche, l'effet significatif du groupe expérimental, favorable à la condition branchée sans robot, est plus difficilement explicable, car les arguments théoriques avancés pour défendre les trois outils que nous avons étudiés sont cohérents et pertinents.

Notre étude constate des différences significatives entre nos trois groupes expérimentaux mais ne permet pas d'expliquer pourquoi nous constatons ces différences. Ching et al. (2018) voient un effet positif potentiel du feedback permis par les logiciels de programmation, mais ils envisagent également que la robotique devrait améliorer la maîtrise des aspects les plus complexes en programmation, ce que nous n'avons pas constaté. Les auteurs reconnaissent cependant la nécessité d'un investissement plus important pour parvenir à exploiter le potentiel des robots pédagogiques. Les outils numériques offrent la possibilité de bénéficier d'un feedback permettant de visualiser en temps réel l'exécution d'un programme créé par l'élève, ce qui pourrait faciliter un apprentissage fondé sur des essais et des erreurs successives. L'influence majeure du feedback sur les apprentissages est déjà bien documentée (Hattie & Timperley, 2007). Le « Chapitre 4 » de notre « Partie théorique » met également en exergue l'impact déterminant de la charge cognitive d'une situation d'apprentissage. Il est possible qu'une situation associant logiciel éducatif et robot pédagogique représente une charge cognitive extrinsèque trop importante pour permettre un apprentissage optimal, du moins auprès d'élèves de cycle 3. L'étude de l'influence du feedback et de la charge cognitive associés à nos trois outils d'enseignement constitue l'objet des trois dernières expériences de cette thèse, détaillées dans le chapitre suivant.

SYNTHÈSE

- Notre première expérience avait pour but de **comparer trois outils** utilisés pour enseigner la programmation auprès d'élèves de CM2.
- Les **performances d'apprentissage sont meilleures dans la condition branchée sans robot**, en comparaison avec les conditions débranchées et branchées avec robot.
- Cependant, les **robots pédagogiques** semblent produire un **effet positif de taille importante sur la motivation** des élèves.
- **Aucun effet** n'a pu être observé concernant l'**attitude envers les sciences**.
- Dans la **condition branchée sans robot**, nous avons pu observer des différences entre les genres : **les filles sont moins motivées que les garçons**.

Chapitre 3. Approfondissements

3.1) Introduction des expériences 2, 3 et 4

Les trois expériences qui seront décrites dans ce chapitre ont pour objectif d'approfondir et d'expliquer les résultats observés dans notre première expérience concernant les performances d'apprentissage en programmation d'élèves de CM2. Nous avons notamment observé de meilleures performances dans la condition branchée sans robot (à l'aide du logiciel Scratch) par rapport aux deux autres conditions. Notre expérience 1 a par ailleurs mis en exergue un effet délétère de l'association d'un robot Thymio à Scratch sur la maîtrise des concepts computationnels et la capacité à résoudre des problèmes algorithmiques. Nous supposons que la présence d'un feedback basique et immédiat, permis par les logiciels de programmation éducatifs, pourrait renforcer les apprentissages et expliquer les moins bonnes performances des élèves en condition débranchée. De plus, la charge cognitive associée à une situation d'apprentissage comprenant à la fois un logiciel et un robot pourrait être trop importante pour des élèves de CM2.

Le rôle du feedback sur l'amélioration des apprentissages

L'une des plus-values intéressantes des logiciels de programmation tels que Scratch repose sur la possibilité de visualiser en temps réel l'exécution du programme qui a été généré par l'élève (Maloney et al., 2010). Ce type particulier de feedback peut être considéré comme immédiat (il suffit de cliquer sur un bouton ou sur une instruction pour en bénéficier), mais aussi relativement basique. En effet, la visualisation de l'exécution d'un programme ne peut que rendre compte d'une différence entre ce qui était attendu et ce qui se passe réellement, conduisant potentiellement l'élève à corriger son programme dans l'optique d'obtenir le résultat désiré. Aucune indication n'est fournie pour préciser à l'élève à quel endroit de son programme une « erreur » serait à corriger ni pour quelle raison l'objectif n'a pas été atteint. On ne peut alors que supposer que cette rétroaction conduira l'apprenant à persévérer dans la tâche afin que cette succession d'essais et d'erreurs le mène au succès.

Dans leur revue de littérature, Hattie et Timperley (2007) considèrent que le feedback a une influence majeure sur les apprentissages et la réussite scolaire, mais cette influence peut être positive ou négative. Pour eux, le feedback peut être conceptualisé comme une information

fournie par un agent (pas nécessairement humain) concernant les performances ou la compréhension d'un apprenant. Le principal objectif du feedback serait ainsi de « réduire l'écart entre les connaissances et performances actuelles et un objectif ». Selon Hattie et Timperley (2007), un feedback efficace doit répondre à trois questions majeures : « Où vais-je ? (Quels sont les objectifs ?) », « Comment y vais-je ? (Quels sont les progrès réalisés pour atteindre l'objectif ?) », et « Que faire ensuite ? (Quelles sont les activités à entreprendre pour mieux progresser ?) ». Ces trois questions correspondent à ce qu'ils appellent *feed up*, *feed back* et *feed forward*. La visualisation de l'exécution d'un programme sur Scratch correspond plutôt à la deuxième question posée (et donc bien au *feed back*), les réponses aux deux autres questions devant plutôt être apportées par l'enseignant. Très souvent, l'attention portée aux progrès de l'élève peut être porteuse d'un jugement qui peut potentiellement freiner les apprentissages. Or, le feedback permis par les logiciels de programmation peut être considéré comme neutre et non menaçant.

Pour Hattie et Timperley (2007), il y a quatre niveaux différents de feedback. Celui-ci peut concerner une tâche (correctement ou incorrectement réalisée), le processus de réalisation de cette tâche (processus de traitement de l'information ou processus d'apprentissage), le niveau d'autorégulation (compétence en matière d'auto-évaluation et confiance en soi pour s'engager dans la tâche) ou encore être plus personnel et s'adresser au « moi » (critique, compliments, etc.). Ce dernier niveau de feedback est très souvent mal relié au niveau de performance dans une tâche. Dans notre contexte, le feedback permis par les logiciels de programmation concerne plutôt la tâche en elle-même. C'est donc un feedback correctif. Hattie et Timperley (2007) remarquent que ce niveau de feedback peut être puissant pour améliorer les apprentissages. Ils mettent en garde toutefois sur les risques d'un feedback trop fréquent au niveau de la tâche qui peut conduire les apprenants à se concentrer sur les objectifs immédiats, plutôt que sur les stratégies pour atteindre ces objectifs. Ceci peut les encourager à multiplier les essais et erreurs et donc à réduire l'effort cognitif consistant à formuler des hypothèses informelles relatives aux liens qui existent entre les instructions données par l'enseignant, le feedback obtenu et l'apprentissage visé. Ce constat nous incite donc à relativiser l'influence positive du feedback des logiciels de programmation, car celui-ci est illimité, en termes de quantité.

Par ailleurs, les tâches simples bénéficieraient davantage de ce type de feedback par rapport aux tâches complexes (Hattie & Timperley, 2007). Notons toutefois que les auteurs rapportent des résultats témoignant de meilleures performances lorsque le feedback était peu

complexe (indication de quelle était la bonne réponse dans un questionnaire à choix multiples) en comparaison à une situation où ce feedback l'est davantage (explications concernant les raisons pour lesquelles les autres réponses étaient incorrectes). Hattie et Timperley (2007) considèrent que le fait de fournir des informations concernant les réponses incorrectes augmente la probabilité que les élèves se souviennent de l'erreur. Ces informations ont également pu être traitées à un niveau superficiel, car ces derniers n'ont pas correctement perçu les liens entre les informations données et l'identification de la bonne réponse. Toujours est-il que la complexité du feedback fourni n'est pas forcément corrélée positivement à son efficacité et que le feedback des logiciels de programmation pourrait être efficace, même s'il peut être considéré comme très basique.

En outre, Hattie et Timperley (2007) se sont également intéressés au délai entre la tâche et le feedback. Leur revue de littérature témoigne d'un impact positif d'un feedback immédiat lorsque celui-ci est relatif à la tâche, ce qui est le cas avec les logiciels éducatifs de programmation. Par ailleurs, le feedback immédiat des outils numériques en programmation initie le processus de débogage par un cycle itératif d'observation, de formulation d'hypothèses, de vérification et d'évaluation de la solution proposée (Sullivan, 2008). Siegfried et al. (2017) ont étudié l'impact d'un feedback immédiat composé d'indices sur l'apprentissage et les performances d'élèves lors d'une tâche de programmation d'un robot à travers un labyrinthe. Les résultats montrent que ceux-ci ont pu mener à bien cette tâche sans l'aide extérieure d'un enseignant, ont amélioré de manière significative le code produit et ont réduit le temps nécessaire pour écrire un programme correct, bien qu'aucune corrélation n'ait pu être observée entre ces performances et les performances à un test d'acquisition des concepts, réalisé ultérieurement.

Cependant, Chevalier et al. (2022) ont mis en place des activités de robotique éducative pour des élèves de 8-9 ans afin de tester si la présence de guidage et le feedback (immédiat ou différé) pouvaient améliorer les performances d'apprentissage. Leur étude met en exergue de meilleures performances lorsque le guidage est présent et lorsque le feedback est différé en ce qui concerne l'une des composantes de la pensée informatique. Les auteurs approfondissent toutefois leur analyse en observant que chaque type de feedback présente des avantages et des inconvénients. Le feedback immédiat serait plus efficace pour mener l'élève à l'accomplissement d'une tâche donnée, mais ne conduirait pas nécessairement au développement des processus cognitifs liés à la pensée informatique, c'est-à-dire aux processus qui relèvent notamment des pratiques et des perspectives computationnelles, telles que définies

par Brennan et Resnick (2012). Pour développer ces processus, le feedback différé serait plus indiqué, même s'il risque de désengager l'élève dans la tâche.

Évaluer la charge cognitive

La mesure de la charge cognitive est d'une importance cruciale pour la recherche en éducation. Généralement, on considère qu'il existe trois types de charge cognitive : la charge intrinsèque correspond à la complexité des informations à traiter, la charge extrinsèque à la manière dont les informations sont présentées et la charge essentielle aux ressources en mémoire de travail qui sont consacrées à la tâche (voir « Chapitre 4 » de la « Partie théorique » pour plus d'informations). Dans les années 1980, lorsque la théorie de la charge cognitive a émergé, très peu d'études avaient pour objectif de mesurer directement la charge cognitive d'une situation d'apprentissage (Sweller, 2018). Généralement, celles-ci comparaient plusieurs conceptions pédagogiques et l'explication des différences observées était associée à certains facteurs liés à la charge cognitive. La charge cognitive était ainsi un facteur supposé, plutôt que mesuré (Sweller, 2018). Il a fallu attendre le début des années 1990 pour qu'une première tentative de mesure de la charge cognitive soit proposée (Paas, 1992). Cette échelle est encore aujourd'hui une des mesures les plus populaires de la charge cognitive, à la fois de par sa fiabilité et de par son utilité pratique. En effet, l'échelle de Paas (1992) comporte un item unique et constitue donc un outil particulièrement facile et rapide à utiliser. Les participants devaient y indiquer leur propre estimation de l'effort mental investi dans une tâche, sur une échelle symétrique allant de 1 (« très, très faible effort mental ») à 9 (« très, très fort effort mental »). Cependant, cet outil ne distingue pas les différents types de charge cognitive (intrinsèque, extrinsèque et essentielle).

Il existe bien sûr des alternatives à l'échelle de Paas (1992). Ayres (2006) a utilisé un outil très similaire (avec cependant une échelle à 7 points au lieu de 9). Dans leur étude (Ayres, 2006), les participants (de niveau 3e) devaient estimer la difficulté de plusieurs tâches de résolution de problèmes. Un effort avait été mis en place pour ne faire varier que la charge cognitive intrinsèque (en jouant sur l'interactivité des éléments) entre les différents problèmes. Ayres (2006) conclut que les mesures subjectives permettent d'identifier des variations de charge cognitive intrinsèque entre plusieurs tâches. Dans leur revue de littérature, Mutlu-Bayraktar et al. (2019) remarquent que les mesures subjectives de la charge cognitive dans le domaine de l'apprentissage multimédia sont les plus utilisées. De nombreuses tentatives ont été mises en place avec succès pour évaluer la charge cognitive à l'aide d'un seul item, même si

cette échelle peut comporter, suivant les études, 5, 6, 7 ou 9 points (Leppink et al., 2013). Récemment, Ouwehand et al. (2021) ont montré que le type d'échelle utilisée (numérique ou picturale) pouvait avoir une influence sur la fiabilité des résultats obtenus. Il semblerait que les échelles numériques permettent de représenter plus fidèlement la charge cognitive d'une tâche de résolution de problèmes complexes alors qu'à l'inverse, les échelles picturales sont plus adaptées pour les tâches de résolution de problèmes simples.

Leppink et al. (2013) proposent une échelle de mesures subjectives autorapportées qui distingue les trois types de charge cognitive, permettant ainsi de les évaluer séparément, et qui n'est donc plus fondée sur un seul item : la *Cognitive Load Scale* (CLS). Leppink et al. (2013) ont entrepris une validation psychométrique très complète de leur outil, qui inclue des analyses factorielles exploratoires et confirmatoires ainsi qu'une analyse en composante principale, ce qui permet de confirmer que leur échelle mesure bien trois construits différents. Cet outil est particulièrement intéressant, et a été largement utilisé depuis, car les variations qui peuvent être observées entre deux situations d'apprentissage ne concernent pas nécessairement la charge cognitive totale, mais peuvent concerner uniquement un type particulier de charge cognitive.

Des études ultérieures ont confirmé la validité de cet outil. Par exemple, Hadie et Yusoff (2016) ont démontré que la CLS présente une bonne validité de construit, une bonne fiabilité et un intérêt certain pour évaluer la charge cognitive d'étudiants en médecine dans un contexte d'apprentissage fondé sur des problèmes. De même, Andersen et Makransky (2021) ont adapté la CLS aux environnements d'apprentissage virtuels en distinguant, au sein de la charge cognitive extrinsèque, trois sous-facteurs pertinents liés à aux instructions, aux interactions et à l'environnement en lui-même. Il est cependant regrettable que l'analyse de Leppink et al. (2013) ne fasse pas état de corrélations positives très significatives entre les items relatifs à la charge cognitive essentielle. Nous pouvons donc supposer que la CLS est particulièrement adaptée pour mesurer la charge cognitive intrinsèque et extrinsèque, mais moins indiquée pour la charge essentielle.

Klepsch et al. (2017) ont également mis en œuvre deux études pour valider leur échelle, distincte de celle de Leppink et al. (2013), mais centrée sur les mêmes objectifs. Les auteurs testent différents modèles et c'est bien le modèle à trois facteurs qui semble le plus pertinent, même si, là encore, il paraît difficile de distinguer la charge cognitive essentielle de la charge cognitive intrinsèque. Le modèle à deux facteurs, qui intègre les items de la charge cognitive essentielle à la charge cognitive intrinsèque, est presque aussi pertinent que le précédent.

Klepsch et al. (2017) ont donc le mérite de proposer une échelle alternative à celle de Leppink et al. (2013), constituée d'un nombre légèrement inférieur d'items. Cependant, il est bon de rappeler que l'ensemble de ces mesures autorapportées supposent que les participants sont capables d'évaluer leur propre charge mentale pendant une tâche et que cette auto-évaluation est bien corrélée à la charge cognitive réelle (Ayres, 2006). En d'autres termes, l'utilisation de mesures autorapportées implique certaines capacités métacognitives.

Concernant les mesures subjectives toujours, Schmeck et al. (2015) se sont intéressés au délai entre l'exécution d'une tâche et la mesure de la charge cognitive associée à cette tâche. Les auteurs observent qu'une simple mesure finale est certes très économique, mais que cette façon de procéder présente un inconvénient majeur, car il est impossible de savoir à quoi correspond cette mesure. À toutes les tâches effectuées précédemment ? À la dernière ? À la plus complexe ? Ou à une combinaison de ces possibilités ? Une évaluation de la charge cognitive après chaque tâche semblerait ainsi plus adaptée. Schmeck et al. (2015) ont observé qu'une seule évaluation finale rend compte d'un niveau de charge cognitive plus élevé que la moyenne de plusieurs mesures réalisées après chaque tâche. Ceux-ci supposent que la perception de la tâche comme une série de problèmes plutôt que comme des problèmes individuels affecte la mesure de la charge cognitive. Lorsqu'une mesure unique finale est utilisée, les élèves pourraient également se focaliser plus fréquemment sur la tâche la plus complexe qu'ils ont dû réaliser, plutôt que sur l'ensemble de ces tâches.

D'autres types de mesures empiriques peuvent être mises en œuvre pour évaluer la charge cognitive : les mesures physiologiques (souvent fondées sur l'activité oculaire du participant), la performance dans une tâche, le temps de réalisation de cette tâche et la réalisation d'une tâche primaire et d'une tâche secondaire (Paas et al., 2003). Ces mesures sont considérées comme plus objectives, mais également plus chronophages et compliquées à mettre en place. Elles ne permettent pas non plus de faire la distinction entre les différents types de charge cognitive. Les techniques fondées sur une tâche primaire et une tâche secondaire consistent à mesurer les performances d'un sujet à ces deux tâches. Dans cette procédure, la performance à la tâche secondaire, généralement constituée d'activités simples nécessitant une attention soutenue, est censée refléter le niveau de charge cognitive imposé par la tâche primaire. Ce type de mesure est sensible et fiable bien que l'exécution d'une tâche secondaire présente un inconvénient majeur, car celle-ci peut interférer considérablement avec la tâche primaire, en particulier lorsqu'elle est complexe, et si les ressources cognitives sont limitées (Paas et al., 2003). Les auteurs recommandent d'associer les tâches d'auto-évaluation de l'effort

mental lié à la réalisation de la tâche avec les tâches de performance (notamment fondées sur les tâches primaire et secondaire) pour une évaluation plus fiable de la charge cognitive.

Chen et al. (2011) comparent quatre méthodes distinctes pour évaluer la charge cognitive dans une tâche d'arithmétique mentale : l'évaluation subjective de la difficulté de la tâche, le temps de réalisation de la tâche, la précision des performances et les mesures physiologiques fondées sur les activités oculaires. Les résultats montrent tout d'abord que les mesures subjectives autorapportées, le temps de réalisation et les mesures physiologiques suivent une même tendance et augmentent lorsque la tâche se complexifie, contrairement à la performance qui est moins bien corrélée à la complexité de la tâche. Ce sont par ailleurs les mesures subjectives autorapportées qui présentent la meilleure capacité à discriminer deux niveaux de complexité adjacents, là où le temps de réalisation et l'activité oculaire ne varient que très peu. Les auteurs concluent ainsi que les mesures autorapportées semblent les plus efficaces pour évaluer la charge cognitive d'une tâche. Précédemment, Ayres (2006) considérait déjà que des preuves considérables étayent l'hypothèse que les mesures subjectives sont fiables, non intrusives et plus sensibles que les mesures physiologiques. Paas et al. (1994) avaient également observé que les mesures subjectives autorapportées étaient plus efficaces que les mesures physiologiques fondées sur les variations du rythme cardiaque, pour évaluer la charge cognitive.

DeLeeuw et Mayer (2008) ont également comparé différentes méthodes utilisées pour évaluer la charge cognitive. Leur étude met en lumière le fait que certaines méthodes d'évaluation sont plus adaptées que d'autres pour mesurer certains types de charge cognitive. Les temps de réaction à une tâche secondaire sont plus sensibles à une manipulation de la charge cognitive extrinsèque. Les auto-évaluations liées à l'effort investi dans la tâche sont plus sensibles à une manipulation de la charge intrinsèque. Enfin, l'évaluation post-test de la difficulté de la tâche semble plus sensible aux processus associés à la charge cognitive essentielle. Il existe ainsi un lien entre le type d'outil de mesure utilisé et le type de charge cognitive que l'on peut évaluer (Zheng & Greenberg, 2017).

Nos études

Notre expérience 2 consiste à comparer un apprentissage sur Scratch réalisé avec et sans feedback. Le feedback du logiciel correspond pour nous à la possibilité de visualiser l'exécution du programme créé en temps réel. Pour ce faire, dans la condition sans feedback, nous avons masqué le personnage virtuel que les élèves devaient programmer. Ainsi, même en cliquant sur

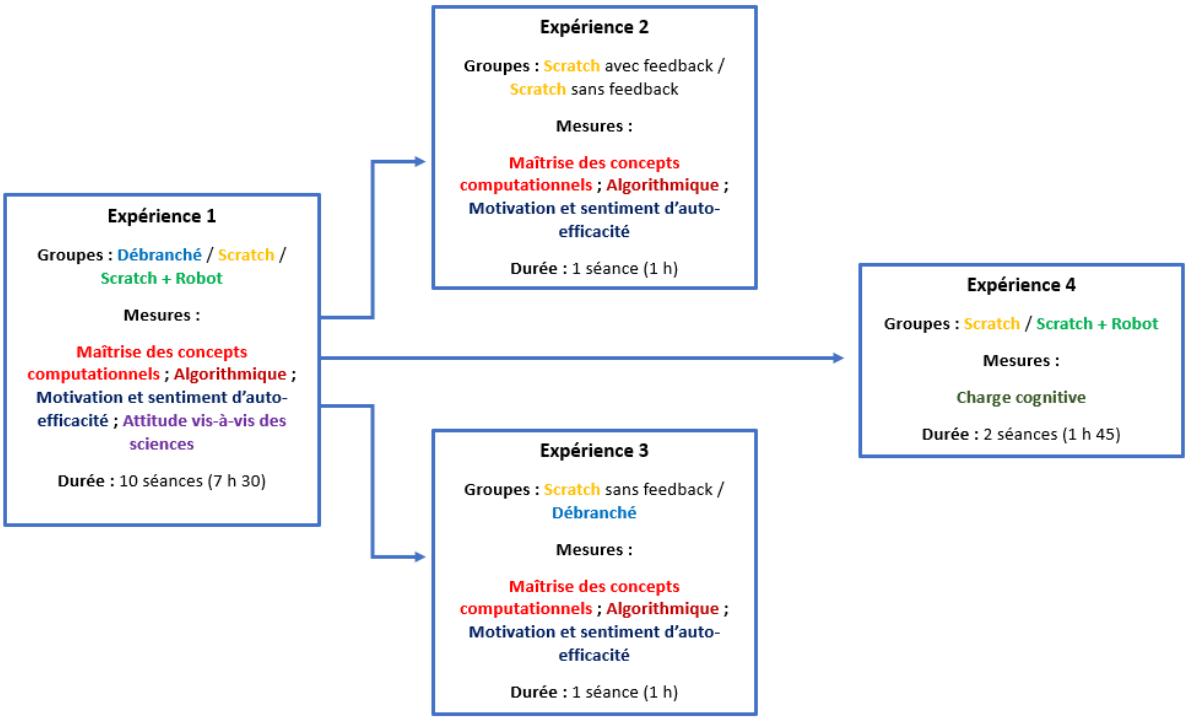
le bouton de visualisation, les déplacements de ce personnage, programmés par les élèves, étaient invisibles. Nous nous attendons à ce que les élèves bénéficiant du feedback soient plus performants que ceux qui n'en bénéficient pas.

Notre expérience 3 permet quant à elle de comparer un apprentissage réalisé sur Scratch sans feedback (toujours en masquant le personnage virtuel) avec un apprentissage débranché. Dans les deux groupes, aucun élève ne pouvait bénéficier d'un feedback du type de celui fourni par Scratch. Nous supposons ici qu'il n'y aura aucune différence significative entre nos deux groupes concernant leurs performances d'apprentissage.

Enfin, notre expérience 4 vise à comparer un apprentissage réalisé sur Scratch sans robot avec un apprentissage réalisé sur Scratch associé à un robot Thymio. Dans cette étude, ce ne sont pas les performances d'apprentissage qui seront mesurées, mais la charge cognitive des deux situations d'apprentissage. Cette charge cognitive sera évaluée à l'aide de plusieurs mesures subjectives. Nous nous attendons à ce que la charge cognitive d'une séance nécessitant l'utilisation d'un logiciel et d'un robot soit plus élevée que celle d'une séance dans laquelle le robot n'est pas utilisé. La figure 13 suivante synthétise les quatre expériences que nous avons menées dans cette thèse.

Figure 13

Synthèse des quatre expériences réalisées au cours de la thèse



3.2) Expérience 2

L'objectif de cette expérience 2 est d'observer comment les élèves apprennent la programmation sur Scratch, selon que la visualisation de l'exécution du programme est disponible (comme c'est le cas habituellement) ou non. Cette étude comparative s'est déroulée en juin 2022. Pour cette expérience (comme pour les expériences 3 et 4), l'objectif était de mettre en place un protocole plus court, faisant appel à un nombre plus réduit de participants comparé à notre première expérience, mais également plus contrôlé. Une session d'apprentissage d'environ une heure a été menée dans chaque classe participante. Les concepts computationnels introduits étaient les suivants : les notions d'algorithme, d'instruction et de boucle.

3.2.1) Méthode de l'expérience 2

Participants

Nous avons recruté 84 élèves de CM2 (44 filles), issus de quatre classes de l'Académie de Montpellier, qui ont participé à l'expérience. Afin de contrôler au maximum les « effet maître » et « effet classe », qui peuvent avoir une influence déterminante sur les performances d'apprentissage des élèves (Hattie, 2003), chaque classe participante a été scindée en deux demi-groupes. Ainsi, dans chaque classe, environ la moitié des élèves a suivi la séance sans possibilité de visualiser l'exécution du programme créé (41 élèves au total) alors que l'autre moitié pouvait bénéficier de cette visualisation (43 élèves). Les deux versions de la séance (avec ou sans visualisation possible) n'ont pas été réalisées en même temps. Lorsqu'une demi-classe effectuait la séance, l'autre demi-classe était dans une autre salle et ne pratiquait pas d'activité de programmation. Les élèves ont été répartis aléatoirement dans les deux conditions expérimentales. Nous avons donc contrôlé l' « effet classe », car des élèves issus de chacune des classes ont suivi l'une ou l'autre des deux versions de la séance. Les deux groupes étaient homogènes en termes d'équilibre entre les genres. Aucun élève n'avait suivi de cours de programmation avant l'expérience. Nous avons contrôlé la sélection des participants, car seules les classes d'écoles situées hors éducation prioritaire (REP et REP+) ont été intégrées pour notre étude, ce qui permet également d'homogénéiser l'échantillon.

Nous avons également contrôlé l' « effet maître ». En effet, un seul enseignant, référent aux usages du numérique (ERUN), ayant plusieurs années d'expérience professionnelle, a

participé à cette étude. Celui-ci a reçu la séance de programmation qu'il devait mettre en œuvre dans les quatre classes. Un entretien a été réalisé avec cet enseignant avant le début de l'expérience, afin de s'assurer qu'il comprenait et approuvait l'importance de respecter au maximum le déroulement des séances pour permettre une comparaison rigoureuse entre les différentes conditions expérimentales. L'enseignant ayant participé à cette étude a volontairement choisi de le faire, en réponse à une demande de recrutement que nous avons formulée auprès de la DRANE de l'Académie de Montpellier, qui a soutenu et fait suivre notre recherche de classes participantes.

Matériel

Dans les deux groupes, les élèves ont utilisé le logiciel de programmation éducatif Scratch 3.0 et ont appris à manipuler des instructions sous forme de blocs. Contrairement au choix réalisé dans notre expérience 1, l'ensemble des élèves ont manipulé une tablette tactile et non un ordinateur. En effet, les « classes mobiles » composées de tablettes tactiles sont aujourd'hui plus fréquemment répandues et utilisées dans les classes d'école élémentaire françaises. Ce choix est donc motivé tant par une facilité pratique, que par une volonté écologique de proposer une expérience proche de celle vécue habituellement en classe. Pour notre première expérience, des discussions préalables avec des enseignants spécialisés dans les usages du numérique, notamment en programmation et en lien avec la robotique, nous avaient permis d'établir que l'utilisation de robots Thymio était particulièrement peu adaptée sur tablettes. Le logiciel Thymio Suite qui permet de programmer un robot via un dispositif numérique est instable sur tablette, quand il n'est pas tout simplement inutilisable selon les modèles. Notre expérience 1 proposait notamment de comparer un apprentissage réalisé sur Scratch sans robot et un apprentissage sur Scratch associé à un robot Thymio. Il nous a ainsi semblé nécessaire de contraindre l'ensemble des classes « branchées » (avec ou sans robot) à utiliser des ordinateurs. Pour cette expérience 2, cette contrainte n'est plus nécessaire puisque les robots Thymio ne sont plus impliqués.

Nous avons contrôlé l'environnement expérimental dans chaque classe en veillant à ce que celles-ci disposent d'un équipement informatique suffisant (une tablette par élève) dans les deux groupes. La séance de programmation réalisée dans chaque groupe expérimental a été conçue avec l'aide d'une conseillère pédagogique de circonscription (CPC), spécialisée dans les technologies numériques et les cours de programmation à l'école primaire, et reprend en grande partie certaines activités déjà mises en place dans notre première expérience (Annexe J). Les

concepts, les compétences et les activités développés au cours de cette séance étaient identiques dans les deux groupes expérimentaux. La seule différence majeure entre les deux groupes reposait sur la possibilité (ou non) de visualiser l'exécution du programme réalisé sur Scratch.

Outils d'évaluation

Nous avons utilisé trois tests différents pour évaluer l'impact de la programmation sur Scratch, avec ou sans possibilité de visualiser l'exécution du programme créé, sur trois variables : la maîtrise des concepts computationnels, la capacité des élèves à résoudre des problèmes algorithmiques et la motivation et le sentiment d'auto-efficacité.

Pour mesurer la maîtrise des concepts computationnels et la capacité des élèves à résoudre des problèmes algorithmiques, nous avons à nouveau exploité les deux outils d'évaluation que nous avons conçus et qui ont été présentés dans le « Chapitre 1 » de la « Partie expérimentale ». Nous avons cette fois-ci utilisé la deuxième version du Test d'Algorithmique. Dans cette deuxième expérience, les élèves n'ont été initiés qu'aux concepts d'algorithme, d'instruction et de boucle. Par conséquent, seuls les items relatifs à ces trois concepts ont été conservés dans le Test de Maîtrise des Concepts Computationnels. De même, nous n'avons conservé que le premier exercice du Test d'Algorithmique version 2 (Annexe F), car la réalisation des autres exercices implique la maîtrise des concepts d'instruction conditionnelle et de variable. Le codage des résultats obtenus pour le Test de Maîtrise des Concepts Computationnels et pour le Test d'Algorithmique a suivi la même logique que celle présentée dans le « Chapitre 1 » de la « Partie expérimentale ». Le Test de Maîtrise des Concepts Computationnels était donc évalué sur 17 points et le Test d'Algorithmique sur 7 points.

Afin de mesurer la motivation et le sentiment d'auto-efficacité des élèves, nous avons à nouveau utilisé l'échelle SAL (*Students' Approaches to Learning*, Marsh, 2006), adaptée à notre contexte. Les items sont regroupés en deux catégories distinctes : le sentiment d'auto-efficacité relatif aux activités de programmation (6 items) et la motivation (5 items). Les items relatifs au sentiment d'auto-efficacité général à l'école, présents dans les évaluations de l'expérience 1, ont été supprimés ici. Nous estimons en effet qu'il est très peu probable qu'une seule séance de programmation puisse affecter cette variable. Les réponses aux questions devaient être données sur une échelle d'accord en quatre points (« *pas du tout d'accord* », « *pas vraiment d'accord* », « *plutôt d'accord* », « *totalement d'accord* »). Zéro à 3 points ont été attribués à chaque réponse pour un total de 33 points maximum. En général, les réponses « *totalement d'accord* » ont rapporté le plus de points, sauf pour l'item 4, formulé négativement, pour lequel la réponse

« *pas du tout d'accord* » a rapporté le maximum de points (3 points). Pour chaque élève, si aucune réponse n'a été donnée pour deux ou plusieurs items, les résultats totaux n'ont pas été pris en considération. En cas d'absence de réponse à un seul item, les résultats totaux de l'élève ont été exploités, mais pas les résultats relatifs à la catégorie à laquelle appartenait l'item laissé sans réponse.

En raison de l'absence de résultat significatif observé dans l'expérience 1 concernant l'attitude des élèves envers les sciences, nous avons pris la décision de ne pas mesurer cette variable dans cette expérience. En effet, il semble peu probable que des différences significatives puissent être constatées concernant cette variable après seulement une séance de programmation alors que ce n'était déjà pas le cas après 10 séances dans l'expérience 1.

Procédure

Les évaluations ont été administrées entre un et deux jours après la fin de la séance. L'objectif étant principalement de mesurer l'acquisition de certains concepts et compétences, plutôt que de tester les capacités de mémoire immédiate des élèves, il nous a semblé nécessaire d'imposer ce délai. Au cours de la séance, les élèves ont été initiés à certains concepts computationnels fondamentaux, nécessaires à la pratique de la programmation et ont appris à manipuler des instructions pour construire des algorithmes afin de programmer les mouvements virtuels d'un personnage sur un écran. Dans une des deux conditions, ces mouvements étaient directement observables via la fenêtre de visualisation présente sur Scratch, alors que dans l'autre condition, ces mouvements étaient masqués, donc invisibles. Cette expérience portait sur l'influence du feedback. Pour cette raison, nous avons contrôlé au maximum les retours qui pouvaient être donnés par les enseignants à chaque étape de la séance. La description détaillée de la séance indique aux enseignants ce qu'ils pouvaient ou ne pouvaient pas faire lorsque les élèves réalisaient les activités de programmation. Nous avons également insisté sur ce point lors des réunions de préparation avant le début de la séance. Dans chaque condition expérimentale, les élèves ont travaillé seuls.

3.2.2) Résultats de l'expérience 2

Pour cette deuxième expérience, nous avons à nouveau utilisé des modèles mixtes afin d'analyser les résultats obtenus. Chaque modèle mixte présenté plus loin propose de mesurer l'influence de deux effets fixes (le groupe expérimental et le genre) et d'un effet aléatoire (la classe) sur nos trois variables dépendantes (maîtrise des concepts computationnels, capacité à

résoudre des problèmes algorithmiques, motivation et sentiment d'auto-efficacité) et les sous-variables correspondantes. Pour cette expérience, nous nous attendons à ce que l'effet aléatoire de la classe soit nul ou très faible. En effet, nous avons contrôlé cet « effet classe » dans notre protocole expérimental puisque, dans chaque classe, les deux conditions expérimentales étaient présentes et chaque élève était réparti aléatoirement dans l'une des deux conditions. Pour cette raison, nous ne mentionnerons pas le coefficient de corrélation intraclasse de notre variable aléatoire qui est nul ou proche de zéro pour l'ensemble des modèles présentés, ce qui est conforme à nos prédictions et traduit l'absence d'influence de la variable « classe ». Ceci ne nous empêche pas cependant de réaliser des modèles mixtes, tout aussi pertinents que les analyses de variance classiques (ANOVA) pour mettre en évidence des différences significatives entre les groupes et les genres des participants. Les analyses préliminaires réalisées pour cette expérience sont identiques à celles présentées dans le cadre de l'expérience 1. Nous avons donc procédé à la suppression des valeurs aberrantes (en nous basant sur la méthode des écarts interquartiles), centré et réduit nos données, puis vérifié la normalité des résidus par le biais d'une analyse graphique, telle que décrite et préconisée par Judd et al. (2018). Les analyses que nous avons menées ont été réalisées sur le logiciel Jamovi, version 2.3.26.0, à l'aide du module Gamlj (*General Analyses for Linear Models in jamovi*).

Maîtrise des concepts computationnels

Le Tableau 19 présente les statistiques descriptives concernant les scores de maîtrise des concepts computationnels (moyennes et écarts-types), dans les groupes expérimentaux, selon le genre : le score total (sur 17 points) et les sous-scores relatifs à la maîtrise du concept d'algorithme (sur 6), d'instruction (sur 4) et de boucle (sur 7).

Tableau 19

Moyennes (et écarts-types entre parenthèses) des scores de maîtrise des concepts computationnels dans les deux groupes expérimentaux, selon le genre

Groupe	Genre	Total	Algorithme	Instruction	Boucle
Scratch, avec feedback	**Féminin**	11.3 (2.88)	4.13 (1.51)	2.58 (0.88)	4.63 (1.47)
	Masculin	11.8 (3.95)	4.28 (1.93)	2.61 (0.85)	4.89 (1.60)
Scratch, sans feedback	**Féminin**	11.3 (3.73)	3.89 (1.82)	2.42 (1.17)	5.00 (1.73)
	Masculin	11.1 (2.90)	4.43 (1.60)	2.10 (1.09)	4.62 (1.69)

L'analyse des résultats ne témoigne d'aucun effet du groupe expérimental ($F(1, 80) = 0.22$, $p = .64$) et du genre ($F(1, 80) = 0.06$, $p = .81$) sur la maîtrise des concepts computationnels. De plus, l'effet de l'interaction entre le groupe et le genre n'est pas significatif ($F(1, 78) = 0.04$, $p = .84$). Concernant les sous-scores, aucun effet significatif du groupe, du genre et de l'interaction entre les deux sur la maîtrise du concept d'algorithme n'est observé (respectivement, $F(1, 80) = 0.12$, $p = .73$; $F(1, 80) = 1.25$, $p = .27$; $F(1, 78) = 0.08$, $p = .78$), tout comme sur la maîtrise du concept d'instruction (respectivement, $F(1, 80) = 0.66$, $p = .42$; $F(1, 80) = 0.11$, $p = .75$; $F(1, 78) = 0.02$, $p = .90$) et sur la maîtrise du concept de boucle (respectivement, $F(1, 80) = 0.12$, $p = .73$; $F(1, 80) = 1.25$, $p = .27$; $F(1, 78) = 0.08$, $p = .78$).

Résolution de problèmes algorithmiques

Le Tableau 20 présente les statistiques descriptives des scores au Test d'Algorithmique dans les groupes expérimentaux selon le genre : le score total (sur 7) et les sous-scores relatifs à la capacité à produire un algorithme avec des instructions pertinentes (sur 2), à la capacité à produire un algorithme correct qui permet au personnage d'atteindre l'arrivée (sur 2), à la capacité à produire un algorithme qui respecte la consigne donnée (sur 2) et à la capacité à utiliser correctement une boucle (sur 1). Pour rappel, les élèves devaient produire deux algorithmes dans cet exercice 1 de la deuxième version du Test d'Algorithmique. Les deux algorithmes devaient permettre à un robot d'atteindre sa maison. Cependant, la consigne était un peu différente pour chaque algorithme. Le premier devait permettre au robot de passer par le chemin le plus rapide. Le deuxième devait comporter le moins d'instructions que possible (et

impliquait l'utilisation de boucles). Ainsi, la capacité à produire un algorithme qui respecte la consigne donnée correspond à la capacité d'évaluation des élèves (une des composantes de la pensée informatique), c'est-à-dire à leur capacité à considérer les différentes solutions disponibles et à évaluer laquelle correspond à ce qui est demandé.

Tableau 20

Moyennes (et écarts-types entre parenthèses) des scores relatifs à la capacité à résoudre des problèmes algorithmiques, selon le genre

Groupe	Genre	Total	Algorithme	Correct	Évaluation	Boucle
Scratch, avec feedback	**Féminin**	3.04 (1.28)	1.78 (0.36)	0.28 (0.54)	0.56 (0.65)	0.42 (0.37)
	Masculin	3.81 (1.86)	1.81 (0.35)	0.58 (0.67)	0.89 (0.83)	0.53 (0.40)
Scratch, sans feedback	**Féminin**	3.21 (1.65)	1.58 (0.61)	0.53 (0.59)	0.74 (0.56)	0.37 (0.44)
	Masculin	4.05 (1.71)	1.75 (0.51)	0.89 (0.74)	0.98 (0.59)	0.43 (0.39)

Le groupe expérimental semble n'avoir aucun impact significatif sur le score total ($F(1, 82) = 0.36$, $p = .55$) tout comme l'interaction entre le groupe et le genre ($F(1, 80) = 0.01$, $p = .93$). Cependant, un effet modéré du genre est observé sur la capacité à résoudre des problèmes algorithmiques ($F(1, 82) = 5.39$, $p < .05$, $\beta = -0.49$, 95% IC [-0.90 ; -0.08]), en faveur des garçons. Pour les quatre sous-scores recueillis, l'analyse graphique de la distribution des résidus montre que ceux-ci s'éloignent trop de la loi normale. Ceci n'est pas vraiment surprenant, compte tenu du fait que ces sous-scores sont évalués sur 1 à 2 points. Pour l'analyse de l'impact du groupe et du genre sur ces sous-scores, nous avons utilisé le test non paramétrique de Kruskal-Wallis, car celui-ci ne suppose pas que la distribution des données suit la loi normale. Nous avons également réalisé des modèles mixtes, comme si nos données respectaient la loi normale, afin de comparer les résultats avec ceux obtenus suite au test de Kruskal-Wallis. Nous avons constaté que les mêmes effets sont observés dans les deux cas de figure (modèle mixte ou test de Kruskal-Wallis). Nous rendrons compte de l'analyse de ces résultats par le biais du test de Kruskal-Wallis.

Le groupe expérimental semble n'avoir aucune influence sur la capacité à produire un algorithme avec des instructions pertinentes ($\chi^2 = 0.41$, $p = .52$), sur la capacité à produire un algorithme qui respecte la consigne ($\chi^2 = 1.91$, $p = .17$) et sur la capacité à utiliser des boucles

($\chi^2 = 0.61$, $p = .44$). Cependant, les élèves qui n'ont pas bénéficié du feedback de Scratch sont plus performants pour produire un algorithme correct qui permet au personnage d'atteindre l'arrivée ($\chi^2 = 5.85$, $p < .05$, $\varepsilon^2 = .07$). Concernant l'influence du genre, celui-ci ne semble pas avoir un impact sur la capacité à produire des algorithmes avec des instructions pertinentes ($\chi^2 = 0.80$, $p = .37$) et la capacité à utiliser des boucles ($\chi^2 = 0.85$, $p = .36$). En revanche, les garçons présentent une meilleure capacité à produire un algorithme correct qui permet au personnage d'atteindre l'arrivée ($\chi^2 = 6.67$, $p = .01$, $\varepsilon^2 = 0.08$) . De même, la capacité des garçons à produire un algorithme qui respecte la consigne donnée est plus élevée que celle des filles ($\chi^2 = 4.09$, $p < .05$, $\varepsilon^2 = 0.05$), même si cet effet est de taille plus faible.

Motivation et sentiment d'auto-efficacité

Le Tableau 21 présente les statistiques descriptives des scores au pré-test concernant la motivation et le sentiment d'auto-efficacité des élèves (moyennes et écarts-types), dans les groupes expérimentaux, selon le genre : le score total (sur 33) et les sous-scores liés au sentiment d'auto-efficacité relatif aux activités de programmation (sur 18) et à la motivation (sur 15).

Tableau 21

Moyennes (et écarts-types entre parenthèses) des scores relatifs à la motivation et au sentiment d'auto-efficacité dans les deux groupes expérimentaux, selon le genre

Groupe	Genre	Total	Auto-efficacité programmation	Motivation
Scratch, avec feedback	**Féminin**	25.7 (4.18)	14.2 (2.28)	11.5 (2.64)
	Masculin	27.4 (3.52)	14.8 (2.31)	12.6 (1.79)
Scratch, sans feedback	**Féminin**	23.4 (4.49)	12.3 (2.75)	11.1 (2.28)
	Masculin	26.5 (3.97)	14.1 (3.05)	12.4 (1.68)

L'analyse des résultats montre un effet marginal non significatif du groupe sur le score global de motivation et d'auto-efficacité ($F(1, 81) = 3.40$, $p = .07$), en faveur du groupe ayant bénéficié du feedback de Scratch. L'impact du genre est en revanche significatif et relativement fort ($F(1, 81) = 7.53$, $p < .01$, $\beta = -0.57$, 95% IC [-0.98 ; -0.16]), en faveur des garçons. Aucun effet d'interaction entre nos deux variables n'est constaté ($F(1, 79) = 0.61$, $p = .44$). Les analyses

suivantes révéleront si les effets observés concernent plutôt le sentiment d'auto-efficacité, la motivation ou les deux.

En ce qui concerne le sentiment d'auto-efficacité relatif aux activités de programmation, l'influence du groupe expérimental est significative et de taille moyenne ($F(1, 81) = 5.50$, $p < .05$, $\beta = -0.48$, 95% IC [-0.89 ; -0.08]). Les élèves du groupe ayant bénéficié du feedback de Scratch présentent un sentiment d'auto-efficacité plus élevé que ceux de l'autre groupe. De même, on observe que le sentiment d'auto-efficacité des garçons est significativement plus élevé que celui des filles et cet effet est également de taille moyenne ($F(1, 81) = 4.88$, $p < .05$, $\beta = -0.46$, 95% IC [-0.86 ; -0.05]). Aucun effet d'interaction significatif n'est constaté ($F(1, 79) = 1.06$, $p = .31$). Pour la motivation, l'effet du groupe expérimental n'est pas significatif ($F(1, 81) = 0.40$, $p = .53$), tout comme l'effet d'interaction entre le groupe et le genre ($F(1, 79) = 0.05$, $p = .83$). Cependant, un impact significatif et relativement fort du genre, en faveur des garçons, est observé sur les scores de motivation ($F(1, 81) = 6.19$, $p < .05$, $\beta = -0.53$, 95% IC [-0.94 ; -0.11]).

Liens de corrélation entre les variables dépendantes

Nous avons mesuré les trois coefficients de corrélation entre nos trois variables dépendantes (la maîtrise des concepts computationnels, la capacité à résoudre des problèmes algorithmiques et la motivation et le sentiment d'auto-efficacité). Un lien de corrélation positif, d'intensité faible à modérée, peut être observé entre la maîtrise des concepts computationnels et la capacité à résoudre des problèmes algorithmiques ($r = .31$, $p < .01$). Une autre corrélation positive significative, plus faible, est constatée entre la capacité à résoudre des problèmes algorithmiques et la motivation et le sentiment d'auto-efficacité ($r = .22$, $p < .05$). En revanche, aucun lien n'a pu être mesuré entre la maîtrise des concepts computationnels et la motivation et le sentiment d'auto-efficacité ($r = .12$, $p = .27$).

3.3) Expérience 3

L'objectif de cette expérience 3 est d'observer comment les élèves apprennent la programmation, selon si cet apprentissage est réalisé sur Scratch, sans possibilité de visualiser l'exécution du programme créé, ou en mode débranché. Cette étude comparative s'est déroulée entre septembre et décembre 2022. Une session d'apprentissage d'environ une heure a été

menée dans chaque classe participante. Les concepts computationnels introduits étaient les suivants : la notion d'algorithme, d'instruction et de boucle.

3.3.1) Méthode de l'expérience 3

Participants

Nous avons recruté 77 élèves de CM2 (35 filles), issus de quatre classes de l'Académie de Montpellier, qui ont participé à l'expérience. Chaque classe participante a été scindée en deux demi-groupes. Ainsi, dans chaque classe, environ la moitié des élèves a suivi la séance sur Scratch, sans possibilité de visualiser l'exécution du programme créé (40 élèves au total) alors que l'autre moitié travaillait en condition débranchée (37 élèves). Les deux versions de la séance (Scratch sans visualisation possible et débranchée) n'ont pas été réalisées en même temps. Lorsqu'une demi-classe effectuait la séance, l'autre demi-classe était dans une autre salle et ne pratiquait pas d'activité de programmation. Les élèves ont été répartis aléatoirement dans les deux conditions expérimentales. Nous avons donc contrôlé l' « effet classe », car des élèves issus de chacune des classes ont suivi l'une ou l'autre des deux versions de la séance. Les deux groupes étaient homogènes en termes d'équilibre entre les genres. Aucun élève n'avait suivi de cours de programmation avant l'expérience. Nous avons contrôlé la sélection des participants, car seules les classes d'écoles situées hors éducation prioritaire (REP et REP+) ont été intégrées à notre étude.

Nous avons également contrôlé l' « effet maître » puisque chaque enseignant a dû réaliser les deux versions de la séance dans sa classe. Ainsi, ce sont les mêmes enseignants qui ont œuvré dans les deux conditions expérimentales. Pour cette expérience, nous n'avons pas pu recruter un seul enseignant pour effectuer la séance dans toutes les classes. Quatre enseignants, ayant plusieurs années d'expérience professionnelle, ont donc participé. Ceux-ci ont reçu la séance de programmation qu'ils devaient mettre en œuvre dans leur classe. Une réunion a été programmée avec ces enseignants avant le début de l'expérience, afin de s'assurer qu'ils comprenaient et approuvaient l'importance de respecter au maximum le déroulement des séances pour permettre une comparaison rigoureuse entre les différentes conditions expérimentales. Les enseignants ayant participé à cette étude ont volontairement choisi de le faire, en réponse à une demande de recrutement que nous avons formulée auprès de la DRANE de l'Académie de Montpellier, qui a soutenu et fait suivre notre recherche de classes participantes.

Matériel

Dans le groupe Scratch sans visualisation, les élèves ont utilisé le logiciel de programmation éducatif Scratch 3.0 et ont appris à manipuler des instructions sous forme de blocs à l'aide de tablettes tactiles. Dans le groupe débranché, des instructions sur papier ont été utilisées. Ces instructions reproduisaient les blocs disponibles sur Scratch : le texte et les couleurs des blocs étaient les mêmes. Pendant les séances d'apprentissage, les élèves ont été invités à assembler ces "blocs de papier" de la même manière que les blocs sont assemblés sur Scratch, afin de construire leurs algorithmes.

Nous avons contrôlé l'environnement expérimental dans chaque classe en veillant à ce que celles-ci disposent d'un équipement informatique suffisant (une tablette par élève) pour le groupe Scratch sans visualisation et en fournissant nous-mêmes l'équipement nécessaire au groupe débranché (instructions au format papier). La séance de programmation réalisée dans chaque groupe expérimental a été conçue avec l'aide d'une conseillère pédagogique de circonscription (CPC), spécialisée dans les technologies numériques et les cours de programmation à l'école primaire, et reprend en grande partie certaines activités déjà mises en place dans notre première expérience (Annexe K). Les concepts, les compétences et les activités développés au cours de cette séance étaient identiques dans les deux groupes expérimentaux. La seule différence majeure entre les deux groupes reposait sur l'utilisation (ou non) du logiciel Scratch.

Outils d'évaluation

Pour cette troisième expérience, nous avons utilisé les mêmes outils d'évaluation que pour l'expérience 2, à savoir le Test de Maîtrise des Concepts Computationnels (uniquement les items relatifs à la maîtrise des concepts d'algorithme, d'instruction et de boucle), le Test d'Algorithmique version 2 (uniquement l'exercice 1, Annexe F) et l'échelle SAL de Marsh (2006) adaptée à notre contexte (uniquement les items relatifs au sentiment d'auto-efficacité en programmation et à la motivation). Pour une description plus détaillée de ces outils, nous vous renvoyons à la sous-partie consacrée aux outils d'évaluations de la partie « Méthode de l'expérience 2 » de ce « Chapitre 3 » de la « Partie expérimentale ».

Procédure

Les évaluations ont été administrées entre un et deux jours après la fin de la séance. L'objectif étant principalement de mesurer l'acquisition de certains concepts et compétences,

plutôt que de tester les capacités de mémoire immédiate des élèves, il nous a semblé nécessaire d'imposer ce délai. Au cours de la séance, les élèves ont été initiés à certains concepts computationnels fondamentaux, nécessaires à la pratique de la programmation et ont appris à manipuler des instructions pour construire des algorithmes afin de programmer les mouvements d'un objet (réel ou virtuel, selon les conditions expérimentales). Dans les deux conditions, ces mouvements n'étaient pas directement visibles et devaient donc être simulés mentalement par les élèves. Cette expérience portait sur l'influence du feedback. Pour cette raison, nous avons contrôlé au maximum les retours qui pouvaient être donnés par les enseignants à chaque étape de la séance. La description détaillée de la séance indique aux enseignants ce qu'ils pouvaient ou ne pouvaient pas faire lorsque les élèves réalisaient les activités de programmation. Nous avons également insisté sur ce point lors des réunions de préparation avant le début de la séance. Dans chaque condition expérimentale, les élèves ont travaillé seuls.

3.3.2) Résultats de l'expérience 3

Chaque modèle mixte présenté ci-après propose de mesurer l'influence de deux effets fixes (le groupe expérimental et le genre) et d'un effet aléatoire (la classe) sur nos trois variables dépendantes (maîtrise des concepts computationnels, capacité à résoudre des problèmes algorithmiques, motivation et sentiment d'auto-efficacité) et les sous-variables correspondantes. Pour les mêmes raisons que dans l'expérience 2, nous ne mentionnerons pas le coefficient de corrélation intraclasse de notre variable aléatoire qui est nul ou proche de zéro pour l'ensemble des modèles présentés, ce qui traduit la faible influence de la variable « classe ». Les analyses préliminaires réalisées pour cette expérience sont identiques à celles des deux expériences précédentes : suppression des valeurs aberrantes (méthode des écarts interquartiles), standardisation des données, analyse graphique de la normalité de la distribution des résidus. Les analyses que nous avons menées ont été réalisées sur le logiciel Jamovi, version 2.3.26.0, à l'aide du module Gamlj (*General Analyses for Linear Models in jamovi*).

Maîtrise des concepts computationnels

Le tableau 22 présente les statistiques descriptives pour les deux scores de maîtrise des concepts computationnels (moyennes et écarts-types), dans les groupes expérimentaux, selon le genre : le score total (sur 17 points) et les sous-scores relatifs à la maîtrise du concept d'algorithme (sur 6), d'instruction (sur 4) et de boucle (sur 7).

Tableau 22

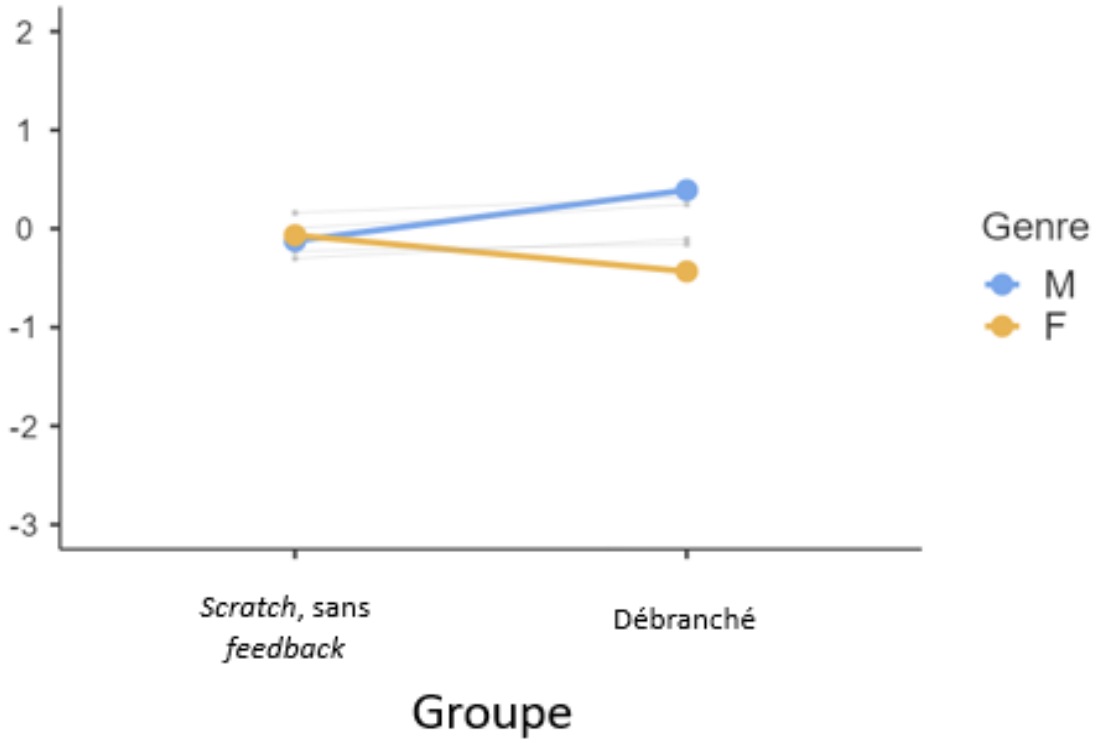
Moyennes (et écarts-types entre parenthèses) des scores de maîtrise des concepts computationnels pour les deux groupes expérimentaux, selon le genre

Groupe	Genre	Total	Algorithme	Instruction	Boucle
Scratch, sans feedback	**Féminin**	11.1 (3.45)	3.95 (1.76)	2.90 (0.97)	4.25 (1.41)
	Masculin	10.7 (3.51	3.94 (1.43)	2.72 (1.18)	4.06 (1.63)
Débranché	**Féminin**	9.93 (2.63)	3.40 (1.59)	2.27 (1.16)	4.27 (1.10)
	Masculin	11.9 (2.72)	4.77 (1.45)	2.68 (0.95)	4.45 (1.37)

L'analyse des résultats ne révèle aucun effet du groupe expérimental ($F(1, 73) = 0.00$, $p = .97$) et du genre ($F(1, 73) = 0.85$, $p = .36$) sur la maîtrise des concepts computationnels. Notons toutefois un effet marginal de l'interaction entre le groupe et le genre ($F(1, 71) = 3.19$, $p = .08$). Cet effet d'interaction est de taille assez importante en ce qui concerne la maîtrise du concept d'algorithme ($F(1, 71) = 4.24$, $p < .05$, $\beta = -0.88$, 95% IC [-1.72 ; -0.04]). Comme en témoigne la Figure 14, les garçons semblent mieux maîtriser le concept d'algorithme que les filles, uniquement dans la condition débranchée. Notons également un effet marginal du genre ($F(1, 73) = 3.17$, $p = .08$), en faveur des garçons. En revanche, l'effet du groupe expérimental n'est pas observé sur cette variable ($F(1, 73) = 0.12$, $p = .73$). De même, aucun effet du groupe, du genre ou de l'interaction entre les deux ne peut être observé sur la maîtrise du concept d'instruction (respectivement, $F(1, 73) = 2.11$, $p = .15$, $F(1, 73) = 0.08$, $p = .78$, $F(1, 71) = 1.79$, $p = .19$) et sur la maîtrise du concept de boucle (respectivement, $F(1, 73) = 0.43$, $p = .52$, $F(1, 73) = 0.00$, $p = .99$, $F(1, 71) = 0.36$, $p = .55$).

Figure 14

Moyennes des scores standardisés de maîtrise du concept d'algorithme, selon le groupe et le genre



Résolution de problèmes algorithmiques

Le Tableau 23 présente les statistiques descriptives des scores au Test d'Algorithmique dans les groupes expérimentaux selon le genre : le score total (sur 7) et les sous-scores relatifs à la capacité à produire un algorithme avec des instructions pertinentes (sur 2), à la capacité à produire un algorithme correct qui permet au personnage d'atteindre l'arrivée (sur 2), à la capacité à produire un algorithme qui respecte la consigne donnée (sur 2) et à la capacité à utiliser correctement une boucle (sur 1).

Tableau 23

Moyennes (et écarts-types entre parenthèses) des scores relatifs à la capacité à résoudre des problèmes algorithmiques, selon le genre

Groupe	Genre	Total	Algorithme	Correct	Évaluation	Boucle
Scratch, sans feedback	**Féminin**	3.73 (1.84)	1.57 (0.57)	0.80 (0.89)	0.95 (0.78)	0.40 (0.22)
	Masculin	3.81 (1.53)	1.75 (0.44)	0.68 (0.75)	0.93 (0.67)	0.45 (0.23)
Débranché	**Féminin**	4.03 (1.96)	1.63 (0.61)	0.80 (0.86)	1.13 (0.72)	0.47 (0.26)
	Masculin	3.62 (1.70)	1.59 (0.61)	0.64 (0.77)	0.98 (0.79)	0.40 (0.24)

Aucun effet significatif du groupe ($F(1, 75) = 0.03$, $p = .85$), du genre ($F(1, 75) = 0.11$, $p = .74$) ou de l'interaction entre les deux n'est observé ($F(1, 73) = 0.56$, $p = .46$). Comme dans notre expérience 2, pour les quatre sous-scores recueillis, l'analyse graphique de la distribution des résidus montre que ceux-ci s'éloignent trop de la loi normale. Nous avons donc utilisé le test non paramétrique de Kruskal-Wallis et nous avons également réalisé des modèles mixtes, comme si nos données respectaient la loi normale, afin de comparer les résultats avec ceux obtenus suite au test de Kruskal-Wallis. Dans les deux cas, aucun effet significatif n'a pu être mis en lumière. Par souci de concision, nous ne détaillerons pas ces résultats qui ne montrent aucun impact du groupe expérimental, du genre ou de l'interaction entre les deux sur ces sous-scores liés à la capacité des élèves à résoudre des problèmes algorithmiques.

Motivation et sentiment d'auto-efficacité

Le Tableau 24 présente les statistiques descriptives des scores au pré-test concernant la motivation et le sentiment d'auto-efficacité des élèves (moyennes et écarts-types), dans les groupes expérimentaux, selon le genre : le score total (sur 33) et les sous-scores liés au sentiment d'auto-efficacité relatif aux activités de programmation (sur 18) et à la motivation (sur 15).

Tableau 24

Moyennes (et écarts-types entre parenthèses) des scores relatifs à la motivation et au sentiment d'auto-efficacité dans les deux groupes expérimentaux, selon le genre

Groupe	Genre	Total	Auto-efficacité programmation	Motivation
Scratch, sans feedback	**Féminin**	20.0 (5.46)	11.5 (3.22)	8.47 (3.27)
	Masculin	23.2 (5.11)	13.2 (2.19)	10.0 (3.84)
Débranché	**Féminin**	18.3 (7.62)	10.6 (4.14)	8.00 (3.36)
	Masculin	22.8 (4.72)	12.1 (2.98)	10.8 (2.99)

L'analyse des résultats ne montre aucun effet significatif du groupe expérimental sur le score total ($F(1, 71) = 0.55$, $p = .46$) et sur les deux sous-scores liés au sentiment d'auto-efficacité en programmation ($F(1, 71) = 2.06$, $p = .16$) et à la motivation ($F(1, 71) = 0.13$, $p = .71$). De même pour l'effet d'interaction entre le groupe et le genre qui n'est pas significatif sur le score total ($F(1, 69) = 0.35$, $p = .56$), tout comme sur les deux sous-scores évoqués précédemment (respectivement, $F(1, 69) = 0.01$, $p = .92$, $F(1, 69) = 0.76$, $p = .39$). Cependant, un effet de taille assez importante est observé, favorable aux garçons, à la fois sur le score total ($F(1, 71) = 8.59$, $p < .01$, $\beta = -0.63$, 95% IC [-1.05 ; -0.21]) et sur les sous-scores liés au sentiment d'auto-efficacité en programmation ($F(1, 71) = 4.68$, $p < .05$, $\beta = -0.49$, 95% IC [-0.93 ; -0.05]) et à la motivation ($F(1, 71) = 7.32$, $p < .01$, $\beta = -0.57$, 95% IC [-0.98 ; -0.16]).

Liens de corrélation entre les variables dépendantes

Nous avons mesuré les trois coefficients de corrélation entre nos trois variables dépendantes (la maîtrise des concepts computationnels, la capacité à résoudre des problèmes algorithmiques et la motivation et le sentiment d'auto-efficacité). Un lien de corrélation positif d'intensité modérée peut être observé entre la maîtrise des concepts computationnels et la capacité à résoudre des problèmes algorithmiques ($r = .55$, $p < .001$). Une autre corrélation positive significative, plus faible, est constatée entre la capacité à résoudre des problèmes algorithmiques et la motivation et le sentiment d'auto-efficacité ($r = .24$, $p < .05$). En revanche, aucun lien n'a pu être mesuré entre la maîtrise des concepts computationnels et la motivation

et le sentiment d'auto-efficacité (r = .23, p = .06), même si le coefficient est très proche du seuil de significativité.

3.4) Expérience 4

L'objectif de cette expérience 4 est précisément de comparer les charges cognitives de deux situations d'apprentissage de la programmation : la première dans laquelle les élèves apprennent sur le logiciel Scratch et la seconde dans laquelle ceux-ci utilisent le robot pédagogique Thymio, programmable sur Scratch. Cette étude comparative s'est déroulée entre avril et juin 2023. Deux séances d'apprentissage, pour un total d'environ 1 heure et 45 minutes, ont été menées dans chaque classe participante. Les concepts computationnels introduits étaient les suivants : la notion d'algorithme, d'instruction et d'instruction conditionnelle.

3.4.1) Méthode de l'expérience 4

Participants

Nous avons recruté 116 élèves de CM2 (63 filles), issus de cinq classes de l'Académie de Montpellier, qui ont participé à l'expérience. Chaque classe participante a été scindée en deux demi-groupes. Ainsi, dans chaque classe, environ la moitié des élèves a suivi la séance sur Scratch sans robot (59 élèves), alors que l'autre moitié travaillait sur Scratch associé au robot pédagogique Thymio (58 élèves). Les deux versions de la séance (avec ou sans robot) n'ont pas été réalisées en même temps. Lorsqu'une demi-classe effectuait la séance, l'autre demi-classe était dans une autre salle et ne pratiquait pas d'activité de programmation. Les élèves ont été répartis aléatoirement dans les deux conditions expérimentales. Nous avons donc contrôlé l' « effet classe », car des élèves issus de chacune des classes ont suivi l'une ou l'autre des deux versions de la séance. Les deux groupes étaient homogènes en termes d'équilibre entre les genres. Aucun élève n'avait suivi de cours de programmation avant l'expérience. Nous avons contrôlé la sélection des participants, car seules les classes d'écoles situées hors éducation prioritaire (REP et REP+) ont été intégrées à notre étude.

Nous avons également contrôlé l' « effet maître » puisque chaque enseignant a dû réaliser les deux versions de la séance dans sa classe. Ainsi, ce sont les mêmes enseignants qui ont œuvré dans les deux conditions expérimentales. Quatre enseignants, ayant plusieurs années d'expérience professionnelle, ont participé (une enseignante a mis en place notre expérience dans sa classe et dans celle d'une de ses collègues). Ceux-ci ont reçu la séance de

programmation qu'ils devaient mettre en œuvre dans leur classe. Une réunion a été programmée avec ces enseignants avant le début de l'expérience, afin de s'assurer qu'ils comprenaient et approuvaient l'importance de respecter au maximum le déroulement des séances pour permettre une comparaison rigoureuse entre les différentes conditions expérimentales. Les enseignants ayant participé à cette étude ont volontairement choisi de le faire, en réponse à une demande de recrutement que nous avons formulée auprès de la DRANE de l'Académie de Montpellier, qui a soutenu et fait suivre notre recherche de classes participantes.

Matériel

Dans les deux groupes, les élèves ont utilisé le logiciel de programmation éducatif Scratch 3.0 et ont appris à manipuler des instructions sous forme de blocs sur des ordinateurs. Dans le groupe qui utilisait un robot, le robot Thymio II a été utilisé, comme dans notre expérience 1.

Nous avons contrôlé l'environnement expérimental dans chaque classe en veillant à ce que celles-ci disposent d'un équipement informatique suffisant (un ordinateur pour deux élèves) en fournissant nous-mêmes les robots dans le groupe correspondant (un robot pour deux élèves). Les deux séances de programmation réalisées dans chaque groupe expérimental ont été conçues avec l'aide d'une conseillère pédagogique de circonscription (CPC), spécialisée dans les technologies numériques et les cours de programmation à l'école primaire, et reprennent en grande partie certaines activités déjà mises en place dans notre première expérience (Annexe L). Les concepts, les compétences et les activités développés au cours de cette séance étaient identiques dans les deux groupes expérimentaux. La seule différence majeure entre les deux groupes reposait sur l'utilisation (ou non) du robot pédagogique Thymio.

Outils d'évaluation

La charge cognitive a été mesurée à deux reprises, au milieu de la deuxième séance de programmation suivie par les élèves, et à la fin de cette même séance. Pour ce faire, nous avons utilisé l'échelle de Klepsch et al. (2017), qui a le mérite de distinguer les trois types de charge cognitive (intrinsèque, extrinsèque et essentielle), ce qui garantit une évaluation plus précise et plus ciblée de cette variable. Cette échelle a été préférée à celle de Leppink et al. (2013), qui poursuit par ailleurs le même objectif. En effet, la CLS de Leppink et al. (2013) nécessite certaines reformulations pour être adaptée à chaque contexte d'apprentissage alors que l'échelle de Klepsch et al. (2017) peut être utilisée telle quelle dans tous les contextes. Fondée sur un

nombre d'items plus restreint, l'échelle de Klepsch et al. (2017) semble également plus adaptée aux interventions courtes, à l'image de celle que nous avons mise en place dans notre étude (deux séances, environ 1 heure et 45 minutes).

Nous avons cependant dû adapter l'échelle de Klepsch et al. (2017) pour que son utilisation soit pertinente dans notre contexte. Premièrement, l'échelle a été initialement validée auprès d'un public adulte alors que nos participants sont des enfants de niveau CM2 (âgés d'environ 9-10 ans). De plus, l'échelle a été validée en langue allemande et traduite en langue anglaise dans l'article de Klepsch et al. (2017). Nous devions donc relever le double défi d'adapter cette échelle en langue française auprès d'un public plus jeune. Pour parvenir à atteindre cet objectif, nous avons effectué une traduction / retraduction de l'échelle (à partir de la traduction anglaise) avec 3 binômes de doctorants en psychologie, bénéficiant d'une bonne maîtrise de la langue anglaise. Dans chaque binôme, un doctorant traduisait les items de l'échelle de l'anglais vers le français puis transmettait le document au deuxième doctorant, chargé d'effectuer la traduction inverse. Ceci nous a permis de réaliser une traduction de l'échelle qui soit la plus fidèle possible. Puis, nous avons fait appel à une professeure des Universités en études germaniques et en sciences du langage de l'Université de Bordeaux. Cette professeure était également spécialiste de la charge cognitive. Ainsi, nous avons pu bénéficier de ses précieux conseils et suggestions pour traduire l'échelle de Klespch et al. (2017) à partir de la langue source (allemand), dans l'optique d'affiner encore davantage notre traduction, mais surtout pour l'adapter aux élèves de niveau CM2. Par la suite, nous avons prétesté cette échelle auprès d'une dizaine d'élèves en fin d'école élémentaire afin de vérifier que la formulation des phrases et le vocabulaire employé étaient compréhensibles pour ce public. Cette adaptation est disponible en annexes (Annexe M).

Cet outil est constitué de 7 items sous forme d'échelles de Likert. Initialement, chaque échelle de Likert proposait 7 choix de réponses. Nous avons réduit ce total à seulement 5 choix de réponses, car Bouranta et al. (2009) ont remarqué qu'une échelle de Likert sur 5 points est moins confuse et suscite davantage de réponses. Or, face à la relative complexité des items proposés (malgré la simplification du vocabulaire, suite à notre travail d'adaptation), il nous a semblé important de limiter le nombre de réponses possibles. Par ailleurs, Royeen (1985) suggère d'utiliser des échelles de Likert fondées sur un nombre réduit de points lorsque le public cible est constitué d'enfants et Dawes (2008) constatent que les échelles à 5 et à 7 points produisent généralement le même score moyen lorsqu'une des deux est rééchelonnée.

En plus de l'échelle de Klepsch et al. (2017), nous avons également utilisé l'échelle de Paas (1992) exclusivement dans la deuxième évaluation, réalisée à la fin de la deuxième séance. Celle-ci permet de mesurer l'effort mental investi dans la totalité de la séance. Il nous a semblé pertinent d'utiliser également cette échelle, en raison de sa simplicité d'utilisation et de sa fiabilité, qui a été démontrée dans de nombreuses études mesurant la charge cognitive. Elle permet également d'obtenir une mesure de la charge cognitive totale qui ne soit pas une addition des trois mesures liées aux trois types de charge cognitive, évaluées par Klepsch et al. (2017). Pour l'échelle de Paas (1992), nous avons conservé l'échelle de Likert en 9 points, car celle-ci est fondée sur un item unique. Une diminution du nombre de réponses possibles aurait pu entraîner une perte d'informations qui aurait été nettement plus préjudiciable sur une échelle à item unique. L'échelle de Paas (1992), traduite en français, a été intégrée en fin d'Annexe M.

Procédure

Les évaluations ont été administrées à deux reprises lors de la deuxième séance de programmation : au milieu et à la fin de cette séance. Chaque évaluation vient clôturer une des deux activités principales qui composaient cette séance. Au cours de la séance 1, les élèves ont été initiés à certains concepts computationnels fondamentaux, nécessaires à la pratique de la programmation et ont appris à manipuler des instructions pour construire des algorithmes afin de programmer les mouvements d'un objet virtuel sur Scratch. Les élèves des deux groupes ont donc suivi une première séance d'initiation à Scratch, sans robot. Dans la deuxième séance, un des deux groupes continuait à travailler exclusivement sur Scratch alors que le second était initié à la manipulation du robot Thymio. Dans chaque condition expérimentale, les élèves ont travaillé par groupes de deux.

3.4.2) Résultats de l'expérience 4

Nous avons réalisé une analyse fondée sur des modèles mixtes pour observer l'influence de deux effets fixes (le groupe expérimental et le genre) et d'un effet aléatoire (la classe) sur nos variables dépendantes : la charge cognitive totale composée des trois types de charges cognitives (mesurée à l'aide de l'échelle de Klepsch et al., 2017) et l'effort mental subjectif (mesuré à l'aide de l'échelle de Paas, 1992). La charge cognitive totale (et les trois sous-mesures correspondantes) ont été évaluées à deux reprises, au milieu et à la fin de la deuxième séance de programmation. Pour analyser ces résultats, nous avons calculé la moyenne des deux évaluations pour chaque élève. Puis, dans un second temps, nous avons observé l'impact de nos variables indépendantes sur les résultats à la première et à la deuxième évaluation, séparément.

En effet, il semblait intéressant de séparer les deux évaluations, car d'une part, la deuxième activité, réalisée après la première évaluation, était la plus complexe à effectuer, et d'autre part, une certaine habituation des élèves à l'outil utilisé (Scratch, avec ou sans robot) pouvait potentiellement influencer nos mesures. Cependant, l'analyse des résultats ne montre aucune différence entre la première et la deuxième évaluation. Les effets observés sont les mêmes dans les deux cas. Par conséquent, les résultats décrits dans les paragraphes suivants ne concerneront que la moyenne des deux évaluations de la charge cognitive.

Les analyses préliminaires réalisées pour cette expérience sont identiques à celles des trois expériences précédentes : suppression des valeurs aberrantes (méthode des écarts interquartiles), standardisation des données, analyse graphique de la normalité de la distribution des résidus. Nos analyses ont été menées sur le logiciel Jamovi, version 2.3.26.0, à l'aide du module Gamlj (*General Analyses for Linear Models in jamovi*).

Le Tableau 25 présente les statistiques descriptives des scores liés à nos mesures de la charge cognitive pour les deux groupes expérimentaux, selon le genre : la charge cognitive totale (sur 28), la charge intrinsèque (sur 8), la charge essentielle (sur 8), la charge extrinsèque (sur 12) et l'effort mental subjectif (sur 9).

Tableau 25

Moyennes (et écarts-types entre parenthèses) des scores relatifs à la charge cognitive (échelle de Klespch et al., 2017) et à l'effort mental subjectif (échelle de Paas, 1992), selon le genre et le groupe

Groupe	Genre	Charge totale	Charge intrinsèque	Charge essentielle	Charge extrinsèque	Effort mental
Scratch, avec robot	**Féminin**	17.0 (4.36)	3.45 (1.81)	5.46 (1.33)	5.13 (2.36)	6.11 (1.59)
	Masculin	15.3 (4.86)	3.43 (1.96)	5.34 (1.86)	3.96 (1.97)	5.17 (2.36)
Scratch, sans robot	**Féminin**	15.1 (4.91)	3.39 (1.69)	5.56 (1.38)	3.94 (2.44)	4.43 (1.94)
	Masculin	12.8 (5.99)	2.82 (1.93)	5.07 (1.96)	2.89 (2.40)	4.00 (2.43)

Concernant la charge cognitive totale, le coefficient de corrélation intraclasse est de .13, ce qui signifie que 13% de la variance s'explique par des différences intersujet (entre les classes) et 87% par des différences intrasujet (au sein d'une même classe, entre les élèves). Par ailleurs, un effet significatif de taille assez importante du groupe expérimental est observé (F(1,

111) = 8.02, $p < .01$, β = 0.49, 95% IC [0.15 ; 0.83]). La charge cognitive totale est plus importante pour les élèves du groupe ayant utilisé le robot Thymio. De même, la charge cognitive est plus importante pour les filles que pour les garçons ($F(1, 111) = 4.63$, $p < .05$, β = -0.37, 95% IC [-0.71 ; -0.03]), même si la taille de cet effet est plus faible. Aucun effet d'interaction entre le groupe et le genre n'est observé ($F(1, 109) = 0.66$, $p = .42$). Il s'agit désormais de déterminer quel type de charge cognitive varie selon le groupe et le genre des élèves.

Le niveau mesuré de la charge cognitive intrinsèque et essentielle varie modérément entre les classes (respectivement, ICC = .19 et ICC = .12). Aucun effet ne peut être observé sur ces deux types de charge cognitive. L'impact du groupe ($F(1, 111) = 2.36$, $p = .13$), du genre ($F(1, 111) = 0.81$, $p = .37$) ou de l'interaction entre les deux ($F(1, 109) = 2.18$, $p = .14$) n'est pas observé sur la charge intrinsèque tout comme l'impact du groupe ($F(1, 111) = 0.51$, $p = .48$), du genre ($F(1, 111) = 0.87$, $p = .35$) et de l'interaction entre les deux ($F(1, 109) = 0.44$, $p = .51$) sur la charge essentielle. Toutefois, les scores de charge essentielle sont relativement élevés, comparativement aux deux autres types de charge, dans les deux groupes, pour les garçons comme pour les filles (voir Tableau 25, entre 5.07/8 et 5.56/8).

Cependant, en ce qui concerne la charge cognitive extrinsèque, nos résultats mettent en lumière un effet de taille assez importante du groupe expérimental ($F(1, 111) = 7.40$, $p < .01$, β = 0.48, 95% IC [0.13 ; 0.83]) et du genre ($F(1, 111) = 6.76$, $p = .01$, β = -0.46, 95% IC [-0.80 ; -0.11]). La charge cognitive extrinsèque est plus élevée dans la condition avec le robot et pour les filles. Là encore, l'effet d'interaction n'est pas significatif ($F(1, 109) = 0.04$, $p = .84$). Pour ce sous-score, le coefficient de corrélation intraclasse est assez faible (ICC = .06).

Enfin, nous observons un impact significatif et de grande taille du groupe expérimental sur l'évaluation subjective de l'effort mental ($F(1, 114) = 13.72$, $p < .001$, β = 0.65, 95% IC [0.30 ; 0.99]). L'effort mental à engager dans la tâche d'apprentissage est considéré comme nettement supérieur dans la condition avec le robot. L'effet du genre est ici marginal ($F(1, 114) = 3.14$, $p = .08$), une nouvelle fois au détriment des filles qui rapportent un effort plus élevé que celui des garçons. L'effet d'interaction entre le groupe et le genre n'est pas significatif ($F(1, 112) = 0.43$, $p = .51$). On peut noter également que le coefficient de corrélation intraclasse est nul pour cette variable.

3.5) Discussion des expériences 2, 3 et 4

Les trois expériences que nous avons menées dans ce chapitre avaient notamment pour but d'expliquer les résultats observés dans notre première expérience. Nous avions notamment mis en évidence des différences significatives concernant les performances d'apprentissage d'élèves de CM2, novices en programmation, en fonction des outils utilisés pour enseigner cette discipline. Les élèves les plus performants avaient utilisé Scratch, un logiciel de programmation éducatif basé sur un langage visuel par blocs. Nous supposions que ceux-ci bénéficiaient d'un meilleur apprentissage que les élèves en condition débranchée, car les outils numériques permettent de visualiser en temps réel l'exécution d'un programme créé, ce qui pourrait favoriser une meilleure compréhension des notions fondamentales en programmation ainsi qu'une meilleure acquisition des compétences liées à la maîtrise de la pensée informatique. Par ailleurs, nous avions constaté que l'association de Scratch avec un robot pédagogique Thymio n'entraînait pas une amélioration des performances d'apprentissage, mais produisait au contraire un effet délétère. L'hypothèse que nous défendions pour expliquer ce constat reposait sur la possibilité d'un accroissement significatif de la charge cognitive d'une situation d'apprentissage, lorsqu'un robot est associé à un logiciel éducatif.

Les expériences 2 et 3 proposaient alors de comparer d'une part, un apprentissage réalisé sur Scratch avec ou sans possibilité de visualiser l'exécution du programme créé, et d'autre part, un apprentissage réalisé sur Scratch sans ce feedback avec un apprentissage exclusivement débranché, durant une séance d'environ une heure. Nous avons également mesuré l'influence de ces conditions expérimentales sur la motivation et le sentiment d'auto-efficacité des élèves. L'expérience 4 proposait quant à elle de mesurer la charge cognitive de deux situations d'apprentissage, selon qu'un robot pédagogique était utilisé ou non, durant deux séances, pour un total d'environ une heure et 45 minutes. L'ensemble des séances ont été conçues pour être similaires entre nos différentes conditions expérimentales, c'est-à-dire centrées sur les mêmes notions à apprendre, les mêmes compétences à développer et les mêmes types de tâches à réaliser. Dans ces trois expériences, l'impact du genre sur nos variables dépendantes était également étudié.

Rôle du feedback permis par les outils numériques

Dans notre deuxième expérience, nous nous attendions à ce que les élèves bénéficiant du feedback de Scratch présentent de meilleures performances d'apprentissage que ceux pour

lesquels ce feedback n'était pas disponible, car nous considérions que ce feedback pouvait être bénéfique aux apprentissages. *A contrario*, la troisième expérience n'était pas censée mettre en lumière une différence significative entre ceux qui ont été initiés à la programmation avec Scratch sans bénéficier du feedback du logiciel et ceux qui l'ont été par le biais d'activités débranchées. En d'autres termes, notre expérience 2 avait pour objectif de mettre en lumière l'apport avantageux de la possibilité de visualiser l'exécution du programme créé, alors que notre expérience 3 pouvait défendre l'idée que cette particularité des outils numériques était la seule plus-value des logiciels éducatifs visuels par blocs.

Nos hypothèses ne sont cependant pas ici confirmées par les faits. Les performances d'apprentissage des élèves ne diffèrent pas significativement, selon s'ils utilisent Scratch avec ou sans le feedback du logiciel, que ce soit en ce qui concerne la maîtrise des concepts computationnels ou la capacité à résoudre des problèmes algorithmiques. Nous avons pu cependant observer un effet intéressant concernant l'une des sous-variables évaluées par notre test d'Algorithmique (version 2), la capacité à produire un algorithme correct, c'est-à-dire un algorithme qui permet au personnage d'atteindre sa maison (l'une des consignes données aux élèves). Contrairement à nos prédictions, cet effet est favorable aux élèves n'ayant pas bénéficié du feedback de Scratch. Il est cependant possible d'expliquer (logiquement) cet effet. Le test d'Algorithmique ayant été réalisé sur papier (sans feedback), on peut supposer que les élèves n'ayant pas bénéficié de ce feedback durant la séance soient plus habitués à cette situation. Durant la séance, en situation « sans feedback », les élèves auraient potentiellement appris à être plus attentifs pour concevoir leurs programmes en comparaison avec ceux qui bénéficiaient du feedback, qui n'avaient pas besoin de faire cet effort. En outre, pour cette même variable, les garçons sont plus performants que les filles, tout comme sur la capacité à évaluer différentes solutions possibles à un même problème pour choisir celle qui était demandée par la consigne (à savoir, produire un algorithme qui passe par le chemin le plus court, puis produire un algorithme avec le moins d'instructions que possible).

Comment pouvons-nous alors expliquer que la présence ou l'absence du feedback du logiciel ne semble pas influencer les performances d'apprentissage des élèves ? Premièrement, il est tout à fait possible que notre hypothèse ne soit tout simplement pas correcte. La possibilité de visualiser l'exécution du programme créé par les élèves ne constitue peut-être pas la plus-value principale des logiciels éducatifs de programmation basés sur des langages visuels par blocs. Hattie et Timperley (2007) mettaient en garde contre les feedback*s* trop fréquents (tels que le feedback illimité de Scratch) qui pourraient empêcher l'apprenant de formuler des

hypothèses ou de déployer des stratégies efficaces et le conduire à multiplier les essais et erreurs, sans investissement cognitif de sa part. De même, Chevalier et al. (2022) considéraient que le feedback immédiat ne produit pas nécessairement un développement des processus cognitifs liés à la pensée informatique et notamment aux pratiques computationnelles. Ces processus étant essentiels pour résoudre des problèmes algorithmiques, ceci pourrait expliquer l'absence de différences concernant cette variable.

Nous pouvons également supposer que d'autres caractéristiques des logiciels tels que Scratch seraient bénéfiques aux apprentissages. Weintrop (2019) a déjà mis en évidence certains atouts des langages visuels par blocs que nous n'avons pas étudiés dans nos expériences. En effet, lorsque deux blocs ne peuvent pas logiquement être assemblés, l'environnement de programmation empêche l'utilisateur de le faire. Les zones de saisie de texte dans les blocs sont également préremplies ce qui facilite la compréhension du type d'informations à donner (nombre, texte…) et prévient toute confusion en ne permettant pas de renseigner un texte lorsque c'est un nombre qui est attendu, par exemple. Ces deux caractéristiques n'ont pas pu être reproduites dans les « blocs papiers » que nous avons utilisés dans la condition débranchée de notre expérience 1. Il est donc possible que cet aspect des logiciels de programmation représente un atout plus significatif pour les apprentissages que la possibilité de visualiser l'exécution d'un programme.

Cependant, nous ne pouvons exclure la possibilité que certaines faiblesses dans notre protocole expérimental aient nuit à l'apparition d'un effet, pourtant bien réel, du feedback sur les apprentissages. Rappelons que notre expérience 2 était réalisée sur une seule séance alors que notre expérience 1 s'étalait sur 10 séances (pré-test et post-test compris). Par conséquent, seuls les concepts computationnels les plus simples (algorithme, instruction, boucle) ont été abordés, nécessairement de manière assez superficielle au vu du peu de temps disponible. Il est probable que le contenu de la séance, centrée sur des concepts plus simples, n'ait pas permis de mettre en lumière des effets notables. Les résultats de notre expérience 1 montrent que l'impact des groupes expérimentaux est nettement plus marqué sur les concepts les plus complexes (boucle, condition, variable). Nous aurions donc dû retrouver un effet a minima sur la maîtrise du concept de boucle et la capacité à les utiliser pour résoudre des problèmes algorithmiques, mais, en une seule séance, ce concept a seulement pu être esquissé.

De même, aucune différence relative aux performances d'apprentissage des élèves, selon qu'ils utilisent Scratch sans feedback ou qu'ils apprennent en condition débranchée, n'a

pu être observée. Ce résultat était attendu et pouvait nous permettre d'exclure l'idée que les logiciels tels que Scratch bénéficient d'autres atouts que le type particulier de feedback qui nous intéresse. Cependant, au vu des résultats non significatifs de l'expérience 2, les conclusions de notre expérience 3 doivent nécessairement être tirées avec prudence. Notons tout de même un effet d'interaction intéressant, mais difficilement explicable en l'état, puisque les garçons semblent mieux maîtriser le concept d'algorithme que les filles, uniquement en condition débranchée, dans cette expérience 3.

Concernant le vécu subjectif des élèves, le seul effet significatif observable de nos groupes expérimentaux concerne un sentiment d'auto-efficacité relatif aux activités de programmation plus élevé dans le groupe ayant bénéficié du feedback de Scratch, en comparaison avec les élèves n'ayant pas eu la possibilité de visualiser l'exécution de leurs programmes. La présence du feedback du logiciel permettrait donc aux élèves de se sentir plus confiants dans leur capacité à maîtriser la programmation, car ceux-ci ont probablement pu, grâce à la visualisation de l'exécution du programme, avoir une confirmation directe et objective de leur capacité à accomplir les tâches qui leur étaient demandées. L'utilisation ou non du logiciel Scratch (que ce soit avec ou sans feedback) ne produit pas d'effet significatif sur la motivation dans nos expériences 2 et 3. Rappelons cependant que ce résultat n'entre pas en contradiction avec notre expérience 1, les effets motivationnels positifs observés dans celle-ci concernaient principalement la condition branchée avec robot, absente de ces deux nouvelles expériences.

Il semblerait également que l'effet du genre soit relativement fort et constant. Pour nos expériences 2 et 3, les garçons sont plus motivés et plus confiants en leur capacité à réussir en programmation que les filles. Dans notre expérience 1, ce même effet était observable sur la motivation, mais pas sur le sentiment d'auto-efficacité. Le plus grand sentiment d'auto-efficacité des garçons dans les disciplines liées à l'informatique a déjà été observé plusieurs fois par le passé (Cheng, 2019 ; Webb et al., 2017) et ne constitue pas un résultat surprenant, en particulier compte tenu du fait que notre test a été soumis aux élèves à l'issue de leur première (et unique) séance de programmation, ce qui n'est certainement pas suffisant pour éliminer les stéréotypes liés au genre. Ceux-ci sont au contraire confirmés par nos résultats expérimentaux, et ce, alors que nos participants sont encore jeunes et viennent à peine d'être initiés à cette discipline.

Enfin, conformément à ce que nous supposions dans le « Chapitre 5 » de notre « Partie théorique » (« Problématique et hypothèses »), les variables cognitives relatives aux performances d'apprentissage (maîtrise des concepts computationnels et capacité à résoudre des problèmes algorithmiques) sont significativement reliées par un lien de corrélation positif d'intensité modérée à assez forte (selon l'expérience concernée). *A contrario*, la motivation et le sentiment d'auto-efficacité ne sont que faiblement (ou pas du tout) corrélés aux deux variables précédemment citées.

Charge cognitive d'une situation d'apprentissage, avec ou sans robot

Notre quatrième expérience devait, selon nos prédictions, mettre en exergue un niveau de charge cognitive plus élevé dans la condition où le logiciel Scratch est associé à un robot Thymio, en comparaison avec la condition pour laquelle les élèves ont appris à programmer sur ce même logiciel, sans utiliser de robot. Notre hypothèse semble ici confirmée par les données empiriques que nous avons recueillies. En effet, l'analyse des résultats témoigne d'une charge cognitive globale, mesurée à l'aide de l'échelle subjective de Klepsch et al. (2017), supérieure dans la condition pour laquelle les élèves utilisaient simultanément le logiciel Scratch et le robot Thymio, en comparaison avec la condition où le robot Thymio n'était pas impliqué.

Une analyse plus précise de l'impact de nos groupes expérimentaux sur les mesures des différents types de charge cognitive révèle que cette augmentation, en condition branchée avec robot, concerne uniquement la charge cognitive extrinsèque, aucune différence significative n'étant observée en relation avec la charge intrinsèque ou essentielle. Ce résultat est assez logique, selon les définitions associées aux trois types de charge cognitive, proposées par Sweller et al. (1998). La charge intrinsèque correspond à la complexité des informations à apprendre. Ici, les concepts, les compétences et les tâches à mettre en œuvre sont très similaires entre nos deux conditions, avec ou sans robot. Il n'est donc pas surprenant que les variations observées ne concernent pas cette charge intrinsèque. En revanche, la charge extrinsèque correspond à toutes les informations à prendre en compte qui sont inutiles aux apprentissages. L'augmentation de la charge extrinsèque, dans ce contexte précis, est probablement causée par la nécessité de maîtriser certains aspects techniques essentiels, propres à la manipulation du robot associé au logiciel, bien que ceux-ci ne constituent pas véritablement un des objectifs d'apprentissage.

Notons tout de même que la mesure du niveau de charge essentielle est très élevée, quels que soient le groupe et le genre des élèves. Ce constat peut déjà s'expliquer par la difficulté de

mesurer cette charge essentielle, en comparaison avec les deux autres types de charge, comme en témoignent les chercheurs à l'origine des principaux outils de mesure des types de charge cognitive (Klepsch et al., 2017 ; Leppink et al., 2013). Par ailleurs, notre adaptation / traduction de l'échelle de Klespch et al. (2017) pourrait également avoir été mise en défaut, les items relatifs à la charge essentielle étant les plus difficiles à adapter pour des élèves de CM2. Par exemple, notre adaptation / traduction de l'item 3 était la suivante : « *J'ai fait un effort pour comprendre toute l'activité, mais aussi les liens entre chaque étape de l'activité* ». La plupart des élèves a présenté des scores élevés à cet item et il est probable qu'il ait été compris en quelque sorte comme une mesure de l'effort et du sérieux des élèves dans la tâche. Une simple modification en « *J'ai **dû** faire des efforts…* » aurait potentiellement gommé cette confusion.

De même, l'effort mental subjectif investi dans la tâche d'apprentissage, mesuré grâce à l'item unique de Paas (1992), est significativement plus élevé dans la condition avec robot, par rapport à la condition sans robot. Cette échelle est supposée représenter une mesure globale de la charge cognitive, mais est souvent plus associée à la charge intrinsèque ou à la charge extrinsèque, selon la manière dont celle-ci est interprétée par le répondant (Leppink et al., 2013). Ici, au vu des résultats décrits précédemment, l'effort investi dans la tâche, jugé supérieur lorsque les robots sont utilisés, pourrait logiquement être imputé au niveau de charge cognitive extrinsèque. Les conclusions de notre expérience 4 incitent à penser que l'effet délétère des robots pédagogiques sur les performances d'apprentissages des élèves, observé dans notre expérience 1, peut au moins partiellement s'expliquer par une charge cognitive trop élevée des dispositifs associant robot et logiciel visuel de programmation pour des élèves de CM2, novices en la matière.

En outre, alors que ceci ne faisait pas partie de nos prédictions, nous pouvons également constater un effet significatif du genre sur la charge cognitive globale et la charge extrinsèque (l'effet étant marginal concernant l'effort mental subjectif). Les filles rapportent des scores plus élevés que les garçons pour ces variables. Or, les filles ne semblaient pas moins performantes que les garçons dans les deux conditions branchées (avec ou sans robot) de notre expérience 1. Au contraire, une observation attentive des statistiques descriptives présentées dans les Tableau 12 et Tableau 13 (« Partie expérimentale », « Chapitre 2 », « Résultats de l'expérience 1 ») témoigne de performances légèrement supérieures pour les filles, tous groupes confondus, mais en particulier en ce qui concerne le groupe branché avec robot. Ceci peut potentiellement s'expliquer par la présence d'autres facteurs, non étudiés dans nos recherches, qui ont permis aux filles de compenser une charge cognitive subjectivement jugée plus élevée, leur permettant

de hisser leur niveau de performance à un niveau équivalent à celui des garçons, malgré cet obstacle. Au vu de la dimension subjective des outils de mesures utilisés, une différence entre filles et garçons dans la manière d'estimer leur propre niveau de charge cognitive, spécifique à l'âge de nos participants, pourrait également être envisagée. Cependant, il semble que l'hypothèse la plus plausible soit celle d'une réduction progressive de l'écart entre les genres concernant l'évaluation de la charge cognitive au fil des séances. En effet, notre expérience 1 comportait 10 séances contre seulement deux séances dans notre expérience 4 (une seule impliquant des robots). Les garçons seraient donc initialement plus à l'aise dans les activités de programmation, mais cette différence entre les genres disparaîtrait rapidement.

SYNTHÈSE

- Les trois expériences décrites dans ce chapitre avaient notamment pour but d'**expliquer les résultats obtenus** dans l'expérience précédente.
- Les expériences 2 et 3 comparaient plusieurs conditions d'apprentissage de la programmation : **débranchée**, **Scratch sans feedback** et **Scratch avec feedback**.
- Ces deux expériences **n'ont pas permis de mettre en évidence l'apport positif du feedback** des logiciels de programmation basés sur des langages visuels par blocs.
- **L'expérience 4 mesurait la charge cognitive** associée à l'utilisation du logiciel Scratch, **avec ou sans robot pédagogique**.
- Les résultats de cette dernière expérience révèlent que la **charge cognitive extrinsèque des élèves est plus élevée lorsqu'un robot pédagogique est associé** au logiciel de programmation Scratch.

DISCUSSION GÉNÉRALE

Cette thèse avait pour objectif de comparer expérimentalement différents outils qui sont utilisés pour enseigner la programmation et la pensée informatique en contexte scolaire, en particulier à la fin de l'école élémentaire, auprès d'élèves novices. L'impact des activités débranchées, des logiciels éducatifs de programmation fondés sur des langages visuels par blocs et des robots pédagogiques a été étudié. L'état actuel de la littérature met en lumière plusieurs arguments théoriques pertinents pour justifier ou encourager l'utilisation de certains outils ou de certaines activités, censés favoriser la capacité des élèves à apprendre cette discipline. Toutefois, ces arguments sont rarement étayés par les faits et peu d'études empiriques proposent de comparer directement plusieurs façons d'enseigner la programmation et la pensée informatique à l'école. Cette thèse contribue à accroître les connaissances dans ce domaine, en proposant une approche écologique centrée sur des expérimentations réalisées en classe, tout en veillant au contrôle des conditions de passation, par la transmission de compétences et l'enseignement de concepts qui sont communs aux groupes expérimentaux, ceux-ci ne se distinguant que par les outils employés pour les enseigner.

Ce travail de recherche nous a menés à une réflexion portant sur les outils d'évaluation de la pensée informatique et de la programmation qui a abouti à la conception de deux nouveaux instruments de mesure satisfaisants, pour lesquels nous avons apporté des indices de fiabilité et de validité. Notre première expérience nous a permis d'observer des différences significatives entre nos trois conditions expérimentales, concernant la capacité des élèves à maîtriser les concepts computationnels mis en jeu, et à résoudre des problèmes algorithmiques, ainsi que sur leur niveau de motivation et leur sentiment d'auto-efficacité. Nous avons par la suite entrepris de mener trois expériences supplémentaires visant à expliquer les résultats observés dans notre première expérience, au sujet des performances d'apprentissage des élèves. L'influence du feedback des outils numériques, permettant de visualiser l'exécution des programmes créés par les élèves, et de la charge cognitive associée aux différentes situations d'apprentissage a ainsi pu être plus précisément explorée. Les principales contributions, limites et perspectives ouvertes par cette thèse seront discutées ci-après, et feront office de premiers éléments de réponse que nous pouvons apporter à la question centrale suivante :

Q : Quel outil (activités débranchées, logiciels visuels de programmation, logiciels visuels associés à des robots pédagogiques) est le plus efficace pour enseigner la programmation et la pensée informatique ?

Évaluer l'apprentissage de la programmation et de la pensée informatique

Afin de mesurer des différences significatives concernant l'apprentissage de la programmation et de la pensée informatique, selon les outils utilisés pour enseigner cette discipline, nous avions besoin de nous interroger sur la manière d'évaluer ces apprentissages, ce qui nous a conduits à poser la question suivante.

Q1 : Comment évaluer l'apprentissage de la programmation et de la pensée informatique à l'école ?

Au vu de l'absence de consensus sur la définition même de la pensée informatique (Brennan & Resnick, 2012 ; Mohaghegh & McCauley, 2016 ; Román-González et al., 2017 ; Moreno-León et al., 2018 ; Tang et al., 2020), il semble illusoire de vouloir apporter une réponse simple et définitive à cette question. C'est la raison pour laquelle nous n'avions formulé aucune hypothèse pour y répondre. La manière d'évaluer ces apprentissages dépend fortement du contexte dans lequel ils sont réalisés et des objectifs qui sont établis. Il est donc finalement assez logique que la revue des outils de mesure existants n'ait pas permis de trouver une solution satisfaisante que nous aurions pu facilement exploiter dans nos études expérimentales. Face à la multitude des cadres conceptuels qui existent pour tenter de définir et d'expliquer ce qu'est la pensée informatique, nous avons pris la décision de considérer cette pensée informatique comme un ensemble de compétences mises en jeu lors d'une tâche de résolution de problème algorithmique, qui s'appuie sur des connaissances théoriques nécessaires pour réaliser cette tâche (la maîtrise des concepts computationnels) et des connaissances pratiques et stratégiques issues de la tâche.

Les deux outils que nous avons conçus, le Test de Maîtrise des Concepts Computationnels et le Test d'Algorithmique, sont complémentaires et permettent d'évaluer à la fois la connaissance et la compréhension des notions fondamentales en programmation, ainsi que la capacité des élèves à mettre en pratique ces notions pour résoudre des problèmes algorithmiques. Ces outils se distinguent des échelles autoévaluatives plus subjectives, des outils d'analyses automatiques du code produit par les élèves et des évaluations réalisées en temps réel pendant le processus d'apprentissage. Ils sont ainsi à ranger parmi les différents

outils qui tentent d'évaluer la maîtrise de la pensée informatique par le biais d'activités de résolution de problèmes, avec cependant quelques différences majeures. Nos outils évaluent la pensée informatique en lien direct avec les activités de programmation, ce qui n'est pas toujours le cas dans ce domaine. En effet, un grand nombre d'instruments de mesure de la pensée informatique, validés et publiés, est essentiellement constitué de problèmes indépendants de toute activité de programmation. Pour nous, ces outils ne peuvent être utilisés comme moyens d'évaluer les apprentissages en programmation, sauf pour mesurer l'effet de transfert, c'est-à-dire la capacité à appliquer ce qui a été appris dans un autre contexte, plus ou moins différent. Cet effet de transfert demeure incertain (Denning et al., 2017) et n'est pas l'objet de cette thèse.

Nos tests ont également été validés en français, auprès d'élèves issus du système éducatif français et, même s'ils sont passés consécutivement, ne devraient pas excéder une heure de passation, ce qui les rend donc particulièrement adaptés pour être mobilisés dans les classes françaises, dans lesquelles le temps d'apprentissage de l'informatique n'excède généralement pas une heure consécutive. De plus, notre approche a consisté à valider et à utiliser nos tests quasi exclusivement en contexte écologique, ce qui est un atout précieux au vu des conclusions d'une récente revue de littérature sur l'évaluation de la pensée informatique (Han, 2023), mettant en évidence une majorité d'articles dans le domaine ayant mesuré les compétences en pensée informatique dans un contexte de recherche, plutôt que dans un contexte pédagogique authentique. Notre contribution majeure consiste également en la tentative de proposer un test pour lequel les élèves devaient produire eux-mêmes un algorithme comme solution à un problème, ce qui n'a été envisagé qu'une seule fois à notre connaissance (Chen et al., 2017), et ce, d'une manière qui exigeait la maîtrise d'une syntaxe spécifique. Nous avons également veillé à ce que nos deux tests couvrent une large partie des objectifs éducatifs, comme défini par la taxonomie révisée de Bloom (Krathwohl, 2002), en évaluant principalement les connaissances factuelles, conceptuelles et procédurales des élèves, ainsi que leur capacité à se remémorer, comprendre, analyser et appliquer les concepts computationnels dans l'optique de créer et d'évaluer des algorithmes permettant de résoudre des problèmes.

Il est important de bien préciser que la conception de ces deux outils ne constituait pas pour nous une fin en soi. Une thèse entière pourrait être consacrée à l'évaluation des connaissances et compétences dans ce domaine, tant définir et mesurer les multiples aspects de la pensée informatique semble une tâche ardue et complexe. Nos outils doivent plutôt être considérés comme un moyen d'évaluer deux aspects complémentaires du développement de la pensée informatique et de l'apprentissage de la programmation, que nous souhaitions étudier

dans nos expériences, et que l'on pourrait grossièrement synthétiser de la manière suivante : les connaissances notionnelles d'une part, et les connaissances procédurales et compétences de résolution de problèmes d'autre part. En ce sens, notre objectif a été atteint. En effet, au vu du processus de validation psychométrique que nous avons entrepris et des résultats que nous avons décrits, il est raisonnable d'estimer que ces deux instruments de mesure évaluent bien les connaissances et compétences à l'œuvre dans l'apprentissage de la programmation et de la pensée informatique.

Il serait toutefois prématuré de penser que le travail de conception et de validation de nos deux tests est terminé. Nous avons mis en lumière certaines faiblesses qu'il serait souhaitable de corriger. En termes de généralisation tout d'abord, nous avons constaté que la première version du Test d'Algorithmique était probablement trop spécifique et pas assez généralisable à l'ensemble de la population des élèves du primaire ou du secondaire. Cette version était certainement trop longue et trop complexe, ce qui a entraîné un taux d'abandon important et une moyenne générale faible. Lorsque nous l'avons utilisé dans notre expérience 1, la moyenne était nettement plus élevée et le taux d'abandon nettement plus faible, ce qui appuie notre argument d'une trop grande spécificité de cette première version. La deuxième version montre une amélioration sur ce plan, même si le dernier exercice présentait des problèmes similaires, en raison de la nécessité d'utiliser des variables, inconnues de la majorité de notre échantillon.

De même, hormis pour le Test de Maîtrise des Concepts Computationnels pour lequel plus de 600 élèves ont été recrutés, les deux versions du Test d'Algorithmique ont chacune été soumises à un peu moins de 200 élèves. Un échantillon plus large aurait pu confirmer davantage nos observations, même si, au vu des analyses que nous avons réalisées (fiabilité interne et coefficient de corrélation), un plus grand nombre de participants n'était pas forcément indispensable. En effet, l'ensemble des corrélations que nous avons mesurées étaient déjà significatives à $p < .001$. Enfin, nous avons pu observer qu'il était particulièrement difficile d'isoler certaines compétences associées à la pensée informatique pour les évaluer indépendamment d'autres compétences ou connaissances. La définition de la pensée informatique, selon Tchounikine (2017), rend palpables les liens qu'entretiennent ces compétences : « savoir décomposer un problème en sous-problèmes plus simples ; savoir réfléchir aux tâches à accomplir pour résoudre un problème en termes d'étapes et d'actions ; savoir décrire les problèmes et les solutions à différents niveaux d'abstraction, ce qui permet d'identifier des similitudes entre problèmes et, par la suite, de pouvoir réutiliser des éléments

de solution » (p. 11). Abstraction, reconnaissance de formes ou de *patterns*, réutilisation et généralisation semblent presque indissociables. On peut également considérer que la pensée algorithmique et la décomposition sont deux compétences proches, car produire un algorithme implique déjà de décomposer un mouvement ou une fonction en une série d'étapes simples. Çakiroğlu et Çevik (2022) estiment par ailleurs que la capacité d'abstraction est impossible à dissocier de la décomposition, de la généralisation ou de la reconnaissance de formes et que ces dernières ne sont que des sous-composantes de l'abstraction, qui constituerait ainsi le cœur de la pensée informatique, ce que Grover et Pea (2013) et Wing (2008) avaient déjà avancé précédemment.

En outre, il serait pertinent d'essayer d'optimiser nos outils par des analyses complémentaires. Premièrement, nous pourrions à l'avenir proposer une plus grande réserve d'items que nous soumettrions à un jury d'experts, chargé d'évaluer la pertinence et la difficulté des items, ainsi que la longueur globale du test, selon le niveau des élèves. Ceci nous permettrait d'affiner la sélection de nos items afin de ne conserver que les plus appropriés pour nos objectifs d'évaluation. En termes de fiabilité, les mesures « test-retest », qui évaluent la cohérence entre deux mesures distinctes, séparées par un intervalle de temps (Polit, 2014), permettraient de s'assurer qu'en l'absence d'intervention relative à l'apprentissage de la programmation et de la pensée informatique, les performances des élèves restent approximativement constantes dans le temps.

La validité de nos tests pourrait également être mesurée par des liens de corrélation éventuels avec d'autres tests, évaluant des connaissances et compétences proches des nôtres (par exemple, le *Computational Thinking Test*, évoqué précédemment). Enfin, les analyses factorielles exploratoires ou confirmatoires sont généralement indiquées pour confirmer la validité d'un outil de mesure. Elles permettent de révéler les facteurs latents (e.g., les différents concepts qui sont évalués), à travers le regroupement de certains facteurs observés (e.g., les items d'un questionnaire), afin d'établir le nombre et la nature des facteurs qui expliquent la variation et la covariation d'un ensemble de données (Brown & Moore, 2012). Celles-ci semblent toutefois difficiles à appliquer à nos deux tests, au vu des chevauchements qui existent entre les facteurs que nous mesurons et ce, y compris pour le Test de Maîtrise des Concepts Computationnels. En effet, pour évaluer la maîtrise d'un concept, nous proposons une assez grande variété de types d'items différents (catégorisation, application, reconnaissance, textes à trous) ce qui crée ainsi un chevauchement entre le concept dont la maîtrise est évaluée et le processus d'évaluation de ce concept.

L'impact des outils d'enseignement sur les performances d'apprentissage

Notre premier objectif concernant cette thèse était de pouvoir déterminer si l'un des trois outils qui nous intéressent permettait un meilleur apprentissage des connaissances et compétences en programmation, comme en témoigne notre deuxième question de recherche.

Q2 : Quel outil a le plus d'impact sur la transmission des notions et compétences fondamentales en lien avec l'apprentissage de la programmation et de la pensée informatique ?

Proposer une hypothèse était assez périlleux, compte tenu du manque de données empiriques comparatives entre nos trois outils. Nous avions cependant observé que les arguments théoriques, avancés pour défendre l'utilisation de tel ou tel outil, reposaient communément sur la diminution de la charge cognitive (y compris pour les langages visuels, plus accessibles, donc potentiellement moins susceptibles de surcharger la mémoire de travail) et sur le feedback permis par les outils numériques. La capacité d'un outil à susciter des interactions motrices et des mouvements du corps, supposés favorables aux apprentissages, selon une perspective incarnée et située de la cognition, était également mise en avant, bien que moins fréquemment envisagée. Ceci nous a conduits à formuler la question de recherche suivante.

Q6 : Quel est l'impact du feedback renvoyé par un outil numérique et de la charge cognitive propre à chaque outil sur les apprentissages des élèves ?

En fin de compte, nous avions estimé que les langages visuels de programmation par blocs, sans association avec un robot, constituaient le meilleur compromis, permettant de bénéficier à la fois du feedback de visualisation de l'exécution d'un programme, tout en étant suffisamment accessibles pour ne pas compromettre la capacité des élèves à apprendre par une charge cognitive extrinsèque trop élevée. Cette hypothèse pouvait être critiquée, puisque les logiciels tels que Scratch étaient également les outils les moins susceptibles de favoriser la motricité des élèves pendant la tâche d'apprentissage, malgré l'apport potentiellement favorable des interactions sensori-motrices. L'hypothèse que nous avons privilégiée était donc la suivante.

H1 : Les langages visuels de programmation, non associés à des robots pédagogiques, constituent l'outil le plus efficace pour transmettre les notions et compétences fondamentales en lien avec la programmation et la pensée informatique.

Notre première expérience met en évidence la supériorité des élèves en condition branchée sans robot, en comparaison avec les élèves en condition débranchée ou en condition branchée avec robot, que ce soit en ce qui concerne la maîtrise des concepts computationnels ou la capacité à résoudre des problèmes algorithmiques. Ces résultats sont donc conformes à notre hypothèse H1, et également à des recherches précédentes qui affirmaient l'apport bénéfique des langages visuels par blocs pour apprendre la programmation, par rapport aux autres outils (Namli & Aybek, 2022 ; Sun, Guo et Zhou, 2022 ; Zhang et al., 2021). Notre étude s'oppose toutefois à d'autres résultats, ayant mis en lumière une absence de différences entre l'apprentissage branché ou débranché de la programmation (Hermans & Aivaloglou, 2017 ; Romero et al., 2018), voire une supériorité des activités débranchées (Wohl et al., 2015), ou une absence de différences entre apprentissages avec ou sans robot (Merkouris & Chorianopoulos, 2015), voire une supériorité des approches impliquant les robots pédagogiques (Bellegarde et al., 2019 ; Kert et al., 2020 ; Saritepeci & Durak, 2017). Cependant, comme nous l'avons évoqué précédemment, les quelques études comparatives réalisées sur ce sujet concernent des élèves de niveau très différents, avec des tailles d'échantillon assez faibles, et poursuivent des objectifs de recherche pas tout à fait similaires, ce qui explique certainement au moins en partie le présence de ces résultats contradictoires.

Par ailleurs, nous avons pu remarquer que cet effet était d'autant plus fort que les concepts abordés étaient complexes (en particulier, les concepts de boucle, d'instruction conditionnelle et surtout, de variable). Ceci nous a logiquement poussés à explorer la dernière hypothèse que nous avions posée.

H6 : La charge cognitive associée à un outil et la présence ou l'absence d'un feedback issu des outils numériques ont un impact significatif sur les performances d'apprentissage.

Pour mettre à l'épreuve cette hypothèse, nous avons conçu trois expériences supplémentaires. Les expériences 2 et 3 comparaient un apprentissage réalisé sur Scratch de manière classique (en bénéficiant de la possibilité de visualiser l'exécution des programmes créés), un apprentissage sur Scratch sans ce feedback et un apprentissage débranché. Là encore, les compétences et concepts mis en jeu étaient les mêmes dans toutes les conditions. Contrairement à nos prédictions, aucune différence significative n'a pu être établie concernant les performances d'apprentissage des élèves. Devons-nous alors en conclure que l'influence du feedback des outils numériques est nulle ? La réponse que nous apportons à cette question est

plus nuancée. La durée des expériences 2 et 3 n'excédait pas une heure, contre près de huit heures pour l'expérience 1. Cette diminution de la durée des interventions n'était pas simplement dictée par des impératifs pratiques, mais également par des enjeux éthiques. En effet, nous pouvons considérer que proposer une séquence d'enseignement relativement longue dans laquelle une partie des élèves apprennent à programmer sur Scratch, sans bénéficier de l'ensemble des atouts du logiciel, constitue une démarche critiquable et potentiellement non éthique, en particulier au vu de l'hypothèse que nous défendons et qui suppose un apport positif de ce feedback. Notons d'ailleurs que les professeurs concernés par ces expériences ont été fortement incités à poursuivre les activités de programmation sur Scratch après notre expérience, pendant au moins une séance supplémentaire, afin de permettre à tous les élèves d'utiliser le logiciel dans les conditions idéales.

Néanmoins, comme nous l'avons déjà évoqué dans la discussion du troisième chapitre expérimental, la faible durée de notre intervention n'a pas permis d'aborder certains concepts complexes (instructions conditionnelles et variables) avec les élèves, alors que ces concepts étaient ceux pour lesquels les différences étaient les plus marquées entre nos trois conditions de l'expérience 1. De même, la version utilisée du test d'Algorithmique n'était pas la même entre l'expérience 1 (version 1) et les expériences 2 et 3 (version 2). Enfin, il est important de remarquer que le matériel utilisé différait également entre ces trois expériences (ordinateur pour l'expérience 1, tablettes pour les expériences 2 et 3), alors que la recherche semble avoir établi des différences suivant si l'enseignement est médié par l'un ou l'autre de ces supports. Papadakis et al. (2016) rapportent par exemple que de nombreuses études font état d'un apport bénéfique des tablettes pour l'apprentissage des plus jeunes (notamment à l'école maternelle), car celles-ci ne nécessitent qu'une formation minimale et aucun matériel supplémentaire, et accroissent la motivation, l'engagement et l'enthousiasme des élèves. L'absence d'étude comparative entre tablettes et ordinateurs, dans le contexte spécifique de l'apprentissage de la programmation, ne nous permet pas de déterminer quel rôle ce changement de support a pu jouer. Toujours est-il que les variations opérées sur l'ensemble de ces facteurs ont pu avoir un impact suffisant pour faire disparaître nos effets.

Il est également possible que notre hypothèse à ce sujet soit tout simplement invalidée et que le feedback de Scratch ne produise aucun effet favorable aux apprentissages. D'autres caractéristiques de ce type de logiciels pourraient également être en jeu. Nous avons notamment déjà abordé, pour les logiciels visuels de programmation, l'impact potentiel de l'impossibilité technique d'associer deux blocs de manière illogique ou de remplir une zone de saisie de texte

par une information non conforme à ce bloc. D'autres apports des langages de programmation visuels pourraient également renforcer la compréhension ou la maîtrise des processus à l'œuvre dans les activités de programmation, comme la possibilité de parcourir les différentes instructions disponibles ou la possibilité de les assembler par un mécanisme intuitif de *drag & drop*, qui permet de bien saisir comment ces instructions sont imbriquées les unes aux autres (Lin & Weintrop, 2021).

Notre expérience 4 visait à comparer deux situations d'apprentissage sur Scratch, avec ou sans robot, pour évaluer la charge cognitive et l'effort mental des élèves dans ces deux situations. Les résultats obtenus montrent que la charge cognitive extrinsèque et l'effort mental, évalués subjectivement par les élèves eux-mêmes, étaient nettement plus élevés dans la condition où les élèves utilisaient le robot pédagogique Thymio. Ce résultat incite à penser que les performances inférieures des élèves en condition branchée avec robot, comparées à celles des élèves en condition branchée sans robot, observées dans l'expérience 1, pourraient s'expliquer par cet accroissement de la charge cognitive. Pour Daniela (2021), la conception des technologies d'apprentissage numériques devrait permettre d'acquérir de nouvelles connaissances, sans créer de charge cognitive superflue, notamment extrinsèque, c'est-à-dire liée à des informations sans importance ou à une profusion de petits détails. Or, le succès des robots pédagogiques s'appuie principalement sur leurs nombreuses fonctionnalités (lumières, sons, déplacements, détection d'obstacles…). Celles-ci participent sans aucun doute à leur attrait et à l'intérêt des élèves pour les robots, mais peuvent également être considérées comme distrayantes, car la variété des fonctionnalités proposées ne garantit pas une meilleure compréhension et maîtrise des concepts ou compétences en programmation. Garner (2002) considérait d'ailleurs que la programmation est une discipline pour laquelle la charge cognitive intrinsèque est très élevée, ce qui implique donc que la charge extrinsèque doit être aussi faible que possible, pour augmenter les performances d'apprentissage.

Cependant, la prudence nous impose de rappeler que, dans notre expérience 4, une seule séance était consacrée à la manipulation de robots, pour le groupe correspondant. Il est possible que la charge cognitive, liée à l'utilisation des robots, diminuent au fil du temps pour atteindre le même niveau que ceux des élèves qui apprennent sur Scratch sans robot, même si nous ne privilégions pas cette hypothèse. Enfin, l'expérience 4 ne mesure pas les performances d'apprentissage (maîtrise des concepts computationnels, capacité à résoudre des problèmes algorithmiques) et ne permet donc pas de vérifier que cet accroissement de la charge cognitive extrinsèque et de l'effort mental est associé à de moins bonnes performances des élèves dans la

condition impliquant le robot. La raison principale qui nous a dissuadés de réaliser des mesures de performances est l'absence de résultat significatif constaté dans les expériences 2 et 3, concernant ces variables. Il nous semble désormais justifier d'avancer que les différences de performances ne peuvent probablement être observées que lorsque la durée d'intervention est suffisante et que les concepts et compétences travaillées en classes sont suffisamment complexes et développés, ce qui ne pouvait être le cas dans cette expérience 4. En conclusion, nous pouvons affirmer que notre hypothèse H6 n'est que partiellement confirmée par les faits et mériterait d'être étayée par de nouvelles données empiriques.

L'impact des outils d'enseignement sur le vécu subjectif des élèves

Notre étude des effets des outils d'apprentissage de la programmation et de la pensée informatique ne se limitait pas à une analyse de l'impact sur les performances. D'autres variables, liées au vécu subjectif des élèves, telles que la motivation, le sentiment d'auto-efficacité ou l'attitude vis-à-vis des sciences, ont également été explorées, afin de tenter de répondre à notre troisième question de recherche.

Q3 : Quel outil a le plus d'impact sur le vécu subjectif des élèves et les variables conatives en lien avec l'apprentissage de la programmation et de la pensée informatique ?

De par leur caractère attrayant et accessible, les outils numériques nous semblaient les plus à même de promouvoir la motivation et le sentiment d'auto-efficacité des élèves. L'impact des activités de programmation sur l'attitude vis-à-vis des sciences étant peu étudié, nous n'avons pu que supposer qu'elles pouvaient avoir un effet positif sur cette variable, sans nous risquer à prédire lequel de nos outils serait susceptible d'optimiser cet effet. Nous avons ainsi proposé les hypothèses suivantes.

H2 : Les approches branchées de la programmation sont les plus efficaces pour maintenir ou susciter une plus grande motivation et un plus grand sentiment d'auto-efficacité chez les élèves.

H3 : Les activités de programmation peuvent influencer positivement l'attitude des élèves vis-à-vis des disciplines scientifiques.

Notre expérience 1 a mis en lumière le rôle positif des robots pédagogiques à ce sujet. D'une manière générale, il semble que les outils numériques accroissent la motivation des élèves, même si la différence observée entre la condition débranchée et la condition branchée sans robot n'est pas significative. En revanche, les élèves ayant utilisé le robot présentent des

scores de motivation nettement plus élevés que les autres et cet impact est observable également sur les deux items, présents uniquement dans le post-test, qui traduisent un certain niveau d'engagement dans les activités de programmation, y compris une volonté de poursuivre ces activités en dehors de l'école. Ces résultats sont conformes à ceux obtenus par Merkouris et al. (2017) qui ont notamment comparé deux apprentissages branchés de la programmation, avec ou sans robot. Les auteurs ont utilisé un protocole à groupes appariés, afin que chaque élève (âgé de 12 ou 13 ans) expérimente avec ou sans le robot, même si, malheureusement, la taille de l'échantillon (36 élèves au total), comme la durée d'intervention (45 minutes par condition), est faible. Leurs conclusions sont proches des nôtres et d'une autre étude qu'ils avaient réalisée précédemment (Merkouris & Chorianopoulos, 2015). Suite à l'intervention de robotique, les élèves se sont montrés plus engagés, faisant part de plus d'émotions positives, et déclarant être plus susceptibles de s'adonner à l'informatique sur leur temps libre. En outre, ceux-ci pensaient avoir plus de compétences en programmation, conformément à ce qu'avaient observé Kert et al. (2020), même si Namli et Aybek (2022) n'avaient montré aucune différence relative au sentiment d'auto-efficacité des élèves selon qu'ils apprenaient la programmation de manière branchée ou débranchée.

Nous pouvons tout de même nous interroger sur l'effet de nouveauté (Cheung & Slavin, 2013) qui n'a pas nécessairement totalement disparu à l'issue des 10 séances de programmation que nous proposions. À ce titre, l'étude de Pedaste et Altin (2020) peut nourrir notre réflexion et nous inciter à une certaine prudence. Les auteurs ont réalisé une intervention de robotique pédagogique centrée sur l'apprentissage de notions de physique au niveau lycée en Estonie, un pays dans lequel la robotique est particulièrement bien implantée, y compris dans les établissements qui ont participé à cette étude. La comparaison avec un groupe contrôle, dans lequel ces notions ont été enseignées sans robot de manière plus traditionnelle, ne montre aucun impact de la robotique à la fois sur les connaissances et compétences enseignées et sur la motivation et le sentiment d'auto-efficacité. L'absence d'effet de nouveauté, dans ces classes rompues à la pratique de la robotique, est l'une des explications avancées par les chercheurs pour expliquer cette absence de différences significatives.

Dans notre expérience 1, un effet plus faible est aussi observé sur le sentiment d'auto-efficacité général à l'école qui semble s'accroître grâce à la pratique de la robotique. En revanche, l'effet sur l'attitude envers les sciences n'est pas observé, sans que nous puissions déterminer si cet effet est réellement inexistant ou s'il nécessite plus de temps pour être visible. Ce constat nous a conduits à abandonner l'étude de cette variable dans les trois expériences qui

ont suivi. Précédemment, Leonard et al. (2016) n'avait pas non plus réussi à montrer un impact positif d'activités de robotique sur l'attitude d'élèves en fin de primaire ou au collège vis-à-vis des disciplines scientifiques STEM et des carrières envisageables dans ces disciplines, tout comme Kandlhofer et Steinbauer (2016) et Zhang et al. (2021). Bers et al. (2014) et Ioannou et Makridou (2018) affirmaient pourtant que la robotique aurait un impact bénéfique sur l'attitude envers les sciences.

Les expériences 2 et 3 ne signalent aucune différence entre nos conditions, concernant la motivation et le sentiment d'auto-efficacité des élèves. Ces deux expériences n'impliquaient pas de robot pédagogique. Ces résultats sont donc plutôt cohérents, car le vrai apport motivationnel des outils numériques semble reposer sur l'utilisation des robots, comme en témoigne notre expérience 1. Notons toutefois que le sentiment d'auto-efficacité relatif aux activités de programmation est plus important chez les élèves ayant bénéficié du feedback de Scratch dans notre expérience 2. Une lecture attentive des résultats de l'expérience 1 montrait déjà un score plus élevé pour cette variable dans les deux conditions branchées, en comparaison avec la condition débranchée, bien que cette différence n'était pas significative. On peut donc supposer que la possibilité de visualiser en temps réel l'exécution d'un programme permettrait d'accroître la confiance de l'élève en sa capacité à réaliser les tâches qui lui sont demandées, en contexte d'apprentissage de la programmation, mais que cet effet tendrait à diminuer (voire à disparaître) avec le temps ou la complexification des concepts et compétences à développer.

Notons également que nous n'avons pas mesuré la motivation ou le sentiment d'auto-efficacité dans notre expérience 4. En effet, quels que soient les résultats que nous aurions observés, il aurait été très difficile d'en tirer des conclusions pertinentes. Lorsque des dispositifs aussi attrayants et novateurs que des robots pédagogiques sont utilisés en classe pendant une ou deux séances, l'effet de nouveauté devrait être assez important et un accroissement de la motivation ou du sentiment d'auto-efficacité ne pourrait pas être véritablement imputé aux caractéristiques pédagogiques de cet outil. De même, la pratique de la robotique (et en particulier du robot pédagogique Thymio que nous avons utilisé) nécessite certaines compétences techniques qui pourraient au contraire susciter une démotivation et un sentiment d'auto-efficacité plus faible. Il ne nous semblait donc pas particulièrement pertinent d'explorer ces variables sur une durée d'intervention aussi courte, au vu des facteurs qui pouvaient influencer les résultats et rendre périlleuse et incertaine toute interprétation de ceux-ci.

Nous pouvons conclure, en nous appuyant sur les données que nous avons recueillies, que notre hypothèse H2 doit être révisée, puisque même si les logiciels visuels de programmation pourraient potentiellement accroître la motivation des élèves, cet effet n'est significatif que lorsque ceux-ci sont associés à un robot pédagogique. Par ailleurs, nous ne pouvons rejeter l'hypothèse nulle face à notre hypothèse H3, car aucun effet n'a pu être observé concernant l'attitude envers les sciences.

L'impact des outils d'enseignement sur la réduction de l'écart entre les genres

Dans le domaine de l'informatique en général, et de la programmation en particulier, il semble indispensable d'évaluer les différences entre les genres, au vu des inégalités maintes fois observées dans la littérature, que celles-ci concernent les performances des élèves, ou leur niveau de motivation, d'engagement et d'auto-efficacité. En effet, il est communément admis que les garçons présentent un intérêt, une motivation et un engagement plus important dans les activités de programmation en général (Baser, 2013 ; Bourgeois et al., 2019 ; Gökçe & Yenmez, 2022 ; Kong & Lai, 2021 ; Korkmaz & Bai, 2019 ; Rubio et al., 2015 ; Weintrop & Wilensky, 2019), en raison de stéréotypes associés aux femmes et à l'informatique (Armoni et al., 2015 ; Bers et al., 2022 ; Ching et al., 2018), même si Román-González et al. (2017) considèrent que cette différence ne s'observe pas vraiment à l'école primaire. Chaque outil que nous avons mis à contribution dans cette thèse présente des caractéristiques susceptibles de diminuer ou accroître ces inégalités, sans que les données empiriques n'aient pu, à ce jour, permettre de conclure quant à cette épineuse préoccupation. Notre question, relative à ce problème, a été formulée de la manière suivante.

Q4 : Quel outil est le plus susceptible de réduire l'écart entre les genres dans l'apprentissage de la programmation et de la pensée informatique ?

Par leur capacité à enseigner la programmation sans utiliser d'outils numériques, de manière non menaçante vis-à-vis des stéréotypes liés à l'informatique, les activités débranchées nous ont semblé plus indiquées pour réduire cet écart entre filles et garçons, conformément à ce qu'avancent Sun et al. (2021), comme en témoigne l'hypothèse que nous avons formulée.

H4 : Les activités débranchées permettent de réduire ou d'éliminer l'écart entre les genres, concernant l'apprentissage de la programmation et de la pensée informatique, en comparaison avec les autres outils.

Les résultats les plus marqués que nous avons obtenus dans nos expériences concernent l'impact du genre sur la motivation et le sentiment d'auto-efficacité des élèves. Ces résultats sont également observables sur les items 19 et 20, présents uniquement dans le post-test de motivation et d'auto-efficacité, qui traduisent un certain engagement de l'élève dans les activités de programmation. Nous avons ainsi pu observer que le niveau de motivation des garçons était significativement plus élevé que celui des filles dans notre expérience 1. Cependant, une analyse plus approfondie des données a révélé que cette différence n'existait que pour la condition branchée sans robot. Pour la condition débranchée et la condition branchée avec robot, les différences ne sont pas significatives. Ce résultat suggère que les robots ne sont pas nécessairement plus favorables aux garçons, comme le supposaient Su et Yang (2023), ou aux filles, comme l'affirmaient Gunes et Kucuk (2022), mais plutôt que la robotique ne semble pas creuser les inégalités entre les genres, conformément à ce que rapportaient Merkouris et Chorianopoulos (2015).

Il serait donc pertinent de s'interroger sur une éventuelle préférence des filles pour les environnements d'apprentissage qui ne sont pas exclusivement virtuels, comme c'est le cas lorsque Scratch est utilisé sans association à un robot pédagogique. Tikva et Tambouris (2019) supposaient au contraire que Scratch permettait de tenir compte des différences entre les genres, et serait donc plus enclin à les réduire, ce que nous n'avons pas observé. Il faut cependant rappeler que les séances que nous avons conçues avaient vocation à être similaires entre nos conditions expérimentales, et pas à être différenciées entre filles et garçons pour tenter de réduire cet écart. L'étude de Wu et Su (2021) révèle que les filles sont plus performantes que les garçons sur une variable d'apprentissage (*pattern recognition*) et les garçons sur une autre (*algorithm design*), uniquement lorsque l'apprentissage de la programmation est réalisé sans utiliser d'environnement virtuel de programmation. Ceci confirme que l'environnement de travail constitue un facteur modulant différemment l'expérience d'apprentissage des élèves selon leur genre.

De plus, dans nos expériences 2 et 3, plus courtes, cet effet du genre sur la motivation est également observé, toujours en faveur des garçons. Notons que pour ces deux expériences, trois des quatre groupes expérimentaux utilisaient le logiciel Scratch sans robot, ce qui explique probablement pourquoi nous retrouvons cet effet. Celui-ci s'accompagne d'un autre effet, concernant cette fois-ci le sentiment d'auto-efficacité relatif aux activités de programmation, supérieur chez les garçons, que nous n'avions pas observé dans notre expérience 1. En réalité, ces différences étaient déjà présentes, en particulier dans les deux groupes branchés, mais non

significatives. Au vu de l'ensemble des résultats obtenus dans nos trois expériences, nous pouvons supposer qu'il existe une supériorité initiale des garçons en termes de motivation et de sentiment d'auto-efficacité, mais que celle-ci tendrait à s'estomper assez rapidement. Elle ne persisterait dans le temps qu'en ce qui concerne la motivation et l'engagement, dans la condition branchée sans robot.

En ce qui concerne les performances d'apprentissage, les données obtenues suite à la validation de notre Test de Maîtrise des Concepts Computationnels et des deux versions du Test d'Algorithmique, relatives à plusieurs centaines d'élèves du CM1 à la 3e, ne font état d'aucune différence significative entre filles et garçons. En revanche, les filles se sont révélées légèrement supérieures dans l'expérience 1, en particulier dans les deux conditions branchées. Cet effet, de faible taille, ne concerne que la maîtrise des concepts computationnels et n'est significatif que sur la maîtrise du concept de boucle. *A contrario*, les garçons se sont montrés plus performants concernant certains aspects de la capacité à résoudre des problèmes algorithmiques dans notre expérience 2 (capacité à produire un algorithme fournissant une solution correcte au problème posé et capacité à évaluer plusieurs solutions pour sélectionner celle qui est en adéquation avec la consigne). Certaines différences relatives à la capacité à résoudre des problèmes algorithmiques, également en faveur des garçons, sont aussi observées pour l'expérience 3, mais ne sont pas significatives.

Nous pourrions ainsi supposer que les garçons sont plus performants en début d'apprentissage de la programmation, lorsque les connaissances et compétences mises en jeu sont peu complexes, avant que cette supériorité ne s'estompe, voire ne s'inverse en faveur des filles. Ceci serait d'ailleurs en accord avec d'autres études qui montraient que les filles ont besoin de plus de temps pour atteindre le même niveau de performance que les garçons (Atmazidou & Demetriadis, 2016), mais que celles-ci progressaient davantage quand les concepts abordés étaient plus complexes, par rapport aux garçons (Delal & Oner, 2020). Cette interprétation ferait toutefois abstraction de la faible taille des effets observés, et de leur spécificité à certaines compétences ou connaissances très ciblées. Au vu de la littérature, particulièrement contradictoire concernant les différences de performances entre filles et garçons en programmation (Cheng, 2019 ; del Olmo-Muñoz et al., 2020 ; Jiang & Li, 2020 ; Li et al., 2021 ; Noh & Lee, 2020 ; Relkin et al., 2020 ; Sun et al., 2022b ; Wang et al., 2022), il serait plus prudent d'affirmer qu'aucune conclusion ne peut être tirée de ces résultats et que, si l'impact du genre sur les performances est avéré, celui-ci est probablement limité.

En outre, nos résultats ont également mis en exergue deux effets d'interaction assez surprenants et difficiles à expliquer. Les garçons semblent plus performants que les filles pour maîtriser les concepts d'algorithme (expérience 3) et d'instruction (expérience 1), uniquement dans la condition débranchée, alors que les filles semblent plus performantes pour maîtriser le concept d'instruction (expérience 1), uniquement dans la condition branchée sans robot. Ces effets ne s'observent pas uniformément dans nos trois expériences et ne concernent que les concepts les plus simples et basiques à appréhender (algorithme et instruction). En l'état, il serait hasardeux de proposer une interprétation à ce constat sans étude plus approfondie. Tout juste pouvons-nous supposer qu'une approche mixte, incluant d'abord des activités débranchées puis des activités branchées, serait plus favorable aux garçons, car les concepts d'algorithme et d'instruction sont généralement les premiers à être enseignés, et les garçons semblent mieux les maîtriser lorsqu'ils sont présentés en condition débranchée.

Enfin, un dernier effet assez paradoxal devrait attirer notre attention. Alors que nous avons mesuré le niveau de charge cognitive dans notre expérience 4, afin d'expliquer la supériorité des performances en condition branchée sans robot, par rapport à la condition branchée avec robot, les filles rapportent une charge cognitive extrinsèque plus élevée que les garçons. Or, celles-ci sont tout aussi performantes que les garçons, dans la condition branchée avec robot de notre expérience 1. Il est possible que l'augmentation de la charge cognitive, associée à une situation d'apprentissage impliquant un logiciel et un robot, ne soit pas le seul facteur qui explique ces différences de performances, mais nous privilégions l'hypothèse selon laquelle les différences d'évaluation de la charge cognitive entre filles et garçons ne s'observent qu'au début de l'apprentissage de la programmation avec ou sans robot, avant que cette variable ne retrouve des niveaux similaires entre les genres par la suite. Ce résultat n'est toutefois pas si surprenant, Chen et al. (2021) avaient également montré que, dans un contexte d'apprentissage par la robotique avec des élèves de primaire, les garçons présentaient un niveau de charge cognitive significativement plus faible que celui des filles. Dans leur étude, les garçons étaient légèrement plus performants que les filles, mais cette différence n'était pas significative.

Nos résultats confirment ceux obtenus par Merkouris et al. (2017), qui observent des performances légèrement meilleures chez les filles, bien que celles-ci se montrent moins confiantes en leurs capacités que les garçons, ainsi qu'un engagement émotionnel identique dans la robotique entre les genres. Globalement, nous pouvons estimer que notre hypothèse H4 semble valide. La condition branchée sans robot est indéniablement celle qui produit le plus d'inégalités. L'ajout d'un robot pédagogique réduit ces inégalités, notamment en ce qui

concerne le niveau de motivation. Mais les différences de performance en faveur des filles sont principalement observées dans les deux conditions branchées. Au final, la condition débranchée est la seule qui ne fait état d'aucune différence entre les genres. Malheureusement, celle-ci est également la moins susceptible de favoriser les apprentissages et de stimuler la motivation des élèves, d'après nos résultats expérimentaux.

Liens entre performances et vécu subjectif des élèves

Plusieurs études ont contribué à éclairer l'impact de tel ou tel outil d'enseignement sur une grande quantité de variables reliées à l'apprentissage de la programmation et au développement de la pensée informatique, mais également à certains facteurs conatifs représentatifs du vécu subjectif des élèves pendant l'expérience d'apprentissage, en particulier la motivation, l'engagement ou le sentiment d'auto-efficacité. Nous pouvons toutefois regretter que peu de recherches se soient consacrées à des comparaisons expérimentales entre ces différents outils, ou aient tenté de contribuer à la compréhension des liens potentiels entre les variables cognitives (plutôt rattachées aux performances) et les variables conatives. Ceci constituait également un objectif de cette thèse que nous avons formulé par la question de recherche suivante.

Q5 : Peut-on mesurer des corrélations significatives entre les variables (cognitives ou conatives) que nous avons choisi d'étudier ?

En raison du paradoxe préférences / performances (Amadieu & Tricot, 2014), il nous semblait plausible qu'il existe une distinction entre d'une part des variables cognitives bien corrélées, et d'autre part des variables conatives qui entretiendraient également des liens forts, sans que ces variables cognitives et conatives ne présentent des liens de corrélation entre elles. Cette prédiction s'exprime dans l'hypothèse suivante que nous avions avancée.

H5 : Il existe un lien de corrélation relativement fort entre les variables cognitives d'une part, et les variables conatives d'autre part, mais le lien de corrélation entre les variables cognitives et conatives est faible ou inexistant.

Notre expérience 1 nous a permis de confirmer que les variables conatives (motivation et sentiment d'auto-efficacité, attitude envers les sciences) sont effectivement unies par un lien de corrélation assez important et significatif. Ce résultat est aisément explicable, car un élève ayant une attitude positive vis-à-vis des sciences devrait logiquement être plus motivé (voire même plus confiant en ses capacités) lorsqu'une discipline scientifique lui est enseignée, même

si ce lien de causalité entre ces deux variables ne peut pas être établi sur la base des corrélations que nous avons observées.

D'autre part, les variables cognitives étudiées dans cette thèse (maîtrise des concepts computationnels et capacité à résoudre des problèmes algorithmiques) étaient assez dissemblables, à la fois dans les processus cognitifs qui étaient mis en jeu (capacité à se remémorer, reconnaître ou comprendre les concepts computationnels ou capacité à créer et évaluer un algorithme) et dans la manière d'évaluer ces apprentissages (QCM ou question ouverte nécessitant la production d'un algorithme), ce qui explique peut-être le faible lien de corrélation observé entre ces deux variables dans notre expérience 1. Cependant, ce lien est renforcé dans l'expérience 2 et plus encore dans l'expérience 3, pour laquelle les corrélations observées sont assez fortes. Notons que l'expérience 3 est celle pour laquelle l'ensemble des élèves ne bénéficiaient d'aucun feedback provenant des outils numériques. Il serait possible de formuler plusieurs hypothèses pour expliquer pourquoi l'absence de feedback semble produire une corrélation plus forte entre la maîtrise des concepts computationnels et la capacité à résoudre des problèmes algorithmiques. Cependant, au vu du manque de robustesse de ce résultat (qui ne se fonde que sur une relation de corrélation plus ou moins forte et pas sur des liens de causalité), nous préférons ne pas nous livrer à des interprétations plus ou moins hasardeuses.

Plus généralement, les résultats observés dans nos trois premières expériences semblent indiquer une décorrélation progressive entre nos deux variables de performances au fil du temps d'enseignement (puisque celles-ci sont mieux corrélées dans nos expériences 2 et 3, plus courtes). Nous pouvons tout de même constater que le lien entre maîtrise des concepts computationnels et capacité à résoudre des problèmes serait assez ténu, peut-être en raison de la multitude de processus qui interviennent dans les activités de résolution de problèmes. Rahman (2019) recense l'ensemble des compétences qui constituent les capacités de résolution de problèmes, considérées comme fondamentales pour le XXIe siècle, et qui interviennent lorsqu'un individu a un but, mais ne sait pas immédiatement comment l'atteindre. L'auteur suggère que la résolution de problèmes peut être décomposée en deux composantes essentielles : l'observation et la pensée critique. Observer consiste à collecter des données, comprendre et interpréter la signification de ces informations. La pensée critique implique la capacité à conceptualiser, raisonner logiquement, appliquer une stratégie, penser de manière analytique, prendre des décisions et synthétiser pour résoudre un problème. Nous pouvons constater que nombre de ces sous-compétences peuvent être sollicitées dans notre Test

d'Algorithmique, mais pas dans notre Test de Maîtrise des Concepts Computationnels, ce qui explique probablement pourquoi les résultats à ces deux tests sont assez faiblement corrélés.

Enfin, conformément à nos prédictions, les liens de corrélations entre nos variables cognitives et conatives, prises conjointement, sont assez faibles, voire inexistants. En particulier, la capacité à résoudre des problèmes algorithmiques est très peu corrélée à la motivation, au sentiment d'auto-efficacité et à l'attitude envers les sciences. Même si chaque élève de notre expérience 1 n'a pas pu tester les différents outils utilisés dans nos trois conditions pour déterminer lequel était leur favori, nos résultats semblent valider le paradoxe préférences / performances (Amadieu & Tricot, 2014). De fait, les élèves qui semblent les plus motivés et engagés dans l'expérience d'apprentissage sont ceux qui ont utilisé le robot Thymio. Or, ceux-ci constituaient le groupe qui s'est révélé le moins performant (même si la différence avec le groupe débranché n'était pas significative). Ainsi, la capacité d'un outil à motiver et à engager les élèves dans l'apprentissage avec un haut niveau d'auto-efficacité ne semble pas garantir que cet outil soit le plus indiqué pour transmettre les notions et compétences fondamentales en programmation.

Par ailleurs, une très récente étude de Safitri et al. (2023) met en évidence un lien de corrélation quasi nul (et non significatif) entre le sentiment d'auto-efficacité et les compétences en pensée informatique (r = -.036) d'élèves de niveau CM2. Les chercheurs considèrent que le sentiment d'auto-efficacité explique seulement 0.12 % des compétences en pensée informatique, ce qui est négligeable. Tsai (2019) avait également observé que les élèves présentant un sentiment d'auto-efficacité faible ou modéré bénéficiaient d'une progression plus marquée de leurs performances, ce qui traduit donc une décorrélation entre sentiment d'auto-efficacité et performances.

Cependant, Sırakaya et al. (2023) font état d'un impact significatif de l'attitude envers les disciplines scientifiques STEM sur les compétences en pensée informatique. Toutefois, leur évaluation de la pensée informatique s'appuie sur les *Computational Thinking Scales* (Korkmaz et al., 2017), dont nous avons évoqué les limites dans l'introduction du premier chapitre expérimental, qui se basent sur des mesures autorapportées, et une conception de la pensée informatique assez éloignée de l'avis de la majorité des chercheurs. Rino et al. (2023) ont également montré un lien de causalité entre l'attitude vis-à-vis des STEM et le développement de la pensée informatique. D'autres études témoignent de ce lien entre attitude envers les sciences et performances d'apprentissage (Alsancak, 2020 ; Lishinski et al., 2016 ; Sun et al.,

2021). Ces quelques études aux résultats contradictoires, auxquelles nous pouvons ajouter la nôtre (expérience 1), présentent de grandes différences dans les façons d'évaluer ces variables d'intérêt, ce qui rend impossible toute conclusion à ce sujet. En définitive, notre hypothèse H5 paraît plutôt étayée par nos données empiriques, bien que les liens qui unissent les variables cognitives relatives aux performances d'apprentissage soient assez faibles. Il serait toutefois pertinent que les études à venir, mesurant des variables cognitives et conatives, prennent soin d'analyser les liens de corrélation qui existent entre celles-ci, au vu du peu de données disponibles concernant cette question.

Limites et perspectives

Forts des résultats rapportés dans cette thèse, qui comprenait la conception et la validation de deux outils d'évaluation ainsi que la réalisation de quatre expériences, nous avons pu apporter des contributions significatives à l'état actuel de la littérature dans le domaine de l'enseignement de la programmation et de la pensée informatique. Ces contributions ne doivent toutefois pas occulter certaines limites que nous avons pu constater, concernant les protocoles expérimentaux que nous avons mis en place. Ces limites nous permettront d'envisager de nouvelles perspectives de recherche afin de poursuivre les travaux relatifs à notre problématique. Les limites des deux outils que nous avons conçus, ainsi que les perspectives d'amélioration futures de ces outils, ont déjà été abordées, à la fois dans la discussion associée à notre premier chapitre expérimental, et dans la première partie de cette « Discussion générale ». Nous ne reviendrons donc pas sur celles-ci. De même, il va de soi que les recherches que nous avons menées peuvent être appliquées à d'autres contextes, avec des élèves de niveaux différents, issus de catégories sociodémographiques défavorisées, dans d'autres pays, etc. Nous nous concentrerons sur les perspectives proches de notre question de recherche, en lien avec l'impact quantitatif des outils d'enseignement de la programmation et de la pensée informatique à la fin du primaire ou au début du secondaire, auprès d'élèves n'ayant pas ou peu d'expérience en la matière.

En premier lieu, il nous semble essentiel de rappeler qu'en raison de certains enjeux pratiques, méthodologiques, écologiques ou éthiques détaillés précédemment, le protocole expérimental employé pour notre expérience 1 est assez largement différent de celui employé pour nos expériences 2, 3 et 4. Dans ces trois dernières expériences, la durée de notre intervention était nettement plus courte (une ou deux séances, contrairement aux 10 séances de l'expérience 1), donc moins approfondie, et nous ne mesurions pas toujours exactement les

mêmes variables dépendantes. La capacité à résoudre des problèmes algorithmiques, évaluée par la première version de notre Test d'Algorithmique dans l'expérience 1, a été mesurée à l'aide de la deuxième version dans les expériences 2 et 3, cette deuxième version s'étant montrée plus valide et plus généralisable, y compris pour des élèves n'ayant suivi qu'une seule séance de programmation. De plus, le contrôle de l'« effet maître » (Hattie, 2003) n'était pas optimal dans notre expérience 1, car, pour des raisons pratiques, il était difficile pour les enseignants de réaliser 10 séances en demi-classes, afin que plusieurs conditions expérimentales soient présentes au sein d'une même classe. De surcroit, le matériel utilisé n'était pas forcément le même d'une expérience à l'autre (ordinateurs dans les expériences 1 et 4, tablettes dans les expériences 2 et 3). Même si ces choix de modification du protocole nous semblaient justifiés, il est possible que ceux-ci ont pu influencer nos résultats et expliquer pourquoi certains effets ne sont pas retrouvés d'une expérience sur l'autre. À l'avenir, il serait pertinent de réfléchir à des solutions permettant de conserver un protocole plus semblable entre deux expériences, lorsque la deuxième est la conséquence logique de la première, et vise à vérifier ou expliquer certaines observations réalisées précédemment. Enfin, nous avons fait le choix de contrôler les compétences et concepts enseignés, afin de s'assurer que nos conditions expérimentales sont comparables. Il est toutefois probable que certaines pratiques pédagogiques soient plus favorables à un apprentissage débranché ou à un apprentissage branché, avec ou sans robot. Nos études ne permettent pas de s'assurer que les résultats obtenus sont généralisables à toutes les façons d'enseigner la programmation en classe.

En outre, même si notre expérience 1 présente une durée relativement élevée, les conséquences d'une année entière de formation et d'apprentissage de la programmation et de la pensée informatique pourraient être bien différentes de celles que nous observons sur un temps plus réduit, en particulier en ce qui concerne certaines variables peu ou pas influencées par nos interventions (attitude envers les sciences). La méta-analyse de Chen et al. (2023) indique une diminution de la taille d'effet lorsqu'une intervention visant à développer la pensée informatique par le biais d'activités débranchées excède les 10 semaines. De même, Lu et al. (2023) constatent que la durée d'intervention idéale pour observer les effets d'un apprentissage fondé sur le jeu de la pensée informatique est assez courte, comprise entre 4 heures et 1 semaine. D'une manière générale, il semblerait que l'augmentation de la durée de l'intervention s'accompagne le plus souvent d'une diminution de la taille des effets (Sun, Guo & Zhou, 2022 ; Zhang et al., 2021). Cependant, les études à long terme sont rares, voire inexistantes. Chen et

al. (2023) reconnaissent que les effets d'un enseignement de la programmation sur une longue durée (une ou plusieurs années) sont encore largement inconnus.

Par ailleurs, afin de garantir une évaluation des performances qui soit strictement la même entre nos conditions expérimentales, tous les élèves ont dû composer sur papier. Nous pouvons tout de même regretter, en particulier pour le test d'Algorithmique, que les élèves n'aient pas pu produire des algorithmes de la même manière qu'ils l'ont fait tout au long des séances (en assemblant des instructions papiers, ou des blocs Scratch, pour programmer un personnage virtuel ou un robot réel). Ceci aurait bien évidemment pu constituer un biais, car il aurait été difficile de considérer ces trois évaluations comme strictement équivalentes. Il serait cependant souhaitable de réfléchir à un mode d'évaluation qui s'adapte aux outils utilisés et maîtrisés par les élèves, sans toutefois mettre en péril la possibilité d'effectuer des comparaisons pertinentes et valides entre ces élèves.

De plus, au moment où nous avons commencé à concevoir nos outils d'évaluation et nos protocoles expérimentaux (2020), il n'y avait pas assez de données empiriques pour affirmer que les dispositifs mixtes (mettant en jeu plusieurs outils d'enseignement différents au cours d'une même séquence pédagogique) se révéleraient nécessairement plus efficaces que les dispositifs centrés sur un outil particulier. Quelques recherches récentes semblent cependant appuyer ce constat (Akiba, 2022), en particulier lorsque les séances débutent par des activités débranchées, ce qui aurait un effet positif sur les performances (Sun et al., 2022b), la motivation (del Olmo-Muñoz et al., 2020), le sentiment d'auto-efficacité (del Olmo-Muñoz et al., 2020 ; Hermans & Aivaloglou, 2017) et la réduction de l'écart entre les genres (del Olmo-Muñoz et al., 2020). Par conséquent, il serait intéressant d'appliquer l'approche comparative qui est au cœur de cette thèse à des dispositifs mixtes (débranché puis branché, branché sans robot puis avec robot, etc.), ce que nous n'avons pas véritablement entrepris pour le moment, même si la première séance de notre expérience 1 (débranchée) était commune à nos trois conditions expérimentales, et la deuxième (branchée sans robot) était commune à nos deux conditions branchées. Dans une méta-analyse consacrée à l'impact des jeux éducatifs sur l'apprentissage de la pensée informatique, Sun, Guo et Hu (2021) ont constaté une augmentation des effets lorsque des dispositifs mixtes sont utilisés, d'abord débranché, puis branché.

D'autre part, parmi les outils que nous avons utilisés, il existe différents types d'activités débranchées, de logiciels de programmation éducatifs basés sur des blocs et de robots pédagogiques, utilisables ou non avec un logiciel. Les activités débranchées désignent toutes

les activités qui peuvent être mises en place pour transmettre des notions et compétences en programmation, sans utiliser d'outils numériques, et sont pratiquement illimitées. Chen et al. (2023) regroupent ces activités en différents types, qui produiraient un effet plus ou moins fort sur les apprentissages (même si les tailles d'effet diffèrent assez peu) : jeux de plateau, jeux de cartes, activités sur papier, blocs, jeux sur papier quadrillé, robotique débranchée, etc. Leur méta-analyse révèle que les activités basées sur des jeux de plateau ou de cartes sont les plus fréquentes, car, selon les auteurs, celles-ci facilitent les interactions et la pensée de « haut niveau », tout en permettant d'aborder naturellement les aspects structurels de la programmation (structure conditionnelle, structure itérative…). Chen et al. (2023) recommandent ainsi une approche visant à « gamifier » les activités proposées pour susciter une plus grande motivation et un plus grand engagement. Les activités débranchées que nous avons proposées étaient d'ailleurs plutôt ludiques, mais il serait tout de même pertinent de tester si d'autres types d'activités pourraient être plus adaptés au contexte d'apprentissage et produire des effets plus ou moins importants.

Même si Scratch demeure encore aujourd'hui le langage visuel par blocs le plus populaire dans les écoles du monde entier, il existe d'autres logiciels éducatifs susceptibles de développer des compétences similaires (Alice, Kodu, Construct, StarLogo, App Inventor, Greenfoot, Blockly, etc). Alice a la particularité d'être à la fois pensé pour des novices et très souvent utilisé avec des élèves assez âgés, voire adultes, puisque plusieurs études font état de son usage à l'université (Rodger et al., 2010 ; Weitnrop & Wilensky, 2015). Ce logiciel est particulièrement indiqué pour réaliser des animations 3D plus ou moins complexes (Rodger et al., 2010). Alice permet également d'appréhender une grande variété de concepts issus de l'informatique (Werner et al., 2012). Il serait intéressant de mesurer l'impact des différents logiciels ou langages sur l'apprentissage de concepts ou de compétences spécifiques, même si, d'après deux études comparatives, Scratch semble plus efficace qu'Alice et App Inventor (Hu et al., 2021) ou Blockly (Seraj et al., 2019) pour transmettre les compétences en programmation.

Dans nos études, nous avons mis à contribution des robots Thymio pour des raisons pratiques (disponibilité des Thymio à la DRANE de Montpellier), méthodologiques (puisque ces robots sont programmables sur Scratch, nous offrant la possibilité d'une comparaison pertinente avec l'usage de Scratch sans robot) et également car aucune étude n'a pour le moment montré une supériorité d'un robot par rapport à un autre, dans l'enseignement de la programmation et de la pensée informatique. Cependant, d'autres alternatives existent (Robot Turtle, Lego Mindstorms, mbot, Blue Bot, etc…) et les apports et limites de ces différents

robots mériteraient d'être mieux documentés. Par exemple, le robot mbot peut être assemblé selon différentes configurations, à la différence de Thymio, et peut également être contrôlé à distance via un logiciel visuel par blocs pour permettre aux élèves d'apprendre à programmer (Pisarov & Mester, 2019). L'utilisation du robot mbot peut produire un haut niveau de satisfaction chez les élèves et les inciter à poursuivre les activités de programmation (Cheng & Hsiao, 2021). D'autres robots programmables, ne nécessitant aucun outil externe pour être programmés, peuvent également être employés. C'est le cas notamment du robot « débranché » Bee-bot, plutôt pensé pour les élèves les plus jeunes de l'école primaire (maternelle et début d'élémentaire), proches des jouets éducatifs. Ceux-ci présentent l'avantage d'une grande simplicité d'utilisation (Angeli & Valandides, 2020 ; Vargová & Círus, 2021), bien que les activités possibles via ces outils soient très limitées (Angeli & Valanides, 2020).

De même, certains outils d'enseignement (ou certaines pratiques pédagogiques) non étudiés dans cette thèse, et moins employés dans les écoles, pourraient favoriser l'apprentissage de la programmation et de la pensée informatique. Ce serait notamment le cas de la gamification (ou ludification) qui consiste à détourner certains mécanismes couramment employés dans les jeux (en particulier les jeux vidéo) pour les appliquer au domaine de l'éducation, dans l'espoir de promouvoir la capacité des élèves à résoudre des problèmes et à surmonter des défis, stimulant ainsi leurs performances et leur motivation. Bien que certaines études témoignent d'un impact bénéfique de la gamification, l'apport de cette pratique demeure controversé (Zhan, He, Tong et al., 2022). Pourtant, dans leur méta-analyse comprenant 14 études, Zhan, He, Tong et al. (2022) révèlent un effet positif modéré de la gamification sur l'apprentissage de la programmation, en particulier en ce qui concerne la motivation et les performances académiques des élèves, même si cet effet est plus prononcé dans l'enseignement supérieur que dans le primaire ou le secondaire. Cependant, l'effet sur la motivation pourrait être plus favorable aux garçons qu'aux filles (Pedro et al., 2015). La gamification semble également accroître légèrement la charge cognitive de la situation d'apprentissage, selon Zhan, He, Tong et al. (2022).

D'autres méta-analyses ont été publiées sur l'impact de la gamification. Manzana-León et al. (2021) confirment que la gamification peut être une stratégie pédagogique efficace, de par ses effets positifs sur la motivation, l'engagement et les performances académiques à différents niveaux scolaires. Toutefois, l'effet sur la motivation, assez largement démontré, concernerait parfois uniquement la motivation extrinsèque des élèves, fondée sur l'obtention de points, badges ou médailles en cas de réussite, ce que regrettent Manzana-León et al. (2021). del Olmo-

Muñoz et al. (2023) ont en quelque sorte répondu à cette limite, évoquée par Manzana-León et al. (2021), en comparant directement les effets de la « *deep gamification* » (centrée sur des approches visant à améliorer la motivation intrinsèque) et de la « *shallow gamification* » (visant à améliorer la motivation extrinsèque), sur le développement des compétences de la pensée informatique de 82 élèves de niveau CE1 (âgés de 7 à 8 ans), pendant trois séances débranchées et deux séances branchées de 45 minutes chacune. Leur étude met en lumière un impact plus important de la « *deep gamification* » sur la motivation, mais révèle aussi que la « *shallow gamification* » produit de meilleures performances d'apprentissage, probablement en raison de l'investissement, de la complexité et de l'engagement nécessaires pour enseigner et apprendre dans un contexte de « *deep gamification* », qui distrait l'élève de cet objectif.

Sun, Guo et Hu (2021) ont également réalisé une méta-analyse révélant l'impact positif des jeux éducatifs sur les compétences associées à la pensée informatique, modéré par la taille de l'échantillon, le niveau scolaire ou les outils utilisés. Contrairement à Zhan, He, Tong et al. (2022), l'effet le plus probant concerne ici les élèves d'école élémentaire. Très récemment, une autre méta-analyse (Lu et al., 2023) confirme à nouveau l'apport positif de l'apprentissage fondé sur des jeux pour développer la pensée informatique, en particulier lorsque l'intervention est assez courte (de 4 heures à 1 semaine). Bien que nous ayons proposé des activités ludiques dans nos trois conditions expérimentales, notamment un projet créatif laissant plus de liberté aux élèves, il serait pertinent à l'avenir de se pencher de manière plus approfondie sur les diverses techniques de gamification qui pourraient moduler l'apprentissage de la programmation et de la pensée informatique.

Parmi les outils disponibles utilisés pour enseigner la programmation, l'usage des cartes programmables, telles que la carte électronique Arduino, n'a pas été étudié dans cette thèse. Dans leur revue systématique de la littérature comprenant neuf articles, García-Tudela et Marín-Marín (2023) constatent que celle-ci n'est pas un outil très répandu en éducation. Les auteurs observent que les systèmes Arduino sont principalement mis à contribution pour enseigner la programmation à l'école élémentaire, à l'aide d'une approche pédagogique fondée sur des problèmes, et que ceux-ci peuvent être utilisés en association avec des langages visuels par blocs, tels que Scratch ou Blockly. D'ailleurs, une récente méta-analyse a mis en évidence l'effet positif de l'association Arduino / Scratch sur les compétences reliées à la pensée informatique (Fidai et al., 2020), même si aucune étude comparative directe entre ce dispositif et d'autres outils ne semble avoir été menée.

Du reste, dans notre « Partie théorique », nous aborde l'effet potentiel des interactions motrices et des mouvements du corps sur l'optimisation des apprentissages. La théorie de la cognition incarnée et située suppose que nos expériences sont encodées sous la forme de représentations multimodales, stockées en mémoire, celles-ci étant réactivées ultérieurement lorsque les connaissances acquises grâce à cette expérience sont nécessaires (Barsalou, 2008). Il serait donc important de proposer des situations d'apprentissage riches en stimulations sensorielles pour favoriser la génération et le maintien de traces mémorielles multimodales, selon cette perspective. À la différence de l'influence de la charge cognitive d'une situation d'apprentissage de la programmation et de l'impact du feedback des outils numériques, ce facteur n'a pas été exploré dans cette thèse. En effet, la capacité à susciter des interactions motrices est plutôt l'apanage des activités débranchées (et dans une moindre mesure des activités de robotique, telles que nous les avons proposées). Or, les résultats de notre expérience 1 font état d'une infériorité des performances des élèves en condition débranchée (et également en condition branchée avec robot), ce qui ne nous a pas incités à poursuivre prioritairement cette piste d'explication des effets observés.

Pour Black (2010), les élèves apprennent à l'école principalement d'une manière totalement déconnectée de leurs expériences sensorielles, stockant en mémoire des connaissances abstraites symboliques, sans être capables de les réutiliser lorsque cela s'avère approprié. Une étude de Fadjo et al. (2009) rapporte qu'une approche consistant à demander aux élèves, travaillant sur Scratch, de produire les mouvements d'un personnage virtuel avant de le programmer pour qu'il fasse ces mouvements était particulièrement efficace. Quelques études ont d'ailleurs montré l'effet positif de l'incarnation pédagogique des concepts étudiés lors de l'apprentissage de la pensée informatique (Black et al., 2012), que cette incarnation soit directe (par la réalisation de certains mouvements) ou explicitement imaginée (par la simulation volontaire et imaginée de ces mouvements). Il serait donc pertinent d'explorer l'impact des approches pédagogiques centrées sur l'incarnation (*embodiment*) sur la maîtrise de la pensée informatique et l'apprentissage de la programmation. Cependant, Merkouris et Chorianopoulos (2019) ont réalisé une expérience impliquant 36 élèves de collège (âgés de 14 à 15 ans) dans des activités de robotique, afin de leur transmettre certains concepts associés à la pensée informatique et d'observer leurs pratiques computationnelles et leur perception de la programmation. Les chercheurs ont testé l'influence de différents niveaux d'incarnation et ont constaté que les élèves ayant utilisé des interfaces à faible niveau d'incarnation (tactiles et multimodales) ou des interfaces sans incarnation ont produit des projets plus sophistiqués que

ceux ayant utilisé des interfaces à fort niveau d'incarnation (interface corporelle). Merkouris et Chorianopoulos (2019) suggèrent qu'il pourrait y avoir un compromis à trouver entre l'attrait et le bénéfice cognitif d'une approche avec un fort niveau d'incarnation, et l'effort et l'investissement nécessaire de cette approche, qui a pu surcharger les ressources cognitives des élèves.

Enfin, même si la motivation et le sentiment d'auto-efficacité ont été étudiés dans nos expériences, force est de constater que nous avons principalement mis l'accent sur les performances d'apprentissage et sur l'exploration des facteurs qui pourraient les influencer. Le test que nous avons utilisé pour évaluer la motivation et le sentiment d'auto-efficacité des élèves (*Student's Approaches to Learning*, Marsh et al., 2006) a été choisi pour sa simplicité d'utilisation et parce qu'il a fait ses preuves dans de nombreux pays de l'OCDE. La conception de ce test est fondée sur une revue des instruments potentiels et des construits psychologiques de l'éducation afin d'identifier les instruments et échelles les plus utiles et les plus solides d'un point de vue psychométrique pour évaluer les approches d'apprentissage des élèves, incluant notamment les préférences motivationnelles et les croyances relatives à soi (dont le sentiment d'auto-efficacité).

Gopalan et al. (2017) recensent cinq modèles ou théories de la motivation. L'échelle de Marsh et al. (2006) s'appuie principalement sur la théorie de la motivation intrinsèque et extrinsèque et la théorie de l'autodétermination (Deci & Ryan, 2000). Cependant, d'autres modèles auraient pu être mis à contribution pour approfondir notre compréhension des effets relatifs à la motivation ou au sentiment d'auto-efficacité des élèves. Le modèle ARCS (*attention, relevance, confidence and satisfaction*) soutient qu'il est nécessaire de capter l'attention de l'élève pour bénéficier de son engagement, de proposer une tâche pertinente par rapport à ses besoins et de susciter la confiance et un sentiment positif de satisfaction face à l'apprentissage (Keller, 2008). La théorie sociale cognitive décrit comment l'environnement social et physique d'un individu a pour conséquence l'acquisition et le maintien de certains schémas de comportements (Bandura, 1989). Enfin, la théorie des attentes met en évidence l'interrelation entre l'effort investi dans une tâche et les chances perçues d'obtenir un résultat souhaité (Van Eerde & Thierry, 1996). Les expériences futures pourraient s'attarder sur la mise à contribution de ces modèles, moins communément utilisés que le modèle de la motivation intrinsèque et extrinsèque, bien que tout aussi pertinents, pour approfondir notre compréhension des déterminants de la motivation et du sentiment d'auto-efficacité des élèves, en contexte d'apprentissage de la programmation et de la pensée informatique.

CONCLUSION

L'apprentissage de la programmation et de la pensée informatique devient un enjeu majeur dans la plupart des systèmes éducatifs, en particulier dans les pays développés. Malgré les incertitudes qui subsistent quant à la définition exacte de la pensée informatique, le développement de celle-ci constitue sans doute l'objectif prioritaire de la mise en place d'activités de programmation en classe. Cette thèse pose la question, rarement abordée, de l'efficacité relative des différents outils qui peuvent être employés pour enseigner cette discipline, et de réaliser des comparaisons fondées sur des données empiriques quantitatives entre les activités débranchées, les logiciels de programmation éducatifs visuels par blocs et les robots pédagogiques. Nos études se sont déroulées dans un contexte écologique et ont veillé à ce que les mêmes notions et compétences soient enseignées entre nos différentes conditions expérimentales. Celles-ci ont été transmises par le biais d'activités très semblables entre les groupes, si bien que la seule différence majeure reposait précisément sur les outils utilisés, auxquels nous pouvons imputer les différences significatives que nous avons pu observer.

Les résultats obtenus, notamment par le biais de deux outils de mesure que nous avons conçus et validés, mettent en lumière une supériorité des performances d'apprentissage des élèves lorsque ceux-ci utilisent les langages visuels de programmation, sans association avec un robot pédagogique. L'effet délétère de l'ajout d'un robot au dispositif d'apprentissage pourrait s'expliquer par l'augmentation de la charge cognitive extrinsèque, telle que mesurée dans notre dernière expérience. Cependant, nous ne sommes pas parvenus à expliquer pourquoi les logiciels éducatifs tels que Scratch semblent plus à même de favoriser les performances des élèves ; d'autres études devraient être menées pour explorer l'impact du feedback des outils numériques, feedback qui permet de visualiser en temps réel l'exécution des programmes créés. Ces outils numériques seraient également les plus susceptibles de maintenir ou de susciter la motivation et l'engagement des élèves, en particulier les robots qui produisent un effet fort sur cette variable. En revanche, l'impact des activités de programmation sur l'attitude des élèves envers les sciences n'a pas pu être observé. Les principales différences entre les genres font état d'une motivation et d'un sentiment d'auto-efficacité globalement plus important chez les garçons, en particulier dans la condition branchée sans robot. D'autres études sont nécessaires pour explorer notamment l'influence des dispositifs mixtes (alliant plusieurs outils) et d'autres outils ou approches, non étudiés dans cette thèse.

BIBLIOGRAPHIE

Ackermann, E. (2001). Piaget's constructivism, Papert's constructionism: What's the difference. *Future of learning group publication*, *5*(3), 438. https://learning.media.mit.edu/content/publications/EA.Piaget%20_%20Papert.pdf

Aho, A. V. (2012). Computation and computational thinking. *The computer journal*, *55*(7), 832-835. https://doi.org/10.1093/comjnl/bxs074

Aivaloglou, E., & Hermans, F. (2016, August). How kids code and how we know: An exploratory study on the Scratch repository. In *Proceedings of the 2016 ACM conference on international computing education research* (pp. 53-61). https://doi.org/10.1145/2960310.2960325

Akiba, D. (2022). Computational Thinking and Coding for Young Children: A Hybrid Approach to Link Unplugged and Plugged Activities. *Education Sciences*, *12*(11), 793. https://doi.org/10.3390/educsci12110793

Alimisis, D. (2013). Educational robotics: Open questions and new challenges. *Themes in Science and Technology Education*, *6*(1), 63-71. http://earthlab.uoi.gr/theste/index.php/theste/article/view/119/85

Alimisis, D., & Kynigos, C. (2009). Constructionism and robotics in education. *Teacher education on robotic-enhanced constructivist pedagogical methods*, 11-26. https://roboesl.eu/wp-content/uploads/2017/08/chapter_1.pdf

Alsancak, D. (2020). Investigating computational thinking skills based on different variables and determining the predictor variables. *Participatory Educational Research*, *7*(2), 102-114. https://doi.org/10.17275/per.20.22.7.2

Alves, N. D. C., Von Wangenheim, C. G., & Hauck, J. C. (2019). Approaches to assess computational thinking competences based on code analysis in K-12 education: A systematic mapping study. *Informatics in Education*, *18*(1), 17. https://doi.org/10.15388/infedu.2019.02

Amadieu, F., & Tricot, A. (2014). *Apprendre avec le numérique: mythes et réalités*. Retz.

Andersen, M. S., & Makransky, G. (2021). The validation and further development of a multidimensional cognitive load scale for virtual environments. *Journal of Computer Assisted Learning*, *37*(1), 183-196. https://doi.org/10.1111/jcal.1247

Angeli, C., & Valanides, N. (2020). Developing young children's computational thinking with educational robotics: An interaction effect between gender and scaffolding strategy. *Computers in human behavior*, *105*, 105954. https://doi.org/10.1016/j.chb.2019.03.018

Anwar, S., Bascou, N. A., Menekse, M., & Kardgar, A. (2019). A systematic review of studies on educational robotics. *Journal of Pre-College Engineering Education Research (J-PEER)*, *9*(2), 2. https://doi.org/10.7771/2157-9288.1223

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From scratch to "real" programming. *ACM Transactions on Computing Education (TOCE)*, *14*(4), 1-15. https://doi.org/10.1145/2677087

Athanasiou, L., Mikropoulos, T. A., & Mavridis, D. (2019). Robotics interventions for improving educational outcomes-A meta-analysis. In *Technology and Innovation in Learning, Teaching and Education: First International Conference, TECH-EDU 2018, Thessaloniki, Greece, June 20–22, 2018, Revised Selected Papers 1* (pp. 91-102). Springer International Publishing. https://doi.org/10.1007/978-3-030-20954-4_7

Atmatzidou, S., & Demetriadis, S. (2014, July). How to support students' computational thinking skills in educational robotics activities. In *Proceedings of 4th International Workshop Teaching Robotics, Teaching with Robotics & 5th International Conference Robotics in Education* (Vol. 18, pp. 43-50).

Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems*, *75*, 661-670. https://doi.org/10.1016/j.robot.2015.10.008

Ayres, P. (2006). Using subjective measures to detect variations of intrinsic cognitive load within problems. *Learning and instruction*, *16*(5), 389-400. https://doi.org/10.1016/j.learninstruc.2006.09.001

Babazadeh, M., & Negrini, L. (2022). How is computational thinking assessed in European K-12 education? A systematic review. *International Journal of Computer Science Education in Schools*, *5*(4), 3-19. https://doi.org/10.21585/ijcses.v5i4.138

Bakala, E., Gerosa, A., Hourcade, J. P., & Tejera, G. (2021). Preschool children, robots, and computational thinking: A systematic review. *International Journal of Child-Computer Interaction*, *29*, 100337. https://doi.org/10.1016/j.ijcci.2021.100337

Bandura, A. (1977). Self-efficacy: toward a unifying theory of behavioral change. *Psychological review*, *84*(2), 191. https://doi.org/10.1037/0033-295X.84.2.191

Bandura, A. (1986). The explanatory and predictive scope of self-efficacy theory. *Journal of social and clinical psychology*, *4*(3), 359-373. https://doi.org/10.1521/jscp.1986.4.3.359

Bandura, A. (1989). Human agency in social cognitive theory. *American psychologist*, *44*(9), 1175. https://doi.org/10.1037/0003-066X.44.9.1175

Bara, F., & Gentaz, E. (2011). Haptics in teaching handwriting: The role of perceptual and visuo-motor skills. *Human movement science*, *30*(4), 745-759. https://doi.org/10.1016/j.humov.2010.05.015

Bara, F., Gentaz, É., & Colé, P. (2004). Les effets des entraînements phonologiques et multisensoriels destinés à favoriser l'apprentissage de la lecture chez les jeunes enfants. *Enfance*, *56*(4), 387-403. https://doi.org/10.3917/enf.564.0387

Baron, G. L. (2014). Élèves, apprentissages et «numérique»: regard rétrospectif et perspectives. *Recherches en éducation*, (18). https://doi.org/10.4000/ree.8525

Baron, G. L., & Boulc'h, L. (2012). Les technologies de l'information et de la communication à l'école primaire. État de question en 2011. *Revue de l'EPI*, 90-120. http://www.epi.asso.fr/revue/articles/a1202b.htm

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community?. *Acm Inroads*, *2*(1), 48-54. https://doi.org/10.1145/1929887.1929905

Barsalou, L. W. (2008). Grounded cognition. *Annu. Rev. Psychol.*, *59*, 617-645. https://doi.org/10.1146/annurev.psych.59.103006.093639

Baser, M. (2013). Attitude, gender and achievement in computer programming. *Online Submission*, *14*(2), 248-255. https://doi.org/10.5829/idosi.mejsr.2013.14.2.2007

Basque, J., & Lundgren-Cayrol, K. (2003). Une typologie des typologies des usages des «TIC» en éducation. *Télé-Université, Québec*. https://tecfa.unige.ch/tecfa/teaching/riat140/0304/typologies.pdf

Basu, S., Rutstein, D. W., Xu, Y., Wang, H., & Shear, L. (2021). A principled approach to designing computational thinking concepts and practices assessments for upper elementary grades. *Computer Science Education*, *31*(2), 169-198. https://doi.org/10.1080/08993408.2020.1866939

Battal, A., Afacan Adanır, G., & Gülbahar, Y. (2021). Computer science unplugged: A systematic literature review. *Journal of Educational Technology Systems*, *50*(1), 24-47. https://doi.org/10.1177/00472395211018801

Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology*, *13*(1), 20-29.

Bell, T., & Vahrenhold, J. (2018). CS unplugged—how is it used, and does it work?. *Adventures between lower bounds and higher altitudes: essays dedicated to Juraj Hromkovič on the occasion of his 60th birthday*, 497-521. https://doi.org/10.1007/978-3-319-98355-4_29

Bellegarde, K., Boyaval, J., & Alvarez, J. (2019). S'initier à la robotique/informatique en classe de grande section de maternelle. Une expérimentation autour de l'utilisation du robot Blue Bot comme jeux sérieux. *Review of Science, Mathematics and ICT Education*, *13*(1), 51-72. https://doi.org/10.26220/rev.3105

Benitti, F. B. V. (2012). Exploring the educational potential of robotics in schools: A systematic review. *Computers & Education*, *58*(3), 978-988. https://doi.org/10.1016/j.compedu.2011.10.006

Bernard, R. M., Borokhovski, E., Schmid, R. F., & Tamim, R. M. (2018). Gauging the effectiveness of educational technology integration in education: What the best-quality meta-analyses tell us. *Learning, design, and technology*, 1-25. https://doi.org/10.1007/978-3-319-17727-4_109-2

Berry, V. (2009). Loisirs numériques et communautés virtuelles: des espaces d'apprentissage?. https://hal.science/hal-03975082/

Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education*, *72*, 145-157. https://doi.org/10.1016/j.compedu.2013.10.020

Bers, M. U., González-González, C., & Armas–Torres, M. B. (2019). Coding as a playground: Promoting positive learning experiences in childhood classrooms. *Computers & Education*, *138*, 130-145. https://doi.org/10.1016/j.compedu.2019.04.013

Bers, M. U., Strawhacker, A., & Sullivan, A. (2022). The state of the field of computational thinking in early childhood education. https://sites.bc.edu/devtech/wp-content/uploads/sites/113/2022/07/TheStateofFieldandComputerationalThinkinginEarlyChildhoodEducation.pdf

Berssanette, J. H., & de Francisco, A. C. (2021). Cognitive load theory in the context of teaching and learning computer programming: A systematic literature review. *IEEE Transactions on Education*, *65*(3), 440-449. https://doi.org/10.1109/TE.2021.3127215

Bibeau, R. (2005). Les TIC à l'école: proposition de taxonomie et analyse des obstacles à leur intégration. *Revue de l'EPI*. https://edutice.archives-ouvertes.fr/edutice-00277818/file/a0511a.htm

Black, J. B. (2010). An embodied/grounded cognition perspective on educational technology. *New science of learning: Cognition, computers and collaboration in education*, 45-52. https://doi.org/10.1007/978-1-4419-5716-0_3

Black, J. B., Segal, A., Vitale, J., & Fadjo, C. L. (2012). Embodied cognition and learning environment design. *Theoretical foundations of learning environments*, *2*, 198-223. https://www.tc.columbia.edu/faculty/jbb21/faculty-profile/files/8Embodied_Cognition2.pdf

Bloom B. (1979). *Caractéristiques individuelles et apprentissages scolaires.* Labor, Bruxelles.

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., & Engelhardt, K. (2016). *Developing computational thinking in compulsory education-Implications for policy and practice* (No. JRC104188). Joint Research Centre (Seville site)

Bosgoed, L., & Fanchamps, N. (2022, June). The Effect of Unplugged Programming and Visual Programming on Computational Thinking in Children Aged 5 to 7. In *CTE-STEM 2022 conference*. https://doi.org/10.34641/ctestem.2022.451

Bouranta, N., Chitiris, L., & Paravantis, J. (2009). The relationship between internal and external service quality. *International Journal of Contemporary Hospitality Management*, *21*(3), 275-293. https://doi.org/10.1108/09596110910948297

Bourgeois, A., Birch, P., & Davydovskaia, O. (2019). *Digital Education at School in Europe. Eurydice Report*. Education, Audiovisual and Culture Executive Agency, European Commission. Available from EU Bookshop.

Brackmann, C. P., Román-González, M., Robles, G., Moreno-León, J., Casali, A., & Barone, D. (2017, November). Development of computational thinking skills through unplugged activities in primary school. In *Proceedings of the 12th workshop on primary and secondary computing education* (pp. 65-72). https://doi.org/10.1145/3137065.3137069

Brennan, K., & Resnick, M. (2012, April). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada* (Vol. 1, p. 25). http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf

Brown, T. A., & Moore, M. T. (2012). Confirmatory factor analysis. *Handbook of structural equation modeling*, *361*, 379.

Buitrago Flórez, F., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). Changing a generation's way of thinking: Teaching computational thinking through programming. *Review of Educational Research*, *87*(4), 834-860. https://doi.org/10.3102/0034654317710096

Caeli, E. N., & Yadav, A. (2020). Unplugged approaches to computational thinking: A historical perspective. *TechTrends*, *64*(1), 29-36. https://doi.org/10.1007/s11528-019-00410-5

Çakiroğlu, Ü., & Çevik, İ. (2022). A framework for measuring abstraction as a sub-skill of computational thinking in block-based programming environments. *Education and Information Technologies*, *27*(7), 9455-9484. https://doi.org/10.1007/s10639-022-11019-2

Çakiroğlu, Ü., Suiçmez, S. S., Kurtoğlu, Y. B., Sari, A., Yildiz, S., & Öztürk, M. (2018). Exploring perceived cognitive load in learning programming via Scratch. *Research in Learning Technology*, *26*. https://doi.org/10.25304/rlt.v26.1888

Carbonneau, K. J., Marley, S. C., & Selig, J. P. (2013). A meta-analysis of the efficacy of teaching mathematics with concrete manipulatives. *Journal of educational psychology*, *105*(2), 380-400. https://doi.org/10.1037/a0031084

Catlin, D., & Woollard, J. (2014, July). Educational robots and computational thinking. In Proceedings of 4th International workshop teaching robotics, teaching with robotics & 5th International conference robotics in education (pp. 144-151). https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=901a93643a8a88ac312456dedb135f8058895d39

Chen, B., Hwang, G. H., & Wang, S. H. (2021). Gender differences in cognitive load when applying game-based learning with intelligent robots. *Educational Technology & Society*, *24*(3), 102-115. https://www.jstor.org/stable/27032859

Chen, G., Shen, J., Barth-Cohen, L., Jiang, S., Huang, X., & Eltoukhy, M. (2017). Assessing elementary students' computational thinking in everyday reasoning and robotics programming. *Computers & education*, *109*, 162-175. https://doi.org/10.1016/j.compedu.2017.03.001

Chen, P., Yang, D., Metwally, A. H. S., Lavonen, J., & Wang, X. (2023). Fostering computational thinking through unplugged activities: A systematic literature review and meta-analysis. *International Journal of STEM Education*, *10*(1), 1-25. https://doi.org/10.1186/s40594-023-00434-7

Chen, S., Epps, J., & Chen, F. (2011, November). A comparison of four methods for cognitive load measurement. In *Proceedings of the 23rd Australian Computer-Human Interaction Conference* (pp. 76-79). https://doi.org/10.1145/2071536.2071547

Cheng, G. (2019). Exploring factors influencing the acceptance of visual programming environment among boys and girls in primary schools. *Computers in Human Behavior*, *92*, 361-372. https://doi.org/10.1016/j.chb.2018.11.043

Cheng, L., Wang, X., & Ritzhaupt, A. D. (2022). The Effects of Computational Thinking Integration in STEM on Students' Learning Performance in K-12 Education: A Meta-analysis. *Journal of Educational Computing Research*, 07356331221114183. https://doi.org/10.1177/07356331221114183

Cheng, Y. H., & Hsiao, J. M. (2021, January). Exploring the intention to continuance of learning programming at elementary school of rural area by the mbot robot. In *Proceedings of the International Conference on Artificial Life and Robotics, Online, Japan* (pp. 21-24). https://alife-robotics.co.jp/members2021/icarob/data/html/data/OS/OS9/OS9-7.pdf

Cheung, A. C., & Slavin, R. E. (2013). The effectiveness of educational technology applications for enhancing mathematics achievement in K-12 classrooms: A meta-analysis. *Educational research review*, *9*, 88-113. https://doi.org/10.1016/j.edurev.2013.01.001

Chevalier, M. (2022). *Évaluation d'une expérimentation randomisée de la pensée informatique, vecteur d'apprentissage des mathématiques au cycle 3 de l'école élémentaire, en CM1 et CM2* [Thèse de doctorat, Université Grenoble Alpes].

Chevalier, M., Giang, C., El-Hamamsy, L., Bonnet, E., Papaspyros, V., Pellet, J. P., ... & Mondada, F. (2022). The role of feedback and guidance as intervention methods to foster computational thinking in educational robotics learning activities for primary school. *Computers & Education*, *180*, 104431. https://doi.org/10.1016/j.compedu.2022.104431

Chiazzese, G., Arrigo, M., Chifari, A., Lonati, V., & Tosto, C. (2019, October). Educational robotics in primary school: Measuring the development of computational thinking skills with the bebras tasks. In *Informatics* (Vol. 6, No. 4, p. 43). MDPI. https://doi.org/10.3390/informatics6040043

Ching, Y. H., & Hsu, Y. C. (2023). Educational Robotics for Developing Computational Thinking in Young Learners: A Systematic Review. *TechTrends*, 1-12. https://doi.org/10.1007/s11528-023-00841-1

Ching, Y. H., Hsu, Y. C., & Baldwin, S. (2018). Developing computational thinking with educational technologies for young learners. *TechTrends*, *62*, 563-573. https://doi.org/10.1007/s11528-018-0292-7

Choppin, A. (1992). *Les Manuels scolaires : histoire et actualité.* Hachette.

Cohen, J. (1992). Statistical power analysis. *Current directions in psychological science*, *1*(3), 98-101. https://doi.org/10.1111/1467-8721.ep10768783

Conrads, J., Rasmussen, M., Winters, N., Geniets, A., & Langer, L. (2017). *Digital education policies in Europe and beyond: Key design principles for more effective policies*. Publications Office of the European Union. https://doi.org/10.2760/462941

CSTA and ISTE (2011). *Computational Thinking in K–12 Education leadership toolkit*. http://csta.acm.org/Curriculum/sub/CurrFiles/471.11CTLeadershiptToolkit-SP-vF.pdf

Daniela, L. (2021). Pedagogical considerations for technology-enhanced learning. In *Makers at School, Educational Robotics and Innovative Learning Environments: Research and Experiences from FabLearn Italy 2019, in the Italian Schools and Beyond* (pp. 57-64). Springer International Publishing. https://doi.org/10.1007/978-3-030-77040-2_8

Dash, C. S. K., Behera, A. K., Dehuri, S., & Ghosh, A. (2023). An outliers detection and elimination framework in classification task of data mining. *Decision Analytics Journal*, 100164. https://doi.org/10.1016/j.dajour.2023.100164

Dawes, J. (2008). Do data characteristics change according to the number of scale points used? An experiment using 5-point, 7-point and 10-point scales. *International journal of market research*, *50*(1), 61-104. https://doi.org/10.1177/147078530805000106

de Souza Oliveira, P. H., Medina, R. D., Nunes, F. B., & Herpich, F. (2022). Analysis of unplugged tools used for the development of computational thinking in the teaching of algorithms: a review of the literature. *RENOTE*, *20*(1), 243-252. https://doi.org/10.22456/1679-1916.126670

de Vries, E. (2001). Les logiciels d'apprentissage : panoplie ou éventail ? *Revue Fraçaise de Pédagogie*, *137*, 105–116. http://www.jstor.org/stable/41201747

Deci, E. L., & Ryan, R. M. (2000). The" what" and" why" of goal pursuits: Human needs and the self-determination of behavior. *Psychological inquiry*, *11*(4), 227-268. https://doi.org/10.1207/S15327965PLI1104_01

del Olmo-Muñoz, J., Bueno-Baquero, A., Cózar-Gutiérrez, R., & González-Calero, J. A. (2023). Exploring gamification approaches for enhancing computational thinking in young learners. *Education Sciences*, *13*(5), 487. https://doi.org/10.3390/educsci13050487

del Olmo-Muñoz, J., Cózar-Gutiérrez, R., & González-Calero, J. A. (2020). Computational thinking through unplugged activities in early years of Primary Education. *Computers & Education*, *150*, 103832. https://doi.org/10.1016/j.compedu.2020.103832

Delal, H., & Oner, D. (2020). Developing middle school students' computational thinking skills using unplugged computing activities. *Informatics in Education*, *19*(1), 1-13. https://doi.org/10.15388/infedu.2020.01

DeLeeuw, K. E., & Mayer, R. E. (2008). A comparison of three measures of cognitive load: Evidence for separable measures of intrinsic, extraneous, and germane load. *Journal of educational psychology*, *100*(1), 223. https://doi.org/10.1037/0022-0663.100.1.223

Denning, P. J. (2009). The profession of IT Beyond computational thinking. *Communications of the ACM*, *52*(6), 28-30. https://doi.org/10.1145/1516046.1516054

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, *60*(6), 33-39. https://doi.org/10.1145/2998438

Denning, P. J., Tedre, M., & Yongpradit, P. (2017). Misconceptions about computer science. *Communications of the ACM*, *60*(3), 31-33. https://doi.org/10.1145/3041047

Di Lieto, M. C., Inguaggiato, E., Castro, E., Cecchi, F., Cioni, G., Dell'Omo, M., ... & Dario, P. (2017). Educational robotics intervention on executive functions in preschool children: A pilot study. *Computers in human behavior*, *71*, 16-23. https://doi.org/10.1016/j.chb.2017.01.018

Dijkstra, E. W. (1974). Programming as a discipline of mathematical nature. *The American Mathematical Monthly*, *81*(6), 608-612. https://doi.org/10.1080/00029890.1974.11993624

Dos Reis, W., Sereno, H., Do Amaral, M., & Dos Reis, P. (2015). Educational robotics as an instrument of formation: a public elementary school case study. In *VI Workshop de Robótica Educacional* (Vol. 6, pp. 70-75). http://www.natalnet.br/wre2015/wre2015.pdf#page=69

Dunn, T. J., Baguley, T., & Brunsden, V. (2014). From alpha to omega: A practical solution to the pervasive problem of internal consistency estimation. *British journal of psychology*, *105*(3), 399-412. https://doi.org/10.1111/bjop.12046

El-Hamamsy, L., Zapata-Cáceres, M., Barroso, E. M., Mondada, F., Zufferey, J. D., & Bruno, B. (2022). The competent computational thinking test: Development and validation of an unplugged computational thinking test for upper primary school. *Journal of Educational Computing Research*, *60*(7), 1818-1866. https://doi.org/10.1177/07356331221081753

Fadjo, C. L., Hallman Jr, G., Harris, R., & Black, J. B. (2009, June). Surrogate embodiment, mathematics instruction and video game programming. In *EdMedia+ innovate learning* (pp. 2787-2792). Association for the Advancement of Computing in Education (AACE). https://www.learntechlib.org/primary/p/31876/

Fagerlund, J., Vesisenaho, M., & Häkkinen, P. (2022). Fourth grade students' computational thinking in pair programming with Scratch: A holistic case analysis. *International Journal of Child-Computer Interaction*, *33*, 100511. https://doi.org/10.1016/j.ijcci.2022.100511

Fidai, A., Capraro, M. M., & Capraro, R. M. (2020). "Scratch"-ing computational thinking with Arduino: A meta-analysis. *Thinking Skills and Creativity*, *38*, 100726. https://doi.org/10.1016/j.tsc.2020.100726

Fluckiger, C. (2020). Les usages effectifs du numérique en classe et dans les établissements scolaires. Cnesco-Cnam. https://www.cnesco.fr/wp-content/uploads/2020/10/201015_Cnesco_Fluckiger_Numerique_Usages-1.pdf

Forsström, S. E., & Kaufmann, O. T. (2018). A literature review exploring the use of programming in mathematics education. *International Journal of Learning, Teaching and Educational Research*, *17*(12), 18-32. https://doi.org/10.26803/ijlter.17.12.2

Galand, B. (2006). La motivation en situation d'apprentissage: les apports de la psychologie de l'éducation. *Revue française de pédagogie. Recherches en éducation*, (155), 5-8. https://doi.org/10.4000/rfp.59

Galand, B. (2020). Le numérique va-t-il révolutionner l'éducation. *Les cahiers de recherche du GIRSEF*, *120*. https://dial.uclouvain.be/pr/boreal/object/boreal%3A254506/datastream/PDF_01/view

García-Tudela, P. A., & Marín-Marín, J. A. (2023). Use of Arduino in Primary Education: A Systematic Review. *Education Sciences*, *13*(2), 134. https://doi.org/10.3390/educsci13020134

García-Valcárcel-Muñoz-Repiso, A., & Caballero-González, Y. A. (2019). Robotics to develop computational thinking in early Childhood Education. *Comunicar. Media Education Research Journal*, *27*(1). https://doi.org/10.3916/C59-2019-06

Garner, S. (2002). *Reducing the cognitive load on novice programmers* (pp. 578-583). Association for the Advancement of Computing in Education (AACE). https://www.learntechlib.org/primary/p/10329/

Gaudiello, I., & Zibetti, E. (2013). La robotique éducationnelle : état des lieux et perspectives. *Psychologie française*, *58*(1), 17-40. https://doi.org/10.1016/j.psfr.2012.09.006

Gaudiello, I., & Zibetti, E. (2019). La robotique éducative en science : pourquoi et comment. *Enfance*, (3), 309-332. https://doi.org/10.3917/enf2.193.0309

Gili, L. (2019). Les usages du numérique éducatif dans le second degré. Académie de Montpellier.

Gökçe, S., & Yenmez, A. A. (2022). Ingenuity of scratch programming on reflective thinking towards problem solving and computational thinking. *Education and Information Technologies*, 1-25. https://doi.org/10.1007/s10639-022-11385-x

Gopalan, V., Bakar, J. A. A., Zulkifli, A. N., Alwi, A., & Mat, R. C. (2017, October). A review of the motivation theories in learning. In *Aip conference proceedings* (Vol. 1891, No. 1). AIP Publishing. https://doi.org/10.1063/1.5005376

Gros, H., Gvozdic, K., Sander, E., & Scheibling-Sève, C. (2018). *Les neurosciences en éducation*. Retz.

Grover, S., Cooper, S., & Pea, R. (2014, June). Assessing computational learning in K-12. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 57-62). https://doi.org/10.1145/2591708.2591713

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational researcher*, *42*(1), 38-43. https://doi.org/10.3102/0013189X12463051

Grugier, O. (2016). Rencontre avec des nouveaux objets à écrans tactiles à l'école et moments d'éducation technologique. *Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation*, *23*(1), 133-157. 16. https://doi.org/10.3406/stice.2016.1695

Gunes, H., & Kucuk, S. (2022). A systematic review of educational robotics studies for the period 2010–2021. *Review of Education*, *10*(3), e3381. https://doi.org/10.1002/rev3.3381

Guo, M., & Ottenbreit-Leftwich, A. (2020, October). Exploring the K-12 computer science curriculum standards in the US. In *Proceedings of the 15th Workshop on Primary and Secondary Computing Education* (pp. 1-6). https://doi.org/10.1145/3421590.3421594

Hadie, S. N., & Yusoff, M. S. (2016). Assessing the validity of the cognitive load scale in a problem-based learning setting. *Journal of Taibah University Medical Sciences*, *11*(3), 194-202. https://doi.org/10.1016/j.jtumed.2016.04.001

Han, J. (2023, July). A Systematic Review of Computational Thinking Assessment in the Context of 21st Century Skills. In *2nd International Conference on Humanities, Wisdom Education and Service Management (HWESM 2023)* (pp. 271-283). Atlantis Press. https://doi.org/10.2991/978-2-38476-068-8_34

Hattie, J.A.C. (2003, October). Teachers make a difference: What is the research evidence? *In Building Teacher Quality: What does the research tell us ACER Research Conference, Melbourne, Australia.* https://ediform.santillana.com.mx/abc/docs/hattie.pdf

Hattie, J., & Timperley, H. (2007). The power of feedback. *Review of educational research, 77*(1), 81-112. https://doi.org/10.3102/003465430298487

Hermans, F., & Aivaloglou, E. (2017, November). To scratch or not to scratch? A controlled experiment comparing plugged first and unplugged first programming lessons. In *Proceedings of the 12th workshop on primary and secondary computing education* (pp. 49-56). https://doi.org/10.1145/3137065.3137072

Hessler, M., Pöpping, D. M., Hollstein, H., Ohlenburg, H., Arnemann, P. H., Massoth, C., ... & Wenk, M. (2018). Availability of cookies during an academic course session affects evaluation of teaching. *Medical Education, 52*(10), 1064-1072. https://doi.org/10.1111/medu.13627

Hickmott, D., Prieto-Rodriguez, E., & Holmes, K. (2018). A scoping review of studies on computational thinking in K–12 mathematics classrooms. *Digital Experiences in Mathematics Education*, *4*, 48-69. https://doi.org/10.1007/s40751-017-0038-8

Hu, Y., Chen, C. H., & Su, C. Y. (2021). Exploring the effectiveness and moderators of block-based visual programming on student learning: A meta-analysis. *Journal of Educational Computing Research*, *58*(8), 1467-1493. https://doi.org/10.1177/0735633120945935

Huang, W., & Looi, C. K. (2021). A critical review of literature on "unplugged" pedagogies in K-12 computer science and computational thinking education. *Computer Science Education*, *31*(1), 83-111. https://doi.org/10.1080/08993408.2020.1789411

Ioannou, A., & Makridou, E. (2018). Exploring the potentials of educational robotics in the development of computational thinking: A summary of current research and practical proposal for future work. *Education and Information Technologies*, *23*, 2531-2544. https://doi.org/10.1007/s10639-018-9729-z

Jiang, B., & Li, Z. (2021). Effect of Scratch on computational thinking skills of Chinese primary school students. *Journal of Computers in Education*, *8*(4), 505-525. https://doi.org/10.1007/s40692-021-00190-z

Judd, C. M., McClelland, G. H., Ryan, C. S., Muller, D., & Yzerbyt, V. (2018). *Analyse des données: une approche par comparaison de modèles*. De Boeck Superieur.

Julià, C., & Antolí, J. Ò. (2016). Spatial ability learning through educational robotics. *International Journal of Technology and Design Education*, *26*, 185-203. https://doi.org/10.1007/s10798-015-9307-2

Kalelioglu, F., Gulbahar, Y., & Kukul, V. (2016). A framework for computational thinking based on a systematic research review. *Baltic Journal of Modern Computing*, *4*, 583-596

Kaloti-Hallak, F., Armoni, M., & Ben-Ari, M. (2015, November). Students' attitudes and motivation during robotics activities. In *Proceedings of the workshop in primary and secondary computing education* (pp. 102-110). https://doi.org/10.1145/2818314.2818317

Kaminski, J. A., Sloutsky, V. M., & Heckler, A. F. (2006, July). Do children need concrete instantiations to learn an abstract concept. In *Proceedings of the XXVIII annual conference of the cognitive science society* (pp. 1167-1172). Erlbaum. https://bpb-us-w2.wpmucdn.com/u.osu.edu/dist/1/56827/files/2018/06/Kaminski-Sloutsky-Heckler-CogSci-2006-2kl3ovl.pdf

Kandlhofer, M., & Steinbauer, G. (2016). Evaluating the impact of educational robotics on pupils' technical-and social-skills and science related attitudes. *Robotics and Autonomous Systems*, *75*, 679-685. https://doi.org/10.1016/j.robot.2015.09.007

Karsenti, T. (2016). *Le tableau blanc interactif (TBI) : usages, avantages et défis?* CRIFPE. http://tbi.crifpe.ca/files/Rapport.pdf

Keller, J. M. (2008). First principles of motivation to learn and e3-learning. *Distance education*, *29*(2), 175-185. https//doi.org/10.1080/01587910802154970

Kert, S. B., ErkoÇ, M. F., & Yeni, S. (2020). The effect of robotics on six graders' academic achievement, computational thinking skills and conceptual knowledge levels. *Thinking Skills and Creativity*, *38*, 100714. https://doi.org/10.1016/j.tsc.2020.100714

Khaneboubi, M. (2009). Description de quelques caractéristiques communes aux opérations de dotations massives en ordinateurs portables en France. *STICEF (Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation)*, *16*, 8-pages. https://hal.science/hal-00696410

Kim, S., & Lee, C. (2016). Effects of robot for teaching geometry to fourth graders. *International Journal of Innovation in Science and Mathematics Education*, *24*(2). https://openjournals.library.sydney.edu.au/CAL/article/view/9048

Kind, P., Jones, K., & Barmby, P. (2007). Developing attitudes towards science measures. *International journal of science education*, *29*(7), 871-893. https://doi.org/10.1080/09500690600909091

Kirschner, P. A., & De Bruyckere, P. (2017). The myths of the digital native and the multitasker. *Teaching and Teacher education*, *67*, 135-142. http://dx.doi.org/10.1016/j.tate.2017.06.001

Kirschner, P. A., Sweller, J., & Clark, R. E. (2006). Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational psychologist*, *41*(2), 75-86. https://doi.org/10.1207/s15326985ep4102_1

Klepsch, M., Schmitz, F., & Seufert, T. (2017). Development and validation of two instruments measuring intrinsic, extraneous, and germane cognitive load. *Frontiers in psychology*, *8*, 1997. https://doi.org/10.3389/fpsyg.2017.01997

Kong, S. C., & Lai, M. (2022). Validating a computational thinking concepts test for primary education using item response theory: An analysis of students' responses. *Computers & Education*, *187*, 104562. https://doi.org/10.1016/j.compedu.2022.104562

Kong, S. C., & Wang, Y. Q. (2021). Item response analysis of computational thinking practices: Test characteristics and students' learning abilities in visual programming contexts. *Computers in Human Behavior*, *122*, 106836. https://doi.org/10.1016/j.chb.2021.106836

Korkmaz, Ö., & Bai, X. (2019). Adapting computational thinking scale (CTS) for Chinese high school students and their thinking scale skills level. *Participatory Educational Research*, *6*(1), 10-26. https://doi.org/10.17275/per.19.2.6.1

Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the computational thinking scales (CTS). *Computers in human behavior*, *72*, 558-569. https://doi.org/10.1016/j.chb.2017.01.005

Krathwohl, D. R. (2002). A revision of Bloom's taxonomy: An overview. *Theory into practice*, *41*(4), 212-218. https://doi.org/10.1207/s15430421tip4104_2

Kukul, V., & Karatas, S. (2019). Computational thinking self-efficacy scale: Development, validity and reliability. *Informatics in Education*, *18*(1), 151-164. https://doi.org/10.15388/infedu.2019.07

Lai, R. P., & Ellefson, M. R. (2022). How Multidimensional is Computational Thinking Competency? A Bi-factor Model of the Computational Thinking Challenge. *Journal of Educational Computing Research*, 07356331221121052. https://doi.org/10.1177/07356331221121052

Lapierre, H. G. (2017). Effet de la robotique éducative sur l'apprentissage et l'intérêt des élèves en sciences et technologies [Thèse de Doctorat, Université du Québec, Montréal].

Larousse. (s. d.). Programmation. Dans *Dictionnaire en ligne*. Consulté le 25 Mai 2023 sur https://www.larousse.fr/dictionnaires/francais/programmation/64205.

Laurent, M., Crisci, R., Bressoux, P., Chaachoua, H., Nurra, C., de Vries, E., & Tchounikine, P. (2022). Impact of programming on primary mathematics learning. *Learning and Instruction*, *82*, 101667. https://doi.org/10.1016/j.learninstruc.2022.101667

Lavonen, J. M., Meisalo, V. P., Lattu, M., & Sutinen, E. (2003). Concretising the programming task: a case study in a secondary school. *Computers & Education*, *40*(2), 115-135. https://doi.org/10.1016/S0360-1315(02)00101-X

Law, K. M., Lee, V. C., & Yu, Y. T. (2010). Learning motivation in e-learning facilitated computer programming courses. *Computers & Education*, *55*(1), 218-228. https://doi.org/10.1016/j.compedu.2010.01.007

Lee, J. (2020). Coding in early childhood. *Contemporary Issues in Early Childhood*, *21*(3), 266-269. https://doi.org/10.1177/1463949119846541

Leonard, J., Buss, A., Gamboa, R., Mitchell, M., Fashola, O. S., Hubert, T., & Almughyirah, S. (2016). Using robotics and game design to enhance children's self-efficacy, STEM attitudes, and computational thinking skills. *Journal of Science Education and Technology*, *25*, 860-876. https://doi.org/10.1007/s10956-016-9628-2

Leppink, J., Paas, F., Van der Vleuten, C. P., Van Gog, T., & Van Merriënboer, J. J. (2013). Development of an instrument for measuring different types of cognitive load. *Behavior research methods*, *45*, 1058-1072. https://doi.org/10.3758/s13428-013-0334-1

Li, Y., Xu, S., & Liu, J. (2021). Development and validation of computational thinking assessment of Chinese elementary school students. *Journal of Pacific Rim Psychology*, *15*, 18344909211010240. https://doi.org/10.1177/18344909211010240

Lim, B. L., & Chen, C. J. (2021). Computational thinking (algorithms) through unplugged programming activities: exploring upper primary students' learning experiences. *International Journal of Academic Research in Business and Social Sciences*, *11*(14), 384-403. http://dx.doi.org/10.6007/IJARBSS/v11-i14/8946

Lin, Y., & Weintrop, D. (2021). The landscape of Block-based programming: Characteristics of block-based environments and how they support the transition to text-based programming. *Journal of Computer Languages*, *67*, 101075. https://doi.org/10.1016/j.cola.2021.101075

Lishinski, A., Yadav, A., Good, J., & Enbody, R. (2016, August). Learning to program: Gender differences and interactive effects of students' motivation, goals, and self-efficacy on performance. In *Proceedings of the 2016 ACM conference on international computing education research* (pp. 211-220). https://doi.org/10.1145/2960310.2960329

Lodi, M., & Martini, S. (2021). Computational thinking, between Papert and Wing. *Science & Education*, *30*(4), 883-908. https://doi.org/10.1007/s11191-021-00202-5

Lu, Z., Chiu, M. M., Cui, Y., Mao, W., & Lei, H. (2023). Effects of game-based learning on students' computational thinking: A meta-analysis. *Journal of Educational Computing Research*, *61*(1), 235-256. https://doi.org/10.1177/07356331221100740

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12?. *Computers in Human Behavior*, *41*, 51-61. https://doi.org/10.1016/j.chb.2014.09.012

Lv, L., Zhong, B., & Liu, X. (2022). A literature review on the empirical studies of the integration of mathematics and computational thinking. *Education and Information Technologies*, 1-23. https://doi.org/10.1007/s10639-022-11518-2

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, *10*(4), 1-15. https://doi.org/10.1145/1868358.1868363

Manzano-León, A., Camacho-Lazarraga, P., Guerrero, M. A., Guerrero-Puerta, L., Aguilar-Parra, J. M., Trigueros, R., & Alias, A. (2021). Between level up and game over: A systematic literature review of gamification in education. *Sustainability*, *13*(4), 2247. https://doi.org/10.3390/su13042247

Mari, E. (2022). Potentialités de la robotique pédagogique pour le développement de connaissances spatiales à l'école [Thèse de Doctorat, Aix Marseille Université (AMU)].

Master, A., Cheryan, S., Moscatelli, A., & Meltzoff, A. N. (2017). Programming experience promotes higher STEM motivation among first-grade girls. *Journal of experimental child psychology*, *160*, 92-106. https://doi.org/10.1016/j.jecp.2017.03.013

Mayer, R. E. (Ed.). (1988). *Teaching and learning computer programming: Multiple research perspectives.* Lawrence Erlbaum Associates, Inc.

McGill, M. M. (2012). Learning to program with personal robots: Influences on student motivation. *ACM Transactions on Computing Education (TOCE)*, *12*(1), 1-32. https://doi.org/10.1145/2133797.2133801

McNeil, N. M., Uttal, D. H., Jarvin, L., & Sternberg, R. J. (2009). Should you show me the money? Concrete objects both hurt and help performance on mathematics problems. *Learning and instruction*, *19*(2), 171-184. https://doi.org/10.1016/j.learninstruc.2008.03.005

Merkouris, A., & Chorianopoulos, K. (2015, November). Introducing computer programming to children through robotic and wearable devices. In *Proceedings of the workshop in primary and secondary computing education* (pp. 69-72). https://doi.org/10.1145/2818314.2818342

Merkouris, A., & Chorianopoulos, K. (2019). Programming embodied interactions with a remotely controlled educational robot. *ACM Transactions on Computing Education (TOCE)*, *19*(4), 1-19. https://doi.org/10.1145/3336126

Merkouris, A., Chorianopoulos, K., & Kameas, A. (2017). Teaching programming in secondary education through embodied computing platforms: Robotics and wearables. *ACM Transactions on Computing Education (TOCE)*, *17*(2), 1-22. https://doi.org/10.1145/3025013

Messer, D., Thomas, L., Holliman, A., & Kucirkova, N. (2018). Evaluating the effectiveness of an educational programming intervention on children's mathematics skills, spatial awareness and working memory. *Education and Information Technologies*, *23*, 2879-2888. https://doi.org/10.1007/s10639-018-9747-x

Ministère de l'Education nationale, de l'enseignement supérieur et de la recherche (2015). Socle commun de connaissances, de compétences et de culture. Décret n° 2015-372. *Bulletin officiel de l'Éducation nationale, de l'enseignement supérieur et de la recherche, 23 avril*.

Ministère de l'Education nationale, de l'enseignement supérieur et de la recherche (2023). Programme du cycle 3 en vigueur à la rentrée 2023. *Bulletin officiel de l'Éducation nationale, de l'enseignement supérieur et de la recherche, 22 juin.*

Mohaghegh, D. M., & McCauley, M. (2016). Computational thinking: The skill set of the 21st century. *International Journal of Computer Science and Information Technologies*, *7*(3), 1524-1530. https://ijcsit.com/docs/Volume%207/vol7issue3/ijcsit20160703104.pdf

Moraiti, I., Fotoglou, A., & Drigas, A. (2022). Coding with block programming languages in educational robotics and mobiles, improve problem solving, creativity & critical thinking skills. *International Journal of Interactive Mobile Technologies*, *16*(20). https://doi.org/10.3991/ijim.v16i20.34247

Moreno-León, J., & Robles, G. (2015, November). Dr. Scratch: A web tool to automatically evaluate Scratch projects. In *Proceedings of the workshop in primary and secondary computing education* (pp. 132-133). https://doi.org/10.1145/2818314.2818338

Moreno-León, J., & Robles, G. (2016, April). Code to learn with Scratch? A systematic literature review. In *2016 IEEE Global Engineering Education Conference (EDUCON)* (pp. 150-156). IEEE. https://doi.org/10.1109/EDUCON.2016.7474546

Moreno-León, J., Román-González, M., Harteveld, C., & Robles, G. (2017, May). On the automatic assessment of computational thinking skills: A comparison with human experts. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (pp. 2788-2795). https://doi.org/10.1145/3027063.3053216

Moreno-León, J., Román-González, M., & Robles, G. (2018, April). On computational thinking as a universal skill: A review of the latest research on this ability. In *2018 IEEE Global Engineering Education Conference (EDUCON)* (pp. 1684-1689). IEEE. https://doi.org/10.1109/EDUCON.2018.8363437

Morris, N. P., Ramsay, L., & Chauhan, V. (2012). Can a tablet device alter undergraduate science students' study behavior and use of technology?. *Advances in Physiology Education*, *36*(2), 97-107. https://doi.org/10.1152/advan.00104.2011

Muradoglu, M., Cimpian, J. R., & Cimpian, A. (2023). Mixed-Effects Models for Cognitive Development Researchers. *Journal of Cognition and Development*, 1-34. https://doi.org/10.1080/15248372.2023.2176856

Mutlu-Bayraktar, D., Cosgun, V., & Altan, T. (2019). Cognitive load in multimedia learning environments: A systematic review. *Computers & Education*, *141*, 103618. https://doi.org/10.1016/j.compedu.2019.103618

Namli, N.A., Aybek, B. (2022). An Investigation of The Effect of Block-Based Programming and Unplugged Coding Activities on Fifth Graders' Computational Thinking Skills, Self-Efficacy and Academic Performance. *Contemporary Educational Technology, 14* (1). https://doi.org/10.30935/cedtech/11477

Nijimbere, C., Laetitia, B. H., Haspekian, M., & Baron, G. L. (2013). Apprendre l'informatique par la programmation des robots. In *Sciences et technologies de l'information et de la communication (STIC) en milieu éducatif*. https://edutice.archives-ouvertes.fr/file/index/docid/875586/filename/D5_NIjimbere.pdf

Nogry, S. (2018). Comment apprennent les élèves au cours d'une séquence de robotique éducative en classe de CP?. *De 0 à 1 ou l'heure de l'informatique à l'école*, *235*. https://doi.org/10.3726/b13411

Noh, J., & Lee, J. (2020). Effects of robotics programming on the computational thinking and creativity of elementary school students. *Educational technology research and development*, *68*, 463-484. https://doi.org/10.1007/s11423-019-09708-w

Nordby, S. K., Bjerke, A. H., & Mifsud, L. (2022). Computational thinking in the primary mathematics classroom: A systematic review. *Digital Experiences in Mathematics Education*, *8*(1), 27-49. https://doi.org/10.1007/s40751-022-00102-5

Nouri, J., Zhang, L., Mannila, L., & Norén, E. (2020). Development of computational thinking, digital competence and 21st century skills when learning programming in K-9. *Education Inquiry*, *11*(1), 1-17. https://doi.org/10.1080/20004508.2019.1627844

Olabe, J. C., Olabe, M. A., Basogain, X., & Castaño, C. (2011). Programming and robotics with Scratch in primary education. *Education in a technological world: Communicating current and emerging research and technological efforts*, 356-363.

Ospennikova, E., Ershov, M., & Iljin, I. (2015). Educational robotics as an inovative educational technology. *Procedia-Social and Behavioral Sciences*, *214*, 18-26. https://doi.org/10.1016/j.sbspro.2015.11.588

Ouahbi, I., Kaddari, F., Darhmaoui, H., Elachqar, A., & Lahmine, S. (2015). Learning basic programming concepts by creating games with scratch programming environment. *Procedia-Social and Behavioral Sciences*, *191*, 1479-1482. https://doi.org/10.1016/j.sbspro.2015.04.224

Ouwehand, K., Kroef, A. V. D., Wong, J., & Paas, F. (2021, September). Measuring cognitive load: Are there more valid alternatives to Likert rating scales?. In *Frontiers in Education* (Vol. 6, p. 702616). Frontiers Media SA. https://doi.org/10.3389/feduc.2021.702616

Oviatt, S. L., & Cohen, A. O. (2010). Toward high-performance communications interfaces for science problem solving. *Journal of science education and technology*, *19*, 515-531. https://doi.org/10.1007/s10956-010-9218-7

Paas, F. G. (1992). Training strategies for attaining transfer of problem-solving skill in statistics: a cognitive-load approach. *Journal of educational psychology*, *84*(4), 429. https://doi.org/10.1037/0022-0663.84.4.429

Paas, F., Tuovinen, J. E., Tabbers, H., & Van Gerven, P. W. (2003). Cognitive load measurement as a means to advance cognitive load theory. *Educational psychologist*, *38*(1), 63-71. https://doi.org/10.1207/S15326985EP3801_8

Paas, F. G., Van Merriënboer, J. J., & Adam, J. J. (1994). Measurement of cognitive load in instructional research. *Perceptual and motor skills*, *79*(1), 419-430. https://doi.org/10.2466/pms.1994.79.1.419

Papadakis, S., Kalogiannakis, M., & Zaranis, N. (2016). Comparing tablets and PCs in teaching mathematics: An attempt to improve mathematics competence in early childhood education. *Preschool and Primary Education*, *4*(2), 241-253. https://www.learntechlib.org/p/187376/

Papert, S. A. (1980). Mindstorms: Children, computers, and powerful ideas. Basic books.

Parent, S. (2022). Pensée informatique: portrait conceptuel des aspects inhérents à la programmation en contexte scolaire. *Formation et profession*, *30*(2), 1-15. https://doi.org/10.18162/fp.2022.651.

Park, I., Kim, D., Oh, J., Jang, Y., & Lim, K. (2015). Learning effects of pedagogical robots with programming in elementary school environments in Korea. *Indian Journal of Science and Technology*, *8*(26), 1-5.

Passey, D., Rogers, C., Machell, J., McHugh, G. & Allaway, D. (2004). The motivational effect of ICT on pupils. Department of Educational Research, Lancaster University.

Pedaste, M., & Altin, H. (2020). Does inquiry-based education using robots have an effect on learners' inquiry skills, subject knowledge and skills, and motivation?. *International Journal on Advanced Science Engineering and Information Technology*, *10*(4), 1403-1409. https://doi.org/10.18517/ijaseit.10.4.12766

Pedro, L. Z., Lopes, A. M., Prates, B. G., Vassileva, J., & Isotani, S. (2015, April). Does gamification work for boys and girls? An exploratory study with a virtual learning environment. In *Proceedings of the 30th annual ACM symposium on applied computing* (pp. 214-219). http://dx.doi.org/ 10.1145/2695664.2695752

Perkins, D. N., & Salomon, G. (1992). Transfer of learning. *International encyclopedia of education*, *2*, 6452-6457.

Piedade, J., & Dorotea, N. (2022). Effects of Scratch-based activities on 4th-grade students' computational thinking skills. *Informatics in Education*. https://doi.org/10.15388/infedu.2023.19

Pisarov, J., & Mester, G. (2019, December). Programming the mbot robot in school. In *Proceedings of the International Conference and Workshop Mechatronics in Practice and Education, MechEdu* (pp. 45-48).

Polit, D. F. (2014). Getting serious about test–retest reliability: a critique of retest research and some recommendations. *Quality of Life Research*, *23*, 1713-1720. https://doi.org/10.1007/s11136-014-0632-9

Popat, S., & Starkey, L. (2019). Learning to code or coding to learn? A systematic review. *Computers & Education*, *128*, 365-376. https://doi.org/10.1016/j.compedu.2018.10.005

Psycharis, S., & Kallia, M. (2017). The effects of computer programming on high school students' reasoning skills and mathematical self-efficacy and problem solving. *Instructional science*, *45*(5), 583-602. https://doi.org/10.1007/s11251-017-9421-5

Qian, Y., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)*, *18*(1), 1-24. https://doi.org/10.1145/3077618

Rahman, M. M. (2019). 21st century skill'problem solving': Defining the concept. Rahman, MM (2019). 21st Century Skill "Problem Solving": Defining the Concept. Asian Journal of Interdisciplinary Research, 2(1), 64-74. https://doi.org/10.34256/ajir1917

Ravestein, J., & Ladage, C. (2014). Ordinateurs et Internet à l'école élémentaire française: Usages déclarés de 907 professeurs d'école. *Éducation et didactique*, *3*, 9-21. https://doi.org/10.4000/educationdidactique.2008

Refvik, K. A. & Bjerke, A. H. (2022). Computational thinking as a tool in primary and secondary mathematical problem solving: a literature review. *Nordic Studies in Education. 27.* 5-27. https://ncm.gu.se/wp-content/uploads/2022/09/27_3_005028_refvik.pdf

Relkin, E., de Ruiter, L., & Bers, M. U. (2020). TechCheck: Development and validation of an unplugged assessment of computational thinking in early childhood education. *Journal of Science Education and Technology*, *29*(4), 482-498. https://doi.org/10.1007/s10956-020-09831-x

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., ... & Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, *52*(11), 60-67. https://doi.org/10.1145/1592761.1592779

Resnick, M., & Siegel, D. (2015). A different approach to coding. *International Journal of People-Oriented Programming*, *4*(1), 1-4. https://educators.brainpop.com/wp-content/uploads/2018/07/A-Different-Approach-to-Coding-Resnick-article-1.pdf

Riedo, F., Chevalier, M., Magnenat, S., & Mondada, F. (2013, November). Thymio II, a robot that grows wiser with children. In 2013 IEEE workshop on advanced robotics and its social impacts (pp. 187-193). IEEE. https://doi.org/10.1109/ARSO.2013.6705527

Rino, R., Siti Irene, A. D., Ariyadi, W., Heri, R., Andi, W., Dyahsih, A. S., & Khasanah, N. H. The impact of STEM attitudes and computational thinking on 21st-century via structural equation modelling. *International Journal of Evaluation and Research in Education (IJERE)*. http://elibrary.almaata.ac.id/id/eprint/3421

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer science education*, *13*(2), 137-172. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=64644ccecc3db4e104e4d3ba2634563cea9cc145

Rodger, S. H., Bashford, M., Dyck, L., Hayes, J., Liang, L., Nelson, D., & Qin, H. (2010, June). Enhancing K-12 education with alice programming adventures. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education* (pp. 234-238). https://doi.org/10.1145/1822090.1822156

Rogalski, J., & Samurçay, R. (1990). Acquisition of programming knowledge and skills. In *Psychology of programming* (pp. 157-174). Academic Press. https://www.cl.cam.ac.uk/teaching/1011/R201/ppig-book/ch2-4.pdf

Rogalski, J., Samurçay, R., & Hoc, J. M. (1988). L'apprentissage des méthodes de programmation comme méthodes de résolution de problème. *Le travail humain*, 309-320. https://www.jstor.org/stable/40657502

Román-González, M. (2015). Computational thinking test: Design guidelines and content validation. In *EDULEARN15 Proceedings* (pp. 2436-2444).

Román-González, M., Moreno-León, J., & Robles, G. (2019). Combining assessment tools for a comprehensive evaluation of computational thinking interventions. *Computational thinking education*, 79-98. S.-C. https://doi.org/10.1007/978-981-13-6528-7_6

Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in human behavior*, *72*, 678-691. https://doi.org/10.1016/j.chb.2016.08.047

Román-González, M., Pérez-González, J. C., Moreno-León, J., & Robles, G. (2018a). Extending the nomological network of computational thinking with non-cognitive factors. *Computers in Human Behavior*, *80*, 441-459. https://doi.org/10.1016/j.chb.2017.09.030

Román-González, M., Pérez-González, J. C., Moreno-León, J., & Robles, G. (2018b). Can computational talent be detected? Predictive validity of the Computational Thinking Test. *International Journal of Child-Computer Interaction*, *18*, 47-58. https://doi.org/10.1016/j.ijcci.2018.06.004

Romero, M., Lille, B., Viéville, T., Duflot-Kremer, M., de Smet, C., & Belhassein, D. (2018, Août). Analyse comparative d'une activité d'apprentissage de la programmation en mode branché et débranché. Dans *Educode-Conférence internationale sur l'enseignement au*

numérique et par le numérique. https://inria.hal.science/hal-01861732/file/Educode%20_UnPlugged-CDS06_soumis.pdf

Romero, M., & Sanabria, J. (2017). Des projets de robotique pédagogique pour le développement des compétences du XXIe siècle. Usages créatifs du numérique pour l'apprentissage au XXIe siècle. Dirigé par: Margarida Romero, Benjamin Lille et Azeneth Patiño. Presses de l'Université du Québec.

Rowe, E., Almeda, M. V., Asbell-Clarke, J., Scruggs, R., Baker, R., Bardar, E., & Gasca, S. (2021). Assessing implicit computational thinking in Zoombinis puzzle gameplay. *Computers in Human Behavior*, *120*, 106707. https://doi.org/10.1016/j.chb.2021.106707

Royeen, C. B. (1985). Adaptation of Likert scaling for use with children. *The Occupational Therapy Journal of Research*, *5*(1), 59-69. https://doi.org/10.1177/153944928500500104

Rubio, M. A., Romero-Zaliz, R., Mañoso, C., & de Madrid, A. P. (2015). Closing the gender gap in an introductory programming course. *Computers & Education*, *82*, 409-420. https://doi.org/10.1016/j.compedu.2014.12.003

Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education*, *97*, 129-141. https://doi.org/10.1016/j.compedu.2016.03.003

Sáez-López, J. M., Sevillano-García, M. L., & Vazquez-Cano, E. (2019). The effect of programming on primary school students' mathematical and scientific understanding: educational use of mBot. *Educational Technology Research and Development*, *67*, 1405-1425. https://doi.org/10.1007/s11423-019-09648-5

Safitri, N., Putra, Z. H., Alim, J. A., & Aljarrah, A. (2023). The relationship between self-efficacy and computational thinking skills of fifth grade elementary school students. *Jurnal Elemen*, *9*(2), 424-439. https://doi.org/10.29408/jel.v9i2.12299

Samurçay, R., & Rouchier, A. (1985). De «faire» à «faire faire»: planification d'actions dans la situation de programmation. *Enfance*, *38*(2), 241-254. Accessible à l'adresse suivante : https://www.persee.fr/doc/enfan_0013-7545_1985_num_38_2_2883

Sarıtepeci, M., & Durak, H. (2017). Analyzing the effect of block and robotic coding activities on computational thinking in programming education. *Educational research and practice*, *490*, 501.

Scherer, R., Siddiq, F., & Sánchez Viveros, B. (2019). The cognitive benefits of learning computer programming: A meta-analysis of transfer effects. *Journal of Educational Psychology*, *111*(5), 764. https://doi.org/10.1037/edu0000314

Scherer, R., Siddiq, F., & Viveros, B. S. (2020). A meta-analysis of teaching and learning computer programming: Effective instructional approaches and conditions. *Computers in Human Behavior*, *109*, 106349. https://doi.org/10.1016/j.chb.2020.106349

Schmeck, A., Opfermann, M., Van Gog, T., Paas, F., & Leutner, D. (2015). Measuring cognitive load with subjective rating scales during problem solving: differences between immediate and delayed ratings. *Instructional Science*, *43*, 93-114. https://doi.org/10.1007/s11251-014-9328-3

Selby, C., & Woollard, J. (2013). Computational thinking: the developing definition. In *Paper presented at the 18th annual conference on innovation and technology in computer science education, Canterbury*. http://eprints.soton.ac.uk/id/eprint/356481

Senkbeil, M., & Ihme, J. M. (2017). Motivational factors predicting ICT literacy: First evidence on the structure of an ICT motivation inventory. *Computers & Education*, *108*, 145-158. https://doi.org/10.1016/j.compedu.2017.02.003

Seraj, M., Katterfeldt, E. S., Bub, K., Autexier, S., & Drechsler, R. (2019, November). Scratch and Google Blockly: How girls' programming skills and attitudes are influenced. In *Proceedings of the 19th Koli Calling International Conference on Computing Education Research* (pp. 1-10). https://doi.org/10.1145/3364510.3364515

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational research review*, *22*, 142-158. https://doi.org/10.1016/j.edurev.2017.09.003

Siegfried, R., Klinger, S., Gross, M., Sumner, R. W., Mondada, F., & Magnenat, S. (2017, June). Improved mobile robot programming performance through real-time program assessment. In *Proceedings of the 2017 ACM conference on innovation and technology in computer science education* (pp. 341-346). https://doi.org/10.1145/3059009.3059044

Sigayret, K., Blanc, N., & Tricot, A. (2022). L'apprentissage de la programmation: quels outils pour évaluer le développement de la pensée informatique à l'école?. *Enfance*, *4*(4), 479-500. https://doi.org/10.3917/enf2.224.0479

Sigayret, K., Tricot, A., & Blanc, N. (2022). Unplugged or plugged-in programming learning: A comparative experimental study. *Computers & Education*, *184*, 104505. https://doi.org/10.1016/j.compedu.2022.104505

Sırakaya, M., Alsancak Sırakaya, D., & Korkmaz, Ö. (2020). The impact of STEM attitude and thinking style on computational thinking determined via structural equation modeling. *Journal of Science Education and Technology*, *29*, 561-572. https://doi.org/10.1007/s10956-020-09836-6

Socratous, C., & Ioannou, A. (2019). Using educational robotics as tools for metacognition: an empirical study in elementary stem education. In *Immersive Learning Research Network Conference, UK* (pp. 64-75). https://doi.org/10.3217/978-3-85125-657-4-11

Sommet, N., & Morselli, D. (2021). Keep calm and learn multilevel linear modeling: A three-step procedure using SPSS, Stata, R, and MPlus. *International Review of Social Psychology*, *34*(1). https://doi.org/10.5334/irsp.555

Spach, M. (2017). *Activités robotiques à l'école primaire et apprentissage de concepts informatiques: quelle place du scénario pédagogique? Les limites du co-apprentissage* [Thèse de Doctorat, Université Sorbonne Paris Cité].

Spolaôr, N., & Benitti, F. B. V. (2017). Robotics applications grounded in learning theories on tertiary education: A systematic review. *Computers & Education*, *112*, 97-107. https://doi.org/10.1016/j.compedu.2017.05.001

Stewart, W., & Baek, K. (2023). Analyzing computational thinking studies in Scratch programming: A review of elementary education literature. *International Journal of Computer Science Education in Schools*, *6*(1), 35-58. https://doi.org/10.21585/ijcses.v6i1.156

Strawhacker, A., Lee, M., & Bers, M. U. (2018). Teaching tools, teachers' rules: Exploring the impact of teaching styles on young children's programming knowledge in ScratchJr. *International Journal of Technology and Design Education*, *28*, 347-376. https://doi.org/10.1007/s10798-017-9400-9

Su, J., & Yang, W. (2023). A Systematic Review of Integrating Computational Thinking in Early Childhood Education. *Computers and Education Open*, 100122. https://doi.org/10.1016/j.caeo.2023.100122

Suárez, C. A. H., Suárez, A. A. G., & Núñez, R. P. (2022). Computational Thinking And Development Of Cognitive Skills Mediated By Unplugged Activities. *Journal of Language and Linguistic Studies*, *18*(2). http://www.jlls.org/index.php/jlls/article/view/4792/1568

Sukirman, Pramudita, D. A., Afiyanto, A., & Utaminingsih, (2022). Block-Based Visual Programming as a Tool for Learning the Concepts of Programming for Novices. *International Journal of Information and Education Technology*, 12(5). https://doi.org/10.18178/ijiet.2022.12.5.1628

Sullivan, A., & Bers, M. U. (2013). Gender differences in kindergarteners' robotics and programming achievement. *International journal of technology and design education*, *23*, 691-702. https://doi.org/10.1007/s10798-012-9210-z

Sullivan, F. R. (2008). Robotics and science literacy: Thinking skills, science process skills and systems understanding. *Journal of Research in Science Teaching: The Official Journal of the National Association for Research in Science Teaching*, *45*(3), 373-394. https://doi.org/10.1002/tea.20238

Sullivan, F. R., & Heffernan, J. (2016). Robotic construction kits as computational manipulatives for learning in the STEM disciplines. *Journal of Research on Technology in Education*, *48*(2), 105-128. https://doi.org/10.1080/15391523.2016.1146563

Sun, L., Guo, Z., & Hu, L. (2021). Educational games promote the development of students' computational thinking: a meta-analytic review. *Interactive Learning Environments*, 1-15. https://doi.org/10.1080/10494820.2021.1931891

Sun, L., Guo, Z., & Zhou, D. (2022). Developing K-12 students' programming ability: A systematic literature review. *Education and Information Technologies*, *27*(5), 7059-7097. https://doi.org/10.1007/s10639-022-10891-2

Sun, L., Hu, L., & Zhou, D. (2021). Which way of design programming activities is more effective to promote K-12 students' computational thinking skills? A meta-analysis. *Journal of Computer assisted learning*, *37*(4), 1048-1062. https://doi.org/10.1111/jcal.12545

Sun, L., Hu, L., & Zhou, D. (2022). Single or combined? A study on programming to promote junior high school students' computational thinking skills. *Journal of Educational Computing Research*, *60*(2), 283-321. https://doi.org/10.1177/07356331211035182

Sung, J. (2022). Assessing young Korean children's computational thinking: A validation study of two measurements. *Education and Information Technologies*, 1-29. https://doi.org/10.1007/s10639-022-11137-x

Sung, E., & Mayer, R. E. (2013). Online multimedia learning with mobile devices and desktop computers: An experimental test of Clark's methods-not-media hypothesis. *Computers in Human Behavior*, *29*(3), 639-647. https://doi.org/10.1016/j.chb.2012.10.022

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive science*, *12*(2), 257-285. https://doi.org/10.1016/0364-0213(88)90023-7

Sweller, J. (2018). Measuring cognitive load. *Perspectives on medical education*, *7*, 1-2. https://doi.org/10.1007/s40037-017-0395-4

Sweller, J., Van Merrienboer, J. J., & Paas, F. G. (1998). Cognitive architecture and instructional design. *Educational psychology review*, 251-296. https://www.jstor.org/stable/23359412

Sweller, J., van Merriënboer, J. J., & Paas, F. (2019). Cognitive architecture and instructional design: 20 years later. *Educational Psychology Review*, *31*, 261-292. https://doi.org/10.1007/s10648-019-09465-5

Talan, T. (2021). The Effect of Educational Robotic Applications on Academic Achievement: A Meta-Analysis Study. *International Journal of Technology in Education and Science*, *5*(4), 512-526. https://doi.org/10.46328/ijtes.242

Tamim, R. M., Borokhovski, E., Bernard, R. M., Schmid, R. F., & Abrami, P. C. (2015, April). A methodological quality tool for meta-analysis: The case of the educational technology literature. In *annual meeting of the American educational research association, Chicago, IL*.

Tang, X., Yin, Y., Lin, Q., Hadad, R., & Zhai, X. (2020). Assessing computational thinking: A systematic review of empirical studies. *Computers & Education*, *148*, 103798. https://doi.org/10.1016/j.compedu.2019.103798

Taslibeyaz, E., Kursun, E., & Karaman, S. (2020). How to Develop Computational Thinking: A Systematic Review of Empirical Studies. *Informatics in Education*, *19*(4), 701-719. https://doi.org/10.15388/infedu.2020.30

Taylor, R. (1990). Interpretation of the correlation coefficient: a basic review. *Journal of diagnostic medical sonography*, *6*(1), 35-39. https://doi.org/10.1177/875647939000600106

Tchounikine, P. (2017). Initier les élèves à la pensée informatique et à la programmation avec Scratch. *Laboratoire d'informatique de Grenoble*. https://pdfbib.com/pdf/0690-initier-les-eleves-a-la-programmation-avec-scratch.pdf

Threekunprapa, A., & Yasri, P. (2020). Unplugged Coding Using Flowblocks for Promoting Computational Thinking and Programming among Secondary School Students. *International Journal of Instruction*, *13*(3), 207-222. https://doi.org/10.29333/iji.2020.13314a

Thurstone, L. L. (1938). Primary mental abilities. University of Chicago Press.

Tikva, C., & Tambouris, E. (2021). Mapping computational thinking through programming in K-12 education: A conceptual model based on a systematic literature Review. *Computers & Education*, *162*, 104083. https://doi.org/10.1016/j.compedu.2020.104083

Tobias, S., & Duffy, T. M. (Eds.). (2009). *Constructivist instruction: Success or failure?*. Routledge.

Tocháček, D., Lapeš, J., & Fuglík, V. (2016). Developing technological knowledge and programming skills of secondary schools students through the educational robotics projects. *Procedia-Social and Behavioral Sciences*, *217*, 377-381. https://doi.org/10.1016/j.sbspro.2016.02.107

Tricot, A. (2020). Quelles fonctions pédagogiques bénéficient des apports du numérique ? Cnesco-Cnam. https://hal-cnam.archives-ouvertes.fr/hal-03249545v1/file/210218_Cnesco_Tricot_Numerique_Fonctions_pedagogiques.pdf

Tricot, A., & Musial, M. (2020). *Précis d'ingénierie pédagogique*. De Boeck Supérieur.

Trizano-Hermosilla, I., & Alvarado, J. M. (2016). Best alternatives to Cronbach's alpha reliability in realistic conditions: congeneric and asymmetrical measurements. *Frontiers in psychology*, *7*, 769. https://doi.org/10.3389/fpsyg.2016.00769

Tsai, C. Y. (2019). Improving students' understanding of basic programming concepts through visual programming language: The role of self-efficacy. *Computers in Human Behavior*, *95*, 224-232. https://doi.org/10.1016/j.chb.2018.11.038

Tsarava, K., Leifheit, L., Ninaus, M., Román-González, M., Butz, M. V., Golle, J., ... & Moeller, K. (2019, October). Cognitive correlates of computational thinking: Evaluation of a blended unplugged/plugged-in course. In *Proceedings of the 14th workshop in primary and secondary computing education* (pp. 1-9). https://doi.org/10.1145/3361721.3361729

Umapathy, K., & Ritzhaupt, A. D. (2017). A meta-analysis of pair-programming in computer programming courses: Implications for educational practice. *ACM Transactions on Computing Education (TOCE)*, *17*(4), 1-13. https://doi.org/10.1145/2996201

Uttl, B., White, C. A., & Gonzalez, D. W. (2017). Meta-analysis of faculty's teaching effectiveness: Student evaluation of teaching ratings and student learning are not related. *Studies in Educational Evaluation*, *54*, 22-42. https://doi.org/10.1016/j.stueduc.2016.08.007

Valls Pou, A., Canaleta, X., & Fonseca, D. (2022). Computational Thinking and Educational Robotics Integrated into Project-Based Learning. *Sensors*, *22*(10), 3746. https://doi.org/10.3390/s22103746

Van Eerde, W., & Thierry, H. (1996). Vroom's expectancy models and work-related criteria: A meta-analysis. *Journal of applied psychology*, *81*(5), 575. https://doi.org/10.1037/0021-9010.81.5.575

Van Lint, S. (2016). La notion de compétence et son évaluation. *Revue technologie*, *202*. https://eduscol.education.fr/sti/sites/eduscol.education.fr.sti/files/ressources/techniques/9999/9999-202-p30.pdf

Vargová, M., & Círus, L. (2021). The Use of a Bee-bot in Pre-primary and Primary Education. *Journal of EducatIon, Technology and Computer Science*, *12*(2 (32), 45-50. https://doi.org/10.15584/jetacomps.2021.2.5

Villemonteix, F., & Khaneboubi, M. (2013). Étude exploratoire sur l'utilisation d'iPads en milieu scolaire: entre séduction ergonomique et nécessités pédagogiques. *Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation*, *20*(1), 445-464. https://doi.org/10.3406/stice.2013.1078

Villemonteix, F., & Nogry, S. (2016). Usages de tablettes à l'école primaire: quelles contraintes sur l'activité pédagogique?. *Recherche & formation*, 79-92. https://doi.org/10.4000/rechercheformation.2628

Wang, C., Shen, J., & Chao, J. (2022). Integrating computational thinking in STEM education: A literature review. *International Journal of Science and Mathematics Education*, *20*(8), 1949-1972. https://doi.org/10.1007/s10763-021-10227-5

Warschauer, M., Cotten, S. R., & Ames, M. G. (2011). One laptop per child Birmingham: Case study of a radical experiment. *International Journal of Learning and Media*, *3*(2). https://doi.org/10.1162/IJLM_a_00069

Webb, M., Davis, N., Bell, T., Katz, Y. J., Reynolds, N., Chambers, D. P., & Sysło, M. M. (2017). Computer science in K-12 school curricula of the 2lst century: Why, what and when?. *Education and Information Technologies*, *22*, 445-468. https://doi.org/10.1007/s10639-016-9493-x

Weese, J. L., & Feldhausen, R. (2017, June). STEM outreach: Assessing computational thinking and problem solving. In *2017 ASEE Annual Conference & Exposition*. https://doi.org/10.18260/1-2--28845

Weinburgh, M. H., & Steele, D. (2000). The modified attitudes toward science inventory: Developing an instrument to be used with fifth grade urban students. *Journal of Women and Minorities in Science and Engineering*, *6*(1). https://doi.org/10.1615/JWomenMinorScienEng.v6.i1.50

Weintrop, D. (2019). Block-based programming in computer science education. *Communications of the ACM*, *62*(8), 22-25. https://doi.org/10.1145/3341221

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of science education and technology*, *25*, 127-147. https://doi.org/10.1007/s10956-015-9581-5

Weintrop, D., & Wilensky, U. (2015, June). To block or not to block, that is the question: students' perceptions of blocks-based programming. In *Proceedings of the 14th international conference on interaction design and children* (pp. 199-208). https://doi.org/10.1145/2771839.2771860

Weintrop, D., & Wilensky, U. (2019). Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms. *Computers & Education*, *142*, 103646. https://doi.org/10.1016/j.compedu.2019.103646

Werner, L., Campe, S., & Denner, J. (2012, February). Children learning computer science concepts via Alice game-programming. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 427-432). https://doi.org/10.1145/2157136.2157263

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35. https://doi.org/10.1145/1118178.1118215

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717-3725. https://doi.org/10.1098/rsta.2008.0118

Wing, J. (2011). Research notebook: Computational thinking—What and why. *The link magazine*, *6*, 20-23. https://people.cs.vt.edu/~kafura/CS6604/Papers/CT-What-And-Why.pdf

Wohl, B., Porter, B., & Clinch, S. (2015, November). Teaching Computer Science to 5-7 year-olds: An initial study with Scratch, Cubelets and unplugged computing. In *Proceedings of the workshop in primary and secondary computing education* (pp. 55-60). https://doi.org/10.1145/2818314.2818340

Wu, S. Y., & Su, Y. S. (2021). Visual programming environments and computational thinking performance of fifth-and sixth-grade students. *Journal of Educational Computing Research*, *59*(6), 1075-1092. https://doi.org/10.1177/0735633120988807

Wu, W. R., & Yang, K. L. (2022). The relationships between computational and mathematical thinking: A review study on tasks. *Cogent Education*, *9*(1), 2098929. https://doi.org/10.1080/2331186X.2022.2098929

Xu, Z., Ritzhaupt, A. D., Tian, F., & Umapathy, K. (2019). Block-based versus text-based programming environments on novice student learning outcomes: A meta-analysis study. *Computer Science Education*, *29*(2-3), 177-204. https://doi.org/10.1080/08993408.2019.1565233

Yadav, A., Good, J., Voogt, J., & Fisser, P. (2017). Computational thinking as an emerging competence domain. *Competence-based vocational and professional education: Bridging the worlds of work and education*, 1051-1067. https://doi.org/10.1007/978-3-319-41713-4_49

Ye, H., Liang, B., Ng, O. L., & Chai, C. S. (2023). Integration of computational thinking in K-12 mathematics education: a systematic review on CT-based mathematics instruction and student learning. *International Journal of STEM Education*, *10*(1), 1-26. https://doi.org/10.1186/s40594-023-00396-w

Zawieska, K., & Duffy, B. R. (2015). The social construction of creativity in educational robotics. In *Progress in Automation, Robotics and Measuring Techniques: Volume 2 Robotics* (pp. 329-338). Springer International Publishing. https://doi.org/10.1007/978-3-319-15847-1_32

Zeng, Y., Yang, W., & Bautista, A. (2023). Computational thinking in early childhood education: Reviewing the literature and redeveloping the three-dimensional framework. *Educational Research Review*, 100520. https://doi.org/10.1016/j.edurev.2023.100520

Zhan, Z., He, L., Tong, Y., Liang, X., Guo, S., & Lan, X. (2022). The effectiveness of gamification in programming education: Evidence from a meta-analysis. *Computers and Education: Artificial Intelligence*, 100096. https://doi.org/10.1016/j.caeai.2022.100096

Zhan, Z., He, W., Yi, X., & Ma, S. (2022). Effect of unplugged programming teaching aids on children's computational thinking and classroom interaction: With respect to Piaget's four stages theory. *Journal of Educational Computing Research*, *60*(5), 1277-1300. https://doi.org/10.1177/07356331211057143

Zhang, L., & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. *Computers & Education*, *141*, 103607. https://doi.org/10.1016/j.compedu.2019.103607

Zhang, Y., Luo, R., Zhu, Y., & Yin, Y. (2021). Educational robots improve K-12 students' computational thinking and STEM attitudes: systematic review. *Journal of Educational Computing Research*, *59*(7), 1450-1481. https://doi.org/10.1177/0735633121994070

Zheng, R. Z., & Greenberg, K. (2017). The boundary of different approaches in cognitive load measurement: strengths and limitations. *Cognitive Load Measurement and Application*, 45-56.

In R. Z. Zheng (Ed.), *Cognitive Load Measurement and Application*, (pp. 45-56). Taylor & Françis.

Zucker, A. A., & Light, D. (2009). Laptop programs for students. *Science*, *323*(5910), 82-85. https://doi.org/10.1126/science.1167705

ANNEXES

- Annexe A : Article publié dans Computers & Education (2022)……..….p.281
- Annexe B : Article publié dans Enfance (2022)……..…..p.295
- Annexe C : Test de Maîtrise des Concepts Computationnels……..……..p.318
- Annexe D : Items vrai / faux du Test de Maîtrise des Concepts Computationnels (supprimés)……..…….p.328
- Annexe E : Test d'Algorithmique (version 1)……...….p.329
- Annexe F : Test d'Algorithmique (version 2)……...….p.334
- Annexe G : Séances de programmation (expérience 1)……p.344
- Annexe H : Échelle de mesure de la motivation et du sentiment d'auto-efficacité (basée sur le SAL de Marsh et al., 2006)…….….p.416
- Annexe I : Échelle de mesure de l'attitude envers les sciences (basée sur les tests de Weinburgh & Steele, 2000 et Kind et al., 2007)……..…..p.419
- Annexe J : Séances de programmation (expérience 2)…….……p.421
- Annexe K : Séances de programmation (expérience 3)……p.435
- Annexe L : Séances de programmation (expérience 4)…….……p.448
- Annexe M : Échelle de mesure de la charge cognitive (basée sur l'échelle de Klepsch et al., 2017 et Paas, 1992)…….p.467

Annexe A

Computers & Education 184 (2022) 104505

Contents lists available at ScienceDirect

Computers & Education

journal homepage: www.elsevier.com/locate/compedu

Unplugged or plugged-in programming learning: A comparative experimental study

Kevin Sigayret *, André Tricot, Nathalie Blanc

Laboratoire Epsylon, EA 4556, Université Paul-Valéry Montpellier 3, France

ARTICLE INFO

Keywords:
Elementary education
Improving classroom teaching
Programming and programming languages
Teaching/learning strategies

ABSTRACT

In recent years, computer programming has reappeared in school curricula with the aim of transmitting knowledge and skills beyond the simple ability to code. However, there are different ways of teaching this subject and very few experimental studies compare plugged-in and unplugged programming learning. The purpose of this study is to highlight the impact of plugged-in or unplugged learning on students' performance and subjective experience. To this end, we designed an experimental study with 217 primary school students divided into two groups and we measured their knowledge of computational concepts, ability to solve algorithmic problem, motivation toward the instruction, self-belief and attitude toward science. The programming sessions were designed to be similar between the two conditions, only the tools were different. Computers and Scratch software were used in the plugged-in group while the unplugged group used paper instructions, pictures, figurines and body movements instead. The results show better learning performance in the plugged-in group. Furthermore, although motivation dropped slightly in both groups, this drop was only significant in the unplugged condition. Gender also seems to be an important factor, as girls exhibit a lower post-test motivation and a lower willingness to pursue their practice in programming outside the school context. However, this effect on motivation was only observable in the plugged-in group which suggests that educational programming software may have a positive but gendered motivational impact.

1. Introduction

Computer science education, and more specifically programming education, has been growing steadily in schools for several years, partly due to the development of new and more accessible programming languages such as Scratch (Resnick et al., 2009). Nowadays, programming and coding skills are learning outcomes in about 20 European education systems for primary education and in about 30 countries for secondary education (European Commission/EACEA/Eurydice, 2019). The same trend can be observed in many developed countries in America or Asia (Webb et al., 2017; Guo & Ottenbreit-Leftwich, 2020). The importance of learning to program is not only based on increasing work opportunities in this field, but also on its supposed educational and cognitive benefits (Scherer, Siddiq, & Sánchez Viveros, 2019; Popat & Starkey, 2019). In addition, being a creator rather than a consumer of new technologies is increasingly seen as a fundamental skill in the future society (Sáez-López, Román-González, & Vázquez-Cano, 2016).

Furthermore, programming is often inextricably linked to the development of computational thinking (CT, first used by Papert,

* Corresponding author. Laboratoire Epsylon, EA 4556, Université Paul-Valéry Montpellier 3, Rue du professeur Henri Serre, 34000, Montpellier, France.
E-mail address: kevin.sigayret@univ-montp3.fr (K. Sigayret).

https://doi.org/10.1016/j.compedu.2022.104505
Received 25 November 2021; Received in revised form 15 March 2022; Accepted 18 March 2022
Available online 30 March 2022

1980), defined as a way of solving problems or designing systems that builds on fundamental concepts of computer science (Wing, 2006) and considered as an essential skill for the 21st century (Mohaghegh & McCauley, 2016; Wing, 2006). Programming can thus be seen as more than just a coding activity as it requires the learner to demonstrate some mastery of CT (Lye & Koh, 2014). In addition, CT does not develop exclusively through programming activities and extends far beyond this discipline (Lye & Koh, 2014).

Although teaching programming and CT is most often carried out through computer-based activities, teachers sometimes use other approaches, based on unplugged activities for which no digital equipment is needed (Brackmann et al., 2017). Several studies have investigated the potential contributions of these two ways of teaching programming (plugged-in and unplugged) but there is still a lack of empirical data to compare their relative abilities to facilitate the transmission of certain essential programming concepts and skills. The aim of our study is to compare the effect of unplugged learning and plugged-in learning on programming concepts and skills acquisition with novice students, in order to provide answers relative to the following research questions:

- Which way of teaching programming (plugged-in or unplugged) enhances learning performances?
- Which way of teaching programming (plugged-in or unplugged) enhances student's subjective experience (motivation, self-belief …)?
- Does the impact of the way of teaching programming (plugged-in or unplugged) on students' learning performance and subjective experience depend on their gender (boys or girls)?

2. Literature review

2.1. Unplugged learning

Unplugged learning involves pencil and paper as well as cards, logic games or simple body movements to represent computational concepts such as algorithms or loops. It does not use digital tools, *i.e.* programming software nor digital objects (computers, tablets, etc.). Unplugged programming would also have some major benefits, summarized by Romero, Viéville, Duflot-Kremer, de Smet and Belhassein (2018): lower cognitive load, embodied learning and concrete analogies. The use of digital tools implies a cognitive load that could have a detrimental effect on students' attention and thus on the learning of the main concepts used in programming. Furthermore, unplugged programming often relies on body movements, gestures and concrete actions which could be conducive to learning according to the embodied view of cognition. Finally, unplugged activities are based on the construction of tangible and concrete analogies, making easier the use of abstract notions related to programming.

Recently, several experimental studies have confirmed the potential benefits of these unplugged activities to increase skills acquisition in computer literacy and CT. For instance, Brackmann et al. (2017) provided empirical evidence of increased CT skills in 10 to 12-year-old students after 10 h of unplugged practice, compared to an inactive control group. These results were confirmed by a similar study (Delal & Oner, 2020) that focused on 6th grade students. Likewise, Threekunprapa and Yasri (2020) reported that both conceptual understanding of coding and CT as well as self-efficacy increased, following unplugged programming activities.

In addition, other works have investigated unplugged learning as part of a mixed approach to teach programming. del Olmo-Muñoz, Cózar-Gutiérrez, and González-Calero (2020) involved 84 students in grade 2 in order to compare learning that was carried out exclusively in digital form (using block-based programming software) with learning that used two different devices, first unplugged and then plugged-in. The results showed that the increase in CT skills was higher in the group that started with unplugged learning. Motivation was similar between the two groups, either at mid- or post-test. It seems that, in the early years of primary education, it is better to introduce CT using mixed devices that combine unplugged and plugged-in activities than to do it only through plugged-in activities. All the studies mentioned above tend to affirm the potential of unplugged activities as a credible alternative to the use of digital devices for learning programming or computer science in general.

Nowadays, a large collection of unplugged activities is available online and used in many countries, for many purposes, to explore computer science education without having to learn computer programming (Bell, Alexander, Freeman, & Grimley, 2009). It should also be remembered that an unplugged approach to programming is the only possible approach for many schools around the world. This is particularly the case in Africa or Asia, where many schools do not have basic computer equipment, but also in most European countries where some rural and remote areas still lack resources (Brackmann et al., 2017). In addition, some students experience difficulties because of having negative attitude toward computer education (Bell et al., 2009; Delal & Oner, 2020). On the contrary, unplugged activities could improve positive attitude and emotional engagement in learning computer science (Threekunprapa & Yasri, 2020), making the learning process more enjoyable, and decrease student's difficulties (Delal & Oner, 2020).

2.2. Plugged-in learning

Scratch is the most widely used software worldwide. It is a visual programming language and a software designed for children to easily create their own interactive content (Olabe, Olabe, Basogain, & Castaño, 2011). The Scratch website also has a social function as it enables the sharing of content between users and therefore promotes collaborative work. Scratch is considered "low floor, high ceilings" *i.e.*, able to offer easily accessible and quickly mastered functionalities ("low floor") while guaranteeing the possibility to design complex and sophisticated programs ("high ceilings"). Since 2009, Scratch is used in most experimental studies investigating the contribution of educational programming software.

Sáez-López et al. (2016) showed that the use of Scratch had the potential to foster programming concepts, logic and computer literacy learning for 5th or 6th grade students. These students found the software fun, motivating and engaging. The authors

recommended the use of Scratch at the end of primary and beginning of secondary school. Some researches with secondary school students also showed the ability of Scratch to motivate students to continue practicing programming, which could result in increased enrolment in computer science classes (Armoni, Meerbaum-Salant, & Ben-Ari, 2015; Ouahbi, Kaddari, Darhmaoui, Elachqar, & Lahmine, 2015). Some limitations were however identified by Aivaloglou and Hermans (2016) who analysed nearly 250,000 Scratch projects. Their findings revealed a large proportion of bad programming practices ("dead script" *i.e.*, a piece of code that is not required and therefore unnecessarily overloads the programme). Certain functionalities and concepts were rarely used and only few projects showed real complexity.

Block-based programming languages like Scratch are supposed to be more accessible and therefore more suitable for school learning. Xu, Ritzhaupt, Tian, and Umapathy (2019) sought to test this claim by conducting a meta-analysis comparing these block-based visual languages with more traditional, text-based languages on teaching programming to novice students. Their study found only a small size effect in favour of block-based languages and the authors deplored a lack of sufficient empirical data to make a definitive statement about the effectiveness of these visual languages. However, more recently, Scherer, Siddiq, and Viveros (2020) found a moderate effect in favour of visualisation tools (especially visual programming software such as Scratch).

Up to now, we can only speculate that the possibilities offered by educational programming software and the immediate feedback from the computer could facilitate the acquisition of the practical skills needed to solve algorithmic problems, although the cognitive load linked to the use of a machine could also be an obstacle to proper learning.

2.3. Plugged-in versus unplugged programming

Despite a number of studies focusing on the potential benefits of unplugged and plugged-in learning, very little research tried to compare their relative efficacy. A few studies have compared a mixed approach (often unplugged at first and then plugged-in) with purely computer-based learning among primary school students (aged 5–11). Most of them reported no significant differences between the two groups in learning programming concepts (Hermans & Aivaloglou, 2017; Romero et al., 2018) or in mathematical abilities and spatial awareness (Messer, Thomas, Holliman, & Kucirkova, 2018). Hermans and Aivaloglou (2017) found a greater belief in self-efficacy and a greater variety of use of Scratch blocks among students who started with unplugged activities. Wohl, Porter, and Clinch (2015) also found that unplugged learning increased young students' mastery (aged 5–7 years old) of several programming concepts. Delal and Oner (2020) believe that computer programming may not be a requirement to teach CT skills.

Webb et al. (2017) recommend further research to identify the relative values of different approaches including "new types of programming languages, unplugged computing activities as well as more general pedagogical approaches made possible by online opportunities". In order to answer questions about the different ways of teaching programming, del Olmo-Muñoz et al. (2020) ask for future studies increasing sample size and length of instruction and the development of measurement tools adapted to young students, especially to assess motivation.

In a literature review concerning the use of Scratch to improve CT skills, Zhang and Nouri (2019) find that experimental results are too rarely replicated which weakens the validity of research in this domain. These authors also state that efforts are needed to explore if current didactic approaches are efficient and call for interventions to compare different ways of learning the same skill. On the other hand, Brackmann et al. (2017) suggest that unplugged programming learning may have some limitations and that computers may be needed to further develop certain skills but question the point at which unplugged activities become less effective.

2.4. The present study

Previous research on programming learning often focuses on CT development, for which there is no real consensus neither on its definition (Selby & Woollard, 2013) nor on the various ways to assess it (Tang, Yin, Lin, Hadad, & Zhai, 2020). Alsancak (2020) argues that attitude toward science (or math) could be an important predictor to CT, although we need empirical evidence to support this claim. For this study, we focused on the assessment of programming knowledge and skills, rather than on the assessment of CT skills, which are closely related to, but distinct from, programming skills. Robins, Rountree, and Rountree (2003) distinguished between programming knowledge and strategies used to program and solve a problem. They also differentiated between the understanding of a programme and the ability to generate it, the former being not necessarily a good predictor of the latter.

Overall, previous studies on unplugged versus plugged-in programming tended to show that unplugged learning is not necessarily less effective than more traditional plugged-in learning, using computers. Unplugged programming might promote greater self-efficacy, but Scratch might provide more motivation and incentive to continue programming activities. Nevertheless, current data are still largely insufficient to be able to make a judgment on the relative effectiveness of these two ways of teaching programming. Furthermore, research in this area has so far been limited to studies with small samples, often conducted over a short period of time and involving students of different ages and school levels, which prevents the generalisation of these results.

For this reason, we conducted a larger-scale comparative study, assessing learning performances in programming (and distinguishing between the comprehension of programming concepts and the ability to solve algorithmic problems) of 5th grade students in France. This study does not propose to test different hypotheses. Indeed, we believe that the current state of the literature does not allow us to favour one hypothesis over another concerning the effectiveness of these two teaching devices (unplugged or plugged-in) in transmitting fundamental concepts and skills in programming.

3. Methods

The aim of this study is to examine how children learn programming at the end of primary school depending on whether this learning is unplugged or plugged-in. This comparative study took place on January and February 2021. Ten 45-min learning sessions were conducted over five weeks. The key programming concepts introduced were: algorithms, instructions, loops, conditionals and variables. This experiment was compatible with the French curriculum according to which grade 4 to 6 students have to know how to "program the movements of a robot or a character on a screen using a programming software".

3.1. Participants

217 grade 5 students (113 females) from south of France participated to the experiment. Classes were randomly assigned to two groups: 4 classes in the plugged-in group (N = 84 students), 6 classes in the unplugged group (N = 133 students). The two groups were homogeneous in terms of gender balance and each involved 4 different schools. No student had ever taken a programming course before the experiment. Only classes from schools in ordinary educational situations were accepted for our study. We thus controlled the selection of participants by not recruiting schools in "éducation prioritaire" areas (in France "priority education", corresponds to areas marked by strong social and economic difficulties, which have an impact on academic achievement). The teachers who notified us of their willingness to participate in this experiment were randomly assigned to each condition (with a minimum of 4 classes per condition, to prevent any imbalance between the populations of the two groups). Our study is quasi-experimental: teachers (and their respective class) were randomly assigned to each group, not students. Ten teachers, with several years of professional teaching experience, participated to this study (5 females). Teachers were given ten programming sessions that they had to implement in their class. Meetings were organized with the participant teachers before the beginning of the experiment, ensuring that they understood and approved the importance of staying as close as possible to the sessions they were given. Teacher participation in this study was voluntary.

3.2. Material

In the plugged-in group, students used Scratch 3.0 educational programming software and learned to manipulate instructions in the form of blocks. Only computers were used during the activities. No tablet, mobile or other digital device were provided to the students. We controlled the experimental environment in each class by ensuring that these classes had sufficient computer equipment (1 computer for 3 or 4 students) in the plugged-in group and by providing the necessary equipment ourselves in the unplugged group.

In the unplugged group, paper instructions were used. These instructions reproduced the blocks available on Scratch: text and colours of the blocks were the same. During learning sessions, students were asked to assemble these "paper blocks" in the same way as blocks are assembled on Scratch, in order to construct their algorithms.

The 10 programming sessions carried out in each experimental group were designed with the help of an educational consultant specialized in digital technology and programming courses in primary school. Concepts, skills and activities developed during this sequence were very similar in the two experimental conditions (see section 2.4). The only major difference between the two groups was the use (or not) of Scratch programming software.

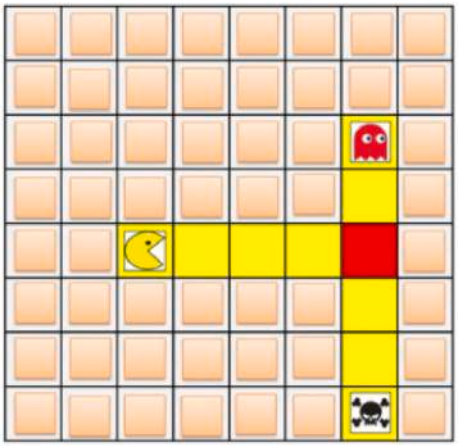


Fig. 1. For this item, taken from the CTt (González, 2015), students had to choose, among 4 programs with conditionals, which one allowed Pac-man (in yellow) to reach the ghost (in red). (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

3.3. Instruments

We designed four different tests to assess the impact of plugged-in or unplugged programming on four variables. These tests are available in appendices (Appendix A). Please note that the students performed them in French and the versions presented in appendices are translated into English.

To produce these new assessment tools, we sometimes used existing questionnaires that had followed a psychometric validation process. We adapted these tests to our context and translated them into French language when necessary.

In order to assess learning performances, we designed a Mastery of computational concepts Test and an Algorithmic Test. These two tests were validated with a sample of 449 students, from 4th to 9th grade (aged 9 to 15). The inclusion of students from different school levels allowed us to benefit from subjects with various levels of programming practice. Their level of programming practice was positively correlated with their performance on these two tests ($r = 0.60$ for the Mastery of computational concepts Test; $r = 0.58$ for the Algorithmic Test), indicating concurrent validity.

3.3.1. Mastery of computational concepts

Brennan and Resnick (2012) and Tchounikine (2017) identified some key computational concepts that need to be mastered in order to learn programming. We relied on these works to design a test assessing the mastery of five fundamental concepts in programming: Algorithm, Instruction, Loop, Conditional and Variable. We relied on the Computational Thinking Test (González, 2015), or CTt, which already assesses the mastery of certain computational concepts (such as loops and conditions). The CTt is a valid and reliable assessment tool, but it has some limitations, since all of the items require the application of computational concepts to solve small practical exercises by mentally executing programs and selecting the correct one, completing or debugging it. We used Bloom's revised taxonomy (Krathwohl, 2002) to provide a wider variety of items and measure different levels of acquisition of the knowledge and skills taught in programming sessions.

Our Mastery of computational concepts Test is composed of 22 multiple-choice items.

- 4 CTt items were translated into French language and integrated into our test (Fig. 1).
- 5 categorization items asking the students to recognize among several propositions a specific concept and to differentiate it from what is not part of this concept (i.e., Item 1 gives 4 different texts and asks the students to identify which texts are algorithms).
- 5 items were affirmations related to the concepts' definitions asking the students to evaluate and check if these were true or false.
- 4 items, students were given a program with arrows pointing to certain parts of this program and were asked to caption it by breaking it down into its constituent parts (algorithm, loop, condition, variable) and detecting how the parts relate to one another.
- 4 items consisted of completing sentences with the names of the 5 different concepts and sometimes required making connections between these different concepts.

We designed two very similar versions of this Mastery of computational concepts test: an unplugged version for the unplugged group and a Scratch version for the plugged-in group. Questions were exactly the same but in the Scratch version, screenshots of programs taken from Scratch are used while these pictures are replaced by algorithms in the form of "paper blocks" in the unplugged version.

For each item, students were told if the question required only one correct answer or if multiple correct answers were possible. When only one answer was expected, a correct answer earned 1 point and any other answer earned 0. When more than one answer was correct, this item was worth as many points as there were correct answers. Each correct answer was worth 1 point and each wrong

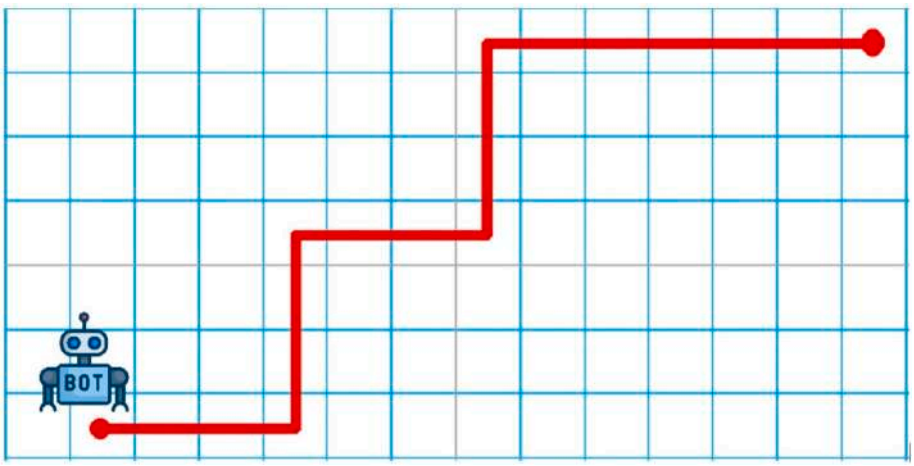


Fig. 2. In this exercise, students were asked to write an algorithm to program "BOT" to move from one point to another, using a loop to repeat certain instructions, and incrementing a variable when the path is completed.

answer was worth −1 point. If the number of points obtained for an item was negative, this value was brought back to 0. Items were grouped in 5 different categories corresponding to the 5 fundamental concepts: algorithm (4 items, 7 points), instruction (2 items, 5 points), loop (6 items, 8 points), condition (6 items, 7 points) and variable (4 items, 4 points). This test is designed to evaluate performance. Thus, total results and results per category were taken into account even when some items were left unanswered.

3.3.2. Ability to solve algorithmic problems

For the purpose of this experiment, we designed a test to evaluate the students' ability to solve algorithmic problems: the Algorithmic Test. This test is designed to assess the higher levels of skill acquisition in programming (corresponding to the "Create" level of Bloom's revised taxonomy, Krathwohl, 2002). It is composed of 4 practical exercises requiring the creation of an algorithm as a solution to the problems described. Unlike what the students did during the 10 programming sessions, where they had to assemble instructions on Scratch (plugged-in group) or in paper format (unplugged group), here they have to write their own instructions to build their algorithms. Even if this new way of working may have disturbed some students, this change was necessary to ensure that neither the plugged-in nor the unplugged group was favoured while ensuring that the test is the same in both conditions.

For each exercise, the students had to program several actions and each step to be programmed was numbered to facilitate their work. The first two exercises are concrete exercises asking students to program the movements of a character on a grid. The last two exercises are more abstract as they did not involve programming moves. The students had to program the character to count indefinitely (exercise 3) and then to count the time passing, like a chronometer (exercise 4). In each exercise, the ability to produce an algorithm with relevant instructions was assessed, as was the ability to use loops (only in exercises 1, 3 and 4), conditions (exercises 2 and 4) or variables (exercises 1, 3 and 4). In the first two exercises, the students were explicitly told that they had to use a loop, a condition or a variable, whereas in the last two exercises, they sometimes had to find this solution by themselves, which makes these exercises much more difficult. Fig. 2 shows the first exercise's path to follow.

In each exercise, the ability to formulate an answer in the form of an algorithm (*i.e.*, a sequence of ordered instructions) was worth 1 point and the relevance of the instructions (precision and coherence of the terms used) was also noted on 1 point. When moves had to be programmed to complete a path, the ability to correctly program this path was awarded with an additional point. When the use of loop or condition was required, 1 point was awarded for the use of the correct statement ("repeat" or "if … then … "). The ability to correctly use loops and conditions to obtain the required result was also scored as 1 point. Finally, the ability to use variables was scored from 1 to 3 points depending on the exercise. The initialization of a variable, its incrementation and the ability to replace a value by a variable were each evaluated on 1-point. Total results and results per category were taken into account even when some items were left unanswered.

3.3.3. Motivation and self-belief

In order to measure students' motivation and self-belief, we used the Students' Approaches to Learning (SAL) Instrument (Marsh, Hau, Artelt, Baumert, & Peschar, 2006) to create our own Motivation and self-belief Test. This instrument measures 14 factors that assess self-regulated learning strategies, self-belief, motivation and learning preferences. On these four dimensions, only items relating to self-belief and motivation have been kept. For the motivation dimension, items linked to the Instrumental motivation factor were also excluded because they were irrelevant for our study. The SAL Instrument is usually used to assess self-concept and interest relative to math and reading. We have adapted these items to fit with self-concept and interest in programming.

To assess motivation and self-belief, we relied on a pre-test/post-test comparison. In the pre-test, some items have been reformulated to address students who have never taken part in programming activities in order to evaluate their motivation a priori, in anticipation of upcoming sessions. The pre-test is composed of 18 items while the post-test contains 20 items. Item 19 (i.e., "When I do programming, I sometimes get totally absorbed") and Item 20 (i.e., "I would like to do programming in my spare time") were excluded from the pre-test since inexperienced students were not able to provide answers on these two statements.

Items are grouped into three distinct categories: General self-belief (9 items), Programming self-belief (6 items) and Motivation (3 items in the pre-test, 5 items in the post-test). Items were answered using a 4-points scale. For Item 1 to 6, all related to General self-efficacy, students used a frequency scale (almost never, sometimes, often, almost always). For other items, students used an agreement scale (disagree, disagree somewhat, agree somewhat, agree). 0 to 3 points were given for each answer for a total of 54 points for the pre-test and 60 points for the post-test. In general, the "almost always" and "agree" answers yielded the most points, except for Item 11, negatively formulated, for which the "disagree" answer yielded the maximum points (i.e., 3 points).

For each student, if no answer was given for two or more items in the pre-test or in the post-test, total results were not taken into consideration. If no answer was given for only one item in one or both tests, total results of the student were exploited, but not the results relating to the category from which the item left unanswered belonged.

3.3.4. Attitude toward science

In order to measure students' attitude toward science, we used two pre-existing tests: the Modified Attitudes toward Science Inventory (Weinburgh & Steele, 2000) and another test assessing the attitude toward science (Kind, Jones, & Barmby, 2007). We combined items from these two instruments to create our own test. Some items were not selected because of their irrelevance to our study, like items related to the perception of the science teacher (present in the Weinburgh and Steel test, 2000) or practical work in science (in the Kind et al. test, 2007). The two tests present some items measuring exactly the same construct, which differ only in their formulation. For this type of item, we have selected the one whose formulation seemed more appropriate to 5th graders.

To assess interest in science, we relied on a pre-test/post-test comparison. The pre-test and the post-test are identical and are both composed of 20 items.

Items are grouped into five distinct categories: Learning science in school (6 items), Self-efficacy in science (5 items), desire to do science in the future (4 items), Value of science in society (3 items) and Anxiety toward science (2 items). Originally, items from the Weinburgh and Steele Test (2000) should be answered on a 6-point agreement scale whereas items from the Kind et al. Test (2007) should be answered on a 5-point agreement scale. In order to avoid the students having to answer the items on two different scales and to facilitate the consistency of the evaluation, we decided to use a 4-point agreement scale, identical to the one used in the Motivation and self-belief Test (disagree, somewhat disagree, somewhat agree, agree). 0 to 3 points were given for each answer for a total of 60 points. In general, the "agree" answers yielded the most important number of points, except for Items negatively formulated (i.e., items 5 to 7 and 11 to 13) for which the "disagree" answers yielded the maximum points (i.e., 3 points).

For each student, if no answer was given for two or more items in the pre-test or in the post-test, total results were not taken into consideration. If no answer was given for only one item in one or both tests, total results of the student were exploited, but not the results relating to the category from which the item left unanswered belonged.

3.4. Procedure

The pre-test and the post-test were administered during the first and last session. Between these two stages, 8 sessions were devoted to programming instruction. Students were introduced to the fundamental computational concepts required to practice programming and learned to manipulate instructions to build algorithms for programming movements and interactions with objects (real or virtual) during the first 4 instruction sessions. Students had to realize a creative project during the last 4 instruction sessions. Fig. 3 summarises the design of the sessions.

In each experimental condition (unplugged or plugged-in), students worked in small groups (3–4 students per group). At the end of sessions 2, 3 and 5, 5 min were taken to write down a trace summarising the main elements seen in class. This written trace was collected in the same way in both conditions. The 10 sessions in each condition are available in appendix A.

4. Results

This section shows the results obtained for both groups in each of the tests. Influence of gender is also described. We used one-way Anova to highlight significant differences between groups and Cohen's *d* to measure effect size (Cohen, 2013). All the tables below give the mean and standard deviation in parenthesis. All values, except the p-value, are rounded to the nearest hundredth. All data on which these analyses are based will be made available on request.

4.1. Mastery of computational concepts

Data reliability was calculated using McDonald's ω coefficient and indicates high enough results ($\omega = 0.79$).

Table 1 shows the average score obtained by students in the two groups on this test, as well as the average scores obtained for each item category.

The differences observed between the two groups were significant in terms of total test performance (F 1, 202) = 31.86; $p < .001$; $d = 0.78$). For items measuring the mastery of the concept of algorithm, the differences were not significant. For the other four concepts that were tested, the plugged-in group performed significantly better than the unplugged group. The students who used the Scratch software were therefore more successful in mastering the concept of instruction ($F(1, 202) = 23.86$; $p < .001$; $d = 0.69$), loop ($F(1, 202) = 18.43$; $p < .001$; $d = 0.62$), condition ($F(1, 202) = 29.70$; $p < .001$; $d = 0.78$) and variable ($F(1, 202) = 33.94$; $p < .001$; $d = 0.84$).

4.2. Ability to solve algorithmic problems

Algorithmic Test's reliability is high ($\omega = 0.85$).

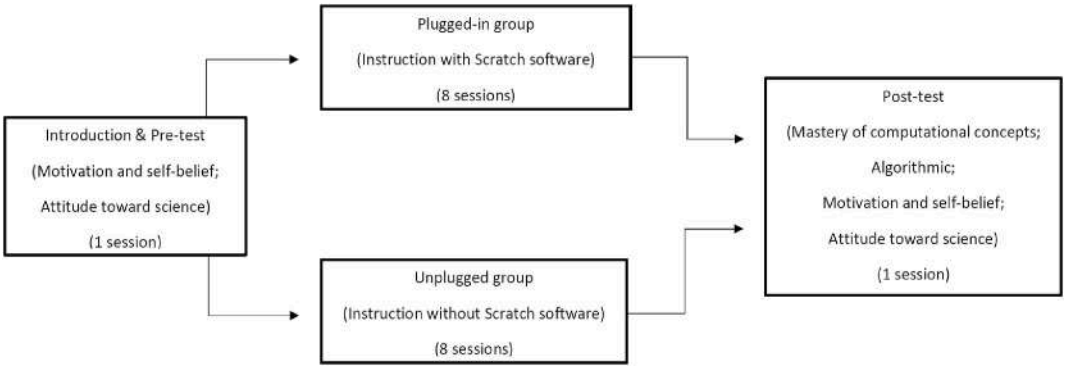


Fig. 3. Design of the sessions.

Table 1
Mastery of computational concepts Test results.

	Total (/31)	Algorithm (/7)	Instruction (/5)	Loop (/8)	Condition (/7)	Variable (/4)
Unplugged	16.68 (5.26)	4.85 (1.70)	2.78 (1.03)	3.75 (2.12)	3.75 (1.47)	1.55 (1.09)
Plugged-in	20.86 (5.03)	4.99 (1.42)	3.58 (1.27)	4.99 (1.86)	4.89 (1.46)	2.41 (0.94)

Table 2 shows the average score obtained by students in the two groups on the Algorithmic Test, as well as the average scores obtained for each item category. Table 3 distinguishes between the results obtained in the first two exercises (basic exercises) and those obtained in the last two (complex exercises).

Again, the plugged-in group obtained a higher total score than the unplugged group ($F(1, 202) = 33.98$; $p < .001$; $d = 0.83$). This superiority was expressed in the ability to create algorithms with relevant instructions ($F(1, 202) = 23.01$; $p < .001$; $d = 0.71$), to use loops ($F(1, 202) = 15.03$; $p < .001$; $d = 0.56$), conditions ($F(1, 202) = 8.50$; $p < .01$; $d = 0.41$) and variables ($F(1, 202) = 57.64$; $p < .001$; $d = 1.04$).

It can be seen on Table 3 that the differences observed between the two groups were not significant with regard to the first two exercises of the Algorithmic Test. However, there were significant differences for the last two exercises with a rather large effect size ($F(1, 202) = 52.46$; $p < .001$; $d = 0.95$).

4.3. Motivation and self-belief

The overall and item category results of the Motivation and self-belief Test are detailed below, especially the pre-test/post-test evolutions for the unplugged and the plugged-in condition (Table 4).

In the unplugged group, the pre-test/post-test evolution of the total scores was not significant, nor was the evolution of the sub-scores related to self-belief (in general or related to programming). On the other hand, the results showed a significant drop in motivation in this group ($F(1, 215) = 6.15$; $p = .014$; $d = 0.34$). In the plugged-in group, the pre-test/post-test evolution was never significant, neither for the total scores nor for the results in the three different item categories.

Two additional motivation items were added in the Motivation and self-belief post-test: item 19 ("When I do programming, I sometimes get totally absorbed") and item 20 ("I would like to do programming in my spare time"). Table 5 compares the pre-test scores of the students in the two groups and results obtained for Items 19 and 20 in the post-test for both groups.

The comparison of the two groups in the pre-test showed that they were equivalent as no significant differences were observed. A significant difference between the two groups was observed on item 20 ($F(1, 197) = 17.95$; $p < .001$; $d = 0.62$) but not on item 19.

4.4. Attitude toward science

The results obtained in the Attitude toward science Test are described in Table 6. The total and item category pre-test/post-test evolutions for the two groups are provided.

In both groups, Table 6 shows that there were no significant evolutions of students' attitude toward science, for the total scores as well as for item category scores.

4.5. Gender

Table 7 describes the influence of gender on the various tests used in this experiment. Table 7 also compares boys and girls on motivation items only, excluding self-belief items. Details of the two additional motivation items (Items 19 & 20) are reported.

There were no significant differences between boys and girls with regard to the total scores obtained in the different tests (Attitude toward science, Motivation and self-belief, Mastery of computational concepts, Algorithmic). At pre-test, the motivation level of boys and girls was very close but the post-test showed statistically significant differences in favour of boys ($F(1, 182) = 9.06$; $p < .01$; $d = 0.44$). Regarding the two additional items, we also observed higher scores for boys than for girls for item 19 ($F(1, 197) = 4.54$; $p < .05$; $d = 0.30$) or for item 20 ($F(1, 197) = 10.18$; $p < .01$; $d = 0.46$). Note that the effect size was larger for item 20.

Table 8 compares boys and girls in both groups in order to highlight potential differences depending on instructional device.

It is interesting to note that the gender effects related to motivation (in favour of boys) are not observable in the unplugged condition. On the other hand, in the plugged-in condition, these effects are strong both for motivation post-test without items 19 & 20 ($F(1, 74) = 19.92$; $p < .001$; $d = 1.02$) and for item 20 ($F(1, 77) = 15.42$; $p < .001$; $d = 0.88$).

Table 2
Algorithmic Test results.

	Total (/26)	Algorithm/Instruction (/10)	Loop (/6)	Condition (/4)	Variable (/6)
Unplugged	9.83 (4.95)	6.21 (2.25)	1.97 (1.71)	1.02 (1.10)	0.62 (1.09)
Plugged-in	14.16 (5.50)	7.65 (1.81)	2.91 (1.67)	1.51 (1.27)	2.08 (1.65)

Table 3
Algorithmic Test results by type of exercise.

	Basic exercises (/11)	Complex exercises (/15)
Unplugged	6.96 (2.39)	2.88 (3.37)
Plugged-in	7.17 (2.00)	6.99 (4.73)

Table 4
Motivation and self-belief Test results evolution in both groups.

	Total (/54)	General self-belief (/27)	Programming self-belief (/18)	Motivation (/9)
Unplugged group				
Pre-test	35.18 (8.74)	18.10 (4.46)	11.74 (3.41)	5.54 (2.33)
Post-test	35.94 (8.53)	19.00 (4.67)	12.14 (3.79)	4.79 (2.14)
Plugged-in group				
Pre-test	35.99 (7.85)	18.22 (5.04)	12.05 (3.06)	5.92 (2.26)
Post-test	36.79 (9.11)	18.38 (5.90)	12.93 (3.79)	5.71 (2.34)

Table 5
Motivation and self-belief pre-Test results and items 19 and 20 results – Comparison unplugged vs. plugged-in.

	Pre-test				Post-test	
	Total (/54)	General self-belief (/27)	Programming self-belief (/18)	Motivation (/9)	Item 19 (/3)	Item 20 (/3)
Unplugged	35.18 (8.74)	18.10 (4.46)	11.74 (3.41)	5.54 (2.33)	1.71 (0.90)	1.16 (1.01)
Plugged-in	35.99 (7.85)	18.22 (5.04)	12.05 (3.06)	5.92 (2.26)	1.94 (0.87)	1.80 (1.07)

Table 6
Attitude toward science Test results evolution in both groups.

	Total (/60)	Science class (/18)	Self-efficacy in science (/15)	Anxiety toward science (/6)	Desire to do science (/12)	Value of science in society (/9)
Unplugged group						
Pre-test	36.74 (10.59)	10.68 (4.52)	10.36 (3.15)	5.06 (1.37)	4.71 (2.59)	6.16 (2.13)
Post-test	36.42 (10.97)	9.94 (4.81)	10.67 (3.47)	5.23 (1.25)	4.47 (2.47)	6.12 (2.08)
Plugged-in group						
Pre-test	38.72 (10.28)	12.20 (4.07)	10.54 (3.18)	5.34 (1.18)	4.95 (2.89)	5.88 (2.17)
Post-test	39.34 (8.44)	11.97 (4.13)	10.89 (2.76)	5.55 (0.92)	5.05 (2.32)	5.94 (1.81)

Table 7
Influence of student's gender on various tests.

	Boys	Girls
Attitude toward science pre-test	38.23 (10.53)	36.68 (10.46)
Motivation and self-belief pre-test	34.73 (9.08)	34.88 (8.68)
Motivation and self-belief post-test	36.62 (8.634)	35.98 (9.16)
Motivation pre-test	5.81 (2.33)	5.43 (2.30)
Motivation post-test (without Items 19 & 20)	5.68 (2.20)	4.70 (2.23)
Item 19	1.95 (1.00)	1.68 (0.77)
Item 20	1.66 (1.05)	1.18 (1.05)
Mastery of computational concepts Test	17.95 (6.12)	18.66 (5.02)
Algorithmic Test	11.65 (6.28)	11.41 (4.87)

4.6. Variation in performance across classes

It has been shown in the literature that achievement in school is influenced by certain factors external to the student himself. Student's home, peers, school and teacher have an impact on achievement (Hattie, 2003). It is therefore necessary to compare classes one by one in each experimental condition to highlight a greater or lesser impact of this effect depending on the instructional device studied (unplugged or plugged-in). Table below show the different mean performances obtained in classes from both groups (Table 9).

Table 8
Influence of student's gender on various tests depending on instructional device.

	Unplugged		Plugged-in	
	Boys	Girls	Boys	Girls
Motivation and self-belief post-test	35.81 (7.55)	36.04 (9.37)	37.71 (9.28)	35.90 (8.96)
Motivation post-test (without Items 19 & 20)	4.86 (2.14)	4.72 (2.14)	6.81 (1.75)	4.67 (2.38)
Item 19	1.79 (0.97)	1.63 (0.83)	2.13 (0.99)	1.74 (0.68)
Item 20	1.28 (1.02)	1.05 (1.00)	2.23 (0.80)	1.36 (1.14)
Mastery of computational concepts Test	16.37 (6.04)	16.95 (4.46)	20.33 (5.38)	21.37 (4.67)
Algorithmic Test	9.57 (5.58)	10.07 (4.35)	14.81 (6.02)	13.54 (4.95)

Table 9
Variation in performance across classes in both groups.

	Unplugged group classes					
	1	2	3	4	5	6
Mastery of computational concepts Test	14.64 (5.31)	16.88 (5.12)	20.36 (4.49)	11.40 (6.22)	17.19 (4.45)	16.61 (3.11)
Algorithmic test	8.30 (2.85)	11.10 (5.60)	6.96 (2.41)	5.43 (2.76)	13.54 (5.02)	8.63 (3.50)
Plugged-in group classes						
		1	**2**	**3**	**4**	
Mastery of computational concepts Test		18.12 (4.75)	21.08 (4.91)	22.43 (4.84)	23.42 (5.22)	
Algorithmic test		13.58 (4.10)	13.02 (7.17)	17.07 (3.85)	10.71 (4.23)	

As expected, there was a significant class effect on performance in the Mastery of computational concepts Test and the Algorithmic Test in both conditions, but this impact was much more pronounced in the unplugged group ($F(5, 118) = 9.07$; $p < .001$; $\eta^2 = 0.228$ and $F(5,118) = 12.8$; $p < .001$; $\eta^2 = 0.351$, respectively) compared to the plugged-in group ($F(3,76) = 3.83$; $p < .01$; $\eta^2 = 0.139$ and $F(3,76) = 3.89$; $p < .001$; $\eta^2 = 0.133$, respectively).

5. Discussion

This study was designed to compare unplugged and plugged-in programming learning at the end of primary school. The ten 45-min sessions were conceived to be similar between the two groups with learning tasks focused on the development of the same skills and addressed the same basic concepts in both experimental conditions. The findings of this experiment are discussed below.

5.1. Learning performance in programming

At the end of the programming instruction, both groups of students were evaluated on their learning performance. The main results showed a better understanding of programming concepts for students who have been introduced to this discipline using Scratch software (plugged-in group) compared to the unplugged group. The concepts of instructions, loops, conditions and variables seemed to be better mastered by students in the plugged-in group. Only the concept of algorithm, which was the most basic and fundamental, did not seem to be better understood in one group than in the other.

Regarding the ability to solve algorithmic problems, again the plugged-in group scored higher compared to the unplugged group. The ability to produce algorithms, using instructions that are consistent with the problems constituting the Algorithmic Test, appeared to be higher for students who used Scratch. A better use of loops, conditions and variables was also observed in this group. However, there were no significant differences between the two groups with regard to the first two exercises proposed in the Algorithmic test. These two exercises were both simpler and more concrete than the last two ones: students were only asked to program the movements of a character to complete a path. The differences were particularly marked on the last two more difficult exercises, requiring a better mastery of the implementation of variables and asking the students to construct an algorithm that did not program any movement in space.

These results seemed to contradict some previous studies in which no difference between plugged-in and unplugged groups were observed (Hermans & Aivaloglou, 2017; Messer et al., 2018; Romero et al., 2018) and those that reported better performance in the unplugged group (Wohl et al., 2015; del Olmo-Muñoz et al., 2020). However, it is necessary to recall that the few studies that have attempted to compare the two learning modes used very different protocols. Indeed, most of them used mixed approaches that combined both unplugged and plugged-in activities. In our study, learning was carried out exclusively in an unplugged or plugged-in manner. Furthermore, while we focused on assessing the understanding of programming concepts and the ability to use them to solve problems, previous research rather assessed computational thinking skills, which are only indirectly related to our variables, or even transfer effects on other cognitive faculties (Messer et al., 2018). Also, studies that have shown a positive effect of unplugged learning (Wohl et al., 2015; del Olmo-Muñoz et al., 2020) were conducted with younger students, aged 5 to 7, which is coherent and predictable given their lack of experience with computers and computing in general. Finally, the samples used in these studies were quite small,

which leads us to be cautious about their conclusions.

Our experimental results clearly showed an advantage to plugged-in learning of programming. This effect was all more salient when the concepts addressed and the problems to solve were complex. For example, results showed a much better assimilation and use of variables in the plugged-in group. These findings highlighted the limits of unplugged programming. Moreover, the cognitive load related to the use of a computer and the Scratch software seemed acceptable, at least for students at the end of primary school and after approximately 7.5 h of practice.

5.2. Students' subjective experience

To take into account the subjective experience of the students with regard to the programming activities or science in general, we integrated a set of self-evaluative variables within our experimental protocol, which were measured pre- and post-test. The Motivation and self-belief pre-test was carried out before the beginning of the programming activities and therefore necessarily reflects students' anticipation of the upcoming activities (certainly influenced by naive preconceptions about the subject). Among the studies that have compared plugged-in and unplugged programming learning, del Olmo-Muñoz et al. (2020) reported very similar motivation levels between the two groups. Nevertheless, several researchers believed that the use of Scratch could enhance students' motivation and in particular may encourage them to continue programming activities (Armoni et al., 2015; Ouahbi et al., 2015; Sáez-López et al., 2016).

Our findings (pre-test/post-test comparison) showed no significant change in motivation in the plugged-in group. However, there was a slight drop in motivation in the unplugged group (significant effect, small size). In both groups, motivation was lower at the end than at the beginning of the activities, which reflects a certain weariness of the students, even more pronounced in the unplugged group. We can suppose that the richness and variety of situations that can be encountered on Scratch facilitate the maintenance of a certain motivation level over time. Furthermore, the results confirmed the ability of the Scratch software to generate a greater willingness to pursue programming activities outside the school setting as evidenced by the significant differences observed in favour of the plugged-in group on item 20 ("I would like to do programming in my spare time") which was included only in the Motivation and self-belief post-test.

Several studies have also noted an increase in self-efficacy, particularly following unplugged programming activities (Hermans & Aivaloglou, 2018; Delal & Oner, 2020; Threekunprapa & Yasri, 2020). In line with these studies, our results showed a slight increase in self-belief (including self-efficacy), but the differences observed were not significant. We also measured Students' Attitude toward science and we did not observe any change in this attitude over time in any of the subcategories of items that were assessed, either in the plugged-in or in the unplugged group. It seems that learning to program does not have an obvious and immediate effect on students' attitudes toward science. However, it is possible that a longer intervention with more practice could have a more noticeable impact.

5.3. Gender

It is fairly well known that the number of women involved in the programming professions is quite low, although a gradual change in this tendency can be observed in recent surveys (Partovi, 2020). Data from an introductory programming course at university indicate that men find programming easier, have more intention to continue programming and are more successful in this discipline compared to women (Rubio, Romero-Zaliz, Mañoso, & Angel, 2015), although this gender gap is much less pronounced in early childhood education (del Olmo-Muñoz et al., 2020).

Concerning the influence of students' gender on the learning performance, we observed no significant difference, in accordance with other studies (Hermans and Aivaloglou, 2017; Delal & Oner, 2020). However, the post-test motivation was higher for boys, compared to girls. This was also the case for the two items that were added only in the post-test (Item 19: "When I do programming, I sometimes get totally absorbed"; Item 20: "I would like to do programming in my spare time"). These two items were removed from the pre-test since they were irrelevant for students who had never participated in any programming activities. The strongest effect was on the willingness to pursue programming activities outside the school context (item 20).

This finding is interesting because it highlights a probable stereotype effect: it seems that girls find it more difficult to consider programming as a leisure activity, possibly because this activity is still perceived as being more reserved to boys. Furthermore, a deeper analysis shows that this increase in motivation for boys is only observable in the plugged-in condition, with a strong effect size. Thus, the motivational benefits of educational programming software such as Scratch seem highly gendered and unplugged activities might therefore be less dependent upon gender. New studies are needed to shed light on this phenomenon.

5.4. Class effect

Finally, as expected (Hattie, 2003), there are differences in performance between classes from the same experimental group. However, this variability is higher in the unplugged condition than in the plugged-in condition. This effect may be due to the greater importance of the teacher in the unplugged condition as the only source of feedback for the students. In the plugged-in condition the students also receive basic but immediate feedback from the software, thus facilitating more effective trial-and-error learning.

Besides, some unplugged classes are close to the average performance of plugged-in classes. This could mean that unplugged learning of programming requires great skill on the part of the teacher to achieve performance levels comparable to plugged-in classes.

5.5. Contributions of research and practice

Our study contributes to the advancement of knowledge both in the research field and in the teaching of computer programming in the classroom. Regarding research, our contribution can be summarized as following:

- We highlighted the importance of empirically comparing different ways of teaching programming and the lack of studies conducted on this subject.
- We compared two groups of students who had been introduced to programming either unplugged or using Scratch software and revealed the limitations of unplugged programming learning.
- We showed evidence for the influence of gender on the motivational variable and the link between gender and the teaching device used (i.e., motivation increased for boys using scratch).

With regards to the teaching of programming in schools, our study provides at least:

- the pedagogical sequence we designed which is composed of 10 sessions of 45 min that teachers can use to introduce their students to programming (plugged-in or unplugged) and the two tests we developed so that teachers can use them to assess their student's mastery of computational concepts and ability to solve algorithmic problems.
- results indicating a positive impact of the Scratch software on learning performances that may help to convince some teachers of the need and interest to provide a plugged-in approach to programming.
- results pointing to the importance of using programming software to build up students' willingness to continue programming outside the school context.
- evidence of a gender effect in the use of programming software, which may remind teachers to be vigilant in this regard.

5.6. Limitations and further research

Our study found significant differences between students learning programming through unplugged activities and those learning it using the educational programming software Scratch. However, these findings did not explain why these differences were observed. We mentioned above the accessibility of Scratch as well as the richness of the possibilities offered, which probably help to maintain a certain motivation level over time. The possibility of receiving rapid feedback from the software to observe in real time the execution of the algorithm constructed by the student could also facilitate learning by trial and error, as shown in previous studies on the importance of feedback (Hattie & Timperley, 2007). Furthermore, the shape of the Scratch blocks is designed to allow only those blocks that can be linked together to be nested. In this way, the software prevents logical errors in the construction of algorithms, which could also have an impact on learning. We therefore need further studies to both confirm our results and explain the differences observed.

In addition, our study did not investigate the use of mixed learning modalities (mixing unplugged and plugged-in activities). Further studies are needed to determine whether the combination of different programming learning devices is beneficial for learning, as some previous research with smaller samples tended to show. It is also essential to remember that the results obtained here are relative to one audience, one school system, and to sessions and tests specific to our experimental protocol. It is highly likely that other studies conducted with a different audience, in another country, and aimed at teaching and assessing other aspects of programming learning, could provide divergent or even contradictory results. Finally, we focused solely on the unplugged teaching of programming and the use of educational programming software. Other devices are also used in this discipline and could potentially be even more effective in conveying fundamental knowledge and skills. In this respect, educational robotics seems to be particularly promising, as shown by some of the studies carried out in this field (Benitti, 2012; Anwar, Bascou, Menekse, & Kardgar, 2019; Atmatzidou & Demetriadis, 2016), although it has not been sufficiently investigated (Scherer et al., 2020).

6. Conclusion

Our study revealed the benefits of educational programming software compared to unplugged learning. Students who used Scratch showed a better mastery of computational concepts and a greater ability to solve algorithmic problems. Regarding subjective experience, it seems that it was easier to keep the motivation of students using Scratch at high levels. Results also confirmed the impact of gender, not on learning performance which was similar between boys and girls, but on motivation and willingness to pursue programming activities, which were lower for girls, but only in the plugged-in group.

This experiment provides answers on the relative effectiveness of the devices that are used to teach programming, an issue that has been insufficiently investigated in the literature. For this study, we used a fairly large sample, at least compared to previous studies, which strengthens our findings' reliability. However, new experimental data are needed to confirm our results and to further explain the observed differences. As programming learning gradually enters (or re-enters) the school curriculum, it is essential to provide teachers with rigorous empirical research on how the subject should be taught in relation to stated learning objectives. We hope that our work will encourage education professionals to appreciate the importance of a plugged-in approach in order to enhance students' performance in programming learning.

Credit author statement

Kevin Sigayret: Conceptualization, Methodology, Formal analysis, Investigation, Writing – original draft, Writing – review & editing. André Tricot: Conceptualization, Methodology, Formal analysis, Writing – review & editing. Nathalie Blanc: Conceptualization, Methodology, Formal analysis, Writing – review & editing.

Acknowledgements

This research was carried out as part of a thesis supported and funded by the Délégation Académique au Numérique Educatif (DANE) of the Académie de Montpellier, France. We would like to thank all the teachers who participated in the design and implementation of the classroom sessions.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.compedu.2022.104505.

References

Aivaloglou, E., & Hermans, F. (2016). How kids code and how we know: An exploratory study on the Scratch repository. In *Proceedings of the 2016 ACM Conference on International computing education research* (pp. 53–61).
Alsancak, D. (2020). Investigating computational thinking skills based on different variables and determining the predictor variables. *Participatory Educational Research, 7*(2), 102–114.
Anwar, S., Bascou, N. A., Menekse, M., & Kardgar, A. (2019). A systematic review of studies on educational robotics. *Journal of Pre-College Engineering Education Research (J-PEER), 9*(2), 2.
Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From scratch to "real" programming. *ACM Transactions on Computing Education (TOCE), 14*(4), 1–15.
Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems, 75*, 661–670.
Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology, 13*(1), 20–29.
Benitti, F. B. V. (2012). Exploring the educational potential of robotics in schools: A systematic review. *Computers & Education, 58*(3), 978–988.
Brackmann, C. P., Román-González, M., Robles, G., Moreno-León, J., Casali, A., & Barone, D. (2017). November). Development of computational thinking skills through unplugged activities in primary school. In *Proceedings of the 12th workshop on primary and secondary computing education* (pp. 65–72).
Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American educational research association* (Vol. 1, p. 25). Vancouver, Canada.
Cohen, J. (2013). *Statistical power analysis for the behavioral sciences*. Academic press.
Delal, H., & Oner, D. (2020). Developing middle school students' somputational thinking skills using unplugged computing activities. *Informatics in Education, 19*(1), 1–13.
European Education and Culture Executive Agency, Eurydice. (2019). *Digital education at school in Europe*. Publications Office. https://data.europa.eu/doi/10.2797/66552.
González, M. R. (2015). Computational thinking test: Design guidelines and content validation. In *Proceedings of EDULEARN15 conference* (pp. 2436–2444).
Guo, M., & Ottenbreit-Leftwich, A. (2020). Exploring the K-12 computer science curriculum standards in the US. In *Proceedings of the 15th workshop on primary and secondary computing education* (pp. 1–6).
Hattie, J. (2003). *Teachers Make a Difference, what is the research evidence?*.
Hattie, J., & Timperley, H. (2007). The power of feedback. *Review of Educational Research, 77*(1), 81–112.
Hermans, F., & Aivaloglou, E. (2017). To scratch or not to scratch ? A controlled experiment comparing plugged first and unplugged first programming lessons. In *Proceedings of the 12th workshop on primary and secondary computing education* (pp. 49–56).
Kind, P., Jones, K., & Barmby, P. (2007). Developing attitudes towards science measures. *International Journal of Science Education, 29*(7), 871–893.
Krathwohl, D. R. (2002). A revision of Bloom's taxonomy: An overview. *Theory Into Practice, 41*(4), 212–218.
Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior, 41*, 51–61.
Marsh, H. W., Hau, K. T., Artelt, C., Baumert, J., & Peschar, J. L. (2006). OECD's brief self-report measure of educational psychology's most useful affective constructs: Cross-cultural, psychometric comparisons across 25 countries. *International Journal of Testing, 6*(4), 311–360.
Messer, D., Thomas, L., Holliman, A., & Kucirkova, N. (2018). Evaluating the effectiveness of an educational programming intervention on children's mathematics skills, spatial awareness and working memory. *Education and Information Technologies, 23*(6), 2879–2888.
Mohaghegh, D. M., & McCauley, M. (2016). *Computational thinking: The skill set of the 21st century*.
Olabe, J. C., Olabe, M. A., Basogain, X., & Castaño, C. (2011). Programming and robotics with Scratch in primary education. In A. Mendez-Vilas (Ed.), *Education in a technological world: Communicating current and emerging research and technological efforts* (pp. 356–363).
del Olmo-Muñoz, J., Cózar-Gutiérrez, R., & González-Calero, J. A. (2020). Computational thinking through unplugged activities in early years of Primary Education. *Computers & Education, 150*, 103832.
Ouahbi, I., Kaddari, F., Darhmaoui, H., Elachqar, A., & Lahmine, S. (2015). Learning basic programming concepts by creating games with scratch programming environment. *Procedia-Social and Behavioral Sciences, 191*, 1479–1482.
Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books.
Partovi, H. (2020). *Young women set records in computer science exams. again!*. Code.org https://codeorg.medium.com/young-women-set-records-in-computer-science-exams-again-d7e3ee4ca6be.
Popat, S., & Starkey, L. (2019). Learning to code or coding to learn? A systematic review. *Computers & Education, 128*, 365–376.
Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., … Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60–67.
Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education, 13*(2), 137–172.
Romero, M., Lille, B., Viéville, T., Duflot-Kremer, M., de Smet, C., & Belhassein, D. (2018). Analyse comparative d'une activité d'apprentissage de la programmation en mode branché et débranché. In *Educode-Conférence internationale sur l'enseignement au numérique et par le numérique*.
Rubio, M. A., Romero-Zaliz, R., Mañoso, C., & Angel, P. (2015). Closing the gender gap in an introductory programming course. *Computers & Education, 82*, 409–420.

Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two years case study using "scratch" in five schools. *Computers & Education, 97*, 129–141.
Scherer, R., Siddiq, F., & Sánchez Viveros, B. (2019). The cognitive benefits of learning computer programming: A meta-analysis of transfer effects. *Journal of Educational Psychology, 111*(5), 764.
Scherer, R., Siddiq, F., & Viveros, B. S. (2020). A meta-analysis of teaching and learning computer programming: Effective instructional approaches and conditions. *Computers in Human Behavior*, 106349.
Selby, C., & Woollard, J. (2013). *Computational thinking: The developing definition*.
Tang, X., Yin, Y., Lin, Q., Hadad, R., & Zhai, X. (2020). Assessing computational thinking: A systematic review of empirical studies. *Computers & Education, 148*, 103798.
Tchounikine, P. (2017). *Initier les élèves à la pensée informatique et à la programmation avec Scratch*. Laboratoire d'informatique de Grenoble (En ligne http:// ligmembres. imag. fr/tchounikine/PenseeInformatiqueEcole. html).
Threekunprapa, A., & Yasri, P. (2020). Unplugged coding using flowblocks for promoting computational thinking and programming among secondary school students. *International Journal of Instruction, 13*(3), 207–222.
Webb, M., Davis, N., Bell, T., Katz, Y. J., Reynolds, N., Chambers, D. P., et al. (2017). Computer science in K-12 school curricula of the 2lst century: Why, what and when? *Education and Information Technologies, 22*(2), 445–468.
Weinburgh, M. H., & Steele, D. (2000). The modified attitudes toward science inventory: Developing an instrument to be used with fifth grade urban students. *Journal of Women and Minorities in Science and Engineering, 6*(1).
Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35.
Wohl, B., Porter, B., & Clinch, S. (2015). Teaching computer science to 5-7 year-olds: An initial study with scratch, cubelets and unplugged computing. In *Proceedings of the workshop in primary and secondary computing education* (pp. 55–60).
Xu, Z., Ritzhaupt, A. D., Tian, F., & Umapathy, K. (2019). Block-based versus text-based programming environments on novice student learning outcomes: A meta-analysis study. *Computer Science Education, 29*(2–3), 177–204.
Zhang, L., & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. *Computers & Education, 141*, 103607.

Annexe B

L'APPRENTISSAGE DE LA PROGRAMMATION : QUELS OUTILS POUR ÉVALUER LE DÉVELOPPEMENT DE LA PENSÉE INFORMATIQUE À L'ÉCOLE ?

Kevin Sigayret, Nathalie Blanc, André Tricot

Presses Universitaires de France | « Enfance »

2022/4 N° 4 | pages 479 à 500

ISSN 0013-7545
ISBN 9782130834779
DOI 10.3917/enf2.224.0479

Article disponible en ligne à l'adresse :

https://www.cairn.info/revue-enfance-2022-4-page-479.htm

L'apprentissage de la programmation : quels outils pour évaluer le développement de la pensée informatique à l'école ?

Kevin Sigayret[a], Nathalie Blanc[a] & André Tricot[a]

RÉSUMÉ

Malgré l'arrivée de la programmation informatique dans les cursus scolaires, il subsiste de nombreuses incertitudes sur les moyens mis en œuvre pour évaluer son apprentissage. L'une des finalités principales de l'apprentissage de la programmation serait la maîtrise de la pensée informatique, dont le développement constituerait un enjeu éducatif majeur pour les décennies à venir. Le présent article propose donc de passer en revue les outils d'évaluation des compétences en pensée informatique et leurs limites. Diverses approches sont discutées : échelles auto-évaluatives, outils d'analyse du code produit par l'élève, tâches de résolution de problèmes. L'importance de distinguer la compréhension des notions et la capacité à résoudre des problèmes dans la construction de ces outils est abordée. L'objectif de cet article est de fournir aux chercheurs comme aux enseignants une synthèse concernant les différentes approches disponibles pour évaluer le développement de la pensée informatique en contexte scolaire. Cette synthèse aura des retombées sur les recherches à venir consacrées à l'évaluation de la pensée informatique et pourra alimenter la réflexion engagée sur les pratiques à l'école.

MOTS-CLÉS : PROGRAMMATION, APPRENTISSAGE, PENSÉE INFORMATIQUE, OUTILS D'ÉVALUATION.

ABSTRACT

Programming learning: what tools to evaluate computational thinking development in school?

Despite the introduction of computer programming in school curricula, there are still many uncertainties about the means used to assess its learning. One of the main purposes of learning programming is to master computational thinking, whose development would constitute a major educational challenge for the decades to come. This article there-

[a] Laboratoire EPSYLON EA 4556, 1 rue du Professeur Henri Serre, 34090 Montpellier, Université Paul Valéry Montpellier 3. *E-mails* : kevin.sigayret@univ-monpt3.fr ; nathalie.blanc@univ-montp3.fr ; andre.tricot@univ-montp3.fr.

fore proposes to review the tools for assessing computational thinking skills and their limitations. Various approaches are discussed: self-assessment scales, tools for analyzing the code produced by the student, problem-solving tasks. The importance of distinguishing between conceptual understanding and problem-solving skills in the construction of these tools is addressed. The aim of this article is to provide researchers and teachers with a synthesis of the different approaches available to assess the development of computational thinking in a school context.

KEYWORDS: PROGRAMMING, LEARNING, COMPUTATIONAL THINKING, ASSESSMENT TOOLS.

INTRODUCTION

La programmation prend incontestablement une importance de plus en plus grande dans notre société où la maîtrise du numérique et de ses possibilités n'est plus une option. Les premières tentatives d'introduction de la programmation dans les classes françaises datent des années 1960, avant que le Plan informatique pour tous de 1985 ne vienne plutôt centrer l'apprentissage sur l'utilisation de l'ordinateur lui-même. La diffusion rapide du web et des outils de communication à l'échelle mondiale à partir du début des années 2000 marque l'avènement d'une nouvelle période dans laquelle la prise en compte de la programmation en tant que discipline scolaire à part entière s'accélère.

De nos jours, selon le rapport Eurydice (Bourgeois, Birch & Davydovskaia, 2019), qui donne un aperçu de l'état de l'éducation numérique en Europe, les compétences en programmation et en codage sont des objectifs d'apprentissage pour l'enseignement primaire dans une vingtaine de systèmes éducatifs européens et dans une trentaine de pays en ce qui concerne l'enseignement secondaire. En France, la compétence numérique est scindée en deux axes dont le premier la définit comme une langue : les langages de programmation et les algorithmes. Le rapport Eurydice classe la programmation comme une compétence issue d'un domaine plus large qui est celui de la création de contenus numériques.

L'apprentissage de la programmation à l'école connaît un essor indéniable depuis vingt ans. Ceci peut s'expliquer en partie par le développement des logiciels de programmation visuelle (par exemple Scratch), qui rendent plus accessible cette discipline parfois jugée trop complexe, ainsi que par la nécessité pour le futur citoyen de pouvoir appréhender et démystifier le numérique et ses enjeux par une réflexion critique (Romero, Viéville, Duflot-Kremer, de Smet & Belhassein, 2018).

Au-delà de la nécessité de faire entrer la programmation dans un contexte scolaire pour former le citoyen de demain, il semblerait que son apprentissage stimule certaines facultés cognitives et permette d'accompagner le développe-

ment intellectuel des enfants et adolescents. Apprendre à programmer pourrait ainsi améliorer les aptitudes au raisonnement logique (Psycharis & Kallia, 2017) et avoir une influence positive sur certaines compétences dites de « haut niveau » (Popat & Starkey, 2019). De plus, Scherer, Siddiq et Sánchez Viveros (2019) ont mis en évidence la possibilité de transférer ce qui a été appris en programmation à d'autres situations, dans un contexte plus ou moins différent.

Alors que la programmation prend de plus en plus de place dans les programmes scolaires, les finalités de cette discipline et les moyens disponibles pour évaluer son apprentissage sont l'objet de discussions et de controverses. Il est aujourd'hui essentiel de permettre aux enseignants, qui ne sont pas toujours suffisamment formés pour faire face à ce défi, de bénéficier d'outils d'évaluation dont les apports et les limites ont été établis de manière rigoureuse.

Dans un premier temps, nous tenterons de définir précisément ce que l'on appelle le développement de la « pensée informatique » qui constitue l'un des enjeux majeurs de l'enseignement de la programmation à l'école. Wing (2006) estime ainsi que la pensée informatique est une compétence fondamentale et devrait faire partie des capacités analytiques de tous les enfants, au même titre que la lecture, l'écriture ou l'arithmétique. La nécessité pour tous les individus de maîtriser cette pensée informatique d'ici le milieu du XXI^{e} siècle est un constat partagé par de nombreux chercheurs et experts du domaine (Mohaghegh & McCauley, 2016 ; Ioannou & Makridou, 2018). De même, Nogry (2018) estime que la transmission d'une culture informatique est devenue un enjeu sociétal. L'initiation à la pensée informatique permettrait ainsi au futur cybercitoyen d'acquérir une certaine maîtrise et donc de dépasser le statut de simple consommateur du numérique.

En français, « pensée informatique » est une traduction imparfaite de l'expression anglaise originale « computational thinking ». Cette dernière comprend le mot « thinking » qui désigne davantage le « processus de penser » que le produit de ce processus, qui est la pensée elle-même. Même si les termes « pensée computationnelle » ou « processus de pensée informatique » seraient plus appropriés, il semble que l'expression « pensée informatique » se soit imposée comme la traduction la plus répandue de « computational thinking ». C'est ainsi que nous désignerons la pensée computationnelle/les processus de pensée informatiques tout au long de cet article.

Cet article présente une revue et une synthèse des outils d'évaluation de la pensée informatique les plus robustes disponibles à ce jour dans la littérature. Les apports et les limites de ces outils sont discutés. L'objectif est de permettre aux chercheurs et aux enseignants d'accroître leurs capacités à évaluer les apprentissages de leurs élèves. Bien que les différentes méthodes, approches ou tests soient généralement décrits en anglais dans les études citées et aient été testés dans des pays étrangers, il est possible et souhaitable que les enseignants s'en inspirent afin de bénéficier d'outils d'évaluation dont la validité a été établie de manière rigoureuse. D'où l'intérêt de cette contribution qui a pour but de faire connaître ces outils et de les rendre accessibles aux professionnels de l'éducation.

Le retour de la programmation en tant que discipline scolaire et la généralisation de l'intérêt porté à la pensée informatique sont deux phénomènes assez récents. Par conséquent, les connaissances actuelles sont encore limitées et ne font pas l'objet d'un véritable consensus scientifique. Il est donc essentiel d'effectuer un tour d'horizon des différentes approches disponibles pour évaluer les apprentissages dans ce domaine. Les outils présentés ont été utilisés et validés auprès d'élèves de l'école élémentaire à la fin du secondaire et comportent des échelles auto-évaluatives, des outils d'analyse des travaux des élèves (focalisés sur le résultat produit ou sur les processus ayant mené à ce résultat) ainsi que des tâches de résolution de problèmes. Nous proposerons également une nouvelle approche qui nous semble pertinente pour évaluer les apprentissages en programmation.

Apprendre à programmer pour développer une pensée informatique

Apprendre à programmer

Le terme « programmation » regroupe généralement un ensemble d'activités relativement disparates qui convergent vers un but identique : développer la capacité de l'élève à produire des algorithmes, c'est-à-dire des successions d'actions qui constituent autant d'étapes vers la résolution d'un problème algorithmique. La programmation est considérée depuis ses débuts comme un processus très complexe impliquant un grand nombre d'activités cognitives et de représentations mentales (Rogalski & Samurçay, 1990). Généralement, les activités de programmation nécessitent l'analyse d'un problème donné afin de déterminer un algorithme adéquat qu'il faudra par la suite retranscrire dans un langage de programmation (Rogalski, Samurçay & Hoc, 1988).

D'un point de vue psychologique, programmer c'est élaborer une conception fonctionnelle de la relation entre la solution d'un problème et la représentation opératoire de cette solution dans un langage de programmation (Samurçay & Rouchier, 1985). Cette nécessaire prise en compte du dispositif d'exécution d'un programme distingue la programmation des activités de résolution de problèmes classiques et induit une modification de la planification des actions. Samurçay et Rouchier (1985) ont soumis des lycéens (âgés de 15-16 ans) à un problème mathématique simple dont la solution était connue de tous et ont montré qu'un grand nombre d'entre eux ne sont pas parvenus à programmer cette solution en prenant en compte les limitations de la machine qui leur étaient imposées. Les auteurs en ont conclu que les élèves avaient des difficultés à se mettre spontanément dans une situation de production de procédures plutôt que dans une situation de production de résultats.

Dans une revue de la littérature, Robins, Rountree et Rountree (2003) distinguent deux composantes de l'apprentissage de la programmation qui sont d'un côté les connaissances et de l'autre les stratégies employées pour program-

mer et résoudre un problème. La question est alors de déterminer pourquoi et comment différentes stratégies peuvent émerger, et comment celles-ci sont liées aux connaissances sous-jacentes. De même, ils différencient la compréhension d'un programme et la capacité à en générer. Pour les auteurs, il est clair que ces deux derniers aspects sont liés, la compréhension jouant un rôle important dans la création d'un programme, mais ils suggèrent par ailleurs que les capacités des individus en ce qui concerne ces deux tâches ne sont pas forcément bien corrélées. Les tâches d'évaluation basées sur la mesure de la compréhension d'un programme ne sont pas nécessairement de bons indicateurs de la capacité à concevoir et rédiger des programmes.

La programmation demeure indissociable de la notion d'algorithme bien que cette notion ne soit pas propre à l'informatique et à la programmation. Dans un document rédigé à l'attention des enseignants, Tchounikine (2017) définit l'algorithme comme « un enchaînement mécanique d'actions, dans un certain ordre, qui chacune a un effet, et dont l'exécution complète permet de résoudre ou de faire quelque chose ». Aujourd'hui, programmer et coder sont parfois considérés à tort comme synonymes. Le codage ne constitue cependant qu'une des phases du processus de programmation, celle où l'on écrit le programme. Ce processus inclut également d'autres phases, notamment celle qui consiste à concevoir et développer des algorithmes (Lodi & Martini, 2021) qui correspond davantage à ce que l'on cherche à transmettre dans l'enseignement scolaire. Le « socle commun de connaissances, de compétences et de culture » (Ministère de l'Éducation nationale, de l'enseignement supérieur et de la recherche, 2015) indique notamment que l'élève « connaît les principes de base de l'algorithmique et de la conception des programmes informatiques. Il les met en œuvre pour créer des applications simples » (p. 4). Ce socle mentionne également les « langages informatiques [qui] sont utilisés pour programmer des outils numériques et réaliser des traitements automatiques de données (p. 4) ».

L'enseignement de la programmation à l'école s'appuie par ailleurs sur différents dispositifs censés faciliter la transmission des connaissances et compétences fondamentales dans cette discipline auprès d'un public jeune, en particulier les élèves d'école primaire. Les robots pédagogiques et les interfaces visuels basés sur des blocs à glisser-déposer (*drag & drop*) constituent l'essentiel des dispositifs qui sont utilisés aujourd'hui pour rendre la programmation accessible et attrayante (Bocconi, Chioccariello, Dettori, Ferrari & Engelhardt, 2016). Plusieurs auteurs ont également mis en évidence l'intérêt des activités débranchées comme moyen d'enseigner la programmation en éliminant complètement la charge cognitive liée à l'utilisation des machines et des outils numériques (Brackmann, Román-González, Robles, Moreno-León, Casali & Barone, 2017 ; Romero et al., 2018). Malgré plusieurs tentatives récentes visant à comparer l'effet de ces différents dispositifs sur les performances d'apprentissage des élèves (del Olmo-Muñoz, Cózar-Gutiérrez & González-Calero, 2020 ; Sigayret, Tricot & Blanc, 2022), beaucoup reste à faire pour identifier les apports et limites de ces outils.

Avant même de pouvoir envisager la possibilité d'évaluer l'apprentissage de la programmation, il est nécessaire de s'interroger sur les connaissances ou les compétences que cet apprentissage est censé développer chez l'apprenant. De nombreuses études ont tenté de mesurer l'impact des activités de programmation sur un large panel de facultés cognitives très générales. Généralement, l'ensemble des compétences cognitives spécifiques et des processus de résolution de problèmes qui interviennent notamment dans les activités de programmation sont regroupés sous le terme de « pensée informatique » qui désigne une capacité bien au-delà de la simple capacité à programmer. La pensée informatique peut plutôt être considérée comme l'ensemble des capacités cognitives de résolution de problèmes qui déterminent (entre autres) les compétences en programmation (Denning, 2017).

Dans une revue de la littérature relative à l'apprentissage de la pensée informatique à travers les activités de programmation, Lye et Koh (2014) affirment que la programmation est plus qu'une simple activité de codage, justement parce qu'elle implique que les apprenants manifestent une certaine maîtrise de la pensée informatique. Cependant, si programmation et pensée informatique sont résolument liées, il est nécessaire de préciser que la programmation peut également amener à la construction de savoirs distincts de cette pensée informatique, par exemple en relation avec la maîtrise technique des outils numériques utilisés. De même, la pensée informatique ne se développe pas exclusivement par le biais des activités de programmation et demeure une attitude et une compétence universellement applicable que tout le monde, pas seulement les informaticiens, devrait apprendre et utiliser (Wing, 2006).

Développer une pensée informatique

Roman-González, Pérez-González et Jiménez-Fernández (2017) distinguent trois types de définitions de la pensée informatique :

les définitions générales plutôt focalisées sur l'ensemble des aptitudes impliquées dans la résolution de problèmes (Wing, 2006 ; Wing, 2011 ; Aho, 2012) ;
les définitions opérationnelles qui fournissent un cadre en catégorisant la pensée informatique en différents sous-domaines ou processus cognitifs (CSTA & ISTE, 2011 ; Selby & Woollard, 2013) ;
les définitions pédagogiques qui listent un ensemble de concepts et de compétences (savoirs et savoir-faire) à enseigner dans un contexte éducatif (Brennan & Resnick, 2012 ; Zhang & Nouri, 2019).

Wing (2006) décrit la pensée informatique, ou pensée computationnelle (*computational thinking*), comme une manière de résoudre des problèmes ou de concevoir des systèmes qui s'appuie sur des concepts fondamentaux de la science informatique. Cette pensée informatique partage de nombreuses similarités avec la programmation, les mathématiques, l'ingénierie et, plus globalement, la pensée scientifique. C'est plus précisément « l'ensemble des processus de pensée impliqués dans la formulation d'un problème et l'expression de sa

solution de manière à ce qu'un ordinateur - humain ou machine - puisse effectivement l'exécuter » (Wing, 2011) ou de manière à ce que « sa solution puisse être représentée sous formes d'étapes de traitement ou de calcul (« *computational steps* ») et d'algorithmes » (Aho, 2012, p. 1).

Bien des années plus tôt, Papert (1980) fut le premier à utiliser l'expression « pensée informatique » en faisant référence à une aptitude mentale que les enfants développent en programmant avec le langage Logo. Il suggérait notamment que l'informatique pouvait offrir aux enfants de nouvelles possibilités d'apprentissage, de réflexion, de développement émotionnel et cognitif et favoriser la pensée procédurale par le biais des activités de programmation.

La définition exacte de la pensée informatique et de ce qu'elle implique demeure cependant sujette à interprétations et il n'existe pas de véritable consensus sur sa description. Selby et Woollard (2013) ont tenté d'en apporter une définition plus précise afin notamment de faciliter l'élaboration d'outils permettant d'évaluer les compétences en pensée informatique, dans un cadre scolaire par exemple. Pour ce faire, les auteurs ont compilé et analysé les articles relatifs à cette pensée informatique, dans l'optique d'en dégager les éléments récurrents qui pourraient contribuer à la définir de manière plus rigoureuse. Il en résulte que trois expressions ou termes apparaissent systématiquement dans la littérature. La pensée informatique semble ainsi inclure inévitablement la notion de « processus de pensée » ainsi que les termes « abstraction » (prise en compte simultanée de plusieurs niveaux d'abstraction) et « décomposition » (capacité à décomposer un problème complexe en plusieurs problèmes plus simples).

D'autres termes sont aussi largement utilisés mais de manière moins consensuelle. C'est le cas d'un ensemble de termes relatifs à des formes de pensée (« pensée logique », « pensée mathématique », « pensée heuristique », etc.) ou aux compétences de « résolution de problèmes ». En particulier, le terme « généralisation », qui désigne la capacité à transférer la résolution d'un problème particulier à tout un ensemble de problèmes plus généraux, et le terme « évaluation », qui correspond à la capacité à reconnaître et évaluer des résultats, sont des termes très souvent usités.

Selby et Woollard (2013) proposent une définition qui inclut ces mots qui sont les plus fréquemment utilisés. La pensée informatique est ainsi définie comme une activité, souvent orientée vers la production de quelque chose, associée (mais pas limitée) à la résolution de problèmes. Il s'agit, plus globalement, d'un processus cognitif qui reflète la capacité à penser en termes d'abstraction, de décomposition, d'évaluation, de généralisation et de manière algorithmique (capacité à écrire des instructions spécifiques et explicites pour chaque étape d'un processus). Cependant, ces processus identifiés par Selby et Woollard (2013) comme composantes principales de la pensée informatique ne sont pas nécessairement spécifiques à ce domaine et peuvent être retrouvés dans un grand nombre d'activités cognitives (liées à la lecture ou aux mathématiques par exemple). Ceci traduit plus généralement la difficulté actuelle des

chercheurs à définir la pensée informatique en des termes suffisamment précis et spécifiques.

La *Computer Science Teachers Association* et *l'International Society for Technology in Education* ont également tenté d'apporter une définition opérationnelle de la pensée informatique (CSTA & ISTE, 2011). La pensée informatique y est ainsi décrite comme un « processus de résolution de problèmes qui comprend les caractéristiques suivantes : formuler les problèmes d'une manière qui permette d'utiliser un ordinateur et d'autres outils pour aider à les résoudre ; organiser logiquement et analyser des données ; représenter des données par des abstractions telles que des modèles et des simulations ; automatiser des solutions à travers la pensée algorithmique ; identifier, analyser et mettre en œuvre des solutions possibles dans le but d'obtenir la combinaison la plus efficiente et efficace d'étapes et de ressources ; généraliser et transférer ce processus de résolution de problèmes à une grande variété de problèmes ». Plus tard, l'ISTE a également affirmé que ce sont d'autres compétences bien connues (la créativité, la pensée algorithmique, la pensée critique, la résolution de problèmes, la pensée coopérative et les compétences communicationnelles) qui, considérées conjointement, constituent cette nouvelle compétence que l'on nomme pensée informatique (ISTE, 2015, citée dans Korkmaz, Cakir & Özden, 2017).

On peut cependant remarquer que les définitions opérationnelles présentées ci-dessus ne sont pas spécifiques à l'apprentissage de la programmation. *A contrario*, Brennan et Resnick (2012) établissent un cadre théorique, comprenant 3 dimensions essentielles, qui permet de définir et d'enseigner la pensée informatique à l'école en s'appuyant sur le logiciel Scratch. Ils y distinguent :

– les concepts computationnels qui regroupent les notions à connaître en programmation : séquences (série d'étapes ou d'instructions), événements (action en provoquant une autre), boucles (répétition d'une partie du programme), opérateurs (expression mathématique ou logique), parallélisme (séquences d'instructions se produisant en même temps), conditions (exécution d'une partie du programme uniquement lorsqu'une certaine condition est remplie), données (stocker, retrouver et actualiser des valeurs).

– les pratiques computationnelles : être incrémental et itératif (planifier la conception d'un programme avant de l'implémenter), tester et débugger (développer des stratégies pour faire face aux problèmes et les anticiper), réutiliser et remixer (s'appuyer sur des programmes antérieurs ou réalisés par d'autres), abstraire et modulariser (construire un programme complexe en assemblant des parts plus petites).

– les perspectives computationnelles : s'exprimer, se connecter, questionner.

Dans leur revue systématique de la littérature consacrée à l'enseignement de la pensée informatique via Scratch, Zhang et Nouri (2019) identifient un ensemble de compétences qui complète les travaux de Brennan et Resnick (2012) : la capacité à lire, interpréter et communiquer du code ; la pensée prédictive (prédire l'exécution d'un programme avant de le concevoir), les notions d'entrées et de sorties, l'utilisation de supports multimodaux (capacité

à synchroniser des images, des sons, des actions…) ou encore les interactions hommes-machines (interactivité du programme créé).

D'autres modèles ont émergé pour tenter d'établir un cadre conceptuel solide permettant de définir précisément ce qui compose la pensée informatique (Grover & Pea, 2013 ; Kalelioglu, Gülbahar, & Kukul, 2016 ; Shute, Sun & Asbell-Clarke, 2017 ; Weintrop et al., 2016). Ces modèles recoupent partiellement ou totalement un ou plusieurs des modèles présentés précédemment. Toutefois, la multiplication de ces propositions théoriques, bien que souvent proches les unes des autres, traduit l'absence de consensus de la communauté sur cette question.

Il semblerait cependant que l'apprentissage de la programmation et de la pensée informatique soient inévitablement rattachés à la notion de compétence (« *Computational thinking skills* »), de façon plus ou moins explicite. Il est donc nécessaire de se demander ce qu'est une compétence et comment elle peut être évaluée. Van Lint (2016) analyse les définitions institutionnelles de la compétence issues de plusieurs pays européens pour mettre en évidence les composantes de cette notion. Les définitions sont globalement assez similaires. En France, la compétence se définit comme « l'aptitude à mobiliser des ressources (connaissances, capacités, attitudes) pour accomplir une tâche ou faire face à une situation complexe ou inédite » (Ministère de l'Éducation nationale, de l'enseignement supérieur et de la recherche, 2015).

Selon Van Lint (2016), la compétence se distingue ainsi de la connaissance qui regroupe un ensemble de concepts, d'énoncés, de définitions, de règles, de formules ou de faits à comprendre et à mémoriser, et de la procédure qui désigne un enchaînement organisé d'actions automatisées qui restent les mêmes dans toutes les situations. La compétence est plutôt une aptitude qui se révèle dans des situations complexes nouvelles qui nécessitent une certaine adaptation. Cette aptitude repose sur la mise en œuvre de connaissances et de procédures adéquates dans une situation donnée (Van Lint, 2016). Pour évaluer une compétence, il convient donc de placer l'élève face à une situation nouvelle et complexe (c'est-à-dire mobilisant plusieurs ressources) qui exige, de la part de l'apprenant, l'accomplissement d'une tâche.

En ce qui concerne l'apprentissage de la programmation dans l'enseignement scolaire français, Tchounikine (2017) identifie 5 notions fondamentales qui constituent les bases que les élèves doivent connaître et comprendre : la notion d'algorithme (ou de programme), la notion d'instruction (ou d'action), la notion de condition (« Si… alors… »), la notion de boucle et, éventuellement, la notion de variable. En outre, l'auteur présente les compétences sous-jacentes à la pensée informatique de la manière suivante : « savoir décomposer un problème en sous-problèmes plus simples ; savoir réfléchir aux tâches à accomplir pour résoudre un problème en termes d'étapes et d'actions ; savoir décrire les problèmes et les solutions à différents niveaux d'abstraction, ce qui permet d'identifier des similitudes entre problèmes et, par la suite, de pouvoir réutiliser des éléments de solution ». On retrouve ici des compétences similaires à celles décrites notamment par Selby et Woollard (2013).

D'une manière plus générale, certains chercheurs considèrent la pensée informatique comme un ensemble de compétences, exigeant des apprenants qu'ils développent à la fois des connaissances spécifiques à un domaine (par exemple, la programmation) et des aptitudes à la résolution de problèmes (Tang, Yin, Lin, Hadad & Zhai, 2020). Ceux-ci suggèrent que ces compétences pourraient être utilisées pour résoudre des problèmes de la vie quotidienne, dans différents domaines et à tous les niveaux scolaires.

Dans la suite de cet article, nous retenons que la pensée informatique peut regrouper : (a) la maîtrise de concepts informatiques : algorithme, instruction, séquence et parallélisme, événement, opérateurs, données, condition, boucle et variable et (b) la compétence à résoudre certains problèmes nouveaux à partir d'abstraction, de décomposition, d'évaluation, de généralisation et de manière algorithmique mais aussi (c) la capacité à lire, interpréter et communiquer du code.

Quels outils pour mesurer l'acquisition de la pensée informatique ?

La question de l'évaluation des compétences en pensée informatique est sujette à débat actuellement. La définition même de la pensée informatique et les compétences qu'elle met en jeu ne font pas consensus. Plusieurs outils sont proposés pour évaluer l'acquisition de la pensée informatique à l'école. De nombreux chercheurs recommandent d'ailleurs de pallier les difficultés rencontrées pour évaluer la pensée informatique en associant plusieurs outils complémentaires pour permettre d'avoir une idée plus précise du niveau des élèves (Roman-González, Moreno-León et Robles, 2019).

La méthode utilisée pour effectuer notre revue systématique de la littérature est décrite ci-dessous.

1. Nous avons utilisé la base de données transdisciplinaire Google Scholar. Le Tableau 1 présente les mots-clés qui ont été utilisés pour notre recherche ainsi que le nombre d'articles retenus. Des milliers d'articles abordent l'évaluation de la pensée informatique, ne serait-ce que dans leur partie théorique, sans pour autant être pertinents pour notre revue de littérature centrée sur les outils d'évaluation. Notre recherche était donc basée uniquement sur le titre des articles, ce qui nous a permis d'effectuer un premier filtrage en excluant tous ceux qui n'étaient pas véritablement centrés sur une présentation détaillée de ces outils d'évaluation. Les mots-clés sont présentés ici en anglais mais leurs équivalents français ont également été utilisés pour mener à bien cette sélection. Très peu d'études comprenant ces mots-clés dans leurs titres ont été publiées en français et aucune ne correspondait à nos critères d'inclusion. Cette recherche a été effectuée pour la dernière fois en décembre 2021.

2. Les références de chaque résultat obtenu à la suite de cette recherche ont été récupérées.

3. Après avoir lu le titre et le résumé, nous avons sélectionné 20 articles respectant les critères suivants :

Critères d'inclusion : seuls les articles, rédigés en français ou en anglais, centrés sur un outil, une méthode ou une approche visant à évaluer la maîtrise de la pensée informatique ont été sélectionnés. Nous avons également veillé à ce que les articles en question présentent une description précise de ces outils et/ou des données psychométriques attestant de leur validité et de leur fiabilité.

Critères d'exclusion : les doublons ont été exclus de notre sélection ainsi que les travaux non aboutis (études préliminaires, projets…) et les tests uniquement à destination d'une population adulte. Les approches qualitatives fondées sur des entretiens ou des observations ont également été exclues. Même si celles-ci peuvent présenter un intérêt certain, nous nous focalisons ici sur les outils d'évaluation permettant la récupération rapide de données quantitatives, objectives et fiables, qui ont l'avantage de faciliter la mesure de différents niveaux de performances et de rendre possible les comparaisons.

4. Les 20 articles ont par la suite été lus dans leur intégralité et analysés, ce qui nous a permis de découvrir 2 nouveaux articles qui apportent une contribution majeure et qui ont été intégrés à une deuxième sélection élargie et définitive (voir Tableau 1). Au total, 22 articles ont été retenus pour les besoins de cette revue de littérature.

Tableau 1. Sélection des articles en fonction des mots-clés utilisés.

Mots-clés	Résultats de la recherche	Articles sélectionnés	Total 1re sélection	Ajout d'articles	TOTAL
Computational Thinking Scale(s)	15	3	20	2	22
Computational Thinking Test	26	4			
Computational Thinking Assess	17	2			
Computational Thinking Assessing	61	4			
Computational Thinking Assessment	100	7			

Des échelles auto-évaluatives

Korkmaz et al. (2017) proposent les *Computational Thinking Scales* (CTS) qui regroupent des items issus de plusieurs tests distincts. Les auteurs s'appuient sur un document de l'ISTE (ISTE, 2015, citée dans Korkmaz et al., 2017) qui définit la pensée informatique comme un ensemble de compétences. Le CTS vise ainsi à mesurer les capacités créatives, la pensée algorithmique, la pensée critique, la capacité à résoudre des problèmes et les capacités coopératives. Tous les items qui les composent ont été récupérés à partir de tests préexistants

(*Problem Solving Scale, Cooperative Learning Attitude Scale, Scale of California Critical Thinking tendency*, etc.).

Le questionnaire est auto-administré, ce qui pose des problèmes liés à la subjectivité des réponses fournies par les élèves, et a été validé auprès de plus de 700 étudiants à l'université qui ont répondu aux 74 affirmations du test à l'aide d'une échelle de Likert en 5 points (jamais, rarement, parfois, souvent, toujours). Le test a donc suivi un processus rigoureux de validation mais la question de sa généralisation à des niveaux inférieurs, et donc de sa présence dans notre sélection, peut légitimement être posée. Pour autant, cette échelle a été testée avec succès auprès d'un millier d'étudiants chinois âgés de 15-16 ans (Korkmaz & Xuemei, 2019) ce qui nous laisse envisager la possibilité de pouvoir l'utiliser également en France à la fin de l'enseignement secondaire.

Cet outil d'évaluation est intéressant mais la pensée informatique est ici considérée comme la simple agrégation d'un ensemble de compétences bien connues et bien identifiées. S'il semble pertinent d'affirmer que la pensée créative ou la pensée critique sont liées à la pensée informatique, la proposition de Korkmaz *et al.* (2017) de considérer la seconde comme la combinaison des deux premières ainsi que d'autres facultés est aussi potentiellement critiquable car relativement réductrice. D'autres échelles du même type, visant à évaluer les perceptions et l'attitude des élèves vis-à-vis de la pensée informatique, ont été validées récemment, en particulier en ce qui concerne le sentiment d'auto-efficacité des élèves en relation avec la maîtrise de la pensée informatique (Weese & Feldhausen, 2017 ; Kukul & Karataş, 2019). Ces échelles présentent les mêmes limites que celle de Korkmaz et al. (2017) et s'adressent à un public plus jeune (plutôt les 10-14 ans). Globalement, les échelles auto-évaluatives peuvent être préconisées pour établir un autodiagnostic, faisant suite à une intervention ayant eu pour objectif de développer la pensée informatique.

Des outils d'analyse objectifs du code produit par l'élève

Brennan et Resnick (2012) proposent différentes façons d'évaluer la pensée informatique. Premièrement, en associant certains blocs du logiciel de programmation Scratch à certains concepts de la pensée informatique, il serait possible d'analyser de manière objective les projets Scratch réalisés par les élèves. Cependant, les chercheurs reconnaissent eux-mêmes que cette manière de procéder est limitée. L'utilisation de certaines instructions ne garantit pas que celles-ci soient parfaitement comprises et que les compétences associées soient bien maîtrisées.

De même, Alves, Von Wangenheim et Hauck (2019) font état de différents outils qui se fondent exclusivement sur l'analyse du code produit pour analyser les programmes réalisés sur des logiciels de programmation visuels, en particulier sur Scratch. Globalement, ce type d'évaluation est critiquable sur au moins un aspect majeur. Il admet que les compétences en pensée informatique sont visibles directement à travers l'analyse du code qui est généré par l'élève, ce qui

reste une supposition. Par ailleurs, cette approche limite l'évaluation de la maîtrise de la pensée informatique aux activités de programmation.

Dans ce registre, Dr. Scratch est un outil capable d'analyser automatiquement des fichiers Scratch, notamment pour évaluer différents aspects de la pensée informatique (Moreno-León & Robles, 2015). Un feedback est ensuite renvoyé à l'utilisateur avec trois niveaux de compétence (basique, développé, maîtrisé). Les auteurs ont réussi à montrer que ce feedback peut être bénéfique pour améliorer les performances de codage des élèves, et donc potentiellement la maîtrise sous-jacente des compétences liées à la pensée informatique (Moreno-León & Robles, 2015). Il existe par ailleurs de fortes corrélations entre les évaluations réalisées automatiquement via Dr. Scratch et les évaluations réalisées par des experts humains (Moreno-León, Román-González, Harteveld & Robles, 2017).

L'abstraction, la synchronisation, la représentation des données, l'interactivité, la pensée logique, le contrôle des flux (lié ici à l'utilisation pertinente de boucles « répéter ») et le parallélisme sont les compétences évaluées par Dr. Scratch. Ce sont des compétences très techniques, qui ne recoupent que partiellement les définitions de la pensée informatique de Brennan et Resnick (2012) ou Grover et Pea (2013). Ces compétences semblent parfois plus liées à la bonne utilisation du logiciel qu'à la pensée informatique elle-même. Certaines ne paraissent pas toujours pertinentes. Par exemple, l'interactivité du programme est un critère d'évaluation pris en compte par Dr. Scratch bien que ce critère ne semble pas directement relié à la maîtrise de la pensée informatique selon la grande majorité des auteurs ayant tenté de définir la pensée informatique.

Ce type d'outils est le plus souvent utilisé dans le cadre d'une évaluation formative-itérative qui consiste à évaluer les élèves pendant le processus d'apprentissage afin de leur permettre de développer et d'améliorer leurs compétences en pensée informatique en amélioration les programmes qui sont conçus. Ils sont conçus pour fonctionner avec un environnement de programmation particulier (et sont donc spécifiques à un langage ou à un logiciel), ce qui constitue également une limite à leur utilisation. Ils peuvent toutefois être utilisés à tous les âges, puisqu'ils se basent uniquement sur les programmes réalisés par les élèves. Par exemple, Dr. Scratch peut être adapté à tous les élèves qui ont appris à programmer sur Scratch, quel que soit leur niveau ou leur âge.

Des tâches de résolution de problèmes

Il existe d'autres démarches intéressantes qui diffèrent de la simple analyse du code produit par un élève et qui consistent en différentes tâches que l'élève doit accomplir. La réussite dans l'accomplissement de ces tâches est ainsi censée traduire la maîtrise de certains aspects de la pensée informatique. Grover, Cooper et Pea (2014) ont utilisé des captures d'écran de Scratch accompagnées de questions pour évaluer les apprentissages (« pourquoi ce pro-

gramme ne fonctionne pas ? » « Quand ce programme est exécuté, quelle valeur nous est renvoyée ? », etc.). Brennan et Resnick (2012) ont demandé aux élèves de partir d'un programme existant pour l'améliorer ou rajouter de nouvelles fonctionnalités définies par l'expérimentateur. De leur côté, Atmazidou et Demetriadis (2016) ont mis en place des exercices de résolution de problème qui impliqueraient l'utilisation de compétences identifiées comme faisant partie de la pensée informatique. Il était par exemple demandé d'identifier les structures communes qui guidaient le comportement d'un robot dans deux tâches différentes (abstraction), de proposer une solution plus générale à un problème (généralisation) etc. La pensée informatique étant très souvent associée au processus de résolution de problèmes, ce type d'outils d'évaluation semble pertinent, notamment dans le cadre d'activités de programmation pour lesquelles il est fréquent de considérer que les connaissances sont construites et évaluées par la résolution de problèmes (Rogalski & Samurçay, 1990).

L'une des tentatives les plus abouties en ce sens reste probablement le *Computational Thinking Test* ou *CTt* (González, 2015), dont la pertinence des items a été mesurée par un jury d'experts. Le *CTt* se compose de 28 items qui visent à estimer la maîtrise de certaines notions d'informatique : les instructions basiques, les boucles, les conditions et les fonctions. Le test est centré sur des exercices nécessitant la réalisation d'un parcours pour lesquels les participants ont un choix de réponse limité et prédéterminé. Ces réponses sont présentées sous forme de texte, de flèches ou de blocs similaires à ceux utilisés dans les logiciels de programmation visuels. Dans ces exercices, il est alternativement demandé au participant de trouver la séquence qui permet de résoudre le problème, de compléter l'instruction manquante dans un programme qui est proposé ou encore d'y trouver l'erreur qui empêche ce programme de répondre au problème posé.

Le test semble un excellent moyen de mesurer la compréhension de certaines notions fondamentales en programmation (« *computational concepts* », Brennan & Resnick, 2012) telles que les boucles ou les structures conditionnelles et la compréhension de leur fonctionnement. Son utilisation est généralement préconisée pour établir un diagnostic du niveau de l'élève à un moment donné et permet de comparer facilement les performances obtenues avant et après une intervention éducative (comparaison pré-test/post-test). Il présente par ailleurs une validité convergente, puisque les performances au *CTt* sont corrélées aux performances à d'autres tests mesurant les capacités spatiales, de raisonnement ou de résolution de problèmes (Roman-González *et al.*, 2017), ainsi que d'une validité prédictive vis-à-vis des performances académiques et de la réussite en programmation au collège (Roman-González, Pérez-González, Moreno-León & Robles, 2018). Enfin, il a été testé (et semble adapté) auprès des élèves dont le niveau se situe de la fin de l'école élémentaire jusqu'au début du lycée (de 9 à 16 ans).

Cependant, ce qui relève des notions et des savoirs semble confondu avec ce qui relève plus de la compétence et du savoir-faire dans ce CTt. González (2015) reconnaît que la structure du test, qui n'est composé que de questions

fermées à choix multiples, ne permet de mesurer le degré de maîtrise de la pensée informatique qu'en ce qui concerne ses aspects les plus simples, c'est-à-dire la capacité à comprendre les concepts mis en jeu et à reconnaître la solution à un problème posé parmi plusieurs possibilités données. Un instrument qui aurait pour objectif de mesurer également les aspects les plus complexes de la pensée informatique devrait alors demander à l'élève de montrer qu'il est capable d'assimiler et d'appliquer ces concepts pour résoudre des problèmes en proposant lui-même les solutions adéquates.

Certains outils proposent par ailleurs une évaluation de la pensée informatique fondée sur la résolution de problèmes indépendants des activités de programmation. Ces outils visent à évaluer dans quelle mesure les élèves sont capables de transférer leurs compétences en pensée informatique à une grande variété de problèmes, situations ou contextes nouveaux. C'est le cas notamment des épreuves du *Bebras International Contest* en Lituanie. La capacité à transférer ces compétences dans d'autres contextes, relatifs à des problèmes de la vie quotidienne, peut y être évaluée, y compris pour des élèves n'ayant jamais été initiés à la programmation. Roman-González *et al.* (2019) ont montré que le *CTt*, Dr. Scratch et les tâches Bebras peuvent être considérés comme des outils complémentaires, présentant des liens de corrélations et permettant d'évaluer différents niveaux d'acquisition de la pensée informatique.

Très récemment, plusieurs études ont proposé et validé de nouveaux outils visant à évaluer la capacité des élèves à résoudre des problèmes de programmation, en utilisant des langages « neutres », permettant à tous les élèves de réaliser ces tests, quels que soient les logiciels ou langages qu'ils ont appris à utiliser (Basu, Rutstein, Xu & Shear, 2020 ; Kong & Wang, 2021 ; Relkin, de Ruiter & Bers, 2020). Ces nouveaux outils ont été validés auprès d'élèves relativement jeunes, âgés de 5 à 11 ans. La résolution des problèmes présentés est ainsi censée révéler, non pas la maîtrise de certains concepts en informatique, comme pour le *CTt*, mais plutôt la maîtrise de certains processus cognitifs (abstraction, pensée algorithmique…) proches de ceux définis par Brennan et Resnick (2012) ou Selby et Woollard (2013) comme faisant partie de la pensée informatique.

À ce titre, le *Computational Thinking Assessment for Chinese Elementary Students* (*CTA-CES*, pas encore adapté dans les pays occidentaux) semble particulièrement prometteur (Li, Xu & Liu, 2021). Là encore, le cadre défini par Selby et Woolard (2013) pour décrire la pensée informatique constituait la base sur laquelle se sont appuyés les auteurs. Ceux-ci ont proposé un test comportant 25 items, similaires aux tâches Bebras dans le sens où les problèmes à résoudre ne sont pas des problèmes de programmation mais plutôt des problèmes de la vie quotidienne, à destination d'élèves âgés de 8 à 12 ans. En plus des méthodes psychométriques classiques utilisées pour valider cet outil, les auteurs ont réalisé des images par résonance magnétique (IRMf). Celles-ci ont montré que les activités neuronales des participants durant la réalisation de ce test et durant la réalisation d'activités de programmation sont corrélées. Ceci suggère que programmer et effectuer le CTA-CES sont deux activités qui impliquent les

mêmes processus cérébraux, ce qui constitue un indice supplémentaire en faveur de la validité de ce test, selon les auteurs.

Cependant, comme les autres tests évoqués précédemment, le CTA-CES ne propose que des questions à choix multiples et n'évalue donc pas la capacité des élèves à générer eux-mêmes leur propre solution à un problème. À notre connaissance, seuls Chen, Shen, Barth-Cohen, Jiang, Huang et Eltoukhy (2017) ont élaboré (et validé) un outil d'évaluation de la pensée informatique (selon la définition opérationnelle proposée par CSTA & ISTE, 2011) centré sur la résolution de problèmes et partiellement composé de questions ouvertes pour lesquelles les élèves (âgés de 9-10 ans) ne pouvaient pas choisir entre plusieurs solutions préétablies. Cependant, cet outil s'intègre dans le cadre d'un apprentissage de la programmation réalisé à l'aide de robots pédagogiques et semble difficile à adapter dans un autre contexte.

Les tâches de résolution de problèmes font généralement appel à des compétences qui dépassent le strict apprentissage de la programmation informatique. Pour cette raison, elles peuvent être utilisées auprès d'élèves n'ayant pas (encore) appris à programmer et se prêtent donc particulièrement bien aux comparaisons pré-test/post-test (évaluations avant et après une séquence pédagogique) qui permettent de facilement mesurer les bénéfices d'une intervention éducative.

Évaluer l'apprentissage « implicite » de la pensée informatique

Rowe *et al.* (2021) ont récemment proposé une approche innovante qui consiste à évaluer l'apprentissage implicite des élèves confrontés à une tâche faisant appel à certaines compétences liées à la pensée informatique. L'apprentissage implicite fait ici référence à un apprentissage dont la manifestation peut être faite lors de l'activité d'apprentissage elle-même (et non de manière différée avec des tests). L'outil présenté par Rowe *et al.* (2021) peut être considéré comme un outil de *data-mining* (exploration des données, en français) qui récupère et enregistre l'activité de l'élève en temps réel.

Les auteurs utilisent le jeu vidéo éducatif *Zoombinis*, composé d'épreuves basées sur la logique et la déduction. L'activité du joueur/élève pendant la résolution des problèmes posés est enregistrée sous la forme de données numériques qui peuvent ainsi être analysées et répliquées. Ici, ce n'est pas la capacité à résoudre les problèmes qui est évaluée mais bien les processus qui ont mené à cette résolution. À travers les données recueillies, les auteurs ont construit manuellement des outils permettant de détecter la maîtrise implicite de la pensée informatique des élèves âgés de 8 à 14 ans.

Les actions réalisées par l'élève lors de la résolution du problème permettent ainsi de mettre en évidence sa capacité à décomposer le problème en sous problèmes plus simples, à identifier une règle sous-jacente au problème, à tester de manière systématique différentes hypothèses ou encore à réutiliser certaines stratégies qui avaient fonctionné précédemment. Ces comportements (abstraction, décomposition…) sont caractéristiques de la pensée informatique selon

de nombreux auteurs (Brennan & Resnick, 2012 ; Selby & Woollard, 2013 ; Shute et al., 2017…).

Cette approche présente l'originalité de proposer une évaluation formative en temps réel fondée sur des activités ludiques, de mettre en évidence des comportements révélateurs d'une maîtrise émergente de la pensée informatique et de se placer aux antipodes des évaluations plus classiques sous forme de tests en ne s'appuyant sur aucune représentation visuelle ou verbale de la pensée informatique. Cependant, les connaissances techniques et le temps nécessaires pour récupérer et analyser les données du jeu et en déduire certaines compétences sont un véritable frein à la généralisation de cette méthode d'évaluation.

La nécessité d'évaluer distinctement la maîtrise de concepts et la compétence à résoudre des problèmes

L'identification des concepts et des compétences à résoudre des problèmes d'une certaine manière, qui constituent la maîtrise de la pensée informatique, est une première étape vers notre capacité à les évaluer. L'apprentissage peut être considéré comme un changement relativement permanent de nos connaissances (Mayer, 2011). Or, certains de ces changements ne sont pas directement observables : ils sont invisibles. Pour résoudre ce problème, Tricot et Musial (2020) proposent de considérer que l'apprentissage concerne un couple {tâche ; connaissance} dans lequel la connaissance est ce qui permet de réaliser une tâche et, réciproquement, la réalisation d'une tâche permet l'acquisition d'une connaissance. La réalisation d'une tâche au moment de l'évaluation peut donc être révélatrice d'une connaissance intrinsèquement invisible.

Les mêmes auteurs définissent la compétence comme un ensemble {Tâche ; Connaissances théoriquement nécessaires à la tâche ; Connaissances issues de la tâche}. La compétence correspond ainsi à la « capacité d'un individu à réaliser une certaine tâche et pour laquelle on peut décrire l'ensemble des connaissances nécessaires à la réalisation de cette tâche », certaines de ces connaissances étant spécifiquement issues de la pratique de la tâche. En suivant ce modèle, et en considérant que la pensée informatique est la compétence majeure que l'enseignement de la programmation cherche à développer, nous pouvons en tirer la conclusion suivante : la pensée informatique peut être considérée comme un ensemble regroupant une tâche (la résolution de problèmes algorithmiques), des concepts nécessaires à la réalisation de cette tâche (les notions fondamentales en programmation) et des connaissances pratiques issues de cette tâche qui permettent au sujet d'utiliser les stratégies adéquates pour y faire face (décomposition, généralisation, abstraction…).

Dans ce modèle, les connaissances pratiques issues des activités de programmation permettent le développement des compétences identifiées précédemment (notamment Selby & Woolard, 2013 ; Tchounikine, 2017) comme relevant de la maîtrise de la pensée informatique et de son utilisation. Ce modèle est également conforme à l'idée que la pensée informatique s'appuie

sur des concepts issus de l'informatique (Wing, 2006) en considérant que les notions fondamentales en programmation constituent les connaissances théoriques nécessaires à la résolution de problèmes algorithmiques, du moins dans le cadre de l'apprentissage de la programmation.

Globalement, les outils de mesure de la pensée informatique que nous avons détaillés précédemment sont des tentatives louables et pertinentes d'évaluer cette compétence. Cependant, ils ne proposent actuellement aucune distinction claire entre d'un côté les connaissances des élèves et d'un autre leur capacité à appliquer ces connaissances pour résoudre des problèmes complexes. Généralement, ces deux aspects fondamentaux sont confondus, si bien qu'on ne sait pas toujours si les outils d'évaluation proposés permettent de mesurer le niveau de compréhension des élèves vis-à-vis de certaines notions ou leur niveau de maîtrise d'une compétence qui leur permet de faire face à des situations inédites pour proposer des solutions en s'appuyant sur leurs connaissances. Or, nous considérons que cette distinction est capitale.

Si l'objectif est de développer des outils d'évaluation permettant de discriminer différents niveaux de maîtrise de la pensée informatique, alors il faut pouvoir en distinguer les aspects les plus fondamentaux, la connaissance et la compréhension des notions et concepts, et les aspects les plus complexes, l'assimilation et l'application pratique de ces concepts et le développement de stratégies permettant la résolution de problèmes. Une évaluation distincte de ces deux aspects de l'apprentissage de la programmation permettrait à l'enseignant d'affiner son diagnostic quant aux éventuelles difficultés rencontrées par ses élèves. Ceux-ci maîtrisent-ils suffisamment les concepts fondamentaux utilisés en programmation ? Si c'est le cas, sont-ils capables de les appliquer dans un contexte pratique en utilisant les stratégies adéquates ?

Cette distinction entre la maîtrise de concepts et compétence à résoudre des problèmes s'accorde d'ailleurs particulièrement bien avec les attendus de fin de cycle 4 (en France) en relation avec l'informatique et la programmation, bien que ceci ne constitue pas un argument scientifique en faveur de notre modèle. La capacité à « écrire, mettre au point et exécuter un programme » y figure ainsi que la maîtrise des « notions d'algorithme et de programme, notion de variable informatique, déclenchement d'une action par un événement, séquences d'instructions, boucles, instructions conditionnelles ».

Conclusion

L'apprentissage de la programmation est une discipline qui intègre ou réintègre progressivement les programmes scolaires pour répondre aux besoins éducatifs entraînés par les changements de notre société. La programmation semble potentiellement indiquée pour acquérir les concepts de la « pensée informatique », une façon de concevoir des systèmes et de résoudre des problèmes, proche de l'informatique et de la méthode scientifique, dont le développement

et la maîtrise constitueraient un enjeu sociétal majeur du XXIe siècle. Cependant, la définition même de la pensée informatique ne fait pas l'objet d'un consensus entre les chercheurs. D'où le problème de l'évaluation de sa maîtrise qui en découle.

Diverses possibilités sont proposées aux enseignants pour tenter de mesurer les compétences en pensée informatique de leurs élèves. Parmi celles-ci, on retrouve des échelles, telles que celle intitulée *Computational Thinking Scales* qui propose d'évaluer certaines facultés identifiées comme proche de la pensée informatique (capacité de résolution de problèmes, pensée créative…) et qui s'adressent plutôt aux élèves les plus âgés (fin du lycée). Le caractère auto-évaluatif de ces échelles demeure cependant une limite importante à cette approche.

Certains outils, comme Dr. Scratch, peuvent également analyser « objectivement » et automatiquement des projets réalisés sur le logiciel Scratch pour estimer la maîtrise de la pensée informatique. Néanmoins, cette approche se limite finalement à l'observation des artefacts d'apprentissage des élèves et ne peut suffire à acquérir une vision globale de leur niveau.

Une autre possibilité consiste à soumettre les élèves à des exercices de résolution de problèmes. À ce titre, le *Computational Thinking Test* semble l'outil le plus fiable aujourd'hui pour permettre d'évaluer la maîtrise de certaines notions fondamentales en programmation. De nombreux outils d'évaluation, basés sur des exercices de résolution de problèmes, sont également disponibles pour mettre en évidence la maîtrise de certains processus cognitifs décrits comme des composantes de la pensée informatique. Cependant, la plupart d'entre eux ne comporte que des questions à choix multiples et ne permettent donc pas d'évaluer la capacité des élèves à générer eux-mêmes leur propre solution à un problème donné.

Enfin, une dernière solution consiste à évaluer non pas la capacité à réaliser une tâche mais les plutôt les processus qui ont mené à la réalisation de cette tâche et qui, dans le cadre de jeux basés sur la résolution de problèmes logiques, peuvent être révélateurs d'une certaine maîtrise de la pensée informatique. Mais cette solution paraît difficile à généraliser à grande échelle.

Ces outils peuvent bien sûr être combinés pour garantir une évaluation plus précise des compétences. Cependant, il subsiste encore de larges incertitudes sur la manière d'évaluer l'acquisition de la pensée informatique auprès d'enfants et d'adolescents, dans le cadre d'activités de programmation. De nouveaux outils, qui distinguent d'une part la compréhension des notions fondamentales en programmation et d'autre part la capacité à résoudre des problèmes algorithmiques, doivent être proposés.

Références

Aho, A. V. (2012). Computation and computational thinking. *The computer journal*, *55*(7), 832-835.

Alves, N. D. C., Von Wangenheim, C. G., & Hauck, J. C. (2019). Approaches to assess computational thinking competences based on code analysis in K-12 education: A systematic mapping study. *Informatics in Education*, *18*(1), 17-39.

Basu, S., Rutstein, D., Xu, Y., & Shear, L. (2020, February). A principled approach to designing a computational thinking practices assessment for early grades. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (pp. 912-918).

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., & Engelhardt, K. (2016). *Developing computational thinking in compulsory education-Implications for policy and practice* (No. JRC104188). Joint Research Centre (Seville site).

Bourgeois, A., Birch, P., & Davydovskaia, O. (2019). *Digital Education at School in Europe. Eurydice Report*. Education, Audiovisual and Culture Executive Agency, European Commission.

Brackmann, C. P., Román-González, M., Robles, G., Moreno-León, J., Casali, A., & Barone, D. (2017, November). Development of computational thinking skills through unplugged activities in primary school. In *Proceedings of the 12th workshop on primary and secondary computing education* (pp. 65-72).

Brennan, K., & Resnick, M. (2012, April). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American educational research association* (pp. 1-25). Vancouver, Canada.

Chen, G., Shen, J., Barth-Cohen, L., Jiang, S., Huang, X., & Eltoukhy, M. (2017). Assessing elementary students' computational thinking in everyday reasoning and robotics programming. *Computers & Education*, *109*, 162-175.

CSTA & ISTE (2011). Operational Definition of Computational Thinking for K–12 Education. *National Science Foundation*.

del Olmo-Muñoz, J., Cózar-Gutiérrez, R., & González-Calero, J. A. (2020). Computational thinking through unplugged activities in early years of Primary Education. *Computers & Education*, *150*, 103832.

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, *60*(6), 33-39.

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational researcher*, *42*(1), 38-43.

Grover, S., Cooper, S., & Pea, R. (2014, June). Assessing computational learning in K-12. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 57-62).

Kalelioglu, F., Gülbahar, Y., & Kukul, V. (2016). A framework for computational thinking based on a systematic research review. *Baltic Journal of Modern Computing*, *4*(3), 583.

Kong, S. C., & Wang, Y. Q. (2021). Item response analysis of computational thinking practices: Test characteristics and students' learning abilities in visual programming contexts. *Computers in Human Behavior*, *122*, 106836.

Korkmaz, Ö., Cakir, R., & Özden, M. Y. (2017). A validity and reliability study of the computational thinking scales (CTS). *Computers in Human Behavior*, *72*, 558-569.

Korkmaz, Ö., & Xuemei, B. A. İ. (2019). Adapting computational thinking scale (CTS) for Chinese high school students and their thinking scale skills level. *Participatory Educational Research*, *6*(1), 10-26.

Kukul, V., & Karatas, S. (2019). Computational thinking self-efficacy scale: Development, validity and reliability. *Informatics in Education*, *18*(1), 151-164.

Li, Y., Xu, S., & Liu, J. (2021). Development and validation of computational thinking assessment of Chinese elementary school students. *Journal of Pacific Rim Psychology*, *15*, 18344909211010240.

Lodi, M., & Martini, S. (2021). Computational thinking, between Papert and Wing. *Science & Education*, *30*(4), 883-908.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12?. *Computers in Human Behavior*, *41*, 51-61.

Mayer, R. E. (2011). Does styles research have useful implications for educational practice? *Learning and Individual Differences*, *21*(3), 319-320.

Ministère de l'Education nationale, de l'enseignement supérieur et de la recherche (2015). Socle commun de connaissances, de compétences et de culture. Décret nº 2015-372. *Bulletin officiel de l'Éducation nationale, de l'enseignement supérieur et de la recherche, 23 avril*.

Mohaghegh, D. M., & McCauley, M. (2016). Computational thinking: The skill set of the 21st century. *International Journal of Computer Science and Information Technologies*, *7*(3), 1524-1530.

Moreno-León, J., & Robles, G. (2015, November). Dr. Scratch: A web tool to automatically evaluate Scratch projects. In *Proceedings of the workshop in primary and secondary computing education* (pp. 132-133).

Moreno-León, J., Román-González, M., Harteveld, C., & Robles, G. (2017, May). On the automatic assessment of computational thinking skills: A comparison with human experts. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (pp. 2788-2795).

Nogry, S. (2018). Comment apprennent les élèves au cours d'une séquence de robotique éducative en classe de CP ?. *De 0 à 1 ou l'heure de l'informatique à l'école* (pp. 235-246). Lausanne : Peter Lang.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books.

Popat, S., & Starkey, L. (2019). Learning to code or coding to learn? A systematic review. *Computers & Education*, *128*, 365-376.

Psycharis, S., & Kallia, M. (2017). The effects of computer programming on high school students' reasoning skills and mathematical self-efficacy and problem solving. *Instructional Science*, *45*(5), 583-602.

Relkin, E., de Ruiter, L., & Bers, M. U. (2020). TechCheck: Development and validation of an unplugged assessment of computational thinking in early childhood education. *Journal of Science Education and Technology*, *29*, 482-498.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer science education*, *13*(2), 137-172.

Rogalski, J., & Samurçay, R. (1990). Acquisition of programming knowledge and skills. In *Psychology of programming* (pp. 157-174). Academic Press.

Rogalski, J., Samurçay, R., & Hoc, J. M. (1988). L'apprentissage des méthodes de programmation comme méthodes de résolution de problème. *Le travail humain*, *51*(4), 309-320.

Román-González, M., Pérez-González, J. C., Moreno-León, J., & Robles, G. (2018). Can computational talent be detected? Predictive validity of the Computational Thinking Test. *International Journal of Child-Computer Interaction*, *18*, 47-58.

Román-González, M., Moreno-León, J., & Robles, G. (2019). Combining assessment tools for a comprehensive evaluation of computational thinking interventions. In *Computational thinking education* (pp. 79-98). Springer, Singapore.

Romero, M., Viéville, T., Duflot-Kremer, M., de Smet, C., & Belhassein, D. (2018, August). Analyse comparative d'une activité d'apprentissage de la programmation en mode branché et débranché. In *Educode-Conférence internationale sur l'enseignement au numérique et par le numérique*.

Rowe, E., Almeda, M. V., Asbell-Clarke, J., Scruggs, R., Baker, R., Bardar, E., & Gasca, S. (2021). Assessing implicit computational thinking in Zoombinis puzzle gameplay. *Computers in Human Behavior*, *120*, 106707.

Samurçay, R., & Rouchier, A. (1985). De « faire » à « faire faire » : planification d'actions dans la situation de programmation. *Enfance*, *38*(2), 241-254.

Scherer, R., Siddiq, F., & Sánchez Viveros, B. (2019). The cognitive benefits of learning computer programming: A meta-analysis of transfer effects. *Journal of Educational Psychology*, *111*(5), 764.

Selby, C., & Woollard, J. (2013). Computational thinking: the developing definition. In *Paper presented at the 18th annual conference on innovation and technology in computer science education, Canterbury*.

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142-158.

Sigayret, K., Tricot, A., & Blanc, N. (2022). Unplugged or plugged-in programming learning: A comparative experimental study. *Computers & Education*, 104505.

Tang, X., Yin, Y., Lin, Q., Hadad, R., & Zhai, X. (2020). Assessing computational thinking: A systematic review of empirical studies. *Computers & Education*, *148*, 103798.

Tchounikine, P. (2017). Initier les élèves à la pensée informatique et à la programmation avec Scratch. *Laboratoire d'informatique de Grenoble*. En ligne : http://ligmembres.imag.fr/tchounikine/PenseeInformatiqueEcole.html.

Tricot, A., & Musial, M. (2020). *Précis d'ingénierie pédagogique*. De Boeck Supérieur.

Van Lint, S. (2016). La notion de compétence et son évaluation. *MARS*. Technologie, *Sciences et techniques industrielles*, 202, 30-33.

Weese, J. L., & Feldhausen, R. (2017, June). STEM outreach: Assessing computational thinking and problem solving. In *2017 ASEE Annual Conference & Exposition*.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of science education and technology*, *25*(1), 127-147.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35.

Wing, J. M. (2011). Research Notebook: Computational Thinking. What and Why? *The Link*. Pittsburg : Carnegie Mellon.

Zhang, L., & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. *Computers & Education*, *141*, 103607.

Annexe C

Test de compréhension des notions de programmation

VERSION SCRATCH

1. Prénom

_____

2. Nom de famille

_____

3. Date

_____

4. Classe

_____

5. Ecole

_____

6. Coche les textes ci-dessous qui sont des algorithmes ou des programmes.

Plusieurs réponses possibles.

☐ Mettre la farine dans un récipient - Ajouter les œufs, le sucre et le beurre - Mélanger délicatement avec un fouet - Cuire la pâte à la poêle

☐ Avancer de 3 cases - Tourner à droite - Avancer de 1 case - Dire « Bravo ! »

☐ Le petit chien entra dans sa niche - Dehors, la pluie ne cessait de tomber - Il coucha sa tête sur le sol - Et attendit le retour du soleil

☐ Avancer une pièce sur l'échiquier - Si certain de capturer le roi au prochain tour - Alors crier « échec et mat ! » - Sinon continuer la partie

7. Coche les phrases qui peuvent être utilisées comme instructions dans un algorithme ou un programme.

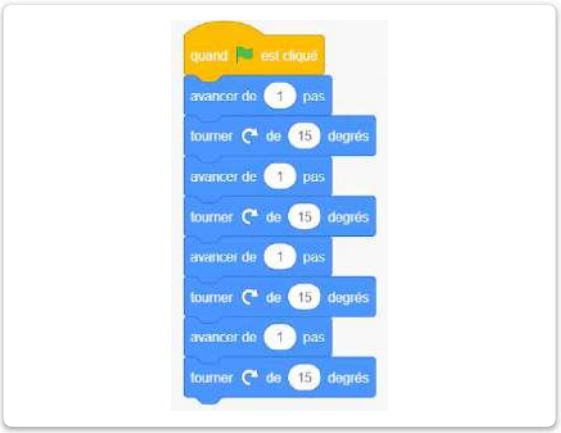
Plusieurs réponses possibles.

- [ ] Répéter 4 fois
- [ ] Le robot est rouge
- [ ] Jouer le son « Musique »
- [ ] Programmer est difficile
- [ ] Tourner de 45°
- [ ] Afficher la réponse

8. Parmi les 4 programmes ci-dessous, lesquels contiennent une boucle ?

Plusieurs réponses possibles.

- [ ] Programme 1

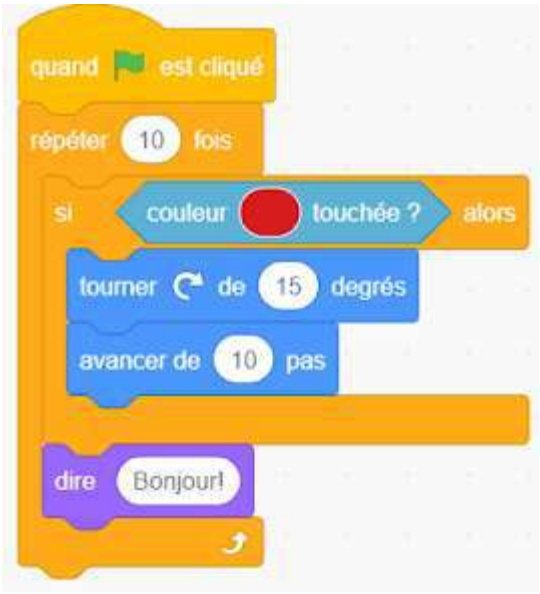


- [ ] Programme 2

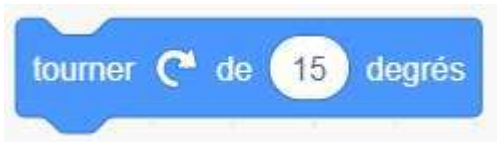
- [ ] Programme 3

- [ ] Programme 4

9. Dans le programme ci-dessous, quelles instructions sont exécutées par l'ordinateur uniquement lorsque le lutin touche une certaine couleur.



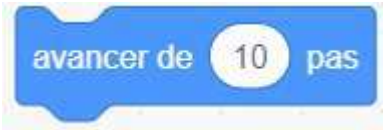
Plusieurs réponses possibles.



☐ Tourner à droite de 15 degrés

☐ Répéter 10 fois

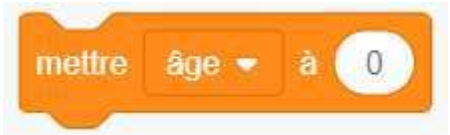
☐ Dire "Bonjour!"



☐ Avancer de 10 pas

10. Dans le programme ci-dessous, quel est le nom de la variable utilisée ? Coche la bonne réponse.

Une seule réponse possible.



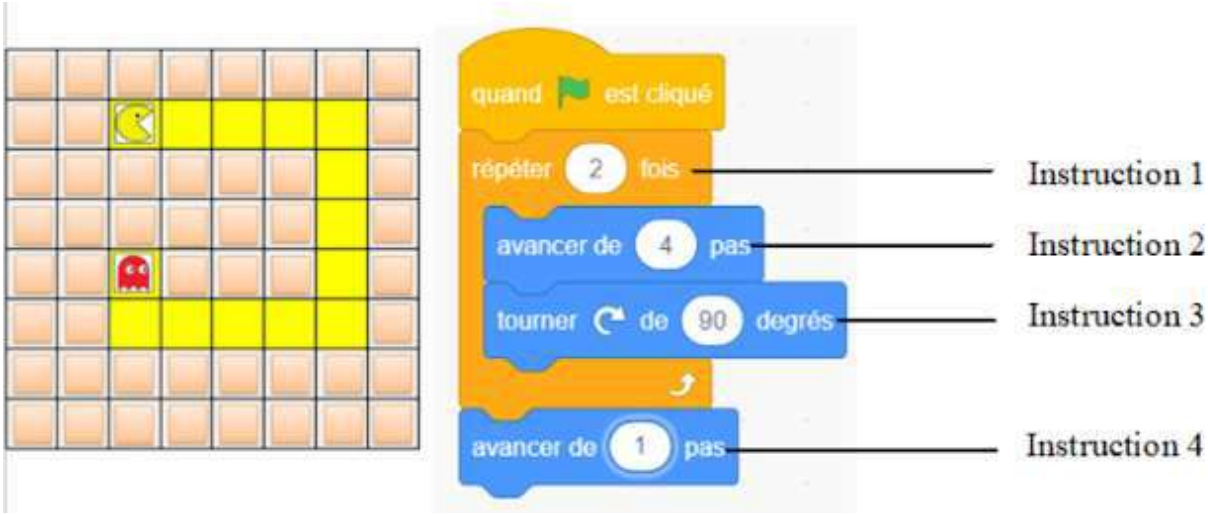
◯ Mettre âge à 0

◯ Ajouter 1 à âge

◯ Pas

◯ Âge

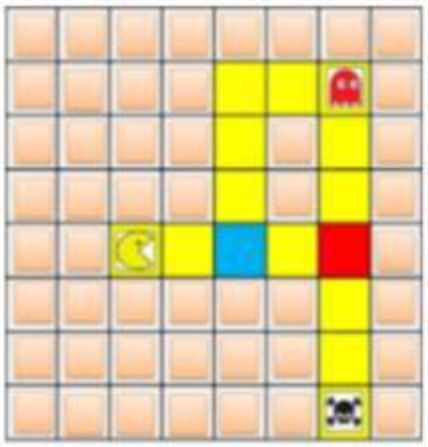
11. Pac-man (en haut) doit atteindre le fantôme (en bas) dans le labyrinthe ci-dessous. Le programme qui est proposé comporte une erreur. Quelle instruction est fausse et empêche Pac-man d'atteindre le fantôme ? (1 pas = 1 case).



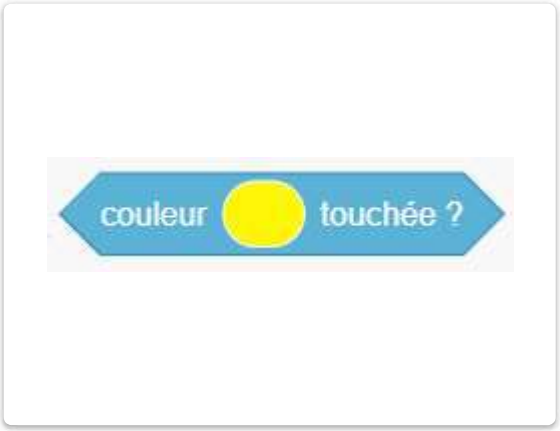
Une seule réponse possible.

- L'instruction 1 est fausse.
- L'instruction 2 est fausse.
- L'instruction 3 est fausse.
- L'instruction 4 est fausse.

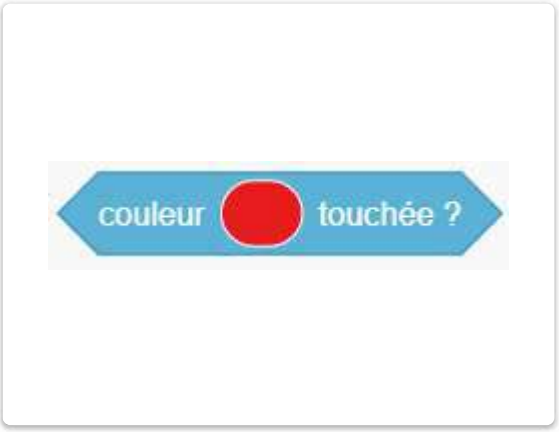
12. Pac-man (en jaune) doit atteindre le fantôme (en rouge). Dans le programme proposé plus bas, il manque une instruction (à l'endroit ou il y a ???). Laquelle ? Coche la bonne réponse. (1 pas = 1 case).



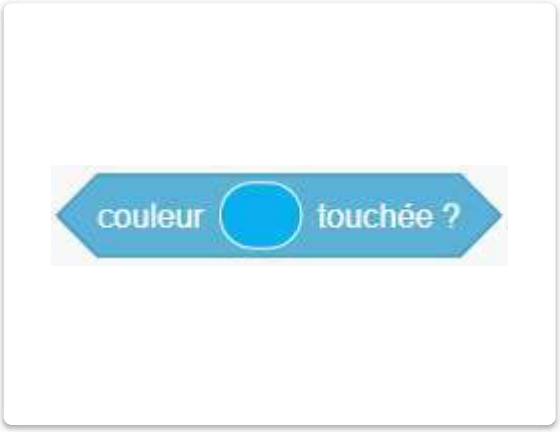
Une seule réponse possible.



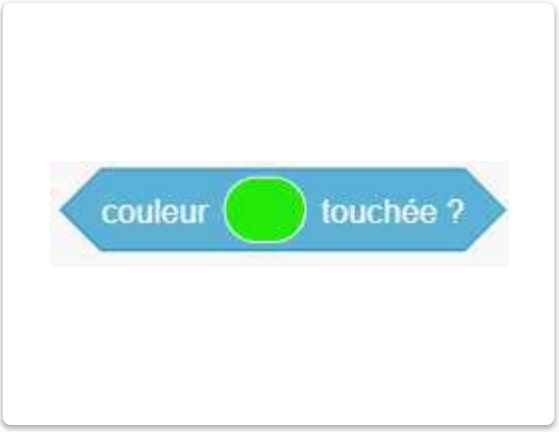
Couleur Jaune touchée



Couleur Rouge touchée

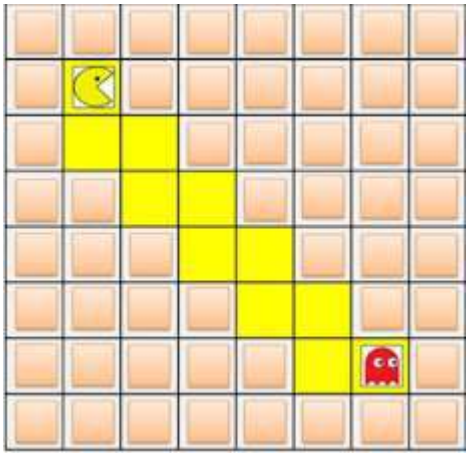


Couleur Bleue touchée



Couleur Verte touchée

13. Pac-man (en haut à gauche) doit atteindre le fantôme (en bas à droite). Parmi les 4 programmes proposés, lequel permet à Pac-man de rejoindre le fantôme ? Coche la bonne réponse. (1 pas = 1 case).



Une seule réponse possible.

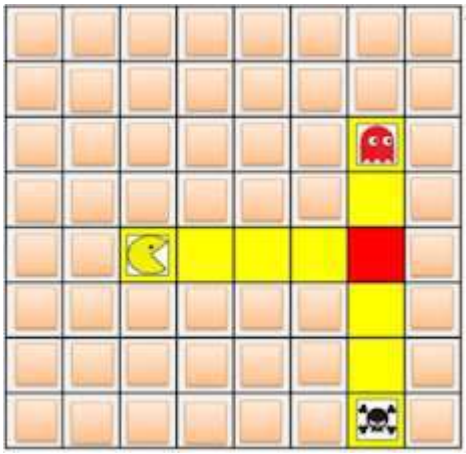
Programme 1

Programme 2

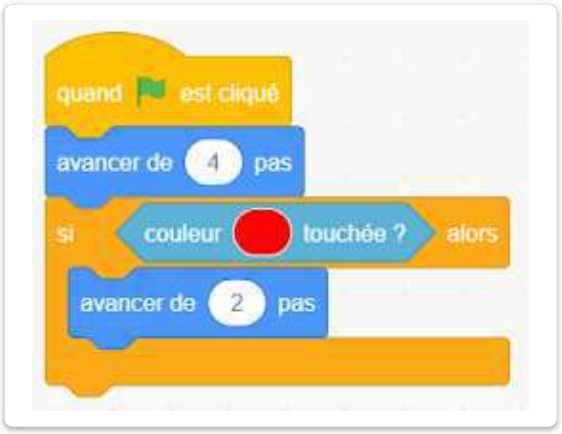
Programme 3

Programme 4

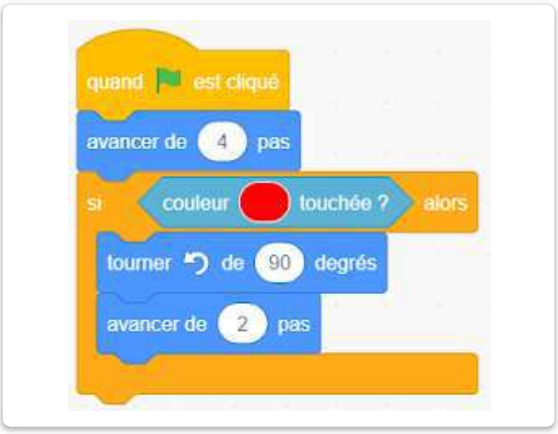
14. Pac-man (en jaune) doit atteindre le fantôme (en rouge). Parmi les 4 programmes proposés, lequel permet à Pac-man de rejoindre le fantôme ? Coche la bonne réponse. (1 pas = 1 case).



Une seule réponse possible.

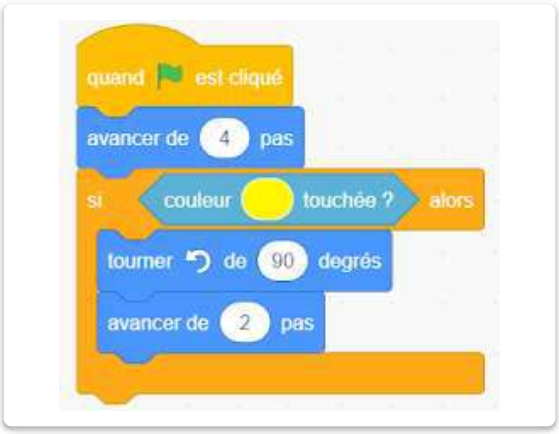


Programme 1



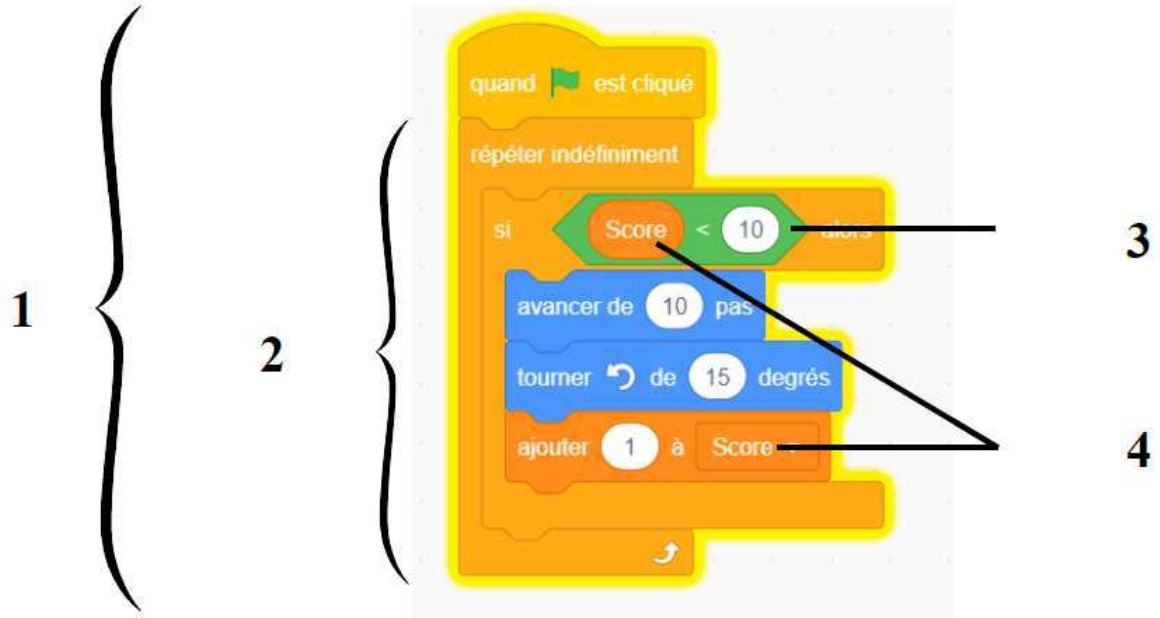
Programme 2

Programme 3



Programme 4

Dans le programme ci-dessous, associe chaque numéro avec un des mots suivants : Variable ; Boucle ; Condition ; Algorithme.



20. Numéro 1 :

_____

21. Numéro 2 :

_____

22. Numéro 3 :

_____

23. Numéro 4 :

_____

Dans le texte ci-dessous, les mots manquants sont remplacés par des numéros. Associe chaque numéro avec un des mots suivants : Algorithme ; Instruction ; Boucle ; Condition ; Variable. Attention, certains mots peuvent être utilisés plusieurs fois.

Un **1** est toujours composé d'une ou plusieurs **2** .
Une **3** permet de stocker et modifier une valeur.
Une **4** permet de répéter plusieurs fois certaines **5** .
On peut exécuter une partie d'un programme seulement lorsqu'une **6** est remplie.

24. Numéro 1 :

_____

25. Numéro 2 :

_____

26. Numéro 3 :

_____

27. Numéro 4 :

_____

28. Numéro 5 :

_____

29. Numéro 6 :

_____

15. Une instruction peut être composée d'un ou plusieurs algorithmes.

Une seule réponse possible.

- ◯ Vrai
- ◯ Faux

16. Une boucle est un type particulier d'instruction.

Une seule réponse possible.

- ◯ Vrai
- ◯ Faux

17. Il est possible d'exécuter une instruction uniquement si une certaine condition est respectée.

Une seule réponse possible.

- ◯ Vrai
- ◯ Faux

18. Une variable est un type particulier d'instruction.

Une seule réponse possible.

- ◯ Vrai
- ◯ Faux

19. Un algorithme peut contenir une boucle, une condition et une variable.

Une seule réponse possible.

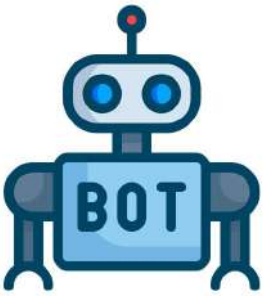
- ◯ Vrai
- ◯ Faux

Annexe E

Prénom : ……. **Nom de famille : …….**

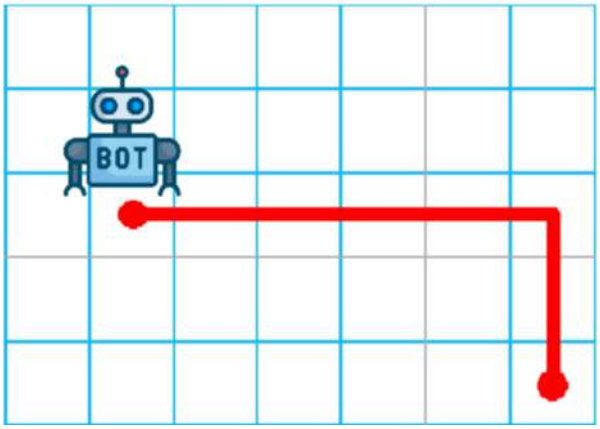
Date : …….

Classe : ……. **École : …….**



BOT est un robot qui comprend le langage humain. Il a besoin d'instructions claires et précises pour réaliser certaines tâches.

Par exemple, **BOT** a besoin d'être programmé pour se déplacer en suivant le parcours affiché ci-dessous (en partant du point situé en dessous de **BOT** sur l'image).



La 1ère ligne du programme peut s'écrire « Avancer de 5 carreaux », la 2ème ligne peut s'écrire « Tourner à droite à angle droit » etc. Peu importe les mots que tu utilises, il comprend le français ! Par contre, il ne comprend que les algorithmes avec des instructions très précises.

Pour déplacer **BOT** sur le parcours affiché en rouge, tu peux donc utiliser les algorithmes suivants :

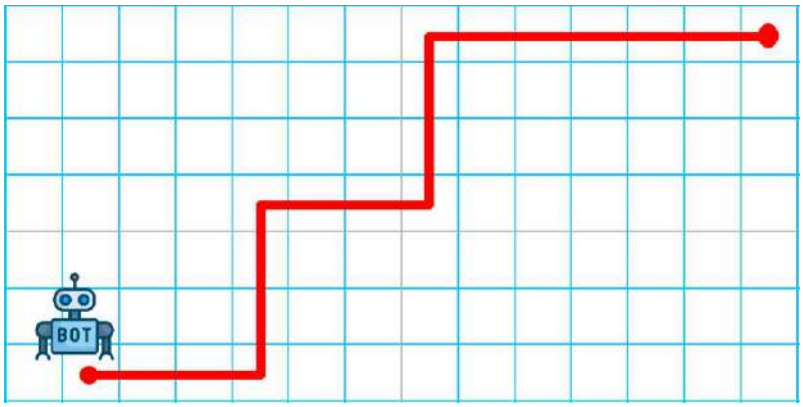
Avancer de 5 carreaux	OU	Répéter 5 fois
Tourner à droite à angle droit		Avancer de 1 case
Avancer de 2 carreaux		Tourner à droite
		Répéter 2 fois
		Avancer de 1 case

Les deux algorithmes produisent le même résultat. Fais attention lorsque tu utilises des boucles à bien séparer les instructions pour que **BOT** sache lesquelles il doit répéter. Ici, on a choisi de décaler ces instructions sur la droite pour que ce soit bien clair !

Pour chaque exercice, **commence à rédiger tes réponses sur une feuille de brouillon** puis quand tu as fini l'exercice, écris ton algorithme au propre.

Exercice 1 : Réaliser un parcours

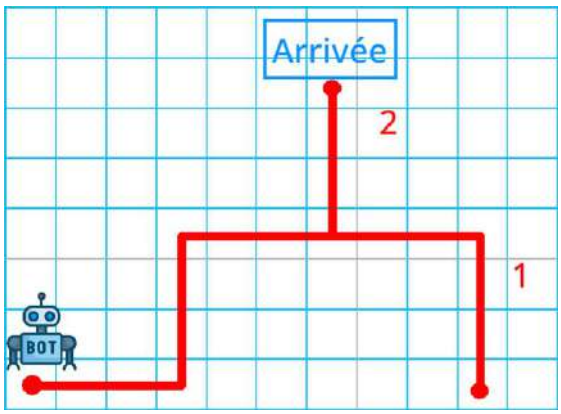
1) Réalise un algorithme pour que **BOT** se déplace en suivant le parcours affiché ci-dessous.



2) Si tu remarques que certaines instructions se répètent, utilise des boucles pour remplacer ces instructions.
3) Crée une variable que tu appelles « Score » et qui augmente de 1 à chaque fois que **BOT** atteint l'arrivée.

Ecris ton algorithme ici !

Exercice 2 : Mauvais chemin !



Sur le parcours ci-dessus, **BOT** est programmé pour suivre le parcours N°1 avec cet algorithme :

Avancer de 3 cases
Gauche
Avancer de 3 cases
Droite
Avancer de 6 cases
Droite
Avancer de 3 cases

1) Modifie cet algorithme pour qu'il suive plutôt le parcours N°2 et qu'il atteigne le panneau « Arrivée ».
2) Complète cet algorithme en utilisant une instruction pour que **BOT** dise : « J'ai réussi ! » s'il touche le panneau arrivée. Tu dois utiliser une condition ici.

Ecris ton algorithme ici !

Exercice 3 : Compter à voix haute

1) Réalise un algorithme pour que **BOT** compte de 1 à 5 à voix haute. Tu peux utiliser l'instruction « Dire » par exemple.
2) Rajoute une instruction pour que **BOT** recommence à compter à partir de 1 quand il arrive à 5.
3) Crée une variable que tu appelles « Nombre ». Mets cette variable à 1 au début de ton programme.
4) Dans l'instruction « Dire 1 », remplace le 1 par la variable « Nombre ».
5) Trouve un moyen pour que **BOT** compte jusqu'à l'infini sans s'arrêter et supprime les instructions qui sont maintenant inutiles.

Ecris ton algorithme ici !

Exercice 4 : Compter le temps qui passe

1) Crée une variable qui s'appelle « Seconde » et qui va permettre à **BOT** de compter le temps qui passe. Utilise une boucle pour que la variable augmente indéfiniment de 1 chaque seconde.
   Tu peux utiliser l'instruction « Attendre 1 seconde » par exemple.
2) Intègre dans ton programme une condition. La variable « Seconde » doit augmenter de 1 jusqu'à 60. Si elle atteint 60, elle doit recommencer à 0 !
3) Crée ensuite une autre variable que tu appelles « Minute » et qui augmente de 1 à chaque fois que 60 secondes se sont écoulées.

Ecris ton algorithme ici !

Annexe F

Test d'algorithmique

1. Est-ce que tu es...
   - Une fille ?
   - Un garçon ?

2. Quel âge as-tu ?

   _____

3. Tu es en quelle classe ?

   _____

4. Est-ce que tu as déjà fait de la programmation informatique (ou du codage) ?
   - Oui
   - Non

Niveau de pratique de la programmation

Si tu as répondu oui à la question précédente, alors continue de répondre aux questions suivantes. Si tu as répondu non, tu peux passer directement à la page 3.

5. Quand est-ce que tu as commencé à faire de la programmation pour la 1ère fois ?
   - Ce mois-ci
   - Cette année scolaire
   - Il y a plus d'un an

6. Quand est-ce que tu as fait de la programmation pour la dernière fois ?
   - Cette semaine
   - Ce mois-ci
   - Cette année scolaire
   - Il y a plus d'un an

7. Depuis la rentrée scolaire de septembre, tu as fait de la programmation...
   - o Plusieurs fois par semaine
   - o Au moins une fois par semaine
   - o Au moins une fois par mois
   - o Au moins une fois
   - o Pas une seule fois

8. A ton avis, dans ta vie, tu as passé combien d'heures à faire de la programmation ?
   - o Moins de 3 heures
   - o Entre 3 et 10 heures
   - o Entre 10 et 50 heures
   - o Plus de 50 heures

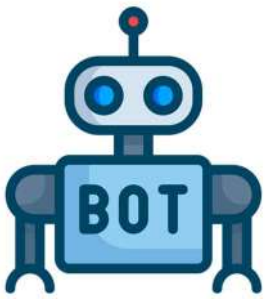
9. Est-ce que tu as déjà fait de la programmation en dehors de l'école ?
   - o Oui
   - o Non

10. Est-ce que tu as déjà fait de la programmation débranchée (sans ordinateur ni aucun outil technologique) ?
    - o Oui
    - o Non

11. Est-ce que tu as déjà fait de la programmation sur un logiciel comme Scratch, Blockly etc. ?
    - o Oui
    - o Non

12. Est-ce que tu as déjà programmé un robot ou un objet connecté ?
    - o Oui
    - o Non

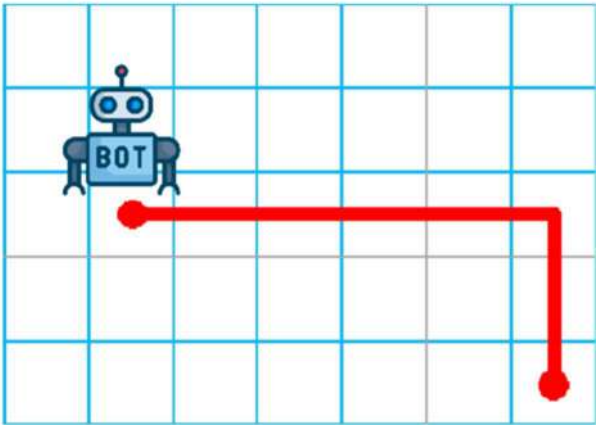
BOT est un robot qui comprend le langage humain. Il a besoin d'instructions claires et précises pour réaliser certaines tâches.



Voici les 10 instructions que **BOT** comprend :

1. Avancer vers la droite de … case(s)
2. Avancer vers la gauche de … case(s)
3. Monter de … case(s)
4. Descendre de … case(s)
5. Répéter … fois
6. Répéter indéfiniment
7. Si … Alors …
8. Attendre … seconde(s)
9. Dire …
10. Ajouter … à …

Par exemple, **BOT** a besoin d'être programmé pour se déplacer en suivant le parcours affiché ci-dessous (en partant du point situé en dessous de **BOT** sur l'image).



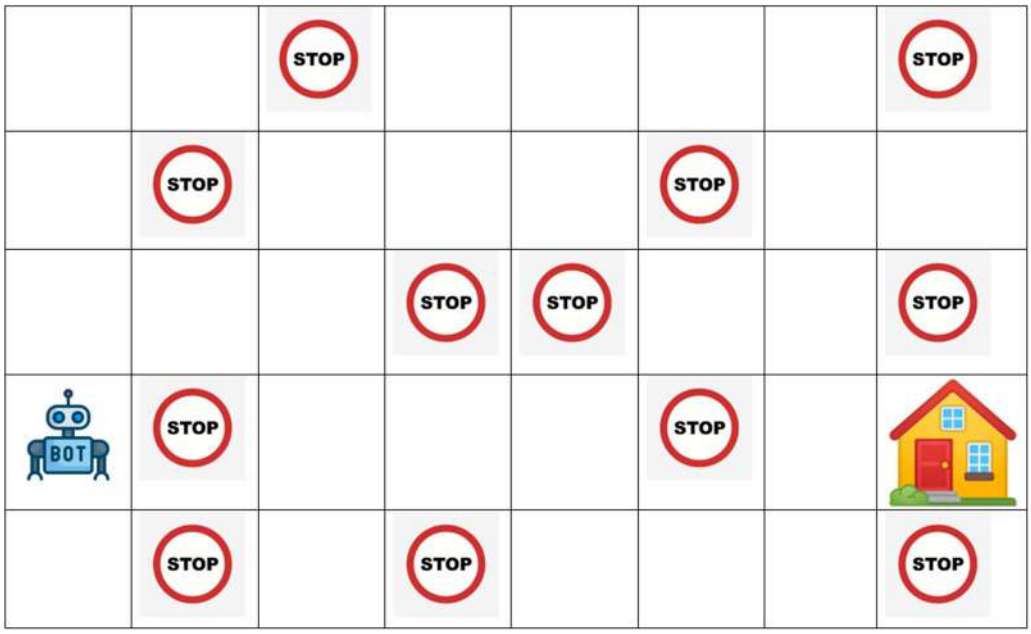
Pour déplacer **BOT** sur le parcours affiché en rouge, tu peux donc utiliser les algorithmes suivants :

Avancer vers la droite de **5** cases	OU	Répéter **5** fois
Descendre de **2** cases		Avancer vers la droite de **1** case
		Répéter **2** fois
		Descendre de **1** case

Les deux algorithmes produisent le même résultat. Fais attention lorsque tu utilises des boucles (« Répéter ») à bien séparer les instructions pour que **BOT** sache lesquelles il doit répéter. Ici, on a choisi de décaler ces instructions sur la droite pour que ce soit bien clair !

Exercice 1 : Plusieurs chemins

BOT doit rejoindre sa maison en évitant les obstacles (**STOP**) qui l'empêchent de passer.
Il y a plusieurs chemins possibles.



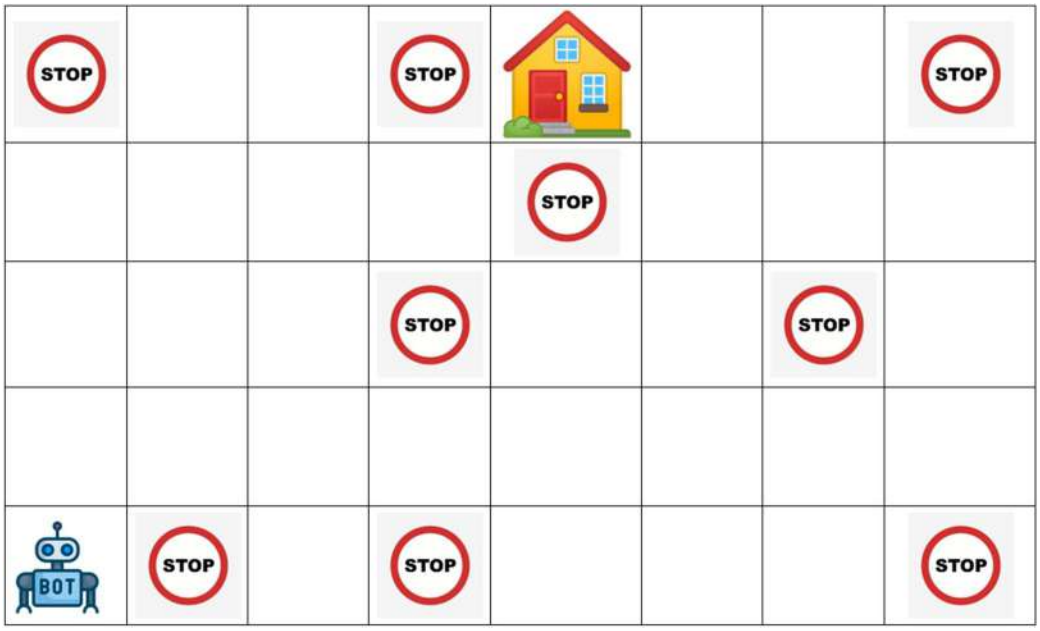
Ecris l'algorithme qui permet à **BOT** d'atteindre l'arrivée par le chemin le plus court, c'est-à-dire en passant par le moins de cases possible (**Algorithme 1**).
Puis, écris l'algorithme qui permet à **BOT** d'atteindre l'arrivée avec le moins d'instructions possibles (le moins de lignes dans l'algorithme). Tu dois utiliser au moins une boucle (« Répéter ») pour répéter certaines instructions (**Algorithme 2**).

Ecris ton Algorithme 1 ici !

Ecris ton Algorithme 2 ici !

Exercice 2 :

BOT doit rejoindre sa maison en évitant les obstacles (**STOP**) qui l'empêchent de passer.



Marie et ***Tom*** ont proposé chacun un algorithme pour réaliser ce parcours mais aucun ne permet au robot d'atteindre l'arrivée car ils ne prennent pas en compte les obstacles sur le chemin (voir plus bas). A partir des algorithmes proposés par ***Marie*** et ***Tom***, écris un **Algorithme 3** pour que **BOT** réalise ce parcours. Attention, tu ne dois utiliser que les instructions déjà utilisées par ***Marie*** et ***Tom*** !

Rajoute une condition (« Si … Alors … ») quelque part dans ton algorithme pour que **BOT** dise « J'ai réussi ! » si la solution est la bonne.

Algorithme de *Marie*	Algorithme de *Tom*
Avancer de 5 cases vers la droite	Monter de 3 cases
Monter de 4 cases	Avancer de 3 cases vers la droite
Avancer d'une case vers la gauche	Monter de 1 case
	Avancer de 1 case vers la droite

Ecris ton Algorithme 3 ici !

Exercice 3 : Compter le temps qui passe

Pauline a réalisé un algorithme pour que **BOT** compte 5 secondes (voir plus bas).
Elle aimerait que **BOT** puisse compter le nombre de secondes qui passent sans s'arrêter. Ecris un **algorithme 4** qui permette à **BOT** de compter les secondes. Dans l'instruction « Dire … », tu peux remplacer le chiffre par la variable « Seconde » puis augmenter cette variable après chaque seconde écoulée et répéter ce processus à l'infini !
Quand tu as terminé, améliore ton **algorithme 4** pour que **BOT** compte (et dise) également les minutes qui se sont écoulées (1 minute = 60 secondes).

Algorithme de *Pauline*
Dire « 1 »
Attendre 1 seconde
Dire « 2 »
Attendre 1 seconde
Dire « 3 »
Attendre 1 seconde
Dire « 4 »
Attendre 1 seconde
Dire « 5 »

Ecris ton Algorithme 4 ici !

Annexe G

Séance 1	Introduction à la programmation	Durée : 45 min	Condition débranchée Condition Scratch Condition Scratch + Robots

Résumé :

- *Deux pré-tests pour estimer le niveau de connaissance préalable de la programmation ainsi que le niveau de motivation, le sentiment d'auto-efficacité et l'intérêt pour les sciences des élèves.*
- *Entre les deux pré-tests, l'enseignant présente la séquence à venir et aborde la notion d'algorithme.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
15 min	**Test de connaissances préalables en programmation**	Répondre à 4 questions très générales portant sur la programmation et ce qui y est associé : - Qu'est-ce que le mot « informatique » évoque pour toi ? - Qu'est-ce qu'un programme informatique ? - A quoi le mot « programmation » te fait penser ? - Qu'est ce qu'un algorithme ?	Veiller à ce que le test soit effectué **individuellement** par la totalité des élèves.	*Le test sera fourni aux enseignants au format papier.*

15 min	**Présentation de la séquence à venir**	Les élèves **expriment leur opinion** autour de la notion d'algorithme et peuvent en débattre entre eux. Ils écoutent ensuite les explications de l'enseignant qui leur donne une **définition exacte** de cette notion et leur présente **différentes formes d'algorithmes** et l'**intérêt de leur utilisation**.	L'enseignant aborde la notion d'**algorithme**. Il demande aux élèves d'essayer de définir ce qu'est un algorithme pour eux (normalement les élèves viennent de réfléchir à cette question lors de la tâche précédente). Il recueille les réponses données par les élèves, les confronte et **anime le « débat »** autour de cette notion. Il finit par en donner une définition lui-même. Un algorithme, c'est une **suite d'instructions** (c'est-à-dire une suite d'actions à exécuter) présentées dans un certain ordre. À quoi cela sert ? À **résoudre des problèmes** ! L'enseignant explique que les élèves vont participer à une séquence dans laquelle ils vont apprendre à manipuler des algorithmes. Il leur présente l'intérêt de cette pratique : **toutes les machines qui nous entourent obéissent à des programmes**, c'est-à-dire à des algorithmes. Il donne des exemples de choses qui ne pourraient pas exister sans les algorithmes : les robots, les jeux vidéo, les ordinateurs...	*L'objectif de cette activité est de **susciter l'intérêt** des élèves en introduisant la séquence à venir et en essayant de mettre en exergue l'intérêt de la pratique de l'algorithmique.* *Le deuxième but visé par cette tâche est de **présenter une grande variété d'algorithmes**, en particulier des algorithmes que les élèves utilisent sans savoir qu'ils en sont ! Dans la mesure du possible, il faut éviter que les élèves qui sont dans la condition "Scratch" par exemple considèrent qu'un algorithme c'est simplement un programme sur Scratch. On doit encourager ici une **vision globale de l'algorithme** qui ne soit pas restreinte à un logiciel particulier ou même à la programmation informatique en général.*

			Il leur explique aussi qu'un algorithme n'est pas simplement utile en informatique mais aussi dans la vie de tous les jours. **Il donne des exemples d'algorithmes du quotidien** et explique pourquoi ce sont des algorithmes : – une recette de cuisine est un algorithme car chaque action à réaliser est une instruction et ces instructions doivent être exécutées dans le bon ordre ! – la méthode que l'on utilise pour chercher un mot dans le dictionnaire est aussi un algorithme qui contient des instructions et même des conditions (« Ouvrir le dictionnaire – Si la première lettre du premier mot de la page se situe dans l'alphabet après la première lettre du mot que je cherche, Alors ouvrir le dictionnaire à une page précédente »). – certaines règles de grammaire sont aussi des algorithmes...	

15 min	**Évaluation de la motivation, du sentiment d'auto-efficacité et de l'intérêt pour les sciences des élèves**	Répondre au pré-test au format papier pour estimer : - le niveau de motivation des élèves : "Est-ce que j'ai envie d'apprendre la programmation ?" - le sentiment d'auto-efficacité : "Est-ce que je me sens capable d'apprendre la programmation ?" - le niveau d'intérêt pour les disciplines scientifiques : "Est-ce que j'aime les sciences ? »	Veiller à ce que le test soit effectué **individuellement** par la totalité des élèves.	*Plusieurs affirmations seront présentées aux élèves qui devront notamment exprimer leur degré d'accord ou de désaccord avec ces affirmations sur une échelle de réponses prédéfinies.*

Séance 2	Découverte de la programmation débranchée	Durée : 45 min (15 + 30min)	Condition débranchée

Résumé :

- *Les élèves découvrent la programmation par des activités débranchées.*
- *Ils apprennent à utiliser un langage algorithmique pour contrôler les déplacements d'un véhicule à distance.*

Matériel :

- *Pour la classe : un vidéo projecteur.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
15 min	**Trouver un langage**	Les élèves doivent **définir quels types d'instructions donner au véhicule pour lui faire suivre le parcours imposé** afin de rejoindre la base. Les déplacements se font carreau par carreau, et pas en diagonale.	L'enseignant présente le contexte de l'activité. Une fusée a été envoyée sur une autre planète. La mission est habitée et, sur la planète, l'équipage dispose déjà d'une base et d'un véhicule terrestre. L'environnement est hostile, donc lors des sorties d'exploration, les astronautes doivent pouvoir diriger le véhicule à distance. Les ordres de déplacement sont donnés au véhicule sous forme d'ondes, mais **il faut inventer un langage pour donner ces ordres.** La question est donc : Quel langage utiliser pour piloter un véhicule à distance ?	*La fiche documentaire est distribuée aux élèves, répartis par groupes.*

			L'enseignant **affiche ou projette une carte de la zone d'exploration** (voir fiche documentaire). Cette zone est quadrillée et un parcours est dessiné de façon à pouvoir rentrer à la base en évitant les zones dangereuses. On ne peut pas faire de raccourci : il faut absolument suivre tout le parcours dessiné, dans le sens de la flèche.	
15 min	**Mise en commun**	La classe **met en commun les travaux des différents groupes.** On peut, par exemple, dessiner ou projeter le parcours au tableau et, lorsqu'un groupe présente sa solution, **vérifier collectivement qu'elle est correcte en l'exécutant au tableau** (on prend un objet quelconque pour représenter le véhicule, cet objet doit suivre scrupuleusement les instructions données).	L'enseignant fait remarquer que **les instructions doivent être exprimées dans un langage particulier,** avec un vocabulaire très restreint, et non ambigu : chaque instruction doit être parfaitement explicite et ne peut pas donner lieu à plusieurs interprétations. Il s'agit d'un langage de « programmation ». Ce langage peut encore être simplifié. Par exemple, il est inutile de dire « Va vers l'Est » quand on peut simplement dire « Est » ou « Va à droite » quand on peut simplement dire « Droite » (si on a bien défini au préalable ce que l'on entend par « Droite », par exemple, « va d'une case vers la droite » et non pas « pivote sur toi-même d'un quart de tour vers la droite »). L'enseignant fait remarquer que l**a grammaire est également très simple**.	*On remarque qu'il existe (au moins) deux logiques pour diriger le véhicule. On peut lui donner des directions « absolues » (va au Nord, va à l'Ouest…) ou, au contraire, des directions relatives, c'est-à-dire qui dépendent de l'orientation du véhicule (tourne vers la droite, avance, tourne vers la gauche, recule…). Il est préférable de découper l'instruction « avance d'une case vers la droite » en 2 instructions bien distinctes : pivoter vers la droite (sous-entendu : en restant sur place), puis avance d'une case.*

			Il n'y a pas de genre, de nombre, de mode, de temps… La seule règle présente ici est celle de la séquence : quand 2 instructions se suivent, par exemple « droite avance », cela signifie qu'elles doivent être exécutées l'une après l'autre. Pour plus de clarté lors de l'écriture et la lecture, **on décide de conserver trois mots pour coder les déplacements : Avance ; Droite** (quart de tour vers la droite sans avancer) ; **Gauche** (quart de tour vers la gauche sans avancer). **On décide aussi d'aller à la ligne après chaque instruction.** **Bien expliquer qu'une instruction, c'est une action à exécuter.**	*La première logique (Nord, Ouest…) est dite **« allocentrée »** tandis que la seconde logique (droite, gauche…) est dite « **autocentrée ».** Les élèves n'ont pas besoin de connaître ces termes, qui ne seront plus utilisés par la suite.*
10 min	**Mesurer l'impact d'une erreur**	La classe **discute des différentes origines possibles d'un bug**. Il peut s'agir d'une erreur dans l'algorithme (la méthode), ou d'une erreur dans le programme (l'expression de l'algorithme dans le langage choisi : une erreur de syntaxe, par exemple), ou encore d'une erreur matérielle (liée, par exemple, à une panne d'un élément de la machine, ou à une erreur dans la transmission des instructions, comme ici).	L'enseignant demande aux élèves ce qui se passe si on introduit une erreur dans un programme (par exemple, si on oublie une instruction). On peut prendre un exemple concret, en se basant sur le parcours initial du véhicule (fiche documentaire). **Que se passe-t-il si l'on saute une instruction ?** On s'aperçoit que quel que soit le langage utilisé, on rate l'objectif.	*On remarquera qu'une erreur dans un langage autocentré peut conduire plus loin de l'objectif qu'une erreur dans un langage allocentré. Cependant, dans les deux cas, il s'agit d'un bug et on notera que l'objectif n'est pas atteint. Donc **c'est un échec aussi important dans un cas que dans l'autre.***

5 min	**Trace écrite**	***"Les machines qui nous entourent ne font qu'exécuter des ordres : les instructions."*** ***"Une instruction est une action qui doit être exécutée avant de passer à l'instruction suivante."*** ***"Un algorithme est un suite d'instructions executées dans un certain ordre."*** ***"Un algorithme est une méthode permettant de résoudre un problème."***

Séance 2	Découverte de Scratch	Durée : 45 min	Condition Scratch Condition Scratch + Robot

Résumé :

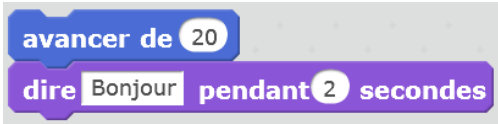
- *Les élèves découvrent Scratch, un environnement de programmation adapté à l'école primaire.*
- *Ils apprennent à lancer le programme et à enchaîner quelques instructions simples.*

Matériel :

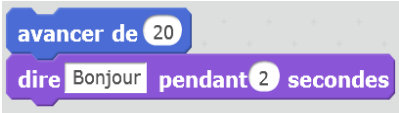
- Pour la classe : un vidéo projecteur.
- *Pour chaque groupe d'élèves : un ordinateur connecté à Internet ou, en l'absence de connexion Internet de bonne qualité, un ordinateur sur lequel le logiciel Scratch a été préalablement installé.*

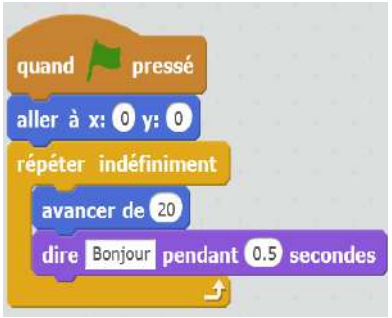
Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
10 min	**Lancer scratch et découvrir son interface**	Chaque groupe **lance Scratch** en double-cliquant sur son icône et **suit les instructions de l'enseignant pour pouvoir reproduire ce qu'il fait** en fin de séance.	L'enseignant explique aux élèves que **Scratch est un langage de programmation conçu spécifiquement pour apprendre à programmer.** Lorsque l'on ouvre le logiciel, il y a un lutin à l'écran (un chat). On peut lui donner des instructions simples. Bien expliquer qu'**une instruction, c'est une action à exécuter.** L'enseignant réalise une petite démonstration. Par exemple, pour demander au chat de se déplacer de 10 pas, il suffit de faire glisser l'instruction « avancer de 10 » depuis la palette d'instructions vers la zone du programme.	*Interface de Scratch :* *Une **« scène »** : c'est là que se déroule le programme !* *Une **zone « lutins »**: les lutins sont les personnages ou les objets qui seront manipulés dans le programme (ils peuvent se déplacer, changer de forme, parler, interagir avec les autres lutins…). Lorsqu'on lance Scratch, il n'y a qu'un seul lutin affiché à l'écran : un chat.* *Une **zone « arrière-plan »**, juste à côté des lutins. L'arrière-plan est fixe, contrairement aux lutins qui peuvent bouger.*

			Si l'on clique ensuite sur cette inscription, on remarque que le chat avance bien de 10 pas (1 pas = 1 pixel de l'écran).  Si l'on souhaite avancer de 20 pas, il suffit de changer « 10 » en « 20 » en cliquant dans la zone dédiée. Si maintenant on souhaite que le chat avance de 20 pas, puis dise « Bonjour », il suffit de coller la nouvelle instruction à la fin du programme. **Écrire un programme se fait simplement en emboitant des instructions entre elles.**  Si maintenant on veut que le chat fasse cela à chaque fois que l'on clique sur le drapeau vert (en haut à droite de la scène, le drapeau vert permet de lancer le programme), alors il faut rajouter l'instruction « Quand drapeau vert pressé » à chercher dans la catégorie « événements » des instructions. Cela donne :	*Par défaut, dans Scratch, l'arrière-plan est un fond blanc uni (plus tard, on pourra le modifier).* *Un onglet **« script »** qui permet d'accéder à :* *- **Une palette d'instructions.** C'est ici que l'on va trouver les instructions (ou « blocs ») que l'on va pouvoir utiliser dans notre programme. Il y a de nombreuses instructions, qui sont regroupées par couleur (exemple : tout ce qui concerne le mouvement du lutin est dans un onglet bleu foncé, tout ce qui concerne son apparence est dans un onglet violet, etc.).* *- Une zone **« programme »**, à droite de la palette d'instructions. C'est ici que l'on va écrire le programme, tout simplement en prenant des instructions depuis la palette et en les faisant glisser dans cette zone. Les autres onglets (costumes, sons) sont inutiles pour le moment.*

			 Finalement, l'enseignant montre **comment supprimer une instruction** (ou tout un bloc d'instructions) : il suffit de faire glisser cette instruction (ou ce bloc) depuis la zone du programme vers la palette des instructions. L'enseignant présente très rapidement l'interface de Scratch (voir remarques).	
10 min	**Explorer librement Scratch**	Les élèves **explorent librement Scratch**. Pour le moment, ils ne doivent pas chercher à modifier la scène ou le lutin mais simplement **manipuler des instructions simples et les agencer** pour observer ce qui se passe.	L'enseignant les encourage à explorer les différentes catégories d'instructions, en particulier : - Catégorie **« mouvement »** (bleu foncé) - Catégorie **« apparence »** (violet) - Catégorie **« événement »** (jaune) - Catégorie **« contrôle »** (orange).	
20 min	**Structuration**	**Exercice 1 :** faire avancer le chat de 10 pas.  **Exercice 2 :** faire avancer le chat de 20 pas.	L'enseignant donne une série de petits exercices (qui reprennent, pour la plupart, ce qui a été fait avant sous forme de démonstration), que les élèves exécutent. Après chaque exercice, une **rapide mise en commun** permet de s'assurer que chacun sait faire l'exercice.	***Exercice 3 :*** ***« dire »*** *bonjour signifie pour nous* ***« écrire »*** *bonjour : on veut faire afficher une bulle à l'écran avec le texte « bonjour » dedans, on ne veut pas faire parler le chat !*

		Plusieurs solutions possibles (on préférera la seconde, plus élégante et facile à lire) :   **Exercice 3 :** faire avancer le chat de 20 pas et lui faire dire « Bonjour ».  **Exercice 4 :** remettre le chat au centre de la scène.  **Exercice 5 :** répéter indéfiniment : faire avancer le chat de 20 et lui faire dire « bonjour ».  	**Exercice 5 :** Inciter les élèves qui ne trouvent pas à chercher dans la catégorie **« contrôle »** (orange). **Cette boucle enserre les autres instructions.** On voit le chat qui s'arrête 2 secondes entre chaque mouvement… Pour raccourcir cette pause, il suffit de raccourcir la durée pendant laquelle il dit « bonjour » (si on écrit « 0.5 » au lieu de 2, il ne s'arrêtera qu'1/2 seconde à chaque fois).	***Exercice 4 :*** *Certains élèves vont sans doute trouver l'astuce… mais pour la plupart, il faudra la leur montrer. Malgré tout, il est indispensable pour eux de voir cette instruction dès maintenant, car, à force de déplacer le chat, ils vont le faire sortir de l'écran et ne sauront pas comment le récupérer.* ***Exercice 6 :*** *Cela se fait très simplement, sur le même modèle que l'exercice précédent, mais avec un autre type de boucle. Ils y trouveront une instruction qui ressemble (« répéter 10 fois ») et qu'il est facile de modifier en remplaçant 10 par 3. Tout ce qui se trouve à l'intérieur de la boucle est exécuté 3 fois.* ***Exercice 7 :*** *Il suffit a priori de rajouter l'instruction « quand drapeau vert pressé » (issue de la catégorie événement), mais cela est encore mieux si on demande au chat de repartir du centre de la scène. Expliquer, à ce moment, le rôle du bouton rouge (situé à côté du drapeau vert). Un clic sur ce bouton rouge met fin à l'exécution du programme (qui, sinon, ne s'arrête jamais dans le cas présent).*

		Exercice 6 : répéter 3 fois : faire avancer le chat de 20 et lui faire dire « bonjour ».  **Exercice 7** : même chose quand on clique sur le drapeau vert  		
5 min	**Trace écrite**	***"Les machines qui nous entourent ne font qu'exécuter des ordres : les instructions."*** ***"Une instruction est une action qui doit être exécutée avant de passer à l'instruction suivante."*** ***"Un algorithme est un suite d'instructions executées dans un certain ordre."*** ***"Un algorithme est une méthode permettant de résoudre un problème."***		

Séance 3	Contrôler les déplacements d'un « robot idiot »	Durée : 45 min	Condition débranchée

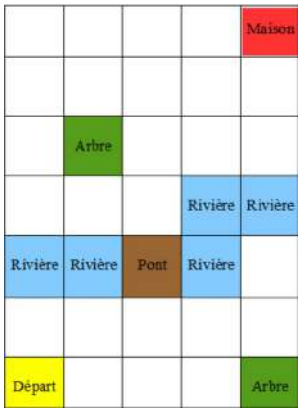
Résumé :

- *Les élèves découvrent le jeu du « robot idiot ».*
  Ils apprennent à créer un algorithme et à enchaîner quelques instructions simples.

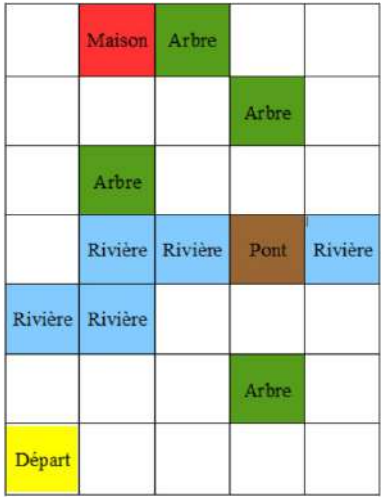
Matériel :

- *<u>Pour chaque groupe d'élèves :</u> un « tapis de jeu », qui peut être constitué d'un drap quadrillé par exemple. La taille minimale pour chaque carreau doit être de 30 X 30 cm.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
5 min	**Consignes pour le jeu du « robot idiot »**	Les élèves **découvrent le "tapis de jeu" et écoutent les consignes** données par l'enseignant.	L'enseignant explique aux élèves qu'ils vont devoir, tour à tour, jouer le rôle d'un « robot idiot » qui ne peut rien faire d'autre qu'écouter et obéir aux instructions qui lui sont données. Dans chaque groupe, **un élève jouera le rôle du robot et un autre celui du programmeur** qui sera chargé de lire un algorithme (une suite d'instructions) que le robot devra exécuter. Le robot doit veiller à ne faire que ce qu'on lui dicte et le programmeur à ne lire que les instructions présentes dans l'algorithme (ni plus, ni moins). S'il y a des élèves dans le groupe qui ne sont ni programmeur ni robot, alors ils jouent **le rôle d'observateurs.**	*Pour démarrer, les éléments sur le tapis sont disposés selon la **configuration 1** (voir fichier **« configurations du tapis de jeu »**) :* 

			Ils doivent vérifier la cohérence entre les déplacements effectués et les instructions qui ont été données.	*Les images représentant les éléments du jeu (pont, rivière, arbre...) sont fournies dans le dossier **« images »**. Elles peuvent être imprimées et plastifiées pour chaque groupe.*
15 min	**Tester les algorithmes**	Les élèves **testent un à un les algorithmes donnés par l'enseignant** en se répartissant les rôles définis précédemment et en changeant de rôle pour chaque algorithme. Ils notent sur leur fiche **quels algorithmes permettent au robot de rentrer chez lui.**	L'enseignant explique aux élèves que **le robot doit partir du premier carreau en bas à gauche du tapis et qu'il doit être guidé pour retourner à sa maison** (en haut à droite). Le robot ne peut traverser la rivière que sur le pont (il ne sait pas nager) et il est bloqué par les arbres. L'enseignant distribue la fiche **"Algorithmes à tester"** et demande aux élèves de tester chacun des algorithmes présentés sur la fiche pour déterminer lequel ou lesquels permet(tent) au robot de retrouver sa maison. Il passe dans les groupes pour vérifier que les consignes sont bien appliquées.	*Le vocabulaire utilisé est le même que celui qui a été choisi lors de la séance précédente. **Avancer** (en spécifiant le nombre de cases) ; **Droite** (pivoter à droite d'un quart de tour sans avancer) ; **Gauche** (pivoter à gauche d'un quart de tour sans avancer).* *Les bonnes réponses sont les algorithmes 1, 2, 4 et 5.* *Les algorithmes 3, 6 et 7 ne sont pas valides.*
10 min	**Créer son propre algorithme.**	Après avoir changé la configuration du tapis de jeu, les élèves réfléchissent ensemble pour **créer un algorithme eux-mêmes.** Ils utilisent les instructions qui leur sont distribuées pour construire leur algorithme en allant à la ligne après chaque instruction.		*En amont, les instructions du fichier **« instructions »** ont été découpées et plastifiées. Les élèves doivent écrire au crayon le nombre de cases pour l'instruction « Avancer de … case(s) ». Ils peuvent disposer leurs instructions **sur une feuille format A3** par exemple.*

		On peut envisager de faire travailler chaque élève du groupe en autonomie pour ensuite tester un à un les algorithmes proposés.	Désormais les éléments du jeu doivent être positionnés selon la **configuration 2** (voir fichier **« configurations du tapis de jeu »**) :  À partir de là, l'enseignant demande aux élèves de créer leur propre algorithme pour aider le robot à retrouver sa maison et de le tester. Les instructions disponibles dans le fichier **« Instructions »** sont distribuées aux élèves. À la fin de cette tâche, l'enseignant **prend en photo au moins un algorithme par groupe qui fonctionne.** Cela permet aux élèves de conserver une trace de ce qu'ils ont réalisé lors des séances précédentes.	*Cette feuille pourra être conservée tout au long des séances de programmation débranchée. Une feuille par groupe devrait suffire.* *Si les élèves finissent un peu vite cette tâche, on peut leur proposer d'essayer de trouver l'algorithme qui nécessite le moins d'instructions (le moins de lignes de code).*

10 min	**Comment faire quand on ne connaît pas le chemin ?**	Les élèves réfléchissent à la possibilité de **guider les déplacements du robot sans savoir exactement par où il doit passer** pour traverser la rivière. **Ils proposent des solutions et en débattent entre eux.** Ils devraient assez rapidement en arriver à la notion de condition : « S'il n'y a pas de pont alors ... ».	L'enseignant fait remarquer que cet exercice est **plutôt facile quand on sais par quel chemin le robot doit passer.** Mais imaginons par exemple que nous ne sachions pas où se trouve le pont pour traverser la rivière. Comment faire pour guider les déplacements du robot ? L'enseignant écoute les propositions des élèves et propose de les tester rapidement si besoin. Si la notion de condition « Si … Alors ... » n'émerge pas d'elle-même, il peut alors suggérer ce début de solution.	*Il ne s'agît pas ici de concevoir un algorithme avec une condition pour résoudre un problème particulier (ce moment arrivera un peu plus tard).* *Il s'agît plutôt de **réfléchir aux possibilités que l'on peut envisager** pour résoudre un problème plus complexe.*
5 min	**Trace écrite**	***"L'exécution d'un algorithme est reproductible (si les instructions ne changent pas, ni les données à manipuler, le programme donne toujours le même résultat)."*** ***"L'ordinateur ne fait qu'exécuter les instructions qu'on lui donne : ni plus, ni moins..."***		

Séance 3	Contrôler les déplacements d'un lutin	Durée : 45 min	Condition Scratch

Résumé :

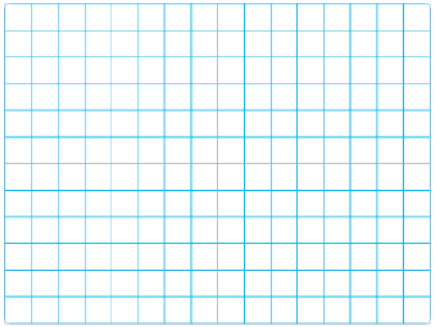
- *Les élèves apprennent à enregistrer leur travail pour le réutiliser plus tard.*
- *Ils réalisent ensuite leur premier programme, leur permettant de contrôler les déplacements d'un lutin.*

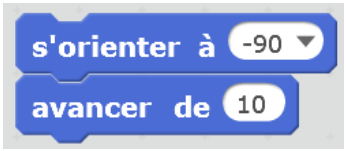
Matériel :

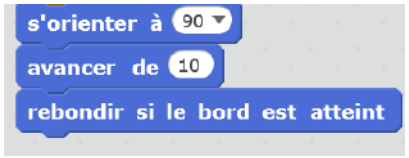
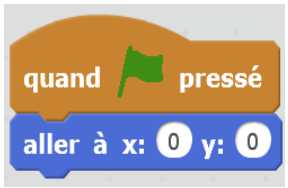
- Pour la classe : un vidéo projecteur.
- *Pour chaque groupe d'élèves : un ordinateur connecté à Internet ou, en l'absence de connexion Internet de bonne qualité, un ordinateur sur lequel le logiciel Scratch a été préalablement installé.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
5 min	**Changer de lutin**	Sélectionner **"Choisir un sprite"** en bas à droite de la scène (version Scratch Desktop) puis une nouvelle fois "Choisir un sprite". L'objectif est de **sélectionner un lutin qui soit "vu du dessus"** pour que ses déplacements paraissent plus naturels. Contrairement au "chat" qui est vu de profil et qui peut se retrouver "la tête en bas" lorsqu'il change de direction.	L'enseignant explique qu'il est possible de supprimer le lutin « chat » et d'en créer un autre à la place.	*Un **sprite** ou un **lutin** est un personnage (ou un objet) qui peut être programmé dans Scratch.* *Pour supprimer le chat, il faut cliquer (bouton droit) sur son icône, dans la zone des lutins, et choisir « supprimer ».*

5 min	**Changer l'arrière-plan**	Pour la suite des séances, il est important de **sélectionner l'arrière-plan "Xy-grid-30px"** qui permet d'afficher un quadrillage dont chaque carreau fait 30 pixels (ou 30 pas) de côté.	De la même façon que précédemment, l'enseignant explique qu'il est possible de changer l'arrière-plan de la scène à partir d'une image issue de la bibliothèque.	*L'arrière-plan sélectionné doit être celui-ci :* 
5 min	**Enregistrer son programme Scratch**	Sauvegarder le fichier	L'enseignant explique qu'il faut enregistrer le programme actuel (même s'il ne contient pas encore grand-chose), pour éviter d'avoir à tout refaire à la prochaine étape.	*Cas n°1 : Scratch installé en local* *L'enregistrement se fait en cliquant sur le menu « fichier » puis l'option « sauvegarder ».* *Cas n°2 : utilisation de Scratch en ligne* *L'enregistrement se fait en cliquant sur le menu « fichier » puis l'option « télécharger dans votre ordinateur ».*
10 min	**Faire avancer le lutin vers la gauche**	Les élèves savent déjà comment faire avancer le lutin vers la droite puisque, par défaut, il est déjà orienté vers la droite. Le faire avancer vers la gauche est une tâche un peu plus difficile, car les élèves doivent d'abord demander au lutin de s'orienter vers la gauche, avant d'avancer.	L'enseignant passant régulièrement dans les groupes pour s'assurer que personne n'est bloqué. Il peut les guider en les incitant à chercher une instruction « s'orienter ». Deux instructions de ce type sont disponibles : **« s 'orienter à… »** qui est celle qui nous intéresse.	*Lorsque l'on clique sur le nombre présent dans l'instruction **« s'orienter à ... »** (en général, ce nombre par défaut est « 90 »), une bulle d'aide apparaît. Suivant les versions de Scratch, elle peut prendre une forme différente pour nous expliquer que l'angle 0° désigne le haut de l'écran ;*

		Ils travaillent en autonomie et tâtonnent. Finalement, le programme qui permet au lutin de se déplacer vers la gauche est :  	« s'orienter vers… » qui ne nous intéresse pas car la seule option disponible, accessible en cliquant sur la petite flèche, est « pointeur de souris » (le lutin, dans ce cas, s'oriente vers la position du pointeur de la souris).	*90° désigne la droite, etc. Dans la version Scratch Desktop, il est possible de modifier l'orientation d'une flèche pour que la valeur de l'angle s'adapte automatiquement. Il faut donc choisir ici « s'orienter à -90 ».*
5 min	**Faire avancer le lutin dans toutes les directions**	Les élèves doivent maintenant être capables de **faire avancer le lutin dans n'importe quelle direction** (droite, gauche, haut et bas), en reprenant exactement la même méthode que précédemment.	L'enseignant demande aux élèves de changer l'orientation du lutin (vers le haut, le bas, revenir vers la droite...).	*Désormais, on a besoin de l'instruction **« s'orienter à 90 »** pour lui dire d'aller à droite… car le lutin n'est plus orienté, par défaut, dans cette direction.*
5 min	**Rebondir sur les bords**	Les élèves cherchent **comment faire en sorte que le lutin rebondisse sur les bords.** Par exemple, si on le dirige vers la droite et qu'il atteint l'extrémité droite de l'écran, le lutin doit rebondir afin de ne pas sortir de l'écran. Cela se fait très simplement en ajoutant l'instruction **« rebondir si le bord est atteint »** en bas de chacun des sous-programmes faits précédemment. Par exemple :	L'enseignant demande aux élèves de trouver un moyen de faire rebondir le lutin sur le bord, pour éviter qu'il ne sorte de l'écran.	

				
5 min	**Initialiser la position du lutin**	Les élèves **reprennent les instructions qu'ils avaient vues** lors de la première séance Scratch. Le programme peut maintenant être exécuté en cliquant sur le drapeau vert. 	L'enseignant rappelle que, **lorsqu'on lance le programme (drapeau vert), le lutin doit se situer au centre de l'écran.**	*Ne pas oublier, à chaque fois, de **sauvegarder** le travail réalisé !*
5 min	**Trace écrite**	***"L'exécution d'un algorithme est reproductible (si les instructions ne changent pas, ni les données à manipuler, le programme donne toujours le même résultat)."*** ***"L'ordinateur ne fait qu'exécuter les instructions qu'on lui donne : ni plus, ni moins..."***		

Séance 3	Contrôler les déplacements du robot	Durée : 45 min	Condition Scratch + Robot

Résumé :

- *Les élèves apprennent à enregistrer leur travail pour le réutiliser plus tard.*
- *Ils réalisent ensuite leur premier programme, leur permettant de contrôler les déplacements d'un robot.*

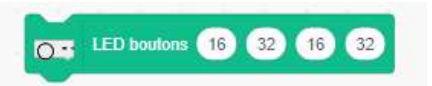
Matériel :

- Pour la classe : un vidéo projecteur.
- *Pour chaque groupe d'élèves : un ordinateur connecté à Internet ou, en l'absence de connexion Internet de bonne qualité, un ordinateur sur lequel le logiciel Scratch a été préalablement installé. Un robot Thymio Wireless.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
5 min	**Lancer le logiciel et connecter le robot**	Suivre les instructions données par l'enseignant pour **commencer à utiliser Scratch avec Thymio.**	L'enseignant donne aux élèves les instructions suivantes : Premièrement, **allumer le robot** en restant appuyé quelques secondes sur le bouton central entouré des quatre flèches directionnelles. Récupérer la clé USB fixée sur le dessus du robot et la **connecter au port USB** de l'ordinateur. **Lancer la suite Thymio** (de préférence via un raccourci sur le bureau) **Sélectionner Scratch** parmi les langages de programmation proposés.	

			Dans la partie droite de la nouvelle fenêtre qui s'ouvre, **sélectionner Thymio-Wireless** en cliquant dessus puis cliquer sur « Programmer avec Scratch » en bas à droite de l'écran.	
5 min	**Enregistrer son programme Scratch**	Sauvegarder le fichier	L'enseignant explique qu'il faut enregistrer le programme actuel (même s'il ne contient pas encore grand-chose), pour éviter d'avoir à tout refaire à la prochaine étape.	*Cas n°1 : Scratch installé en local* *L'enregistrement se fait en cliquant sur le menu « fichier » puis l'option « sauvegarder ».* *Cas n°2 : utilisation de Scratch en ligne* *L'enregistrement se fait en cliquant sur le menu « fichier » puis l'option « télécharger dans votre ordinateur ».*
5 min	**Faire avancer le robot**	**Les élèves travaillent en autonomie et tâtonnent.** Pour que le robot avance, il suffit d'utiliser le bloc "Avancer de 50". Il est souhaitable d'utiliser un événement pour **déclencher le début du programme**. Ici, l'événement choisi est "Quand drapeau vert est cliqué". On obtient le programme suivant : 	L'enseignant fait remarquer tout d'abord que la connexion du robot Thymio à l'ordinateur a permis de **rajouter une nouvelle catégorie de blocs, la catégorie "Thymio"** tout en bas, en dessous de la catégorie "Mes blocs". L'enseignant attire aussi l'attention sur la possibilité d'**avancer tout en contrôlant la vitesse du robot** avec le l'instruction "Avancer de 50 à vitesse 50". Il est possible pour les élèves de tester différentes vitesses. 	*Si le robot s'apprête à sortir du périmètre défini pour chaque groupe, les élèves sont autorisés à soulever le robot pour le remettre à sa position initiale. Si la distance renseignée est négative le robot se dirige vers l'arrière.*

		On constate que la valeur "50" peut être modifiée pour que le robot avance sur une plus grande distance. **Les élèves peuvent faire le test.**		
10 min	**Faire avancer le robot dans toutes les directions**	Les élèves **testent les différentes instructions possibles pour changer l'orientation du robot.** 	L'enseignant fait remarquer que t**rois instructions sont disponibles** pour faire changer de direction au robot. La première est une **instruction basique**, la seconde permet de **contrôler la vitesse de rotation** du robot, la troisième permet de **contrôler le temps effectué** pour réaliser cette rotation. Il est possible de réaliser un petit programme qui utilise ces trois instructions en faisant varier l'angle, la vitesse et le temps pour bien marquer les différences observées (Voir ci-contre).	*On remarque que le robot tourne vers la droite par défaut. Il est possible de le faire tourner dans l'autre sens en donnant **des valeurs négatives** à l'angle de rotation.* *Il est possible de demander aux élèves de n'utiliser que certaines valeurs d'angle : 45 et -45 pour légèrement tourner à droite ou à gauche ; 90 et -90 pour tourner à droite ou à gauche ; 180 pour faire un demi-tour et 360 pour faire un tour complet. Ainsi, **chaque élément de langage est associé à une valeur** d'angle de rotation.*
15 min	**Explorer les fonctionnalités**	Les élèves **explorent librement la programmation de Thymio avec Scratch.** Pour le moment, ils doivent uniquement **manipuler des instructions simples et les agencer** pour observer ce qui se passe.	L'enseignant encourage les élèves à explorer les instructions qui permettent à Thymio de se **déplacer en cercle**, de **changer de couleur** ou d'**émettre des sons.** Il est aussi souhaitable de souligner que certaines instructions vues lors de la séance précédente peuvent être utilisées avec Thymio ! Notamment les boucles qui permettent de répéter certaines parties du programme.	*Les couleurs des LED sont associées à des **valeurs numériques** pour l'utilisation de Thymio dans Scratch. Ces valeurs peuvent représenter l'**intensité** des LED comme dans l'instruction ci-dessous :* 

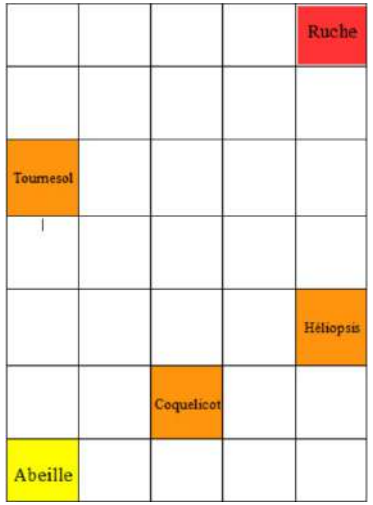
			 Dans ce cas, il est utile de rappeler qu'il est possible de **stopper l'exécution du programme** en cliquant sur le cercle rouge en haut à droite, juste à côté du drapeau vert.	*Ici, plus le nombre associé aux 4 LED entourant le bouton central est élevé, plus la LED correspondante s'éclaire intensément (en rouge).* *Dans d'autres cas, la valeur numérique représente un code RVB qui permet de **coder une couleur particulière** à partir des 3 couleurs primaires (Rouge, Vert, Bleu). Dans l'instruction ci-dessous, en donnant une valeur maximale (32) au Rouge et au Vert et une valeur minimale au Bleu (0), le robot s'éclaire en jaune :*  *Les élèves n'ont pas besoin de comprendre les combinaisons de couleur. Il suffit qu'ils comprennent qu'en modifiant les valeurs numériques présentes dans ce type d'instruction, ils peuvent changer la couleur ou l'intensité des LED qui y sont associées.*
5 min	**Trace écrite**	***"L'exécution d'un algorithme est reproductible (si les instructions ne changent pas, ni les données à manipuler, le programme donne toujours le même résultat)."*** ***"L'ordinateur ne fait qu'exécuter les instructions qu'on lui donne : ni plus, ni moins..."***		

Séance 4	Récolter des ressources	Durée : 45 min	Condition débranchée

Résumé :

- *Les élèves construisent un algorithme pour aider une abeille à ramener des ressources (du nectar) à sa ruche.*
- *Ils apprennent à programmer des boucles de répétition, des instructions conditionnelles (si… alors) et à utiliser une boucle pour tester une condition indéfiniment.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
5 min	**Présentation du plateau de jeu**	L'élève doit **prendre connaissance du nouveau plateau de jeu et écouter les consignes** données par l'enseignant.	L'enseignant distribue en début de séance les **photos des programmes réalisés la séance précédente** qu'il a pris soin d'imprimer. Il distribue également de **nouvelles instructions** qui pourront être utiles plus tard (voir fichier **« Instructions supplémentaires »**). Ensuite, il présente aux élèves le **nouveau plateau de jeu**, plus petit, qui ne permet pas que l'on joue au « robot idiot ». Il va falloir cette fois guider les déplacements d'une abeille pour qu'elle rentre à sa ruche. Mais attention, elle ne peut pas rentrer les mains vides ! Elle doit d'abord butiner les trois fleurs présentes sur son parcours pour **ramener le nectar** à la ruche.	*Pour démarrer, les éléments sur le plateau sont disposés selon la **configuration 1** (voir fichier **« configurations du plateau de jeu »**) :* 

				*Les images représentant les éléments du jeu (pont, rivière, arbre...) sont fournies dans le dossier **« images »**. Elles peuvent être imprimées et plastifiées pour chaque groupe.*
10 min	**Programmer le déplacement de l'abeille**	Les élèves collaborent pour **concevoir l'algorithme demandé** ensemble en réutilisant les instructions qui avaient été distribuées lors de la séance précédente. Ils **testent ensuite cet algorithme** en déplaçant l'image de l'abeille selon les instructions qu'ils ont rédigées ensemble.	L'enseignant demande aux élèves de proposer rapidement un algorithme pour guider les déplacements de l'abeille afin qu'elle aille butiner le coquelicot, puis l'héliopsis et enfin le tournesol (dans cet ordre) avant de rentrer à la ruche.	
10 min	**Programmer le déplacement de l'abeille avec une boucle de répétition.**	Les élèves intègrent une **boucle pour répéter une partie de leur programme.** Ils suppriment donc les instructions redondantes qui ne sont plus nécessaires.	L'enseignant pose la questions aux élèves : « Que peut-on remarquer entre le point de départ et le coquelicot et entre le coquelicot et l'héliopsis ? ». Si les élèves ne trouvent pas, il leur indique que déplacement de l'abeille entre le point de départ et le coquelicot et le déplacement entre le Coquelicot et l'Héliopsis sont **les mêmes** (Avancer 1 case – Droite – Avancer 2 cases – Gauche). Il incite les élèves à utiliser une **boucle de répétition** dans ce cas. Il est possible d'utiliser l'instruction « Répéter 2 fois » pour simplifier et aller plus vite. L'instruction « Répéter » est placée avant ce qui doit être répété.	*Une confusion sur la notion de répétition est possible. Pour effectuer un même déplacement 2 fois, les élèves peuvent avoir tendance à utiliser l'instruction « Répéter 1 fois ». Dans le langage courant, la notion de répétition implique déjà qu'une chose est réalisée plusieurs fois. Ici, c'est bien **« Répéter 2 fois »** qu'il faut utiliser car cette instruction sert à signifier combien de fois les instructions présentes dans la boucle doivent être exécutées.*

5 min	**Comprendre comment organiser son algorithme**	À ce stade, les élèves auront probablement fait « l'erreur » de mettre toutes les instructions du programme à la suite, sur un même plan, comme ceci : Répéter 2 fois Avancer de 1 case Droite Avancer de 2 cases Gauche Les élèves trouvent par eux-mêmes (ou avec le soutien de l'enseignant) **un moyen simple de distinguer ce qui fait partie de la boucle de ce qui n'en fait pas partie.** Il suffit de décaler le contenu de la boucle vers la droite. Le programme doit donc débuter ainsi : Répéter 2 fois Avancer de 1 case Droite Avancer de 2 cases Gauche Avancer de 1 case	L'enseignant fait remarquer que si toutes les instructions sont mises sur le même plan, **on ne peut pas savoir quelle partie de notre programme doit être répétée** et quelle partie ne doit pas l'être. Ici, doit-on répéter « Avancer 1 case ; Droite » ou simplement « Avancer 1 case » par exemple ? Il demande aux élèves quelle solution on pourrait trouver. La solution la plus simple et naturelle qui doit émerger consiste à **décaler le contenu du la boucle sur la droite.** L'enseignant explique aux élèves qu'il est nécessaire d'**organiser l'algorithme** pour qu'il n'y ait aucune ambiguïté quant à son exécution.	

10 min	**Retourner la ressource lorsqu'elle est touchée (1ère étape)**	Les élèves apprennent à utiliser une **structure conditionnelle** en « si … alors … ». Ils réflechissent ensemble à comment remplir cette condition. Une solution possible serait « si **la fleur est touchée par l'abeille** alors **la fleur est retournée** ».	L'enseignant indique qu'on cherche maintenant à créer un algorithme qui permette de retourner l'image de la fleur lorsqu'elle est touchée par l'abeille (pour montrer que cette ressource est bien récoltée). Il précise bien aux élèves que l'on doit **créer un tout nouvel algorithme**, puisque ce n'est plus les déplacements de l'abeille que l'on doit contrôler mais le mouvement des fleurs. Les deux algorithmes vont pouvoir s'exécuter en même temps, **en parallèle**. L'enseignant indique que l'instruction à utiliser est une condition **« si … alors … »** qui va exécuter une partie du programme **uniquement quand la ressource est touchée.**	*Le nouvel algorithme dédié à la programmation du comportement des fleurs peut être créé sur une feuille à part ou sur le côté droit de la feuille si celle-ci est grand format (A3). Il peut être souhaitable de faire noter aux élèves **« Algorithme de l'abeille »** ou **« Algorithme des fleurs »** sur les feuilles pour éviter toute confusion.*

10 min	**Retourner la ressource lorsqu'elle est touchée (2ème étape)**	Les élèves **testent le programme** et remarquent que le test de la condition « si … alors … » ne se fait **qu'une seule fois** au début du programme. Ils **réflechissent alors à une solution qui leur permettrait de tester la condition en permanence.** L'algorithme final qui doit régir le « comportement » des fleurs est le suivant : Répéter indéfiniment Si **Fleur touchée** Alors **Retourner fleur**	L'enseignant propose aux élèves de tester ce nouveau programme en faisant lire chaque algorithme à un élève (un élève pour le déplacement de l'abeille et un élève pour l'algorithme des fleurs). Pour éviter que les élèves ne se coupent la parole, on fait lire une instruction par élève **chacun son tour**. Quand l'élève arrive à la fin de son algorithme, il s'arrête de parler. On remarque ainsi que le test de la condition n'est effectué qu'une seule fois, au lancement du programme. Or, à ce moment-là, l'abeille ne touche aucune fleur. Donc, si on suit rigoureusement l'algorithme, **il n'y a aucune raison de retourner les fleurs !** Pour que le programme fonctionne tel que souhaité, **il faut que le test soit effectué en permanence**, de façon à pouvoir déclencher l'action voulue dès que la condition sera remplie. Pour cela, il suffit de placer ce test dans une boucle **« répéter indéfiniment »**. L'enseignant laisse les élèves tatônner puis **les aiguille vers la réponse si nécessaire.** À la fin de la séance, il est possible d'effectuer **un nouveau test** avec deux élèves. L'élève chargé de lire l'algorithme des fleurs répétera la même instruction à chaque fois.	*Ici, la condition « si … alors … » est considérée comme **une seule et même instruction !*** *Il est nécessaire d'insister sur la manière dont s'exécute un programme ou un algorithme. Les instructions sont exécutées une par une à la suite et il est impossible de garder en mémoire une instruction. Les élèves auront peut-être tendance à prendre en compte la structure conditionnelle même si celle-ci n'est plus répétée, sous prétexte qu'elle a déjà été exécutée une fois. Mais un algorithme ne fonctionne pas comme cela. **On peut seulement répéter plusieurs instructions mais pas revenir en arrière.***

Séance 4	Récolter une ressource	Durée : 45 min	Condition Scratch

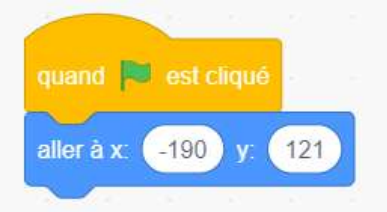
Résumé :

- *Les élèves complètent leur programme en ajoutant des ressources à aller chercher (nouveaux lutins).*
- *Ils apprennent à programmer des instructions conditionnelles (si… alors), à utiliser des capteurs et à utiliser une boucle pour tester une condition indéfiniment.*

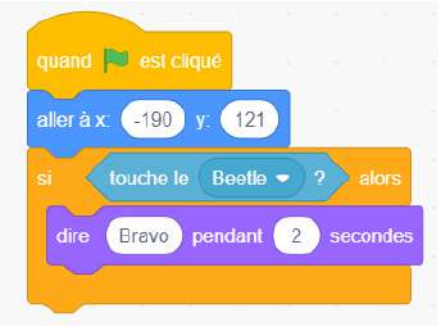
Matériel :

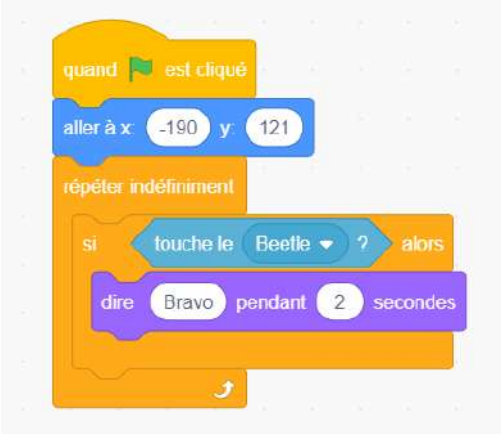
- Pour la classe : un vidéo projecteur.
- *Pour chaque groupe d'élèves : un ordinateur connecté à Internet ou, en l'absence de connexion Internet de bonne qualité, un ordinateur sur lequel le logiciel Scratch a été préalablement installé.*

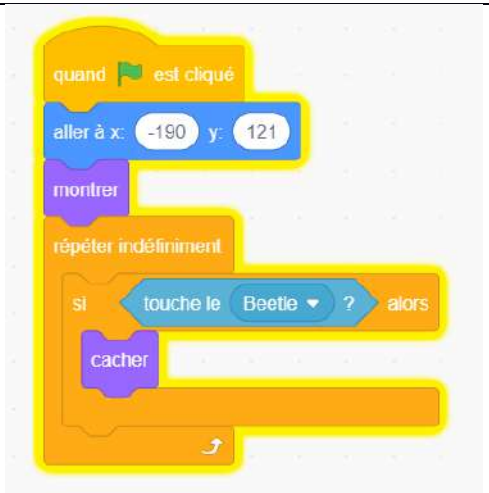
Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
5 min	**importer une ressource sous la forme d'un lutin**	Les élèves ouvrent le fichier sauvegardé lors de la dernière séance et **reprennent l'arrière-plan et le lutin** sélectionnés pendant cette séance (qui sera notre lutin principal). Ils peuvent supprimer les programmes qui avaient été réalisés. **Ils importent un nouveau lutin** qui pourrait être une ressource à récupérer par le lutin principal.	L'enseignant veille à ce qu'ils pensent à initialiser la position de cette ressource de manière aléatoire, en donnant au hasard des valeurs positives ou négatives à x et y dans l'instruction suivante : 	*La ressource peut être positionnée n'importe où, pourvu que les deux lutins **ne se chevauchent pas.***

			Il fait remarquer que chaque lutin possède **sa propre zone de programme** (on passe du programme d'un lutin à l'autre en cliquant sur le lutin voulu). Il y a potentiellement autant de programmes que de lutins : tous ces programmes s'exécutent **en parallèle**.	
5 min	**Comprendre le quadrillage**	Normalement, tous les élèves devraient avoir en arrière-plan un repère (une grille). Les élèves peuvent rapidement **tester cette grille** en programmant un déplacement de 60 pas par exemple qui doit faire avancer le lutin de deux carreaux.	Dans cette étape, il est nécessaire d'expliquer que chaque carreau qui est visible mesure 30 pas de côté.	*On conserve ici le terme **« pas »** et on abandonne la notion de « pixels ». Ainsi, pour déplacer un lutin d'un carreau, il faut l'orienter dans la bonne direction et le faire avancer de 30 pas.*
10 min	**Programmer le déplacement vers la ressource**	Les élèves doivent **compter le nombre de carreaux à traverser.** Plutôt que de multiplier le nombre de pas par le nombre de carreaux, on préférera **conserver l'instruction « Avancer de 30 pas »** qui correspond à un carreau et l'**intégrer dans une boucle** qui répète cette action autant de fois qu'il y a de carreaux à traverser.	L'enseignant indique qu'il est d'abord nécessaire que les élèves **orientent leur lutin vers la position de la ressource.** Problème : si la ressource est positionnée en diagonale par rapport à notre lutin, il est impossible de savoir combien de pas le lutin doit réaliser pour toucher la ressource. On va donc demander aux élèves de s'orienter d'abord sur l'**axe horizontal** (vers la gauche ou vers la droite) pour avancer le nombre de pas nécessaire puis sur l'**axe vertical** (vers le haut ou vers le bas).	

		Exemple, s'il y a 3 carreaux : 		
15 min	**Faire dire « Bravo ! » à la ressource lorsqu'elle est touchée par le lutin principal**	**Les groupes doivent modifier le programme de la ressource pour que celle-ci dise « bravo » lorsqu'elle est touchée par le lutin principal.** Le programme attendu :  Malheureusement, lorsque l'on lance le programme et que le lutin principal se dirige vers la ressource, cela ne fonctionne pas. Pourquoi ? **La classe peut discuter collectivement de ce que fait ce programme** :	L'enseignant indique que l'instruction à utiliser est une condition **« si … alors »** qui va exécuter une partie du programme **uniquement quand la ressource est touchée** et que le bloc qui permet de détecter si la ressource est touchée se trouve dans la catégorie **« capteurs »**. Il laisse les élèves tâtonner, puis passe dans les groupes pour **les guider en cas de blocage.** L'enseignant fait remarquer en lisant le programme proposé par les élèves (ci-contre), que le test **n'est effectué qu'une seule fois**, au lancement du programme (juste après l'initialisation de la ressource). Or, à ce moment-là, les 2 lutins ne se touchent pas. Donc, le message « bravo » ne s'affiche pas. C'est normal. Pour que le programme fonctionne correctement, **il faut que le test soit effectué en permanence**, de façon à pouvoir déclencher l'action voulue dès que la condition sera remplie.	*Objectifs :* *- **faire dire « bravo »** au lutin (tous les élèves savent le faire à ce stade).* *- **déclencher une instruction uniquement lorsqu'une certaine condition est remplie.** Cela se fait via la catégorie « contrôle », dans lequel on trouve l'instruction « si… alors… ».*  *- **détecter quand un lutin en touche un autre.** Cela se fait via la catégorie « capteurs » de la palette d'instructions (instruction « touche le … » sur la version Desktop). Une fois qu'on l'a sélectionné, un clic sur la petite flèche permet de faire défiler les lutins déjà créés.*

		La ressource est placée dans la position que l'on a choisie. Un test est effectué : si la ressource touche le lutin principal, alors elle dit « bravo ». Puis… plus rien. Les élèves suivent les instructions de l'enseignant pour que ce **test soit effectué tout au long de l'exécution du programme** et non pas seulement au début.	Pour cela, il suffit de placer ce test dans une boucle **« répéter indéfiniment »**, que l'on trouve dans la catégorie **« contrôle »**. Le programme de la ressource devient : 	*Ici, on est dans le programme de la ressource et on veut tester si le lutin principal (ici nommé "Beetle") est touché, on clique donc sur "Beetle".* 
10 min	**Faire disparaître la ressource quand elle est touchée**	Les élèves **remplacent, dans le programme de la ressource, l'instruction « dire Bravo » par une instruction faisant disparaître le lutin ressource**. Il s'agit de l'instruction **« cacher »** que l'on trouve dans la catégorie **« apparence ».**	L'enseignant indique aux élèves qu'il faut remplacer **« dire bravo »** par une instruction qui permet de cacher un lutin dans la catégorie **« Apparence »**. Attention, une fois qu'on exécute le programme, la ressource est désormais toujours cachée (car on ne lui a jamais dit de se montrer à nouveau !). Il faut donc ajouter l'instruction **« montrer »** juste après l'instruction **« quand drapeau vert pressé »**. Le programme de la ressource devient donc :	

			 quand est cliqué aller à x: -190 y: 121 montrer répéter indéfiniment si touche le Beetle ? alors cacher	

Séance 4	Atteindre la station électrique	Durée : 45 min	Condition Scratch + Robot

Résumé :

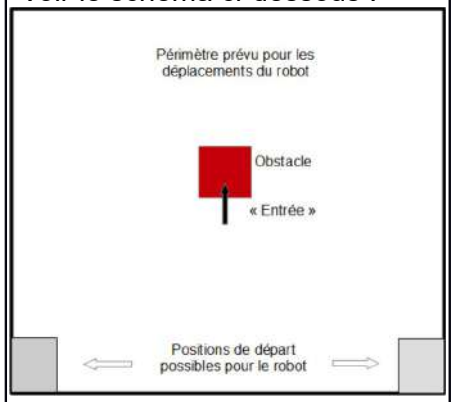
- *Les élèves réalisent un programme avec un obstacle à détecter.*
- *Ils apprennent à programmer des instructions conditionnelles (si… alors), à utiliser des capteurs et à utiliser une boucle pour tester une condition indéfiniment.*

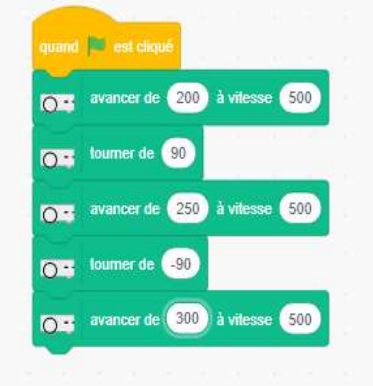
Matériel :

- Pour la classe : un vidéo projecteur.
- *Pour chaque groupe d'élèves : un ordinateur connecté à Internet ou, en l'absence de connexion Internet de bonne qualité, un ordinateur sur lequel le logiciel Scratch a été préalablement installé. Un robot Thymio Wireless.*

Déroulement :

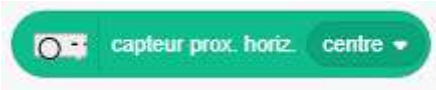
Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
5 min	**Placer un obstacle dans le périmètre**	Les élèves doivent **placer le robot dans un angle du périmètre carré** (l'arrière du robot sur la ligne d'un des côtés, le côté gauche ou droit du robot sur la ligne d'un côté adjacent). **Placer ensuite un obstacle** (d'une hauteur de 10 cm minimum pour être sûr que le robot puisse le détecter) approximativement au centre du périmètre.	L'enseignant demande aux élèves de mettre le robot en position dans un angle du périmètre et de placer un obstacle au centre de celui-ci. Cet obstacle représente la **station électrique** dans laquelle le robot doit aller recharger ses batteries. **L'entrée** de la station est orientée vers le côté où se trouve le robot initialement. L'enseignant indique que Thymio ne pourra pas accéder à la station s'il arrive par la droite ou la gauche de l'obstacle, il doit nécessairement se diriger tout droit en face de l'entrée.	*Il sera demandé aux élèves de programmer le déplacement du robot pour qu'il atteigne la station par **« l'entrée »** de celle-ci. L'objectif est de **complexifier le programme demandé**. Sans cette consigne, il serait possible de faire le déplacement en 3 instructions seulement : Avancer – Pivoter – Avancer. Avec cette consigne, il en faut au moins 4 et il faut faire tourner le robot vers la gauche puis vers la droite (ou l'inverse).*

		Voir le schéma ci-dessous :  		
5 min	**Comprendre l'unité de déplacement du robot**	Les élèves placent une règle à côté du robot, avec le 0 aligné sur l'arrière du robot et **réalisent les tests demandés par l'enseignant.** Ils devraient arriver à la conclusion que l'unité de mesure est **le millimètre**. Donc 10 pas correspondent à 1 cm, 50 à 5 cm, 100 à 10 cm etc.	L'enseignant fait remarquer la distance parcourue par défaut par le robot est de 50, mais 50 quoi ? On ne connaît pas **l'unité utilisée.** L'enseignant demande donc aux élèves de faire des tests. Il demande d'avancer de 50, puis de 100 et enfin de 10 en notant à chaque fois de combien de centimètres le robot a avancé.	
10 min	**Programmer les déplacements du robot**	Maintenant qu'ils connaissent l'unité de déplacement du robot, **les élèves essaient de trouver le chemin approprié pour que le robot atteigne la station**, par essais et erreurs. Ils se servent de leur règle pour estimer les distances à parcourir. Ci-dessous, un programme possible pour atteindre l'objectif :	L'enseignant demande aux élèves de programmer les déplacements du robot pour qu'il atteigne la station. Il rappelle que pour cela, le robot doit impérativement arriver de face, que 10 => 1 cm et que pour tourner à droite, il faut tourner de 90 et de -90 pour tourner à gauche.	*Le problème c'est que, pour le moment, Thymio **ne détecte pas la station** et peut donc foncer dessus sans s'arrêter !*

				
15 min	**Stopper les moteurs lorsque la station est détectée**	**Les élèves doivent modifier la dernière instruction** (dans l'exemple ci-dessus, l'instruction « Avancer de 300 à vitesse 500 ») et la remplacer par l'instruction suivante, qui ne donne pas d'indications de distance :  Le robot avancera tout droit tant qu'on ne l'arrêtera pas. Ils doivent ensuite améliorer le programme pour que le robot **s'arrête lorsqu'il détecte un obstacle** (la station). Le programme attendu est le suivant :	L'enseignant fait remarquer que, pour le moment, **on est obligé de connaître la distance exacte qui sépare Thymio de la station** si on ne veut pas qu'il soit trop court ou trop long (en fonçant sur la station sans s'arrêter). Il explique que Thymio est équipé de capteurs qui permettent de **détecter des obstacles.** Il demande aux élèves de remplacer la dernière instruction de leur programme par une instruction qui fait avancer le robot sans indication de distance. Il leur demande ensuite de suivre et de reproduire les étapes permettant au robot de détecter un obstacle et de s'arrêter.	*La tâche étant relativement complexe, les élèves **suivent pas à pas les instructions** données par l'enseignant.* *La valeur associée au capteur ne correspond PAS à la distance entre le capteur et l'obstacle. Plus le robot se rapproche de l'obstacle plus la valeur du capteur augmente également. Il ne faut donc pas se tromper d'instruction : c'est bien dans un bloc **« > 2500 »** (ou toute autre valeur) qu'il faut intégrer le capteur et pas dans un bloc « < 2500 »*

<table>
<tr>
<td></td>
<td></td>
<td>


Malheureusement, lorsque l'on lance le programme et que le robot se dirige vers la station, cela ne fonctionne pas. Pourquoi ?
La classe peut discuter collectivement de ce que fait ce programme :
La station est placée au centre du périmètre. Un test est effectué : si le robot détecte l'obstacle alors il s'arrête. Puis … plus rien.
Les élèves suivent les instructions de l'enseignant pour que ce test soit effectué tout au long de l'exécution du programme et non pas seulement au début.
</td>
<td>
Il indique que l'instruction à utiliser est une condition « si … alors » qui va exécuter une partie du programme uniquement quand un obstacle est détécté. Pour détecter un obstacle, on utilise l'instruction :

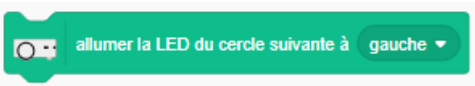


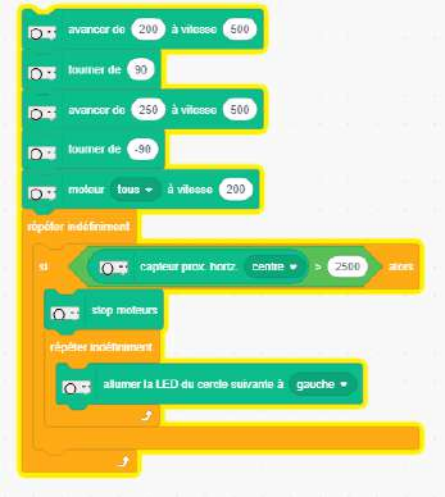
Mais l'enseignant fait remarquer que ce bloc a des bords arrondis, il ne peut pas être placé dans une condition qui nécessite un bloc à bords hexagonaux :

En réalité, c'est tout à fait normal. Une valeur numérique est associé au capteur. Lorsque le robot détecte quelque chose, cette valeur augmente ! Il faut donc placer le capteur à l'intérieur d'un autre bloc. Ce bloc peut être trouvé dans la catégorie « opérateurs », on obtient l'ensemble suivant :

</td>
<td></td>
</tr>
</table>

			Ici, la valeur a été fixée à 2500 mais cette valeur n'est pas essentielle. L'enseignant peut cependant indiquer que plus cette valeur est haute, plus le robot s'arrêtera proche de l'obstacle. Il ne reste plus qu'à placer une instruction pour stopper les moteurs si cette condition est remplie. On obtient le programme détaillé précédemment dans la colonne de gauche. L'enseignant fait remarquer en lisant le programme que **le test n'est effectué qu'une seule fois** juste après avoir mis les moteurs. Or, à ce moment-là, le robot ne détecte aucun obstacle (il est trop loin). Donc, il ne s'arrête pas. C'est normal. Pour que le programme fonctionne correctement, **il faut que le test soit effectué en permanence**, de façon à pouvoir déclencher l'action voulue dès que la condition sera remplie. Pour cela, il suffit de placer ce test dans une boucle **« répéter indéfiniment »**, que l'on trouve dans la catégorie **« contrôle »**. Le programme devient alors :	

				
10 min	**Activer les LED lorsque la station est détectée**	Les élèves réfléchissent à un moyen d'**activer successivement les LED** qui entourent le bouton central pour montrer que le robot est en charge. Ils devraient arriver à la conclusion qu'une **boucle de répétition est nécessaire**. Ils doivent veiller à ce que cette boucle soit intégrée dans la première. On obtient ainsi le programme final suivant :	L'enseignant indique qu'une fois qu'il a atteint la station, le robot peut enfin se mettre en charge ! On voudrait maintenant programmer le robot pour qu'il montre clairement qu'il est en charge. Pour se faire , on peut utiliser l'instruction suivante :  L'enseignant demande aux élèves de tester ce que fait cette instruction en cliquant plusieurs fois dessus. On remarque qu'elle permet d'allumer successivement les LED qui entourent le bouton central. Il demande aux élèves comment on pourrait procéder pour que les LED **s'allument successivement et indéfiniment** lorsque le robot a atteint l'obstacle pour bien montrer qu'il est en train de charger.	*Il est possible (si le temps le permet) d'activer les LED uniquement un certain temps (avec l'instruction **« répéter x fois »**) puis de faire produire un son au robot, comme pour signifier qu'il a fini de charger ! Le robot pourrait aussi **changer de couleur** lorsqu'il atteint la station ou lorsqu'il a fini de charger. Lorsque le robot a atteint la station, stoppé ses moteurs et activé l'animation « en charge » (en activant successivement les LED), on devrait obtenir le résultat suivant :*

		 Les valeurs des distances dans les 4 premières instructions dépendent bien sûr du contexte.		 *(Le gif ci-dessus ne rend pas très bien, en réalité l'animation est plus fluide et plus homogène).*

Séance 5	Créer une variable	Durée : 45 min	Condition débranchée

Résumé :

- *Les élèves apprennent à créer une variable qui comptabilise le nombre de ressources récoltées et à la modifier.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	***Remarques***
5 min	**Créer une variable « score »**	En s'aidant des photos distribuées par l'enseignant, les élèves **reconstituent les programmes réalisées** lors de la séance précédente. Les élèves **réalisent une boîte** (par exemple avec des morceaux de carton et du Scotch). Ils peuvent aussi simplement récupérer une boîte et, si possible, écrire « Score » ou « Ressources » dessus (par exemples). Ce sera le nom de cette variable.	L'enseignant **distribue en début de séance les photos des programmes réalisés** la séance précédente qu'il a pris soin d'imprimer. Il distribue également de nouvelles instructions qui pourront être utiles plus tard (voir fichier **« Instructions supplémentaires 2 »**). L'enseignant indique aux élèves qu'ils doivent créer un score maintenant et l'augmenter à chaque fois qu'une ressource est récupérée. Il prend soin de leur expliquer que l'on a besoin de créer ce que l'on appelle **une variable**, qui est en quelque sorte une « boîte » permettant de stocker des valeurs et de les modifier.	*Pour qu'un programme soit facile à comprendre, il est important de **donner des noms explicites aux variables** que l'on crée. Cette bonne habitude limite également les bugs de programmation. Le nom de la variable peut donc être, tout simplement : « score » ou « ressource ». Certains élèves utilisent des noms qui n'ont pas de sens, ou qui révèlent une confusion entre la variable et les manipulations de cette variable (par exemple, ils nomment la variable « ajouter 1 au score »).*

10 min	**Augmenter le score**	Les élèves **décident de la façon dont le score doit être augmenté** (par exemple, on l'augmente de 1 à chaque fois que l'abeille touche la ressource). Dans un second temps, les élèves **cherchent l'instruction** permettant d'augmenter le score de 1 (par exemple) : Ajouter ..... à la variable ..... Il suffit de remplir correctement cette instruction (par exemple « Ajouter 1 à la variable « Score » ») et de la **placer dans le programme de la fleur, à l'intérieur du test**, juste en-dessous ou au-dessus de l'instruction **« retourner fleur » :** Répéter indéfiniment Si **Fleur touchée** Alors **Retourner fleur** Ajouter **1** à la variable **"Score"** Les élèves **testent les programmes** (toujours un élève pour chaque programme). Lorsqu'une fleur est touchée par l'abeille, les élèves peuvent mettre 1 point dans la boîte « Score » (voir l'image **« 1 point »** dans le dossier **« Images »**)	L'enseignant demande aux élèves d'augmenter cette variable « score » quand la ressource est touchée.	

10 min	**Initialiser le score à 0**	Pour initialiser la variable à zéro, il suffit d'ajouter l'instruction **« mettre le score à 0 »** en début de programme : Mettre la variable **Score** à **0** Les élèves peuvent maintenant **tester le programme final.** Lorsqu'ils doivent mettre la variable « Score » à 0, ils sont autorisés à **retourner la boîte** pour la vider de son contenu.	L'enseignant fait remarquer que si on relance plusieurs fois le programme, **le score n'est pas remis à zéro**, car aucune instruction ne le permet. Il demande aux élèves de trouver un moyen de remettre le score à 0 quand on relance le programme.	
15 min	**Placer de nouvelles ressources**	Les élèves changent la configuration du plateau de jeu et optent pour la **configuration 2** (voir fichier **« configurations du plateau de jeu »**). Désormais, de nouvelles ressources sont présentes sur le plateau, et il y a aussi des obstacles (dangers) à éviter. Les étapes suivantes sont à réaliser à nouveau en autonomie : - **Programmation des déplacements de l'abeille** qui doit toujours aller récolter les trois premières fleurs AVANT de récolter les nouvelles. Il est donc nécessaire de conserver les premiers déplacements qui ont été programmés précédemment et de rajouter simplement une suite à ces déplacements qui	Une fois que l'enseignant est assuré que les étapes précédentes ont bien été réalisées et comprises par les élèves, il leur demande de changer la configuration du plateau puis de recommencer le processus pour **augmenter le nombre de ressources récoltées**.	*Les éléments sur le plateau sont disposés selon la **configuration 2** (voir fichier **« configurations du plateau de jeu »**) :* 

		permet à l'abeille d'avancer jusqu'aux nouvelles fleurs après avoir récolté les premières. - **Programmation de nouvelles fonctionnalités** lorsque l'abeille touche une fleur. Par exemple, l'abeille accélère lorsqu'elle touche une fleur. Les élèves peuvent ainsi créer leurs propres instructions et les intégrer dans leur programme. Dans cet exemple, les élèves devraient commencer la lecture de l'algorithme très lentement et accélérer à chaque fois qu'une fleur est touchée.		*En gris, les **danger à éviter.** Les images nécessaires pour les dangers et les nouvelles fleurs sont disponibles dans le dossier **« Images ».*** *Elles peuvent être imprimées et plastifiées pour chaque groupe.*
5 min	**Trace écrite**	***« Une boucle permet de répéter plusieurs fois la même action. Certaines boucles ne s'arrêtent jamais, d'autres peuvent être répétées un nombre prédéfini de fois ».*** ***« Une condition est une expression qui est soit vraie, soit fausse. On peut choisir quelle action effectuer si une condition est vérifiée ou non ».*** ***« Une variable permet de stocker une valeur pour la modifier ou la réutiliser plus tard »***		

Séance 5	Créer une variable	Durée : 45 min	Condition Scratch

Résumé :

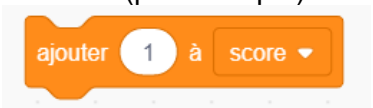
- *Les élèves apprennent à créer une variable qui comptabilise le nombre de ressources récoltées et à la modifier.*

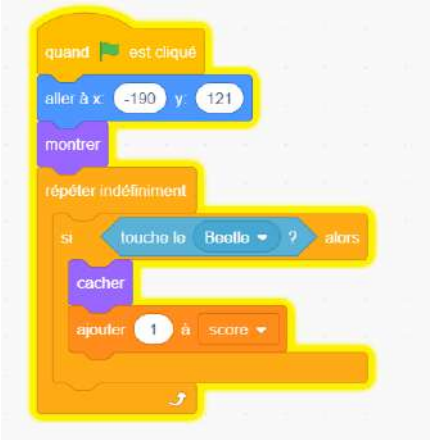
Matériel :

- Pour la classe : un vidéo projecteur.
- *Pour chaque groupe d'élèves : un ordinateur connecté à Internet ou, en l'absence de connexion Internet de bonne qualité, un ordinateur sur lequel le logiciel Scratch a été préalablement installé.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
5 min	**Créer une variable « score »**	Les élèves **reprennent le programme réalisé** lors de la dernière séance. Ils peuvent explorer les différentes catégories d'instructions : la réponse se trouve dans la catégorie orange intitulée **« Variables »**, via la commande **« créer une variable »**. La variable ainsi créée peut être accessible par un seul lutin ou par tous.	L'enseignant indique aux élèves qu'ils doivent créer un score maintenant et l'augmenter à chaque fois qu'une ressource est récupérée. Il fait remarquer aux élèves qu'ils ont déjà **manipulé des variables** dans les séances précédentes (l'abscisse X et l'ordonnée Y, qui donnent la position d'un lutin à l'écran). Ces variables étaient déjà disponibles et les élèves ont pu les manipuler sans avoir besoin de les créer. Il prend soin de leur expliquer que l'on a besoin de créer ce que l'on appelle **une variable,** qui est en quelque sorte une « boîte » permettant de stocker et modifier des valeurs.	*Pour qu'un programme soit facile à comprendre, il est important de **donner des noms explicites aux variables** que l'on crée. Cette bonne habitude limite également les bugs de programmation. Le nom de la variable peut donc être, tout simplement : « score » ou « ressource ». Certains élèves utilisent des noms qui n'ont pas de sens, ou qui révèlent une confusion entre la variable et les manipulations de cette variable (par exemple, ils nomment la variable « ajouter 1 au score »).*

				On remarque que lorsque la variable est créée, elle est affichée à l'écran, ainsi que sa valeur. Ici, il s'agit de compter un score. Cette variable pourra être manipulée par plusieurs lutins lorsqu'on décidera plus tard d'ajouter de nouvelles ressources : on doit donc la rendre accessible à tous les lutins.
10 min	**Augmenter cette variable**	Les élèves **décident de la façon dont le score doit être augmenté** (par exemple, on l'augmente de 1 à chaque fois que le lutin touche la ressource). Dans un second temps, les élèves **cherchent l'instruction** permettant d'augmenter le score de 1 (par exemple) :  Il suffit de la **placer dans le programme de la ressource, à l'intérieur du test,** juste en-dessous ou au-dessus de l'instruction « cacher ».	**L'enseignant demande aux élèves d'augmenter cette variable « score » quand la ressource est touchée.**	

				
10 min	**Initialiser le score à zéro**	Les élèves **testent à plusieurs reprises** ce qui se passe quand le lutin principal touche la ressource. Le programme semble bien fonctionner (la ressource est présente, le lutin principal la touche, elle disparaît et le score est augmenté de 1). Pour initialiser la variable à zéro, il suffit d'ajouter l'instruction **« mettre score à 0 »** en début de programme. 	L'enseignant fait remarquer que si on arrête le programme et qu'on le relance, **le score n'est pas remis à zéro.** Il demande aux élèves de trouver un moyen de remettre le score à 0 quand on relance le programme.	*A priori, cette initialisation peut être faite dans le programme de n'importe quel lutin : l'important étant qu'elle soit faite une fois, et une seule. Mais le score est une variable qui sera sans doute manipulée par d'autres lutins (les autres ressources que nous importerons dans la tâche suivante). Pour cette raison, nous conseillons de le faire dans le programme du lutin (qui est notre **programme « principal »**), juste en dessous de l'initialisation de sa position.*

15 min	**Importer de nouvelles ressources**	Les étapes suivantes sont à réaliser à nouveau en autonomie. - **Importation d'une nouvelle ressource** et initialisation de sa position. - **Programmation des déplacements du lutin principal** qui doit toujours aller récolter la première ressource avant de récolter la seconde. Il est donc nécessaire de conserver les premiers déplacements qui ont été programmés précédemment et de rajouter simplement une suite à ces déplacements qui permet au lutin principal d'avancer jusqu'à la nouvelle ressource après avoir récolté la première. - **Programmation de la nouvelle ressource** pour qu'elle puisse aussi disparaître quand elle est touchée et augmenter la valeur du score.	Une fois que l'enseignant est assuré que les étapes précédentes ont bien été réalisées et comprises par les élèves, il peut les inciter à importer une (ou plusieurs) nouvelle(s) ressource(s) et à recommencer le processus pour **augmenter le nombre de ressources récoltées.**	*Il est conseillé d'indiquer aux élèves que la position des nouvelles ressources à importer **doit être assez éloignée** de la première. Il est important d'inciter les élèves à être "paresseux", c'est-à-dire à réfléchir à quelle partie de leur code peut être **simplement recopiée** et quelle partie doit être **modifiée.***
5 min	**Trace écrite**	***« Une boucle permet de répéter plusieurs fois la même action. Certaines boucles ne s'arrêtent jamais, d'autres peuvent être répétées un nombre prédéfini de fois ».*** ***« Une condition est une expression qui est soit vraie, soit fausse. On peut choisir quelle action effectuer si une condition est vérifiée ou non ».*** ***« Une variable permet de stocker une valeur pour la modifier ou la réutiliser plus tard »***		

Séance 5	Créer une variable	Durée : 45 min	Condition Scratch + Robot

Résumé :

- *Les élèves apprennent à créer une variable qui comptabilise le nombre de ressources récoltées et à la modifier.*

Matériel :

- Pour la classe : un vidéo projecteur.
- *Pour chaque groupe d'élèves : un ordinateur connecté à Internet ou, en l'absence de connexion Internet de bonne qualité, un ordinateur sur lequel le logiciel Scratch a été préalablement installé. Un robot Thymio Wireless.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
5 min	**Placer une ressource et créer une variable « score »**	Les élèves **reprennent le programme réalisé** lors de la dernière séance. Sur le chemin suivi par le robot, ils rajoutent une ressource : **une batterie** que le robot doit récupérer avant de parvenir à la station. La ressource peut être représentée par une image, imprimée puis plastifiée, disposée sur le parcours du robot (voir l'image **« Batterie »**). Ils peuvent explorer les différentes catégories d'instructions.	L'enseignant indique aux élèves qu'ils doivent maintenant créer un score et l'augmenter à chaque fois qu'une ressource est récupérée. Il prend soin de leur expliquer que l'on a besoin de créer ce que l'on appelle **une variable**, qui est en quelque sorte une « boîte » permettant de stocker des valeurs et de les modifier.	*Pour qu'un programme soit facile à comprendre, il est important de **donner des noms explicites aux variables** que l'on crée. Cette bonne habitude limite également les bugs de programmation. Le nom de la variable peut donc être, tout simplement : « score » ou « ressource ». Certains élèves utilisent des noms qui n'ont pas de sens, ou qui révèlent une confusion entre la variable et les manipulations de cette variable (par exemple, ils nomment la variable « ajouter 1 au score »).*

		La réponse se trouve dans la catégorie orange intitulée **« Variables »**, via la commande **« créer une variable »**.		*Lors de la création de la variable, il est demandé si elle doit être accessible pour ce sprite (lutin) seulement ou pour tous les sprites. Dans notre cas, ce choix importe peu car nous programmons un robot et pas un lutin de Scratch.*
10 min	**Augmenter le score**	Les élèves **décident de la façon dont le score doit être augmenté** (par exemple, on l'augmente de 1 à chaque fois que le robot touche la batterie). Dans un second temps, les élèves **cherchent l'instruction** permettant d'augmenter le score de 1 (par exemple) :  Il suffit de la placer juste en-dessous d'un événement **« Quand la touche espace est pressée »**:  Ces deux blocs sont séparés des autres blocs qui ont été assemblés lors de la séance précédente.	L'enseignant demande aux élèves d'augmenter cette variable « score » quand la batterie est touchée par le robot en pressant la barre d'espace. Il précise qu'en effet, cette fois, le robot ne sera pas capable de détecter la batterie. Ce sera aux élèves **d'augmenter le score « manuellement »**.	*Pour se familiariser avec ces commandes, nous conseillons de laisser les élèves les expérimenter quelques minutes avec l'affichage des variables qu'ils manipulent.* *Le nouveau programme qui permet d'incrémenter le score en pressant la barre d'espace **s'exécute en parallèle** de l'ancien qui codait les déplacements et les actions du robot.*

10 min	**Initialiser le score**	Les élèves **testent à plusieurs reprises le programme** qui semble bien fonctionner (la batterie est bien présente sur le parcours du robot, quand le robot la touche, un des élèves appuie sur la barre espace et le score est augmenté de 1). Pour initialiser la variable à zéro, il suffit d'ajouter l'instruction **« mettre score à 0 »** en début de programme. 	L'enseignant fait remarquer que si on arrête le programme et qu'on le relance, **le score n'est pas remis à zéro.** Il demande aux élèves de trouver un moyen pour initialiser le score à 0 lorsque le programme est lancé.	
15 min	**Placer de nouvelles ressources**	Les étapes suivantes sont à réaliser à nouveau en autonomie. - **Placement d'une (ou plusieurs) nouvelle(s) batterie(s)** dans le périmètre. - **Programmation des déplacements du robot** qui doit toujours aller récolter la première batterie avant de récolter la seconde.	Une fois que l'enseignant est assuré que les étapes précédentes ont bien été réalisées et comprises par les élèves, il peut les inciter à placer une (ou plusieurs) nouvelle(s) batterie(s) et à recommencer le processus pour **augmenter le nombre de batteries récoltées.**	*Il est conseillé d'indiquer aux élèves que la position des nouvelles ressources à importer doit être **assez éloignée** de la première. Il est important d'inciter les élèves à être "paresseux", c'est-à-dire à réfléchir à quelle partie de leur code peut être **simplement recopiée** et quelle partie doit être **modifiée**.*

		Il est donc nécessaire de conserver les premiers déplacements qui ont été programmés précédemment et de rajouter simplement une suite à ces déplacements qui permet au robot d'avancer jusqu'à la nouvelle batterie après avoir récolté la première. - **Programmation de nouvelles fonctionnalités** lorsque le robot touche la batterie (donc lorsque la barre espace est pressée). Par exemple, le robot change de couleur lorsqu'il touche une batterie.		
5 min	**Trace écrite**	***« Une boucle permet de répéter plusieurs fois la même action. Certaines boucles ne s'arrêtent jamais, d'autres peuvent être répétées un nombre prédéfini de fois ».*** ***« Une condition est une expression qui est soit vraie, soit fausse. On peut choisir quelle action effectuer si une condition est vérifiée ou non ».*** ***« Une variable permet de stocker une valeur pour la modifier ou la réutiliser plus tard ».***		

Séance 6	Projet créatif – Partie 1	Durée : 45 min	Condition Débranchée

Résumé :

- *L'enseignant demande aux élèves de réaliser un projet créatif en lien avec la programmation.*
- *Les élèves prennent connaissance des consignes de ce projet et réfléchissent en groupe à ce qu'ils veulent réaliser.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
10 min	**Présentation du projet**	Les élèves **écoutent l'enseignant présenter le projet** qu'ils doivent réaliser. Une fois que la présentation a été faîte, ils peuvent demander des précisions s'ils n'ont pas tout compris.	L'enseignant explique aux élèves qu'ils vont devoir réaliser un projet en autonomie pendant plusieurs séances avant de le présenter à la classe. Les consignes à donner sont les suivantes : – Vous allez avoir une carte du monde quadrillée (avec des cases). Voir fichier **« Carte »** pour une proposition de carte possible. – Ensuite vous devez **programmer des actions à réaliser** pour un personnage de votre choix (n'importe quelle figurine peut faire l'affaire) pour que celui-ci fasse un « tour du monde ».	*Dans l'idéal, les projets des élèves doivent tous impliquer au moins une variable. Mais il faut reconnaître que la notion de variable est parfois complexe à appréhender et difficile à intégrer dans un projet qui ne s'y prête pas. Par conséquent, si les élèves buttent sur la variable à intégrer, il est possible de* ***supprimer cette consigne.***

			Vous pouvez réutiliser les instructions déjà utilisées lors des séances précédentes mais vous pouvez aussi en créer de nouvelles ! – Le programme doit contenir au moins **une boucle** (« Répéter »), **une condition** (« Si … Alors ... ») et **une variable.**	
15 min	**1ères réflexions sur le projet**	Les élèves **réfléchissent ensemble (par groupes) au projet qu'ils veulent mettre en place.** Ils commencent à écrire sur une feuille leurs idées.	L'enseignant demande aux élèves de réfléchir à ce qu'ils souhaitent réaliser comme projet. Il insiste sur le fait que les consignes doivent être scrupuleusement respectées malgré la relative liberté des élèves pour construire leur projet. Il peut laisser les élèves **travailler en autonomie** sans intervenir.	*Il est préférable de conserver **les mêmes groupes** que dans les séances précédentes, si aucun problème particulier n'a été observé.*
10 min	**Mise en commun**	Un élève par groupe vient **présenter à l'ensemble de la classe les premières réflexions** de son groupe. Les autres élèves peuvent interagir pour demander des précisions, faire remarquer que certaines consignes ne sont pas bien respectées etc.	L'enseignant demande aux élèves de présenter leurs premières réflexions. Il essaie d'encourager les interactions entre les élèves. Il s'assure surtout que les premières pistes de réflexions **respectent les consignes**, que les projets semblent **réalisables**, qu'ils sont **suffisamment complexes** pour présenter un intérêt mais **pas non plus excessivement complexes.**	*Il est possible de **donner des rôles** au sein du groupe pour organiser la répartition des tâches collaboratives : secrétaire, rapporteur…*
15 min	**Approfondissement du projet**	Suite à la mise en commun des premières idées de chaque groupe, les élèves ont pu bénéficié d'un retour sur leurs réflexions.	L'enseignant guide les élèves pour qu'ils finalisent leurs réflexions concernant leur projet.	*De préférence, le projet proposé par les élèves **ne devrait pas évoluer** au-delà de cette séance.*

		Ils profitent de cette fin de séance pour corriger ce qui doit l'être, **approfondir et préciser leur projet**. Ils doivent aussi écrire sur une feuille **un résumé** de leur projet qui doit être définitif.	Il s'assure que les retours qui ont été faits précédemment ont été bien pris en compte. Il passe dans les groupes pour s'assurer que les élèves ont tous une idée claire de **ce qu'ils doivent réaliser pour leur projet.**	*Leurs réflexions sur le contenu du projet qu'ils vont proposer devraient être abouties à l'issue de cette tâche.*

Séance 6	Projet créatif – Partie 1	Durée : 45 min	Condition Scratch

Résumé :

- *L'enseignant demande aux élèves de réaliser un projet créatif en lien avec la programmation.*
- *Les élèves prennent connaissance des consignes de ce projet et réfléchissent en groupe à ce qu'ils veulent réaliser.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
10 min	**Présentation du projet**	Les élèves **écoutent l'enseignant présenter le projet** qu'ils doivent réaliser. Une fois que la présentation a été faîte, ils peuvent demander des précisions s'ils n'ont pas tout compris.	L'enseignant explique aux élèves qu'ils vont devoir réaliser un projet en autonomie pendant plusieurs séances avant de le présenter à la classe. Les consignes à donner sont les suivantes : – Vous allez importer une **image satellite de votre école** (prise sur Google maps) en arrière-plan de votre programme Scratch. – Ensuite vous devez **programmer des actions à réaliser** pour un ou plusieurs lutins en relation avec l'école (par exemple, un lutin peut présenter son école à un autre lutin).	*De préférence, l'arrière-plan Google Map de l'école **devrait être réalisé par le professeur** en amont. On peut cependant aussi imaginer que cette tâche puisse être réalisée par les élèves eux-mêmes, **en cours de géographie** par exemple.* *Dans l'idéal, les projets des élèves doivent tous impliquer au moins une variable. Mais il faut reconnaître que la notion de variable est parfois complexe à appréhender et difficile à intégrer dans un projet qui ne s'y prête pas.*

			– Le programme doit contenir au moins **une boucle** (« Répéter »), **une condition** (« Si … Alors ... ») et **une variable.**	*Par conséquent, si les élèves buttent sur la variable à intégrer, il est possible de **supprimer cette consigne**.*
15 min	**1ères réflexions sur le projet**	Les élèves **réfléchissent ensemble (par groupes) au projet qu'ils veulent mettre en place.** Ils commencent à écrire sur une feuille leurs idées.	L'enseignant demande aux élèves de réfléchir à ce qu'ils souhaitent réaliser comme projet. Il insiste sur le fait que les consignes doivent être scrupuleusement respectées malgré la relative liberté des élèves pour construire leur projet. Il peut laisser les élèves **travailler en autonomie** sans intervenir.	*Il est préférable de conserver **les mêmes groupes** que dans les séances précédentes, si aucun problème particulier n'a été observé.*
10 min	**Mise en commun**	Un élève par groupe vient **présenter à l'ensemble de la classe les premières réflexions** de son groupe. Les autres élèves peuvent interagir pour demander des précisions, faire remarquer que certaines consignes ne sont pas bien respectées etc.	L'enseignant demande aux élèves de présenter leurs premières réflexions. Il essaie d'encourager les interactions entre les élèves. Il s'assure surtout que les premières pistes de réflexions **respectent les consignes**, que les projets semblent **réalisables,** qu'ils sont **suffisamment complexes** pour présenter un intérêt mais **pas non plus excessivement complexes.**	*Il est possible de **donner des rôles** au sein du groupe pour organiser la répartition des tâches collaboratives : secrétaire, rapporteur…*
15 min	**Approfondissement du projet**	Suite à la mise en commun des premières idées de chaque groupe, les élèves ont pu bénéficié d'un retour sur leurs réflexions. Ils profitent de cette fin de séance pour corriger ce qui doit l'être, **approfondir et préciser leur projet.**	L'enseignant remet les élèves aux travail pour qu'ils finalisent leurs réflexions concernant leur projet. Il s'assure que les retours qui ont été faits précédemment ont été bien pris en compte. Il passe dans les groupes pour s'assurer que les élèves ont tous une idée claire de **ce qu'ils doivent réaliser pour leur projet.**	*De préférence, le projet proposé par les élèves **ne devrait pas évoluer** au-delà de cette séance. Leurs réflexions sur le contenu du projet qu'ils vont proposer devraient être abouties à l'issue de cette tâche.*

		Ils doivent aussi écrire sur une feuille **un résumé** de leur projet qui doit être définitif.		

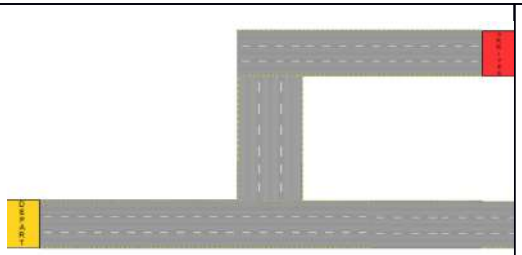
Séance 6	Projet créatif – Partie 1	Durée : 45 min	Condition Scratch + Robot

Résumé :

- *L'enseignant demande aux élèves de réaliser un projet créatif en lien avec la programmation.*
- *Les élèves prennent connaissance des consignes de ce projet et réfléchissent en groupe à ce qu'ils veulent réaliser.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
10 min	**Présentation du projet**	Les élèves écoutent l'enseignant **présenter le projet** qu'ils doivent réaliser. Une fois que la présentation a été faîte, ils peuvent demander des précisions s'ils n'ont pas tout compris.	L'enseignant explique aux élèves qu'ils vont devoir réaliser un projet en autonomie pendant plusieurs séances avant de le présenter à la classe. Les consignes à donner sont les suivantes : – Vous allez avoir un **plan avec des rues dessinées** sur une grande feuille (Format A1 idéalement, ou 2 X A2). Les routes doivent se croiser comme sur le schéma ci-dessous (en jaune le départ, en rouge l'arrivée) :	*Il est possible de faire réaliser le plan par les élèves eux-mêmes, à partir d'un modèle ou non. Mais cette tâche **ne peut pas être réalisée au cours des séances de programmation.** Cela peut être effectué lors d'une **séance de géométrie** sur la construction de droites parallèles et perpendiculaires. Il faut veiller à ce que les rues soient assez larges pour que le robot puisse y circuler (environ 12 cm) Dans l'idéal, les projets des élèves doivent tous impliquer au moins une variable.*

			 – Ensuite vous **programmerez des actions à réaliser** pour que le robot se déplace dans les rues et atteigne son objectif. Il peut bien sûr changer de couleur, émettre des sons et utiliser toutes les fonctionnalités possibles. – Le programme doit contenir au moins **une boucle** (« Répéter »), **une condition** (« Si … Alors ... ») et **une variable.**	*Mais il faut reconnaître que la notion de variable est parfois complexe à appréhender et difficile à intégrer dans un projet qui ne s'y prête pas.* *Par conséquent, si les élèves buttent sur la variable à intégrer, il est possible de* ***supprimer cette consigne.***
15 min	**1ères réflexions sur le projet**	Les élèves **réfléchissent ensemble (par groupes) au projet qu'ils veulent mettre en place.** Ils commencent à écrire sur une feuille leurs idées.	L'enseignant demande aux élèves de réfléchir à ce qu'ils souhaitent réaliser comme projet. Il insiste sur le fait que les consignes doivent être scrupuleusement respectées malgré la relative liberté des élèves pour construire leur projet. Il peut laisser les élèves **travailler en autonomie** sans intervenir.	*Il est préférable de conserver l****es mêmes groupes*** *que dans les séances précédentes, si aucun problème particulier n'a été observé.*
10 min	**Mise en commun**	Un élève par groupe vient **présenter à l'ensemble de la classe les premières réflexions** de son groupe.	L'enseignant demande aux élèves de présenter leurs premières réflexions. Il essaie d'encourager les interactions entre les élèves.	*Il est possible de* ***donner des rôles*** *au sein du groupe pour organiser la répartition*

		Les autres élèves peuvent interagir pour demander des précisions, faire remarquer que certaines consignes ne sont pas bien respectées etc.	Il s'assure surtout que les premières pistes de réflexions **respectent les consignes**, que les projets semblent **réalisables**, qu'ils sont **suffisamment complexes** pour présenter un intérêt mais **pas non plus excessivement complexes.**	*des tâches collaboratives : secrétaire, rapporteur…*
15 min	**Approfondissement du projet**	Suite à la mise en commun des premières idées de chaque groupe, les élèves ont pu bénéficié d'un retour sur leurs réflexions. Ils profitent de cette fin de séance pour corriger ce qui doit l'être, **approfondir et préciser leur projet.** Ils doivent aussi écrire sur une feuille **un résumé** de leur projet qui doit être définitif.	L'enseignant remet les élèves aux travail pour qu'ils finalisent leurs réflexions concernant leur projet. Il s'assure que les retours qui ont été faits précédemment ont été bien pris en compte. Il passe dans les groupes pour s'assurer que les élèves ont tous une idée claire de **ce qu'ils doivent réaliser pour leur projet.**	*De préférence, le projet proposé par les élèves **ne devrait pas évoluer** au-delà de cette séance. Leurs réflexions sur le contenu du projet qu'ils vont proposer devraient être abouties à l'issue de cette tâche.*

Séance 7	Projet créatif – Partie 2	Durée : 45 min	Condition débranchée Condition Scratch Condition Scratch + Robots

Résumé :

- *Les élèves apprennent à décomposer leur projet en un ensemble de problèmes simples à résoudre.*
- *Ils apprennent à réutiliser les algorithmes qu'ils ont réalisé lors des séances précédentes et commencent la réalisation de leur projet.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
5 min	**Relecture des projets**	Par groupes, les élèves relisent ce qu'ils ont écrit lors de la séance précédente et **se remémore donc le projet** qu'ils ont choisi de réaliser.	L'enseignant demande aux élèves de se remémorer le projet sur lequel ils ont réfléchi précédemment.	*On peut par exemple choisir de désigner un élève par groupe qui sera chargé de relire le projet à haute voix pour les autres membres du groupe.*
15 min	**Décomposition du projet en sous-problèmes**	Dans chaque groupe, les élèves **décomposent le projet** qu'ils souhaitent réaliser en **plusieurs sous-problèmes** qu'ils sont capables de résoudre. Ils écrivent les **différentes étapes de programmation** à suivre pour réaliser ce projet.	L'enseignant demande aux élèves de décomposer leur projet en étapes de programmation (sous-problèmes simples à résoudre). Il passe dans les groupes pour s'assurer que la consigne est bien comprise. Une étape doit nécessairement **correspondre à une action à programmer**. Ex : Déplacer le robot/lutin/personnage à tel endroit et le faire tourner sur lui-même correspond à deux étapes de programmation (1. déplacer ; 2. pivoter)	*Pour plus de clarté sur la tâche à accomplir, il est possible d'imprimer une feuille pour chaque groupe avec écrit : **"D'abord, je dois programmer … ; Ensuite, je dois programmer … etc."***

10 min	**Reconnaissance de formes**	Pour chaque étape de programmation de leur projet, les élèves s'interrogent sur la **possibilité de réutiliser des parties de code** réalisées lors des séances précédentes. Ils peuvent ainsi reprendre les algorithmes conçus précédemment (programmes Scratch enregistrés sur l'ordinateur). **Les élèves apprennent à reconnaître des formes et à les réutiliser.**	L'enseignant demande aux élèves, pour chaque étape de programmation déterminée précédemment, d'indiquer si oui ou non ils peuvent récupérer un algorithme (ou une partie d'algorithme) qu'ils ont réalisé lors des séance précédentes.	*L'objectif est de montrer aux élèves qu'en programmation, il n'est pas forcément nécessaire de partir de zéro. Il est souvent possible de réutiliser des parties de code qui ont servi pour d'autres projets.*
15 min	**Réalisation du projet 1**	Les élèves commencent à **concevoir en pratique les algorithmes** dont ils ont besoin pour réaliser leur projet.	L'enseignant demande aux élèves de commencer à réaliser leur projet. Ils doivent donc concevoir les algorithmes qui, une fois exécutés, permettront la réalisation des actions souhaitées. L'enseignant reste disponible pour **aider les élèves en cas de blocage.**	*La réalisation du projet se terminera lors de la séance suivante.*

Séance 8	Projet créatif – Partie 3	Durée : 45 min	Condition débranchée Condition Scratch Condition Scratch + Robots

Résumé :

- *Les élèves continuent à concevoir leurs programmes pour réaliser leur projet.*
- *En milieu de séance, une tâche de débuggage est proposée pour permettre aux élèves de communiquer leurs difficultés et de réfléchir ensemble à un moyen de résoudre leurs blocages.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
15 min	**Réalisation du projet 2**	Les élèves continuent de travailler sur leur projet. Ils **conçoivent et testent les algorithmes** dont ils ont besoin pour le mener à bien.	L'enseignant laisse les élèves travailler **en autonomie** et n'intervient que si son aide est demandée.	*La progression des élèves est relativement libre. Il faut juste s'assurer que certains élèves ne soient pas passifs ou inactifs.*
15 min	**Débuggage**	Les élèves profitent d'un temps dédié au **« débuggage »**, c'est-à-dire **à l'analyse et au repérage des dysfonctionnements** du programme. Par groupe, les élèves expliquent aux autres soit ce qu'ils **n'arrivent pas à faire**, soit **ce qu'ils leur restent à faire** et comment ils comptent s'y prendre.	L'enseignant interrompt le travail des groupes sur leur projet. Il demande aux élèves d'expliquer avec précision ce qui peut éventuellement les bloquer ou ce qu'il leur reste encore à faire. Il encourage les interactions entre les élèves. Il veille à ce que les élèves soient **précis dans leurs demandes.** Ils doivent indiquer exactement quelle partie de leur programme ne fonctionne pas, ce qu'ils souhaitent réaliser sans y parvenir etc.	*Dans cette tâche, on essaie de faire travailler la compétence d'**abstraction** des élèves (compétence souvent associée à la maîtrise de la pensée informatique). Les élèves doivent être capables d'**isoler ce qui est pertinent par rapport à une tâche** et donc de laisser tomber le reste.*

				Ici, ils doivent isoler le problème précis qui les bloque, l'étape de programmation qu'ils ne parviennent pas à réaliser...
15 min	**Réalisation du projet 3**	Les élèves **terminent la réalisation de leurs projets.** Ils sont prévenus qu'ils n'auront pas de temps supplémentaire pour aboutir leur travail.	L'enseignant s'assure que les groupes finalisent leurs projets. Si certains groupes finissent prématurément, il peut les inciter à aller plus loin en **améliorant certaines parties du programme** ou en rajoutant de nouvelles étapes de programmation.	*Idéalement, les projets devraient tous être terminés à la fin de cette séance. Si ce n'est pas le cas, alors les élèves présenteront **une version incomplète** de leur programme. L'enseignant est libre de leur laisser bénéficier d'un peu de temps supplémentaire lors de la prochaine séance mais il faut alors veiller à bien gérer le timing de cette prochaine séance pour que toutes les activités proposées puissent être réalisées.*

Séance 9	Présentation des projets	Durée : 45 min	Condition débranchée Condition Scratch Condition Scratch + Robots

Résumé :

- *Les élèves présentent leur projet à la classe.*
- *En fin de séance, l'enseignant incite les élèves à comprendre que les programmes qu'ils ont réalisés peuvent être généralisés à d'autres situations similaires.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
35-40 min	**Présentation des projets à la classe**	Un élève de chaque groupe **présente le projet** qu'ils ont réalisé. Il explique tout d'abord ce qu'ils ont voulu faire (potentiellement en relisant ce qu'ils avaient écrit lors de la première séance dédiée au projet). Ensuite, **un élève issu d'un autre groupe exécute le programme.**	L'enseignant demande aux élèves de venir présenter leur projet à la classe. Il veille à ce que chaque présentation dure entre 5 et 10 min maximum pour que tout le monde puisse passer dans le temps imparti. Lorsque la présentation été faîte, il peut demander aux élèves (en fonction du temps disponible) ce qu'ils pourraient proposer pour **améliorer leur programme**, les **difficultés qu'ils ont rencontrées**...	*Il suffit de **lancer le programme** pour en faire la démonstration.*
5-10 min	**Réflexions sur la notion de généralisation**	Les élèves réfléchissent ensemble et répondent aux questions posées par l'enseignant. Ils appréhendent la notion de **généralisation.** Une solution à un problème particulier (en l'occurrence,	L'enseignant demande aux élèves : « Si par exemple le robot / lutin Scratch / personnage que vous avez programmé était placé initialement à un autre endroit, est-ce que le reste du programme que vous avez réalisé fonctionnerait correctement ? ». La réponse est évidemment non. Les déplacements du robot / lutin /	*Il n'est pas forcément nécessaire d'insister sur le terme « généralisation ». Les élèves n'ont pas à le connaître, ils doivent juste comprendre **ce que cela implique.***

		ici, le projet qu'ils souhaitaient réaliser) peut souvent être généralisée à un ensemble de problèmes qui sont similaires.	personnage ne seraient plus appropriés. Alors, l'enseignant demande : « Donc on est obligé de repartir de 0 ? » Normalement, la réponse des élèves devrait aussi être non. Ce n'est pas parce qu'on change un seul paramètre d'un programme que tout doit être jeté à la poubelle. En réalité, l'enseignant explique qu'un programme est une solution à un problème (ex : je veux que mon robot se déplace de telle manière). Mais la solution à un problème peut servir à résoudre tout un éventail de problèmes similaires. Ainsi, les programmes qu'ils ont réalisés peuvent être utiles pour programmer n'importe quel **tour du monde réalisé par un personnage** (condition débranchée), pour n'importe quel **parcours d'un robot** dans un ensemble de rues ou dans un labyrinthe (condition Scratch + Robot) ou pour n'importe quelle **présentation d'une école** sur Scratch (condition Scratch). C'est ce que l'on appelle **la généralisation.**	

Séance 10	Évaluations	Durée : 1 h (20 + 25 + 15min)	Condition débranchée Condition Scratch Condition Scratch + Robots

Résumé :

- *Cette dernière séance est consacrée à des tests permettant notamment d'évaluer et de mesurer les performances des élèves.*
- *Il est nécessaire, pour ne pas surcharger les élèves, de scinder cette séance en 3 parties (une pour chaque test). Les 3 tests doivent être réalisés sur un délai de 2 jours maximum.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	***Remarques***
20 min	**Test de connaissances et de compréhension des notions de programmation**	L'élève doit **répondre à un ensemble de questions** (une dizaine) qui portent sur les notions d'algorithme, d'instruction, de boucle, de condition et de variable.	Veiller à ce que le test soit effectué **individuellement** par la totalité des élèves.	*Le test sera fourni aux enseignants au format papier dans la condition débranchée et au format numérique dans les deux autres conditions.*
25 min (Différée dans le temps)	**Test de maîtrise de la pensée informatique.**	L'élève doit modifier ou créer des programmes pour **résoudre des problèmes** permettant d'évaluer les compétences associées à la pensée informatique (généralisation, abstraction, reconnaissance de formes, décomposition, algorithmique).	Veiller à ce que le test soit effectué **individuellement** par la totalité des élèves.	*Le test sera fourni aux enseignants au format papier dans la condition débranchée et au format numérique dans les deux autres conditions.* *Cette partie sera différée pour ne pas surcharger inutilement les élèves.*

15 min (Différée dans le temps)	**Évaluation de la motivation, du sentiment d'auto-efficacité et de l'intérêt pour les sciences des élèves**	**Répondre au post-test** au format papier pour estimer : - le niveau de motivation des élèves : "Est-ce que j'aime apprendre la programmation ?" - le sentiment d'auto-efficacité : "Est-ce que je me sens capable de réaliser des tâches de programmation ?" - le niveau d'intérêt pour les disciplines scientifiques : "Est-ce que j'aime les sciences ? »	Veiller à ce que le test soit effectué **individuellement** par la totalité des élèves.	*Plusieurs affirmations seront présentées aux élèves qui devront notamment exprimer leur degré d'accord ou de désaccord avec ces affirmations sur une échelle de réponses prédéfinies. Cette partie sera différée pour ne pas surcharger inutilement les élèves.*

Annexe H

Prénom : ..... Nom de famille : .....

Date : .....

Classe : ..... École : ….....

Première partie :

*Pour chacune des phrases numérotées, indique si elle correspond à quelque chose que tu ne fais **quasiment jamais**, que tu fais **parfois**, que tu fais **souvent** ou que tu fais **presque toujours**. Entoure la réponse qui te correspond.*
Sois le plus honnête possible, personne ne te jugera !

1. **Je suis certain(e) de pouvoir comprendre les choses les plus difficiles présentées en classe.**

Quasiment jamais / Parfois / Souvent / Presque toujours

2. **Quand je veux apprendre quelque chose de vraiment difficile, je peux l'apprendre.**

Quasiment jamais / Parfois / Souvent / Presque toujours

3. **Si je décide de ne pas avoir de mauvaises notes, je peux vraiment le faire.**

Quasiment jamais / Parfois / Souvent / Presque toujours

4. **Si je décide de ne pas me tromper, je peux vraiment le faire.**

Quasiment jamais / Parfois / Souvent / Presque toujours

5. **Si je veux bien apprendre quelque chose, je le peux.**

Quasiment jamais / Parfois / Souvent / Presque toujours

6. **Je suis certain(e) de pouvoir maîtriser les compétences enseignées en classe.**

Quasiment jamais / Parfois / Souvent / Presque toujours

Deuxième partie :

*Pour chacune des phrases numérotées, indique si tu n'es **pas du tout d'accord**, si tu n'es **pas vraiment d'accord**, si tu es **plutôt d'accord** ou si tu es **totalement d'accord.** Entoure la réponse qui te correspond.*
Sois le plus honnête possible, personne ne te jugera !

1. **Je suis certain(e) que je peux comprendre les choses les plus difficiles présentées en cours de programmation.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

2. **Je suis convaincu(e) que je pourrai faire un excellent travail sur les devoirs et les tests en programmation.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

3. **J'apprends rapidement dans la plupart des matières scolaires.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

4. **Je suis certain(e) que je peux maîtriser les compétences enseignées en programmation.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

5. **Je ne pense pas pouvoir réussir en programmation.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

6. **La programmation est une matière amusante, j'aimerais continuer à en faire.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

7. **Je suis fort(e) dans la plupart des matières scolaires.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

8. **J'apprends vite en cours de programmation.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

9. **Je pense pouvoir avoir de bonnes notes en programmation.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

10. **La programmation est une de mes matières préférées.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

11. **J'obtiens de bons résultats aux tests dans la plupart des matières scolaires.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

12. **La programmation est une matière importante pour moi.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

13. **Quand je fais de la programmation, je suis parfois totalement absorbé par ce que je fais.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

14. **J'aimerais faire de la programmation sur mon temps libre.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

Annexe I

Prénom : ..... Nom de famille : .....

Date : .....

Classe : ..... École : ….....

*Pour chacune des phrases numérotées, indique si tu n'es **pas du tout d'accord**, si tu n'es **pas vraiment d'accord**, si tu es **plutôt d'accord** ou si tu es **totalement d'accord.** Entoure la réponse qui te correspond.*
Sois le plus honnête possible, personne ne te jugera !

1. **J'apprends des choses intéressantes en cours de science.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

2. **J'attends avec impatience mes cours de sciences.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

3. **J'aimerais faire plus de science à l'école.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

4. **Je préfère la science à la plupart des autres matières à l'école.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

5. **La science est ennuyeuse.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

7. **Je trouve la science difficile.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

8. **Je ne suis pas doué pour les sciences.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

9. **J'ai de bonnes notes en sciences.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

10. **J'apprends rapidement les sciences.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

11. **La science est l'une de mes matières préférées.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

12. **Même en faisant des efforts, je n'arrive pas à comprendre les sciences.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

13. **Quand j'entends le mot « Science », je me sens mal à l'aise ou tendu(e).**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

14. **J'ai peur des cours de sciences.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

15. **C'est important pour moi de comprendre ce que je fais en cours de sciences.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

16. **J'aimerais étudier davantage la science à l'avenir.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

17. **J'aimerais devenir professeur de sciences.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

18. **J'aimerais devenir un scientifique.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

19. **La science est utile pour résoudre les problèmes de la vie quotidienne.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

20. **La plupart des gens devraient étudier la science.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

21. **La science est utile pour comprendre le monde.**

Pas du tout d'accord / Pas vraiment d'accord / Plutôt d'accord / Totalement d'accord

Annexe J

1 séance	Initiation à la programmation	Durée : 1h30	Condition Scratch 1

Résumé :

- *Les élèves découvrent les notions d'algorithme (ou de programme), d'instruction et de boucle.*
- *Ils apprennent à programmer les déplacements d'une « sprite » (personnage) sur Scratch 3.0.*
- *L'objectif de cette étude est de comparer une séance d'initiation à la programmation réalisée sur Scratch avec d'autres modalités d'enseignement.*
- *Il est préférable que cette séance soit scindée en deux parties rapprochées. Par exemple, la dernière activité (Evaluation, 30 min) peut être réalisée le lendemain.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
15 min	**Découverte des notions d'algorithme et d'instruction.**	**Objectif :** Par la conception d'une recette de cuisine, les élèves **découvrent ce qu'est un algorithme**. Ils prennent conscience de la présence des algorithmes partout autour de nous.		
		Consignes (en collectif) : « Quels sont les ingrédients pour faire des crêpes ? » « Dans quel ordre ajoute-t-on les ingrédients pour réaliser cette recette ? » « Quelqu'un saurait expliquer ce qu'est **un algorithme** ? »	L'enseignant **demande aux élèves de donner les ingrédients nécessaires** pour réaliser une pâte à crêpes et les note aux tableaux. Il leur demande ensuite de **formuler les actions à réaliser dans le bon ordre**, comme si on voulait donner la recette à quelqu'un qui ne sait pas faire des crêpes, en commençant par un verbe à l'infinitif (ex : « Mettre la farine dans un récipient »).	*Pour la dernière partie de cette activité qui consiste à expliquer l'importance des algorithmes dans les objets qui nous entourent, l'objectif est de susciter l'intérêt et la motivation des élèves.* *Les définitions des mots « algorithme » et « instruction » doivent être prises en notes par les élèves.*

			Une fois la recette écrite au tableau, l'enseignant explique aux élèves qu'ils ont réalisé ce que l'on appelle un algorithme. *« Un algorithme est une suite d'instructions présentées dans un certain ordre qui permettent de résoudre un problème (ici, comment faire des crêpes ?) ».* *« Une instruction est une action (ou un ordre) à exécuter. »* **Il leur présente l'intérêt des algorithmes** : toutes les machines qui nous entourent obéissent à des programmes, c'est-à-dire à des algorithmes qui ont été **traduits dans un langage de programmation** compréhensible par une machine. Il donne des exemples de choses qui ne pourraient pas exister sans les algorithmes : les robots, les jeux vidéo, les ordinateurs...	
10 min	**Découverte de Scratch**	**Objectif :** Les élèves apprennent à **ouvrir un fichier Scratch** (soit en double-cliquant dessus, soit en ouvrant le fichier depuis le logiciel Scratch) et à **assembler des blocs** pour constituer un programme. Ils apprennent également à **sauvegarder** un programme.		
		"Ouvrez le fichier scratch qui est sur votre ordinateur Je vais vous présenter **les différents espaces**"	L'enseignant demande aux élèves **d'ouvrir le fichier « Scratch 1 »,** qui aura préalablement été déposé par l'enseignant sur l'ordinateur de l'élève. Il leur présente le	*Nous avons utilisé des blocs préprogrammés pour faciliter la tâche des élèves. Ainsi, ils n'ont pas à estimer eux-mêmes la « taille » d'une case*

		«Vous voyez ces deux instructions ? **Vous ne devez pas les supprimer !** »	fonctionnement de Scratch. A gauche, il y a les « blocs » (les différentes instructions que l'on peut utiliser pour constituer un programme) qui sont classées par catégories. L'enseignant demande de **cliquer sur la catégorie « Mes blocs »** qui contient les blocs préprogrammés pour cette séance car ce sont les seuls blocs qui vont être utilisés pour le moment. Au centre, il y a la partie « code » où les élèves peuvent assembler leurs instructions. Tout en haut, deux blocs sont déjà présents :  « Quand drapeau vert est cliqué » permet de lancer le programme en cliquant sur le drapeau vert en haut à droite. « Repartir du début » est un bloc préprogrammé qui permet de replacer le personnage à programmer au début du parcours. Les élèves **ne doivent pas supprimer ces deux**	*lorsque le personnage à programmer devra se déplacer de case en case par exemple. Cependant, à l'ouverture du programme, les élèves risquent de tomber sur des blocs « définir » au centre de l'écran qui nous ont permis de préprogrammer nos blocs. Ces blocs « définir » doivent être ignorés. L'enseignant doit demander aux élèves d'utiliser la barre de défilement (à droite de la partie centrale de l'écran) pour remonter tout en haut de la partie « code ».* *Il est également essentiel de bien demander aux élèves de constituer des programmes en assemblant les blocs, puis en cliquant sur le drapeau vert. Certains pourraient être tentés de cliquer directement sur les blocs pour les exécuter un par un mais cette façon de procéder est à proscrire car elle ne permet pas aux élèves de travailler la logique algorithmique !* *Enfin, il faut également préciser que pour supprimer une instruction, il suffit de la*

		« Vous pouvez maintenant utiliser les différentes instructions présentes dans « Mes blocs » pour **créer un programme** ». Vous devez **sauvegarder votre programme** en indiquant nom et prénom à la suite de « Scratch 1 ».	**instructions !** En haut à droite, il y a un quadrillage qui permet de visualiser les déplacements du personnage à programmer. La partie en bas à droite est à ignorer. Après cette courte présentation, **l'enseignant incite les élèves à utiliser les différentes instructions** présentes dans « Mes blocs » pour créer un programme permettant à l'abeille d'aller butiner une fleur sur son chemin, en faisant glisser les blocs depuis la gauche vers le centre, en les emboîtant les uns avec les autres et en exécutant le programme (clic sur le drapeau vert). L'enseignant demande enfin à chaque élève de **sauvegarder le programme** en renseignant leur nom et prénom à la suite de « Scratch 1 ».	*faire glisser depuis le centre vers la gauche de l'écran (là où se trouvent tous les blocs). Si l'enseignant ne le montre pas, la question risque d'émerger rapidement.*
15 min	**Programmer le déplacement de l'abeille**	**Objectif :** Les élèves **apprennent à programmer les déplacements d'une abeille** pour qu'elle réalise le parcours demandé. Ils sont informés qu'ils doivent **travailler seul** sans être aidé par le professeur. A la fin, ils prennent conscience que **plusieurs solutions étaient possibles** pour résoudre ce problème.		
		« Vous allez devoir **guider l'abeille** pour qu'elle aille butiner 3 fleurs avant de	L'enseignant demande aux élèves de créer un programme pour guider les déplacements de	*L'expérience que nous réalisons propose de comparer différentes façons*

		retourner à sa ruche » « Vous devez **sauvegarder votre travail** lorsque vous avez fini. » « Voici les programmes que vous avez réalisés, que remarquez-vous ? »	l'abeille afin qu'elle aille butiner les trois fleurs **avant** de rentrer à la ruche (située en bas à droite du quadrillage). Il informe les élèves qu'ils **doivent travailler en autonomie** et qu'il ne peut les aider que s'ils ont un problème pour utiliser le logiciel correctement. Il leur demande de **sauvegarder** leur travail lorsqu'ils ont fini l'activité. Lorsque tout le monde a terminé, l'enseignant montre aux élèves qu'il existe **plusieurs chemins permettant de réaliser ce parcours.**	*d'enseigner la programmation et s'intéresse notamment au rôle du feedback sur les apprentissages. Ainsi, il est important pour nous de contrôler au maximum le feedback qui sera reçu par les élèves. Pour cette raison, il est essentiel que l'enseignant n'intervienne pas pour guider les élèves dans la réalisation de cette tâche. Il ne peut intervenir que pour répéter la consigne ou si une question technique est posée (Comment je peux supprimer un bloc ? Comment j'exécute le programme ? Comment je sauvegarde ? etc.).* *Si le matériel nécessaire est à sa disposition, l'enseignant peut projeter les écrans de plusieurs élèves pour montrer plusieurs solutions possibles. Sinon, il peut préparer lui-même plusieurs solutions qu'il projette à partir de son propre écran.* *Il est essentiel que chaque élève pense bien à sauvegarder son travail car les programmes seront*

				récupérés pour être analysés.
20 min	**Programmer le déplacement de l'abeille avec des boucles de répétition.**	**Objectif : réaliser un nouveau parcours** qui comporte plus de fleurs mais également des obstacles à éviter. Ils intègrent une ou plusieurs **boucle(s) pour répéter une partie de leur programme.**		
		Les élèves ouvrent le fichier Scratch 2. Phase 1 : « Vous allez devoir **guider l'abeille** pour qu'elle aille butiner 3 fleurs avant de retourner à sa ruche, attention, cette fois-ci, il y a **des obstacles** ! L'abeille doit éviter tout ce qui n'est pas une fleur ou sa ruche.» « Que peut-on remarquer entre le point de départ et le coquelicot (fleur rouge, 2ème colonne) et entre le coquelicot et l'héliopsis (fleur jaune, 3ème colonne) ? ». Phase 2 : Pour inciter les élèves à utiliser des boucles, l'enseignant leur demande de réaliser un **programme comportant le moins d'instructions possible.**	L'enseignant demande aux élèves d'ouvrir le fichier « Scratch 2 » et de **réaliser le nouveau parcours** (il faut toujours « récupérer » les fleurs puis se rendre à la ruche). Il attire leur attention sur les **obstacles** qui sont maintenant sur le chemin (frelon, guêpier, insecticides…). Au cours de son exploration auprès des élèves, l'enseignant leur fait éventuellement remarquer que certaines parties de leur programme se répètent. Si ce n'est pas le cas au bout d'un certain temps, il leur indique que les déplacements de l'abeille entre le point de départ et le coquelicot et le déplacement entre le Coquelicot et l'Héliopsis sont **les mêmes** (Avancer 1 case – Droite – Avancer 2 cases – Gauche). Il incite les élèves à utiliser une **boucle de répétition** dans ce cas. Il est possible d'utiliser l'instruction « Répéter 2 fois » pour simplifier et aller plus vite. L'instruction	*Une confusion sur la notion de répétition est possible. Pour effectuer un même déplacement 2 fois, les élèves peuvent avoir tendance à utiliser l'instruction « Répéter 1 fois ». Dans le langage courant, la notion de répétition implique déjà qu'une chose est réalisée plusieurs fois. Ici, c'est bien **« Répéter 2 fois »** qu'il faut utiliser car cette instruction sert à signifier combien de fois les instructions présentes dans la boucle doivent être exécutées.* *Lorsque l'abeille passe sur un obstacle, un message d'erreur apparaît. Cela signifie que le parcours n'est pas réalisé correctement.* *La définition d'une boucle doit être prise en note par les élèves.* *Comme pour l'activité précédente, les élèves doivent*

			« Répéter » est placée avant ce qui doit être répété. L'enseignant montre aux élèves comment utiliser les boucles (l'instruction « répéter … fois » est un bloc à compléter, présent dans la catégorie « Contrôle »). *« Une boucle est une instruction qui permet de répéter plusieurs fois un ensemble d'instructions ».* Vu que le parcours à réaliser est plus long et plus complexe, il incite les élèves à **utiliser des boucles aussi souvent que possible**, en repérant les déplacements qui se répètent, pour que le programme ne soit pas trop long.	*travailler en autonomie et bien sauvegarder leur travail.*
30 min	**Evaluations**	Les élèves sont évalués par le biais des **trois tests** suivants au format papier : - Test de compréhension des notions de programmation - Test de résolution de problèmes - Test de motivation et d'auto-efficacité en programmation	L'enseignant donne trois tests à aux élèves. Il ne peut pas les aider mais il peut **reformuler** ou répéter les consignes si les élèves en font la demande.	*Les tests sont réalisés INDIVIDUELLEMENT par chaque élève.*

1 séance	Initiation à la programmation	Durée : 1h30	Condition Scratch 2

Résumé :

- *Les élèves découvrent les notions d'algorithme (ou de programme), d'instruction et de boucle.*
- *Ils apprennent à programmer en utilisant sur Scratch 3.0, mais sans pouvoir visualiser l'exécution du programme.*
- *L'objectif de cette étude est de comparer une séance d'initiation à la programmation réalisée sur Scratch (avec ou sans visualisation de l'exécution du programme) avec d'autres modalités d'enseignement.*
- *Il est préférable que cette séance soit scindée en deux parties rapprochées. Par exemple, la dernière activité (Evaluation, 30 min) peut être réalisée le lendemain.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
15 min	**Découverte des notions d'algorithme et d'instruction.**	**Objectif :** Par la conception d'une recette de cuisine, les élèves **découvrent ce qu'est un algorithme**. Ils prennent conscience de la présence des algorithmes partout autour de nous.		
		Consignes (en collectif) : « Quels sont les ingrédients pour faire des crêpes ? » « Dans quel ordre ajoute-t-on les ingrédients pour réaliser cette recette ? » « Quelqu'un saurait expliquer ce qu'est **un algorithme** ? »	L'enseignant **demande aux élèves de donner les ingrédients nécessaires** pour réaliser une pâte à crêpes et les note aux tableaux. Il leur demande ensuite de **formuler les actions à réaliser dans le bon ordre**, comme si on voulait donner la recette à quelqu'un qui ne sait pas faire des crêpes, en commençant par un verbe à l'infinitif (ex : « Mettre la farine dans un récipient »).	*Pour la dernière partie de cette activité qui consiste à expliquer l'importance des algorithmes dans les objets qui nous entourent, l'objectif est de susciter l'intérêt et la motivation des élèves.* *Les définitions des mots « algorithme » et « instruction » doivent être prises en notes par les élèves.*

			Une fois la recette écrite au tableau, l'enseignant explique aux élèves qu'ils ont réalisé ce que l'on appelle un algorithme. *« Un algorithme est une suite d'instructions présentées dans un certain ordre qui permettent de résoudre un problème (ici, comment faire des crêpes ?) ».* *« Une instruction est une action (ou un ordre) à exécuter. »* **Il leur présente l'intérêt des algorithmes** : toutes les machines qui nous entourent obéissent à des programmes, c'est-à-dire à des algorithmes qui ont été **traduits dans un langage de programmation** compréhensible par une machine. Il donne des exemples de choses qui ne pourraient pas exister sans les algorithmes : les robots, les jeux vidéo, les ordinateurs...	
10 min	**Découverte de Scratch**	**Objectif :** Les élèves apprennent à **ouvrir un fichier Scratch** (soit en double-cliquant dessus, soit en ouvrant le fichier depuis le logiciel Scratch) et à **assembler des blocs** pour constituer un programme. Ils apprennent également à **sauvegarder** un programme.		
		"Ouvrez le fichier scratch qui est sur votre ordinateur Je vais vous présenter **les différents espaces**"	L'enseignant demande aux élèves **d'ouvrir le fichier « Scratch A »,** qui aura préalablement été déposé par l'enseignant sur l'ordinateur de l'élève. Il leur présente le	*Nous avons utilisé des blocs préprogrammés pour faciliter la tâche des élèves. Ainsi, ils n'ont pas à estimer eux-mêmes la « taille » d'une case*

			fonctionnement de Scratch. A gauche, il y a les « blocs » (les différentes instructions que l'on peut utiliser pour constituer un programme) qui sont classées par catégories. L'enseignant demande de **cliquer sur la catégorie « Mes blocs »** qui contient les blocs préprogrammés pour cette séance car ce sont les seuls blocs qui vont être utilisés pour le moment. Au centre, il y a la partie « code » où les élèves peuvent assembler leurs instructions. Tout en haut, un bloc « Quand drapeau vert est cliqué » est déjà présent. Habituellement, il permet de lancer le programme mais ici, la visualisation de l'exécution du programme n'est pas possible car nous avons volontairement masqué l'abeille à programmer. Ce bloc est utilisé simplement pour servir de base à partir de laquelle les élèves vont pouvoir assembler leurs instructions. En haut à droite, il y a un quadrillage qui représente le parcours à réaliser. La partie en bas à droite est à ignorer. Après cette courte présentation, **l'enseignant incite les élèves à**	*lorsque le personnage à programmer devra se déplacer de case en case par exemple. Cependant, à l'ouverture du programme, les élèves risquent de tomber sur des blocs « définir » au centre de l'écran qui nous ont permis de préprogrammer nos blocs. Ces blocs « définir » doivent être ignorés. L'enseignant doit demander aux élèves d'utiliser la barre de défilement (à droite de la partie centrale de l'écran) pour remonter tout en haut de la partie « code ».* *Il est également essentiel de bien demander aux élèves de constituer des programmes en assemblant les blocs, puis en cliquant sur le drapeau vert.* *Enfin, il faut également préciser que pour supprimer une instruction, il suffit de la faire glisser depuis le centre vers la gauche de l'écran (là où se trouvent tous les blocs). Si l'enseignant ne le montre pas, la question risque d'émerger rapidement.*

		« Vous pouvez maintenant utiliser les différentes instructions présentes dans « Mes blocs » pour **créer un programme** ». Vous devez **sauvegarder votre programme** en indiquant nom et prénom à la suite de « Scratch 1 ».	**utiliser les différentes instructions** présentes dans « Mes blocs » pour créer un programme permettant à l'abeille d'aller butiner une fleur sur son chemin, en faisant glisser les blocs depuis la gauche vers le centre, en les emboîtant les uns avec les autres et en exécutant le programme (clic sur le drapeau vert). L'enseignant demande enfin à chaque élève de **sauvegarder le programme** en renseignant leur nom et prénom à la suite de « Scratch A ».	
15 min	**Programmer le déplacement de l'abeille**	**Objectif :** Les élèves **apprennent à programmer les déplacements d'une abeille** pour qu'elle réalise le parcours demandé. Ils sont informés qu'ils doivent **travailler seul** sans être aidé par le professeur. A la fin, ils prennent conscience que **plusieurs solutions étaient possibles** pour résoudre ce problème.		
		« Vous allez devoir **guider l'abeille** pour qu'elle aille butiner 3 fleurs avant de retourner à sa ruche »	L'enseignant demande aux élèves de créer un programme pour guider les déplacements de l'abeille afin qu'elle aille butiner les trois fleurs **avant** de rentrer à la ruche (située en bas à droite du quadrillage). Il informe les élèves qu'ils **doivent travailler en autonomie** et qu'il ne peut les aider que s'ils ont un problème pour utiliser le logiciel	*L'expérience que nous réalisons propose de comparer différentes façons d'enseigner la programmation et s'intéresse notamment au rôle du feedback sur les apprentissages. Ainsi, il est important pour nous de contrôler au maximum le feedback qui sera reçu par les élèves. Pour cette raison, il est*

		« Vous devez **sauvegarder votre travail** lorsque vous avez fini. » « Voici les programmes que vous avez réalisés, que remarquez-vous ? »	correctement. Il leur demande de **sauvegarder** leur travail lorsqu'ils ont fini l'activité. Lorsque tout le monde a terminé, l'enseignant montre aux élèves qu'il existe **plusieurs chemins permettant de réaliser ce parcours.**	*essentiel que l'enseignant n'intervienne pas pour guider les élèves dans la réalisation de cette tâche. Il ne peut intervenir que pour répéter la consigne ou si une question technique est posée (Comment je peux supprimer un bloc ? Comment j'exécute le programme ? Comment je sauvegarde ? etc.).* *Si le matériel nécessaire est à sa disposition, l'enseignant peut projeter les écrans de plusieurs élèves pour montrer plusieurs solutions possibles. Sinon, il peut préparer lui-même plusieurs solutions qu'il projette à partir de son propre écran.* *Il est essentiel que chaque élève pense bien à sauvegarder son travail car les programmes seront récupérés pour être analysés.*
20 min	**Programmer le déplacement de l'abeille avec des boucles de répétition.**	**Objectif : réaliser un nouveau parcours** qui comporte plus de fleurs mais également des obstacles à éviter. Ils intègrent une ou plusieurs **boucle(s) pour répéter une partie de leur programme.**		
		Les élèves ouvrent le fichier Scratch B.	L'enseignant demande aux élèves d'ouvrir le fichier « Scratch B » et de **réaliser le nouveau parcours**	*Une confusion sur la notion de répétition est possible. Pour effectuer un même*

		Phase 1 : « Vous allez devoir **guider l'abeille** pour qu'elle aille butiner 3 fleurs avant de retourner à sa ruche, attention, cette fois-ci, il y a **des obstacles** ! L'abeille doit éviter tout ce qui n'est pas une fleur ou sa ruche.» « Que peut-on remarquer entre le point de départ et le coquelicot (fleur rouge, 2ème colonne) et entre le coquelicot et l'héliopsis (fleur jaune, 3ème colonne) ? ». Phase 2 : Pour inciter les élèves à utiliser des boucles, l'enseignant leur demande de réaliser un **programme comportant le moins d'instructions possible.**	(il faut toujours « récupérer » les fleurs puis se rendre à la ruche). Il attire leur attention sur les **obstacles** qui sont maintenant sur le chemin (frelon, guêpier, insecticides…). Au cours de son exploration auprès des élèves, l'enseignant leur fait éventuellement remarquer que certaines parties de leur programme se répètent. Si ce n'est pas le cas au bout d'un certain temps, il leur indique que les déplacements de l'abeille entre le point de départ et le coquelicot et le déplacement entre le Coquelicot et l'Héliopsis sont **les mêmes** (Avancer 1 case – Droite – Avancer 2 cases – Gauche). Il incite les élèves à utiliser une **boucle de répétition** dans ce cas. Il est possible d'utiliser l'instruction « Répéter 2 fois » pour simplifier et aller plus vite. L'instruction « Répéter » est placée avant ce qui doit être répété. L'enseignant montre aux élèves comment utiliser les boucles (l'instruction « répéter … fois » est un bloc à compléter, présent dans la catégorie « Contrôle »). *« Une boucle est une instruction*	*déplacement 2 fois, les élèves peuvent avoir tendance à utiliser l'instruction « Répéter 1 fois ». Dans le langage courant, la notion de répétition implique déjà qu'une chose est réalisée plusieurs fois. Ici, c'est bien **« Répéter 2 fois »** qu'il faut utiliser car cette instruction sert à signifier combien de fois les instructions présentes dans la boucle doivent être exécutées.* *La définition d'une boucle doit être prise en note par les élèves.* *Comme pour l'activité précédente, les élèves doivent travailler en autonomie et bien sauvegarder leur travail.*

			qui permet de répéter plusieurs fois un ensemble d'instructions ». Vu que le parcours à réaliser est plus long et plus complexe, il incite les élèves à **utiliser des boucles aussi souvent que possible**, en repérant les déplacements qui se répètent, pour que le programme ne soit pas trop long.	
30 min	**Evaluations**	Les élèves sont évalués par le biais des **trois tests** suivants au format papier : - Test de compréhension des notions de programmation - Test de résolution de problèmes - Test de motivation et d'auto-efficacité en programmation	L'enseignant donne trois tests à aux élèves. Il ne peut pas les aider mais il peut **reformuler** ou répéter les consignes si les élèves en font la demande.	*Les tests sont réalisés INDIVIDUELLEMENT par chaque élève.*

Annexe K

1 séance	Initiation à la programmation	Durée : 1h30	Condition Débranchée

Résumé :

- *Les élèves découvrent les notions d'algorithme (ou de programme), d'instruction et de boucle.*
- *Ils apprennent à programmer en assemblant des instructions au format papier.*
- *L'objectif de cette étude est de comparer une séance d'initiation à la programmation réalisée en débranché, sans outil numérique, avec une séance réalisée à l'aide du logiciel Scratch.*
- *Il est préférable que cette séance soit scindée en deux parties rapprochées. Par exemple, la dernière activité (Evaluation, 30 min) peut être réalisée le lendemain.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
15 min	**Découverte des notions d'algorithme et d'instruction.**	**Objectif :** Par la conception d'une recette de cuisine, les élèves **découvrent ce qu'est un algorithme**. Ils prennent conscience de la présence des algorithmes partout autour de nous.		
		Consignes (en collectif) : « Quels sont les ingrédients pour faire des crêpes ? » « Dans quel ordre ajoute-t-on les ingrédients pour réaliser cette recette ? » « Quelqu'un saurait expliquer ce qu'est **un algorithme ?** »	L'enseignant **demande aux élèves de donner les ingrédients nécessaires** pour réaliser une pâte à crêpes et les note aux tableaux. Il leur demande ensuite de **formuler les actions à réaliser dans le bon ordre**, comme si on voulait donner la recette à quelqu'un qui ne sait pas faire des crêpes, en commençant par un verbe à l'infinitif (ex : « Mettre la farine dans un récipient »).	*Pour la dernière partie de cette activité qui consiste à expliquer l'importance des algorithmes dans les objets qui nous entourent, l'objectif est de susciter l'intérêt et la motivation des élèves.* *Les définitions des mots « algorithme » et « instruction » doivent être prises en notes par les élèves.*

			Une fois la recette écrite au tableau, l'enseignant explique aux élèves qu'ils ont réalisé ce que l'on appelle un algorithme. *« Un algorithme est une suite d'instructions présentées dans un certain ordre qui permettent de résoudre un problème (ici, comment faire des crêpes ?) ».* *« Une instruction est une action (ou un ordre) à exécuter. »* **Il leur présente l'intérêt des algorithmes** : toutes les machines qui nous entourent obéissent à des programmes, c'est-à-dire à des algorithmes qui ont été **traduits dans un langage de programmation** compréhensible par une machine. Il donne des exemples de choses qui ne pourraient pas exister sans les algorithmes : les robots, les jeux vidéo, les ordinateurs...	
10 min	**Découverte de la programmation débranchée**	**Objectif :** Les élèves apprennent à **assembler des blocs** au format papier pour constituer un algorithme.		
		« Vous allez maintenant jouer au **jeu du robot** en utilisant les instructions que vous avez à disposition »	L'enseignant fournit aux élèves les **instructions (blocs) au format papier** présentes sur le document « Instruction.pdf » (sauf les boucles « Répéter » qui seront utilisées plus tard). Il leur montre	*L'objectif ici n'est pas de produire un algorithme pour résoudre un problème donné mais simplement de permettre à l'élève de comprendre comment il peut réaliser des*

			comment construire un algorithme en les mettant à la suite les unes après les autres. Il demande aux élèves de **jouer au « jeu du robot »** en petits groupes. Un élève du groupe assemble des instructions pour former un algorithme et lis chaque instruction à haute voix pour qu'un autre élève puisse jouer le rôle du « robot idiot » chargé d'exécuter les ordres qui lui sont donnés pour se déplacer dans la classe	*algorithmes à partir d'un petit nombre d'instructions qui sont mises à sa disposition.* *Les instructions seront fournies aux enseignants découpées et plastifiées avant le début de la séance.*
15 min	**Programmer le déplacement de l'abeille**	**Objectif :** Les élèves **apprennent à programmer les déplacements d'une abeille** pour qu'elle réalise le parcours demandé. Ils sont informés qu'ils doivent **travailler seul** sans être aidé par le l'enseignant. A la fin, ils prennent conscience que **plusieurs solutions étaient possibles** pour résoudre ce problème.		
		« Vous allez devoir **guider l'abeille** pour qu'elle aille butiner 3 fleurs avant de retourner à sa ruche » « Voici les algorithmes que vous avez trouvés, que remarquez-vous ? »	L'enseignant fournit aux élèves le **document « Parcours 1.jpg »** préalablement imprimé et plastifié. Il demande aux élèves de créer un algorithme à partir des instructions utilisées dans l'activité précédente (« Instructions.pdf ») pour guider les déplacements de l'abeille afin qu'elle aille butiner les trois fleurs **avant** de rentrer à la ruche (située en bas à droite du quadrillage). Il informe les élèves qu'ils **doivent travailler en autonomie** et qu'il ne peut les aider que s'ils ont un	*L'expérience que nous réalisons propose de comparer différentes façons d'enseigner la programmation et s'intéresse notamment au rôle du feedback sur les apprentissages. Ainsi, il est important pour nous de contrôler au maximum le feedback qui sera reçu par les élèves. Pour cette raison, il est essentiel que l'enseignant n'intervienne pas pour guider les élèves dans la réalisation*

			problème de compréhension de la consigne. A la fin de l'activité, l'enseignant **prend en photo les algorithmes** qui ont été réalisés par ses élèves et les affiche au tableau pour qu'ils puissent commenter. Il leur fait remarquer que **plusieurs solutions étaient possibles** pour réaliser ce parcours.	*de cette tâche. Il ne peut intervenir que pour répéter la consigne.* *Si le matériel nécessaire est à sa disposition, l'enseignant peut projeter les photos des algorithmes proposés par les élèves pour montrer plusieurs solutions possibles. Sinon, il peut préparer lui-même plusieurs solutions qu'il projette à partir de son propre écran.* *Il est essentiel de récupérer les photos des algorithmes des élèves car ceux-ci seront analysés ultérieurement.*
20 min	**Programmer le déplacement de l'abeille avec des boucles de répétition.**	**Objectif** *:* Les élèves doivent **réaliser un nouveau parcours** qui comporte plus de fleurs mais également des obstacles à éviter. Ils intègrent une ou plusieurs **boucle(s) pour répéter une partie de leur programme.**		
		Phase 1 : « Vous allez devoir **guider l'abeille** pour qu'elle aille butiner 3 fleurs avant de retourner à sa ruche, attention, cette fois-ci, il y a **des obstacles** ! L'abeille doit éviter tout ce qui n'est pas une fleur ou sa ruche.» « Que peut-on remarquer entre le point de départ et le	L'enseignant donne aux élèves le document « parcours 2.jpg », préalablement imprimé et plastifié et leur demande de **réaliser le nouveau parcours** (il faut toujours « récupérer » les fleurs puis se rendre à la ruche). Il attire leur attention sur les **obstacles** qui sont maintenant sur le chemin (frelon, guêpier, insecticides…). Au cours de son exploration	*Une confusion sur la notion de répétition est possible. Pour effectuer un même déplacement 2 fois, les élèves peuvent avoir tendance à utiliser l'instruction « Répéter 1 fois ». Dans le langage courant, la notion de répétition implique déjà qu'une chose est réalisée plusieurs fois. Ici, c'est bien* ***« Répéter 2 fois »***

		coquelicot (fleur rouge, 2ème colonne) et entre le coquelicot et l'héliopsis (fleur jaune, 3ème colonne) ? ». Phase 2 : Pour inciter les élèves à utiliser des boucles, l'enseignant leur demande de réaliser un **programme comportant le moins d'instructions possible.**	auprès des élèves, l'enseignant leur fait éventuellement remarquer que certaines parties de leur programme se répètent. Si ce n'est pas le cas au bout d'un certain temps, il leur indique que les déplacements de l'abeille entre le point de départ et le coquelicot et le déplacement entre le Coquelicot et l'Héliopsis sont **les mêmes** (Avancer 1 case – Droite – Avancer 2 cases – Gauche). Il incite les élèves à utiliser une **boucle de répétition** dans ce cas et leur donne les instructions correspondantes (voir « Instructions.pdf »). Il est possible d'utiliser l'instruction « Répéter 2 fois » pour simplifier et aller plus vite. L'instruction « Répéter » est placée avant ce qui doit être répété. L'enseignant **montre aux élèves comment utiliser les boucles**. *« Une boucle est une instruction qui permet de répéter plusieurs fois un ensemble d'instructions ».* Vu que le parcours à réaliser est plus long et plus complexe, il incite les élèves à **utiliser des boucles aussi souvent que possible**, en repérant les déplacements qui se	*qu'il faut utiliser car cette instruction sert à signifier combien de fois les instructions présentes dans la boucle doivent être exécutées.* *Pour que le contenu de la boucle soit bien délimité, il est nécessaire de décaler les instructions à répéter d'un cran vers la droite. Voir l'exemple ci-dessous (le contenu de la boucle est en rouge) :* *Avancer de 2 cases* *Répéter 4 fois* *Avancer de 2 cases* *Pivoter à droite* *Pivoter à gauche* *Comme pour l'activité précédente, les élèves doivent travailler en autonomie et leur nouvel algorithme doit être pris en photo par l'enseignant.*

			répètent, pour que le programme ne soit pas trop long.	
30 min	**Evaluations**	Les élèves sont évalués par le biais des **trois tests** suivants au format papier : - Test de compréhension des notions de programmation - Test de résolution de problèmes - Test de motivation et d'auto-efficacité en programmation	L'enseignant donne trois tests aux élèves. Il ne peut pas les aider mais il peut **reformuler** ou répéter les consignes si les élèves en font la demande.	*Les tests sont réalisés INDIVIDUELLEMENT par chaque élève.*

1 séance	Initiation à la programmation	Durée : 1h30	Condition Scratch 2

Résumé :

- *Les élèves découvrent les notions d'algorithme (ou de programme), d'instruction et de boucle.*
- *Ils apprennent à programmer en utilisant sur Scratch 3.0, mais sans pouvoir visualiser l'exécution du programme.*
- *L'objectif de cette étude est de comparer une séance d'initiation à la programmation réalisée sur Scratch (avec ou sans visualisation de l'exécution du programme) avec d'autres modalités d'enseignement.*
- *Il est préférable que cette séance soit scindée en deux parties rapprochées. Par exemple, la dernière activité (Evaluation, 30 min) peut être réalisée le lendemain.*

Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
15 min	**Découverte des notions d'algorithme et d'instruction.**	**Objectif :** Par la conception d'une recette de cuisine, les élèves **découvrent ce qu'est un algorithme**. Ils prennent conscience de la présence des algorithmes partout autour de nous.		
		Consignes (en collectif) : « Quels sont les ingrédients pour faire des crêpes ? » « Dans quel ordre ajoute-t-on les ingrédients pour réaliser cette recette ? » « Quelqu'un saurait expliquer ce qu'est **un algorithme** ? »	L'enseignant **demande aux élèves de donner les ingrédients nécessaires** pour réaliser une pâte à crêpes et les note aux tableaux. Il leur demande ensuite de **formuler les actions à réaliser dans le bon ordre**, comme si on voulait donner la recette à quelqu'un qui ne sait pas faire des crêpes, en commençant par un verbe à l'infinitif (ex : « Mettre la farine dans un récipient »).	*Pour la dernière partie de cette activité qui consiste à expliquer l'importance des algorithmes dans les objets qui nous entourent, l'objectif est de susciter l'intérêt et la motivation des élèves.* *Les définitions des mots « algorithme » et « instruction » doivent être prises en notes par les élèves.*

			Une fois la recette écrite au tableau, l'enseignant explique aux élèves qu'ils ont réalisé ce que l'on appelle un algorithme. *« Un algorithme est une suite d'instructions présentées dans un certain ordre qui permettent de résoudre un problème (ici, comment faire des crêpes ?) ».* *« Une instruction est une action (ou un ordre) à exécuter. »* **Il leur présente l'intérêt des algorithmes** : toutes les machines qui nous entourent obéissent à des programmes, c'est-à-dire à des algorithmes qui ont été **traduits dans un langage de programmation** compréhensible par une machine. Il donne des exemples de choses qui ne pourraient pas exister sans les algorithmes : les robots, les jeux vidéo, les ordinateurs...	
10 min	**Découverte de Scratch**	**Objectif :** Les élèves apprennent à **ouvrir un fichier Scratch** (soit en double-cliquant dessus, soit en ouvrant le fichier depuis le logiciel Scratch) et à **assembler des blocs** pour constituer un programme. Ils apprennent également à **sauvegarder** un programme.		
		"Ouvrez le fichier scratch qui est sur votre ordinateur Je vais vous présenter **les différents espaces**"	L'enseignant demande aux élèves **d'ouvrir le fichier « Scratch A »,** qui aura préalablement été déposé par l'enseignant sur l'ordinateur de l'élève. Il leur présente le	*Nous avons utilisé des blocs préprogrammés pour faciliter la tâche des élèves. Ainsi, ils n'ont pas à estimer eux-mêmes la « taille » d'une case*

			fonctionnement de Scratch. A gauche, il y a les « blocs » (les différentes instructions que l'on peut utiliser pour constituer un programme) qui sont classées par catégories. L'enseignant demande de **cliquer sur la catégorie « Mes blocs »** qui contient les blocs préprogrammés pour cette séance car ce sont les seuls blocs qui vont être utilisés pour le moment. Au centre, il y a la partie « code » où les élèves peuvent assembler leurs instructions. Tout en haut, un bloc « Quand drapeau vert est cliqué » est déjà présent. Habituellement, il permet de lancer le programme mais ici, la visualisation de l'exécution du programme n'est pas possible car nous avons volontairement masqué l'abeille à programmer. Ce bloc est utilisé simplement pour servir de base à partir de laquelle les élèves vont pouvoir assembler leurs instructions. En haut à droite, il y a un quadrillage qui représente le parcours à réaliser. La partie en bas à droite est à ignorer. Après cette courte présentation, **l'enseignant incite les élèves à**	*lorsque le personnage à programmer devra se déplacer de case en case par exemple. Cependant, à l'ouverture du programme, les élèves risquent de tomber sur des blocs « définir » au centre de l'écran qui nous ont permis de préprogrammer nos blocs. Ces blocs « définir » doivent être ignorés. L'enseignant doit demander aux élèves d'utiliser la barre de défilement (à droite de la partie centrale de l'écran) pour remonter tout en haut de la partie « code ».* *Il est également essentiel de bien demander aux élèves de constituer des programmes en assemblant les blocs, puis en cliquant sur le drapeau vert.* *Enfin, il faut également préciser que pour supprimer une instruction, il suffit de la faire glisser depuis le centre vers la gauche de l'écran (là où se trouvent tous les blocs). Si l'enseignant ne le montre pas, la question risque d'émerger rapidement.*

		« Vous pouvez maintenant utiliser les différentes instructions présentes dans « Mes blocs » pour **créer un programme** ». Vous devez **sauvegarder votre programme** en indiquant nom et prénom à la suite de « Scratch 1 ».	**utiliser les différentes instructions** présentes dans « Mes blocs » pour créer un programme permettant à l'abeille d'aller butiner une fleur sur son chemin, en faisant glisser les blocs depuis la gauche vers le centre, en les emboîtant les uns avec les autres et en exécutant le programme (clic sur le drapeau vert). L'enseignant demande enfin à chaque élève de **sauvegarder le programme** en renseignant leur nom et prénom à la suite de « Scratch A ».	
15 min	**Programmer le déplacement de l'abeille**	**Objectif :** Les élèves **apprennent à programmer les déplacements d'une abeille** pour qu'elle réalise le parcours demandé. Ils sont informés qu'ils doivent **travailler seul** sans être aidé par le professeur. A la fin, ils prennent conscience que **plusieurs solutions étaient possibles** pour résoudre ce problème.		
		« Vous allez devoir **guider l'abeille** pour qu'elle aille butiner 3 fleurs avant de retourner à sa ruche »	L'enseignant demande aux élèves de créer un programme pour guider les déplacements de l'abeille afin qu'elle aille butiner les trois fleurs **avant** de rentrer à la ruche (située en bas à droite du quadrillage). Il informe les élèves qu'ils **doivent travailler en autonomie** et qu'il ne peut les aider que s'ils ont un problème pour utiliser le logiciel	*L'expérience que nous réalisons propose de comparer différentes façons d'enseigner la programmation et s'intéresse notamment au rôle du feedback sur les apprentissages. Ainsi, il est important pour nous de contrôler au maximum le feedback qui sera reçu par les élèves. Pour cette raison, il est*

		« Vous devez **sauvegarder votre travail** lorsque vous avez fini. » « Voici les programmes que vous avez réalisés, que remarquez-vous ? »	correctement. Il leur demande de **sauvegarder** leur travail lorsqu'ils ont fini l'activité. Lorsque tout le monde a terminé, l'enseignant montre aux élèves qu'il existe **plusieurs chemins permettant de réaliser ce parcours.**	*essentiel que l'enseignant n'intervienne pas pour guider les élèves dans la réalisation de cette tâche. Il ne peut intervenir que pour répéter la consigne ou si une question technique est posée (Comment je peux supprimer un bloc ? Comment j'exécute le programme ? Comment je sauvegarde ? etc.).* *Si le matériel nécessaire est à sa disposition, l'enseignant peut projeter les écrans de plusieurs élèves pour montrer plusieurs solutions possibles. Sinon, il peut préparer lui-même plusieurs solutions qu'il projette à partir de son propre écran.* *Il est essentiel que chaque élève pense bien à sauvegarder son travail car les programmes seront récupérés pour être analysés.*
20 min	**Programmer le déplacement de l'abeille avec des boucles de répétition.**	**Objectif : réaliser un nouveau parcours** qui comporte plus de fleurs mais également des obstacles à éviter. Ils intègrent une ou plusieurs **boucle(s) pour répéter une partie de leur programme.**		
		Les élèves ouvrent le fichier Scratch B.	L'enseignant demande aux élèves d'ouvrir le fichier « Scratch B » et de **réaliser le nouveau parcours**	*Une confusion sur la notion de répétition est possible. Pour effectuer un même*

		Phase 1 : « Vous allez devoir **guider l'abeille** pour qu'elle aille butiner 3 fleurs avant de retourner à sa ruche, attention, cette fois-ci, il y a **des obstacles** ! L'abeille doit éviter tout ce qui n'est pas une fleur ou sa ruche.» « Que peut-on remarquer entre le point de départ et le coquelicot (fleur rouge, 2ème colonne) et entre le coquelicot et l'héliopsis (fleur jaune, 3ème colonne) ? ». Phase 2 : Pour inciter les élèves à utiliser des boucles, l'enseignant leur demande de réaliser un **programme comportant le moins d'instructions possible.**	(il faut toujours « récupérer » les fleurs puis se rendre à la ruche). Il attire leur attention sur les **obstacles** qui sont maintenant sur le chemin (frelon, guêpier, insecticides…). Au cours de son exploration auprès des élèves, l'enseignant leur fait éventuellement remarquer que certaines parties de leur programme se répètent. Si ce n'est pas le cas au bout d'un certain temps, il leur indique que les déplacements de l'abeille entre le point de départ et le coquelicot et le déplacement entre le Coquelicot et l'Héliopsis sont **les mêmes** (Avancer 1 case – Droite – Avancer 2 cases – Gauche). Il incite les élèves à utiliser une **boucle de répétition** dans ce cas. Il est possible d'utiliser l'instruction « Répéter 2 fois » pour simplifier et aller plus vite. L'instruction « Répéter » est placée avant ce qui doit être répété. L'enseignant montre aux élèves comment utiliser les boucles (l'instruction « répéter … fois » est un bloc à compléter, présent dans la catégorie « Contrôle »). *« Une boucle est une instruction*	*déplacement 2 fois, les élèves peuvent avoir tendance à utiliser l'instruction « Répéter 1 fois ». Dans le langage courant, la notion de répétition implique déjà qu'une chose est réalisée plusieurs fois. Ici, c'est bien **« Répéter 2 fois »** qu'il faut utiliser car cette instruction sert à signifier combien de fois les instructions présentes dans la boucle doivent être exécutées.* *La définition d'une boucle doit être prise en note par les élèves.* *Comme pour l'activité précédente, les élèves doivent travailler en autonomie et bien sauvegarder leur travail.*

			qui permet de répéter plusieurs fois un ensemble d'instructions ». Vu que le parcours à réaliser est plus long et plus complexe, il incite les élèves à **utiliser des boucles aussi souvent que possible**, en repérant les déplacements qui se répètent, pour que le programme ne soit pas trop long.	
30 min	**Evaluations**	Les élèves sont évalués par le biais des **trois tests** suivants au format papier : - Test de compréhension des notions de programmation - Test de résolution de problèmes - Test de motivation et d'auto-efficacité en programmation	L'enseignant donne trois tests à aux élèves. Il ne peut pas les aider mais il peut **reformuler** ou répéter les consignes si les élèves en font la demande.	*Les tests sont réalisés INDIVIDUELLEMENT par chaque élève.*

Annexe L

Séance 1	Découverte de Scratch	Durée : 45 min	Condition Scratch Condition Scratch + Robot

Résumé :

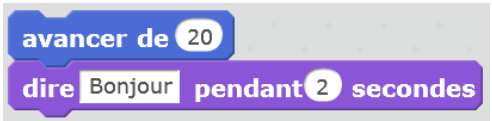
- *Les élèves découvrent Scratch, un environnement de programmation adapté à l'école primaire.*
- *Ils apprennent à lancer le programme et à enchaîner quelques instructions simples.*

Matériel :

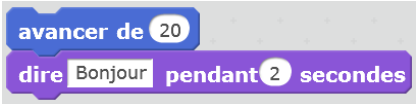
- Pour la classe : un vidéo projecteur.
- *Pour chaque groupe d'élèves : un ordinateur connecté à Internet ou, en l'absence de connexion Internet de bonne qualité, un ordinateur sur lequel le logiciel Scratch a été préalablement installé.*

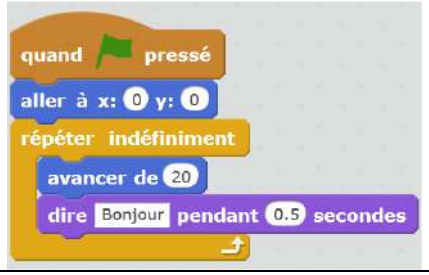
Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
10 min	**Lancer scratch et découvrir son interface**	Chaque groupe **lance Scratch** en double-cliquant sur son icône et **suit les instructions de l'enseignant pour pouvoir reproduire ce qu'il fait** en fin de séance.	L'enseignant explique aux élèves que **Scratch est un langage de programmation conçu spécifiquement pour apprendre à programmer.** Lorsque l'on ouvre le logiciel, il y a un lutin à l'écran (un chat). « Lutin » est le nom qu'on donne aux personnages ou objets que l'on peut programmer avec Scratch. On peut lui donner des instructions simples. Bien expliquer qu'**une instruction, c'est une action à exécuter.** L'enseignant réalise une petite démonstration (pour cette première activité, les élèves sont passifs et ne reçoivent pas de	*Interface de Scratch :* *Une **« scène »** : c'est là que se déroule le programme !* *Une **zone « lutins »**: les lutins sont les personnages ou les objets qui seront manipulés dans le programme (ils peuvent se déplacer, changer de forme, parler, interagir avec les autres lutins…). Lorsqu'on lance Scratch, il n'y a qu'un seul lutin affiché à l'écran : un chat.* *Une **zone « arrière-plan »**, juste à côté des lutins. L'arrière-plan est fixe, contrairement aux lutins qui peuvent bouger.*

			consigne). Par exemple, pour demander au chat de se déplacer de 10 pas, il suffit de faire glisser l'instruction « avancer de 10 » depuis la palette d'instructions vers la zone du programme. Si l'on clique ensuite sur cette inscription, on remarque que le chat avance bien de 10 pas (1 pas = 1 pixel de l'écran).  Si l'on souhaite avancer de 20 pas, il suffit de changer « 10 » en « 20 » en cliquant dans la zone dédiée. Si maintenant on souhaite que le chat avance de 20 pas, puis dise « Bonjour », il suffit de coller la nouvelle instruction à la fin du programme. **Écrire un programme se fait simplement en emboitant des instructions entre elles.**  Si maintenant on veut que le chat fasse cela à chaque fois que l'on clique sur le drapeau vert (en haut à droite de la scène, le drapeau vert permet de lancer le programme), alors il faut rajouter l'instruction «	*Par défaut, dans Scratch, l'arrière-plan est un fond blanc uni.* *Un onglet **« script »** qui permet d'accéder à :* *- **Une palette d'instructions.** C'est ici que l'on va trouver les instructions (ou « blocs ») que l'on va pouvoir utiliser dans notre programme. Il y a de nombreuses instructions, qui sont regroupées par couleur (exemple : tout ce qui concerne le mouvement du lutin est dans un onglet bleu foncé, tout ce qui concerne son apparence est dans un onglet violet, etc.).* *- Une zone **« programme »**, à droite de la palette d'instructions. C'est ici que l'on va écrire le programme, tout simplement en prenant des instructions depuis la palette et en les faisant glisser dans cette zone.*

			Quand drapeau vert pressé » à chercher dans la catégorie « événements » des instructions. Cela donne :  Finalement, l'enseignant montre **comment supprimer une instruction** (ou tout un bloc d'instructions) : il suffit de faire glisser cette instruction (ou ce bloc) depuis la zone du programme vers la palette des instructions. L'enseignant présente très rapidement l'interface de Scratch (voir remarques).	
10 min	**Explorer librement Scratch**	Les élèves **explorent librement Scratch**. Pour le moment, ils ne doivent pas chercher à modifier la scène ou le lutin mais simplement **manipuler des instructions simples et les agencer** pour observer ce qui se passe.	L'enseignant leur demande d'explorer les différentes catégories d'instructions, en particulier : - Catégorie **« mouvement »** (bleu foncé) - Catégorie **« apparence »** (violet) - Catégorie **« événement »** (jaune) - Catégorie **« contrôle »** (orange).	
25 min	**Structuration**	**Exercice 1 :** faire avancer le chat de 10 pas.	L'enseignant donne une série de petits exercices (qui reprennent, pour la plupart, ce qui a été fait avant sous forme de démonstration), que les	***Exercice 3 :*** ***« dire »*** *bonjour signifie pour nous* ***« écrire »*** *bonjour : on veut faire afficher une bulle à l'écran*

		 **Exercice 2 :** faire avancer le chat de 20 pas. Plusieurs solutions possibles (on préférera la seconde, plus élégante et facile à lire) :   **Exercice 3 :** faire avancer le chat de 20 pas et lui faire dire « Bonjour ».  **Exercice 4 :** remettre le chat au centre de la scène.  **Exercice 5 :** répéter indéfiniment : faire avancer le chat de 20 et lui faire dire «	élèves exécutent. Après chaque exercice, une **rapide mise en commun** permet de s'assurer que chacun sait faire l'exercice. **Exercice 5 :** Inciter les élèves qui ne trouvent pas à chercher dans la catégorie **« contrôle »** (orange). **Cette boucle enserre les autres instructions.** On voit le chat qui s'arrête 2 secondes entre chaque mouvement… Pour raccourcir cette pause, il suffit de raccourcir la durée pendant laquelle il dit « bonjour » (si on écrit « 0.5 » au lieu de 2, il ne s'arrêtera qu'1/2 seconde à chaque fois).	*avec le texte « bonjour » dedans, on ne veut pas faire parler le chat !* ***Exercice 4 :*** *Certains élèves vont sans doute trouver l'astuce… mais pour la plupart, il faudra la leur montrer. Malgré tout, il est indispensable pour eux de voir cette instruction dès maintenant, car, à force de déplacer le chat, ils vont le faire sortir de l'écran et ne sauront pas comment le récupérer.* ***Exercice 6 :*** *Cela se fait très simplement, sur le même modèle que l'exercice précédent, mais avec un autre type de boucle. Ils y trouveront une instruction qui ressemble (« répéter 10 fois ») et qu'il est facile de modifier en remplaçant 10 par 3. Tout ce qui se trouve à l'intérieur de la boucle est exécuté 3 fois.* ***Exercice 7 :*** *Il suffit a priori de rajouter l'instruction « quand drapeau vert pressé » (issue de la catégorie événement), mais cela est encore mieux si on demande au chat de repartir du centre de la scène. Expliquer, à ce moment, le rôle du bouton rouge (situé à côté du drapeau vert). Un clic sur ce bouton rouge met fin à l'exécution du programme (qui, sinon, ne*

		bonjour ».  **Exercice 6 :** répéter 3 fois : faire avancer le chat de 20 et lui faire dire « bonjour ».  **Exercice 7** : même chose quand on clique sur le drapeau vert  		*s'arrête jamais dans le cas présent).*

Séance 2	Contrôler les déplacements du robot	Durée : 1h	Condition Scratch + Robot

Résumé :

- *Les élèves apprennent à contrôler les déplacements d'un robot et à le faire stopper lorsqu'il détecte un obstacle.*

Matériel :

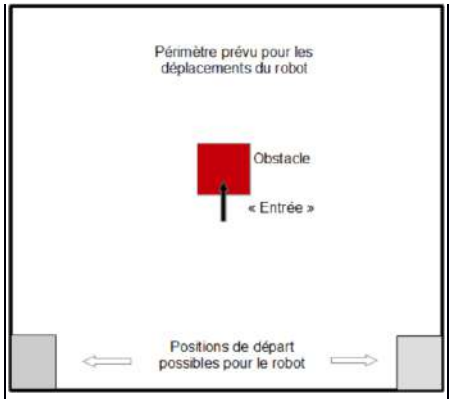
- Pour la classe : un vidéo projecteur.
- *Pour chaque groupe d'élèves : un ordinateur connecté à Internet ou, en l'absence de connexion Internet de bonne qualité, un ordinateur sur lequel le logiciel Scratch a été préalablement installé. Un robot Thymio Wireless.*

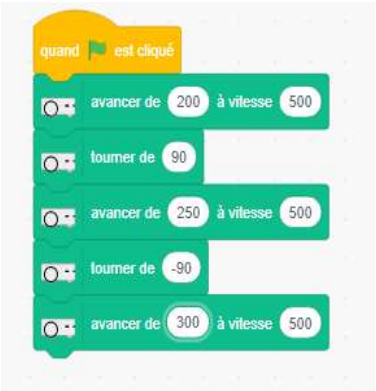
Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
5 min	**Lancer le logiciel et connecter le robot**	Suivre les instructions données par l'enseignant pour **commencer à utiliser Scratch avec Thymio.**	L'enseignant donne aux élèves les instructions suivantes : Premièrement, **allumer le robot** en restant appuyé quelques secondes sur le bouton central entouré des quatre flèches directionnelles. Récupérer la clé USB fixée sur le dessus du robot et la **connecter au port USB** de l'ordinateur. **Lancer la suite Thymio** (de préférence via un raccourci sur le bureau) **Sélectionner Scratch** parmi les langages de programmation proposés.	*Au préalable, l'enseignant devra installer la suite Thymio sur les ordinateurs.* *Il est aussi souhaitable, pour éviter le désordre, de prévoir un périmètre d'environ 1m² pour chaque élève manipulant le robot. Les élèves ne doivent pas faire sortir le robot de ce périmètre. L'enseignant peut expliquer cette consigne au tout début de la séance ou même en amont, avant que celle-ci ne débute.*

			Dans la partie droite de la nouvelle fenêtre qui s'ouvre, **sélectionner Thymio-Wireless** en cliquant dessus puis cliquer sur « Programmer avec Scratch » en bas à droite de l'écran.	
5 min	**Faire avancer le robot**	**Les élèves travaillent en autonomie et tâtonnent.** Pour que le robot avance, il suffit d'utiliser le bloc "Avancer de 50". Il est souhaitable d'utiliser un événement pour **déclencher le début du programme**. Ici, l'événement choisi est "Quand drapeau vert est cliqué". On obtient le programme suivant :  On constate que la valeur "50" peut être modifiée pour que le robot avance sur une plus grande distance. **Les élèves peuvent faire le test.**	L'enseignant fait remarquer tout d'abord que la connexion du robot Thymio à l'ordinateur a permis de **rajouter une nouvelle catégorie de blocs, la catégorie "Thymio"** tout en bas, en dessous de la catégorie "Mes blocs". L'enseignant attire aussi l'attention sur la possibilité d'**avancer tout en contrôlant la vitesse du robot** avec l'instruction "Avancer de 50 à vitesse 50". Il est possible pour les élèves de tester différentes vitesses.  	*Si le robot s'apprête à sortir du périmètre défini pour chaque élève, ils sont autorisés à soulever le robot pour le remettre à sa position initiale. Si la distance renseignée est négative le robot se dirige vers l'arrière.*
10 min	**Faire avancer le robot dans toutes les directions**	Les élèves **testent les différentes instructions possibles pour changer l'orientation du robot.**	L'enseignant fait remarquer que t**rois instructions sont disponibles** pour faire changer de direction au robot. La première est une **instruction basique**, la seconde permet de **contrôler la vitesse de rotation** du robot, la troisième permet de **contrôler le temps effectué** pour	*On remarque que le robot tourne vers la droite par défaut. Il est possible de le faire tourner dans l'autre sens en donnant **des valeurs négatives** à l'angle de rotation.* *Il est préférable de demander aux élèves de n'utiliser que*

			réaliser cette rotation. Il est possible de réaliser un petit programme qui utilise ces trois instructions en faisant varier l'angle, la vitesse et le temps pour bien marquer les différences observées (Voir ci-contre).	*certaines valeurs d'angle : 90 et -90 pour tourner à droite ou à gauche ; 180 pour faire un demi-tour et 360 pour faire un tour complet. Ainsi,* ***chaque élément de langage est associé à une valeur*** *d'angle de rotation.*
5 min	**Placer un obstacle dans le périmètre**	Les élèves doivent **placer le robot dans un angle du périmètre carré** (l'arrière du robot sur la ligne d'un des côtés, le côté gauche ou droit du robot sur la ligne d'un côté adjacent). **Placer ensuite un obstacle** (d'une hauteur de 10 cm minimum pour être sûr que le robot puisse le détecter) approximativement au centre du périmètre. Voir le schéma ci-dessous :	L'enseignant demande aux élèves de mettre le robot en position dans un angle du périmètre et de placer un obstacle au centre de celui-ci. Cet obstacle représente la **station électrique** dans laquelle le robot doit aller recharger ses batteries. **L'entrée** de la station est orientée vers le côté où se trouve le robot initialement. L'enseignant indique que Thymio ne pourra pas accéder à la station s'il arrive par la droite ou la gauche de l'obstacle, il doit nécessairement se diriger tout droit en face de l'entrée.	*Il sera demandé aux élèves de programmer le déplacement du robot pour qu'il atteigne la station par* ***« l'entrée »*** *de celle-ci. L'objectif est de* ***complexifier le programme demandé****. Sans cette consigne, il serait possible de faire le déplacement en 3 instructions seulement : Avancer – Pivoter – Avancer. Avec cette consigne, il en faut au moins 4 et il faut faire tourner le robot vers la gauche puis vers la droite (ou l'inverse).*

				
5 min	**Evaluation**	Les élèves doivent **répondre à un court questionnaire** permettant d'évaluer la charge cognitive de la tâche qu'ils ont réalisé.	L'enseignant veille à ce que chaque élève remplisse ce questionnaire INDIVIDUELLEMENT.	*Voir questionnaire1.pdf*
10 min	**Programmer les déplacements du robot**	**Les élèves essaient de trouver le chemin approprié pour que le robot atteigne la station**, par essais et erreurs. Ci-dessous, un programme possible pour atteindre l'objectif :	L'enseignant demande aux élèves de programmer les déplacements du robot pour qu'il atteigne la station. Il rappelle que pour cela, le robot doit impérativement arriver de face, et que pour tourner à droite, il faut tourner de 90 et de -90 pour tourner à gauche.	*Le problème c'est que, pour le moment, Thymio **ne détecte pas la station** et peut donc foncer dessus sans s'arrêter !*

				
15 min	**Stopper les moteurs lorsque la station est détectée**	**Les élèves doivent modifier la dernière instruction** (dans l'exemple ci-dessus, l'instruction « Avancer de 300 à vitesse 500 ») et la remplacer par l'instruction suivante, qui ne donne pas d'indications de distance :  Le robot avancera tout droit tant qu'on ne l'arrêtera pas. Ils doivent ensuite améliorer le programme pour que le robot **s'arrête lorsqu'il détecte un obstacle** (la station). Le programme attendu est le suivant :	L'enseignant fait remarquer que, pour le moment, **on est obligé de connaître la distance exacte qui sépare Thymio de la station** si on ne veut pas qu'il soit trop court ou trop long (en fonçant sur la station sans s'arrêter). Il explique que Thymio est équipé de capteurs qui permettent de **détecter des obstacles.** Il demande aux élèves de remplacer la dernière instruction de leur programme par une instruction qui fait avancer le robot sans indication de distance. Il leur demande ensuite de suivre et de reproduire les étapes permettant au robot de détecter un obstacle et de s'arrêter. Il indique que l'instruction à utiliser est une condition **« si … alors »** qui va exécuter une partie du programme **uniquement quand un obstacle est**	*La tâche étant relativement complexe, les élèves **suivent pas à pas les instructions** données par l'enseignant.*

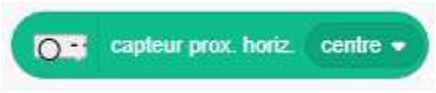
Malheureusement, lorsque l'on lance le programme et que le robot se dirige vers la station, cela ne fonctionne pas. Pourquoi ?

La classe peut discuter collectivement de ce que fait ce programme :

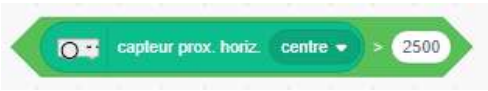
La station est placée au centre du périmètre. Un test est effectué : si le robot détecte l'obstacle alors il s'arrête. Puis … plus rien.

Les élèves suivent les instructions de l'enseignant pour que ce **test soit effectué tout au long de l'exécution du programme** et non pas seulement au début.

détecté. Pour détecter un obstacle, on utilise l'instruction :



Il faut ensuite **placer ce capteur à l'intérieur d'un autre bloc**. Ce bloc peut être trouvé dans la catégorie **« opérateurs »**, on obtient l'ensemble suivant :



Ici, la valeur a été fixée à 2500 mais cette valeur n'est pas essentielle. Ces deux blocs doivent ensuite être intégrés dans une condition « Si … Alors … » (catégorie « contrôle »)

Il ne reste plus qu'à placer une instruction pour stopper les moteurs si cette condition est remplie. On obtient le programme détaillé précédemment dans la colonne de gauche.

L'enseignant fait remarquer en lisant le programme que **le test n'est effectué qu'une seule fois** juste après avoir mis les moteurs.

Or, à ce moment-là, le robot ne

			détecte aucun obstacle (il est trop loin). Donc, il ne s'arrête pas. C'est normal. Pour que le programme fonctionne correctement, **il faut que le test soit effectué en permanence**, de façon à pouvoir déclencher l'action voulue dès que la condition sera remplie. Pour cela, il suffit de placer ce test dans une boucle **« répéter indéfiniment »**, que l'on trouve dans la catégorie **« contrôle »**. Le programme devient alors : 	
5 min	**Evaluation**	Les élèves doivent **répondre à un court questionnaire** permettant d'évaluer la charge cognitive de la tâche qu'ils ont réalisé.	L'enseignant veille à ce que chaque élève remplisse ce questionnaire INDIVIDUELLEMENT.	*Voir questionnaire2.pdf*

Pour aller plus loin :

- *Une valeur numérique est associé au capteur. Lorsque le robot détecte quelque chose, cette valeur augmente ! La valeur associée au capteur ne correspond PAS à la distance entre le capteur et l'obstacle. Plus le robot se rapproche de l'obstacle plus la valeur du capteur augmente également. Il ne faut donc pas se tromper d'instruction : c'est bien dans un bloc **« > 2500 »** (ou toute autre valeur) qu'il faut intégrer le capteur et pas dans un bloc « < 2500 ». Plus cette valeur est haute, plus le robot s'arrêtera proche de l'obstacle.*

Séance 2	Contrôler les déplacements d'un lutin	Durée : 1h	Condition Scratch

Résumé :

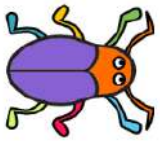
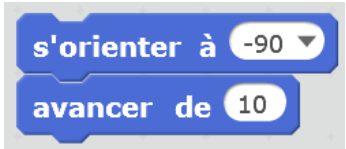
- *Les élèves apprennent à contrôler les déplacements d'un lutin pour atteindre une ressource placée sur le chemin.*

Matériel :

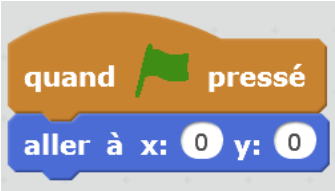
- Pour la classe : un vidéo projecteur.
- *Pour chaque groupe d'élèves : un ordinateur connecté à Internet ou, en l'absence de connexion Internet de bonne qualité, un ordinateur sur lequel le logiciel Scratch a été préalablement installé.*

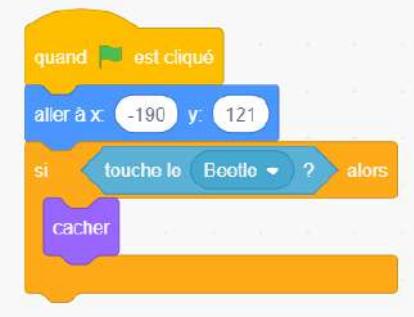
Déroulement :

Durée	Désignation	Tâche de l'élève	Rôle de l'enseignant	*Remarques*
5 min	**Changer de lutin**	Sélectionner **"Choisir un sprite"** en bas à droite de la scène (version Scratch Desktop) puis une nouvelle fois "Choisir un sprite". L'objectif est de **sélectionner un lutin qui soit "vu du dessus"** pour que ses déplacements paraissent plus naturels. Contrairement au "chat" qui est vu de profil et qui peut se retrouver "la tête en bas" lorsqu'il change de direction.	L'enseignant explique qu'il est possible de supprimer le lutin « chat » et d'en créer un autre à la place.	*Un **sprite** ou un **lutin** est un personnage (ou un objet) qui peut être programmé dans Scratch.* *Pour supprimer le chat, il faut cliquer (bouton droit) sur son icône, dans la zone des lutins, et choisir « supprimer ».*

		Exemple : 		
5 min	**Faire avancer le lutin vers la gauche**	Les élèves savent déjà comment faire avancer le lutin vers la droite puisque, par défaut, il est déjà orienté vers la droite. Le faire avancer vers la gauche est une tâche un peu plus difficile, car les élèves doivent d'abord demander au lutin de s'orienter vers la gauche, avant d'avancer. **Ils travaillent en autonomie et tâtonnent.** Finalement, le programme qui permet au lutin de se déplacer vers la gauche est : 	L'enseignant passant régulièrement dans les groupes pour s'assurer que personne n'est bloqué. Il peut les guider en les incitant à chercher une instruction « s'orienter ». Deux instructions de ce type sont disponibles : **« s 'orienter à… »** qui est celle qui nous intéresse. « s'orienter vers… » qui ne nous intéresse pas car la seule option disponible, accessible en cliquant sur la petite flèche, est « pointeur de souris » (le lutin, dans ce cas, s'oriente vers la position du pointeur de la souris).	*Lorsque l'on clique sur le nombre présent dans l'instruction **« s'orienter à ... »** (en général, ce nombre par défaut est « 90 »), une bulle d'aide apparaît. Suivant les versions de Scratch, elle peut prendre une forme différente pour nous expliquer que l'angle 0° désigne le haut de l'écran ; 90° désigne la droite, etc. Dans la version Scratch Desktop, il est possible de modifier l'orientation d'une flèche pour que la valeur de l'angle s'adapte automatiquement. Il faut donc choisir ici « s'orienter à -90 ».*
5 min	**Faire avancer le lutin dans toutes les directions**	Les élèves doivent maintenant être capables de **faire avancer le lutin dans n'importe quelle direction** (droite, gauche, haut et bas), en reprenant exactement la même méthode que précédemment.	L'enseignant demande aux élèves de changer l'orientation du lutin (vers le haut, le bas, revenir vers la droite...). 	*Désormais, on a besoin de l'instruction **« s'orienter à 90 »** pour lui dire d'aller à droite… car le lutin n'est plus orienté, par défaut, dans cette direction.*

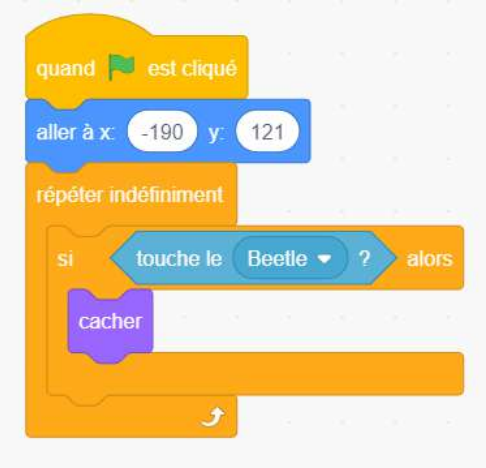
5 min	**Rebondir sur les bords**	Les élèves cherchent **comment faire en sorte que le lutin rebondisse sur les bords.** Par exemple, si on le dirige vers la droite et qu'il atteint l'extrémité droite de l'écran, le lutin doit rebondir afin de ne pas sortir de l'écran. Cela se fait très simplement en ajoutant l'instruction **« rebondir si le bord est atteint »** en bas de chacun des sous-programmes faits précédemment. Par exemple : 	L'enseignant demande aux élèves de trouver un moyen de faire rebondir le lutin sur le bord, pour éviter qu'il ne sorte de l'écran.	
5 min	**Initialiser la position du lutin**	Les élèves **reprennent les instructions qu'ils avaient vues** lors de la première séance Scratch. Le programme peut maintenant être exécuté en cliquant sur le drapeau vert.	L'enseignant rappelle que, **lorsqu'on lance le programme (drapeau vert), le lutin doit se situer au centre de l'écran.**	

				
5 min	**Evaluation**	Les élèves doivent **répondre à un court questionnaire** permettant d'évaluer la charge cognitive de la tâche qu'ils ont réalisé.	L'enseignant veille à ce que chaque élève remplisse ce questionnaire INDIVIDUELLEMENT.	*Voir questionnaire1.pdf*
5 min	**importer une ressource sous la forme d'un lutin**	Les élèves ouvrent le fichier sauvegardé lors de la dernière séance et **reprennent l'arrière-plan et le lutin** sélectionnés pendant cette séance (qui sera notre lutin principal). Ils peuvent supprimer les programmes qui avaient été réalisés. **Ils importent un nouveau lutin** qui pourrait être une ressource à récupérer par le lutin principal. Exemple (une étoile) : 	L'enseignant veille à ce qu'ils pensent à initialiser la position de cette ressource de manière aléatoire, en donnant au hasard des valeurs positives ou négatives à x et y dans l'instruction suivante :  Il fait remarquer que chaque lutin possède **sa propre zone de programme** (on passe du programme d'un lutin à l'autre en cliquant sur le lutin voulu). Il y a potentiellement autant de programmes que de lutins : tous ces programmes s'exécutent **en parallèle**.	*La ressource peut être positionnée n'importe où, pourvu que les deux lutins **ne se chevauchent pas.***

10 min	**Programmer le déplacement vers la ressource**	Les élèves doivent **programmer le déplacement du lutin vers la ressource.** Ils progressent par tâtonnements pour trouver le nombre de pas qui permet au lutin d'atteindre la ressource.	L'enseignant indique qu'il est d'abord nécessaire que les élèves **orientent leur lutin vers la position de la ressource.** Problème : si la ressource est positionnée en diagonale par rapport à notre lutin, il est plus compliqué d'estimer combien de pas le lutin doit réaliser pour toucher la ressource. On va donc demander aux élèves de s'orienter d'abord sur l'**axe horizontal** (vers la gauche ou vers la droite) pour avancer le nombre de pas nécessaire puis sur l'**axe vertical** (vers le haut ou vers le bas).	
10 min	**Faire disparaître la ressource quand elle est touchée**	**Les élèves doivent modifier le programme de la ressource pour que celle-ci disparaisse lorsqu'elle est touchée par le lutin principal.** Le programme attendu :  Malheureusement, lorsque l'on lance le programme et que le lutin principal se dirige vers la	**L'enseignant explique que, désormais, on souhaite faire disparaître la ressource lorsque celle-ci est touchée par le lutin principal pour bien montrer qu'elle a été récupérée.** Il indique que l'instruction à utiliser est une condition **« si … alors »** qui va exécuter une partie du programme **uniquement quand la ressource est touchée** et que le bloc qui permet de détecter si la ressource est touchée se trouve dans la catégorie **« capteurs »**. Il laisse les élèves tâtonner, puis passe pour **les guider en cas de blocage.** L'enseignant fait remarquer en lisant le programme proposé par les élèves (ci-contre), que le test **n'est effectué qu'une seule fois**, au lancement du	*Objectifs :* *- **faire disparaître** un lutin* *- **déclencher une instruction uniquement lorsqu'une certaine condition est remplie.** Cela se fait via la catégorie « contrôle », dans lequel on trouve l'instruction « si… alors… ».*  ***- détecter quand un lutin en touche un autre.** Cela se fait via la catégorie « capteurs » de la palette d'instructions (instruction « touche le … » sur la version Desktop). Une fois qu'on l'a*

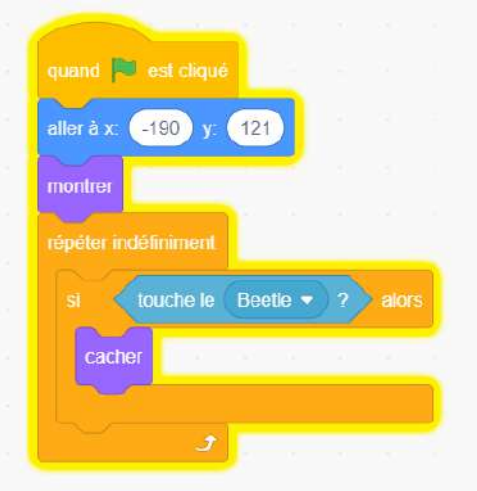
ressource, cela ne fonctionne pas. Pourquoi ? **La demi-classe peut discuter collectivement de ce que fait ce programme** :
La ressource est placée dans la position que l'on a choisie.
Un test est effectué : si la ressource touche le lutin principal, alors elle dit « bravo ». Puis… plus rien.
Les élèves suivent les instructions de l'enseignant pour que ce **test soit effectué tout au long de l'exécution du programme** et non pas seulement au début.

programme (juste après l'initialisation de la ressource). Or, à ce moment-là, les 2 lutins ne se touchent pas. Donc, la ressource ne disparaît pas. C'est normal. Pour que le programme fonctionne correctement, **il faut que le test soit effectué en permanence**, de façon à pouvoir déclencher l'action voulue dès que la condition sera remplie.
Pour cela, il suffit de placer ce test dans une boucle **« répéter indéfiniment »**, que l'on trouve dans la catégorie **« contrôle »**. Le programme de la ressource devient :



Attention, une fois qu'on exécute le programme, la ressource est désormais toujours cachée (car on ne lui a jamais dit de se montrer à

sélectionné, un clic sur la petite flèche permet de faire défiler les lutins déjà créés.
Ici, on est dans le programme de la ressource et on veut tester si le lutin principal (ici nommé "Beetle") est touché, on clique donc sur "Beetle".

			nouveau !). Il faut donc ajouter l'instruction **« montrer »** juste après l'instruction **« quand drapeau vert pressé »**. Le programme de la ressource devient donc : 	
5 min	**Evaluation**	Les élèves doivent **répondre à un court questionnaire** permettant d'évaluer la charge cognitive de la tâche qu'ils ont réalisé.	L'enseignant veille à ce que chaque élève remplisse ce questionnaire INDIVIDUELLEMENT.	*Voir questionnaire2.pdf*

Annexe M

Prénom : **Nom :** **Classe (CM1 ou CM2 ?) :**

Évaluation 2

*Pour chacune des phrases numérotées, indique si tu n'es **pas du tout d'accord**, si tu n'es **pas vraiment d'accord**, si tu es **ni d'accord ni pas d'accord**, si tu es **plutôt d'accord** ou si tu es **totalement d'accord.** Entoure ou coche la réponse qui te correspond.*
Sois le plus honnête possible, personne ne te jugera !

1. **Pour réaliser cette activité, je devais garder beaucoup de choses en tête en même temps.**

Pas du tout d'accord	Pas vraiment d'accord	Ni d'accord ni pas d'accord	Plutôt d'accord	Totalement d'accord

2. **Cette activité était très compliquée.**

Pas du tout d'accord	Pas vraiment d'accord	Ni d'accord ni pas d'accord	Plutôt d'accord	Totalement d'accord

3. **J'ai fait un effort pour comprendre toute l'activité mais aussi les liens entre chaque étape de l'activité.**

Pas du tout d'accord	Pas vraiment d'accord	Ni d'accord ni pas d'accord	Plutôt d'accord	Totalement d'accord

4. **Mon but en faisant cette activité était de tout comprendre correctement.**

Pas du tout d'accord	Pas vraiment d'accord	Ni d'accord ni pas d'accord	Plutôt d'accord	Totalement d'accord

5. **Pendant cette activité, c'était fatiguant de reconnaître les informations importantes par rapport à celles qui ne l'étaient pas.**

Pas du tout d'accord	Pas vraiment d'accord	Ni d'accord ni pas d'accord	Plutôt d'accord	Totalement d'accord

6. **La manière dont cette activité était présentée n'était pas pratique pour vraiment apprendre quelque chose.**

Pas du tout d'accord	Pas vraiment d'accord	Ni d'accord ni pas d'accord	Plutôt d'accord	Totalement d'accord

7. **Durant l'activité, j'ai eu du mal à reconnaître et à relier les notions essentielles entre elles.**

Pas du tout d'accord	Pas vraiment d'accord	Ni d'accord ni pas d'accord	Plutôt d'accord	Totalement d'accord

8. **Tu viens de participer à une séance de programmation qui a duré environ 1h.**
   Est-ce que pour réaliser cette séance tu as dû faire beaucoup d'effort mental ?
   Réponds sur une échelle de 1 à 9 en entourant la réponse qui te correspond.
   1 = Effort très très faible
   9 = Effort très très élevé

1 2 3 4 5 6 7 8 9