L'enseignement et l'apprentissage de la programmation à l'école

Ghizlane Amezrhar

L'enseignement et l'apprentissage de la programmation à l'école

DOI : 10.5281/zenodo.10303119

Amezrhar Ghizlane
CY Cergy Paris Université
ghizlane.amezrhar@gmail.com
ORCID : 0009-0002-2544-4225

Résumé : Dans cet article nous passons en revue les repères clés de l'intégration de la programmation à l'école, en mettant l'accent sur le développement de la pensée computationnelle. Nous abordons les difficultés d'apprentissage auxquelles sont confrontés les novices et nous fournissons un aperçu sur l'état actuel des dispositifs et des approches pédagogiques utilisés pour soutenir l'apprentissage de la programmation.

Abstract: In this review, we highlight the key points of reference for the integration of programming in school, with a focus on the development of computational thinking skills. We analyze and discuss the learning difficulties confronted by novices and we provide a state-of-the-art overview of pedagogical environments and approaches used to support and improve programming learning.

Mots clés : pensée computationnelle, programmation scolaire, langage de programmation visuel, langage de programmation textuel, robotique pédagogique.
Ce travail a été effectué sous la supervision du Pr. Alain Jaillet.

1. INTRODUCTION

En 1967 Seymour Papert invente la première version du langage Logo sur la base du langage LISP (Solomon, C. et al., 2020, 33). Un langage de programmation pour les enfants, avec le projet de faire découvrir les potentiels de l'algorithmique comme instrument de formation, d'enseignement, d'éducation et de créativité.

A partir du langage LOGO, les années 80 ont vu se déployer plusieurs propositions qui allaient dans ce sens. Après plus de quatre décennies de croissance, Logo subi encore des changements en phase avec le développement de la technologie informatique ce qui a donné naissance à une grande famille d'environnements Logo : Blockly, Scratch, Snap!, MBlock, … etc.

Le codage comme instrument de programmation et de découverte de la complexité constituait une nouvelle voie pour développer plusieurs compétences comme la résolution de problème, la collaboration, la créativité, et la pensée logique et critique… etc. (Buitrago Flórez, F. et al., 2017, 3-4).

Les curricula qui introduisent la programmation dans les cursus scolaires utilisent des termes différents pour la désigner, Zhang, L.C; Nouri, J. (2019,1-2) ont repéré quatre termes souvent cités dans la littérature à savoir : la programmation, le codage, l'informatique, et la pensée computationnelle. Ces termes semblent avoir des limites floues et sont parfois confondus. Dans le langage courant le codage et la programmation sont souvent interchangeables, cependant le codage n'est qu'une sous-tâche de la programmation qui implique plusieurs autres tâches (décomposition d'un problème, conception, codage, débogage... etc.). L'informatique est plus vaste comme concept mais qui reste fortement attaché à la programmation. "The core of computing is computer science, in which pupils are taught the principles of information and computation, how digital systems work, and how to put this knowledge to use through programming" (Department of Education UK, 2013, 1). Quant à la pensée computationnelle, elle est encore plus vaste comme concept, et peut être développée à travers plusieurs activités mais la littérature montre que la programmation reste la manière la plus adaptée pour développer cette pensée et ses compétences sous-jacentes (Drot-Delange, B. et al., 2019, 15).
Force est de constater que les différents termes qui figurent dans les curricula s'articulent tous autour de la programmation. Ce qui fait de la programmation toute une stratégie pédagogique et pas seulement un ensemble de techniques de codage à apprendre (Romero, M. et al., 2017, 2). Cependant l'introduction à la programmation pour les novices présente toujours des difficultés, et soulèvent des questions autours du langage de programmation à choisir (textuel/visuel/hybride), quel objet programmer (virtuel/tangible), et quelles approches pédagogiques mobiliser. Dans ce sens nous présenterons ici une revue de littérature qui illustre les apports des recherches portant sur l'enseignement et l'apprentissage de la programmation. Nous commençons par décrire notre stratégie de recherche documentaire, ensuite nous passons en revue les repères clés de l'enseignement et l'apprentissage de la programmation, puis nous abordons les difficultés auxquelles sont confronté les débutants ainsi que les dispositifs d'apprentissage utilisés pour soutenir l'introduction à la programmation. Après, nous donnons un aperçu des approches pédagogiques utilisées dans les cours d'introduction à la programmation, nous soulignons l'impact des activités de collaboration, l'apprentissage par le jeu, la résolution de problème, et les stratégies de rétroaction. Nous présentons ensuite une synthèse du travail en mettant l'accent sur les convergences et les divergences des études examinées. Et enfin nous terminons par une conclusion.

2. MÉTHODOLOGIE

Notre démarche commence par une exploration simple du sujet, nous avons lancé une recherche sur google scholar en utilisant des mots clés et leurs synonymes comme : pensée computationnelle, apprentissage enseignement programmation, apprentissage codage, difficultés programmation, environnements programmation blocs, programmation graphique, programmation textuelle, robotique pédagogique, robotique tangible, programmation simulation, approches enseignement/apprentissage programmation, introduction programmation, programmation novices…etc. Comme la recherche sur google scholar est ouverte à tout type de publication incluant les articles approuvés ou non par des comités de lecture, nous avons tenté d'explorer d'autres bases de données indexées comme : ACM Digital Library, IEEE Xplore Digital Library, SpringerLink, ScienceDirect, et Dimensions, ainsi que les revues spécialisées dans l'enseignement e l'informatique et la programmation

comme : The Review of Educational Research, Computer Science Education, Educational Technology & Society, etc. L'utilisation des opérateurs booléens (and/or/not) nous a permis de relier les mots clés lors de la recherche afin d'obtenir les résultats les plus pertinents que possible. Nous avons trouvé des articles de journaux et de revues, des comptes rendus, des actes de communication et des publications gouvernementales.

Pour filtrer les sources obtenues nous avons procédé par une première lecture sélective qui consiste à lire le titre, le résumé, et les mots clés de chaque article, ce qui nous a permis d'éliminer les sources parasites qui sont loin de répondre à notre intérêt de recherche et de repérer rapidement les sources qui semblent être pertinentes. Ensuite nous avons effectués une deuxième lecture basée sur le texte intégral de chaque document, cette étape s'appuie sur une grille d'analyse (tableau 1) qui permet d'évaluer la qualité scientifique de chaque recherche. Dans notre cas les critères d'évaluation portent sur les questions suivantes : La question de recherche est-elle précisée ? hypothèse qui est vérifiée est-elle annoncée ? La partie "Matériel et Méthodes" décrit-elle l'expérimentation qui a été menée ? Est-il possible avec cette description de répéter la même expérimentation ? Les résultats sont-ils interprétés lors de leur présentation ou bien dans la discussion ? Cette interprétation correspond-elle aux résultats présentés ? Y a-t-il des extrapolations non vérifiées ? A la base de ces questions nous avons sélectionné les documents forment notre corpus d'études (détails en Annexe). Nous précisons que nous n'avons pris en compte aucune restriction concernant le pays où l'étude a été menée ni sa date de publication.

Tableau 1 : grille de lecture et d'analyse (adaptée et enrichie de Hart, C., 2009, 146)

Titre, Auteur(s), Année	Problématique + Questions et Hypothèses de recherche (éventuellement)	Méthodologie (Approche méthodologique et outils) Corpus de données	Principaux résultats	Pistes éventuelles

L'étape suivante consiste à regrouper, catégoriser, comparer, et organiser les références sélectionnées entre elles afin de mieux comprendre et interpréter les données. Pour cela nous avons utilisé une fiche d'annotation de références sous Excel, les colonnes du tableau répondent aux questions suivantes : quelle est la référence de l'article, auteur(s) et titre, lien d'accès, de quoi parle l'article ? quel est le but de l'étude ? pourquoi c'est important? quelle est l'approche / la méthode utilisée pour acquérir les données ? quelle est l'approche / la méthode utilisée pour analyser les données ? quelles sont les principales conclusions ? y a-t-il des limitations à l'étude ? Lesquelles y a-t-il des perspectives d'applications et de recherches futures ? Quelles sont les principales conclusions et implications dans un contexte plus large? commentaires personnels (Mémos). Le but de cette fiche est de pouvoir faire le maximum de croisements, de liens possibles et de faire différents tris : par année, méthodes, thématiques, etc.

Pour maitriser les informations recueillies et identifier les grandes orientations de la recherche en relation avec l'enseignement et l'apprentissage de la programmation à l'école, nous avons opté pour une classification des sources collectées par thématique traitée. Il en résulte trois grands axes : le premier concerne la pensée computationnelle et les repères clés de

l'enseignement et l'apprentissage de la programmation, le deuxième est consacré aux difficultés auxquelles sont confrontés les débutants ainsi que les dispositifs d'apprentissage utilisés pour soutenir l'introduction à la programmation, et le troisième pour les approches pédagogiques utilisées dans l'enseignement de la programmation.

3. REPÈRES CLÉS DE L'ENSEIGNEMENT ET L'APPRENTISSAGE DE LA PROGRAMMATION

En parcourant l'historique de l'enseignement et l'apprentissage de la programmation dans la littérature, on peut constater que la pensée computationnelle est un concept central qui est à l'origine de l'intégration de la programmation dans les programmes scolaires. La littérature met en avant l'importance de la pensée computationnelle pour le transfert des compétences dans d'autres domaines ainsi que les compétences sous-jacentes impliquées. Cependant, la définition de la pensée computationnelle et son évaluation posent des défis, en raison de la diversité des cadres de référence qui entourent ce concept.

3.1 Pensée computationnelle et programmation

La pensée computationnelle ou bien la pensée informatique a pour origine les travaux de Seymour Papert en 1981, Son intérêt à la pensée découle d'un héritage Piagétien : «understanding how we know what we know, how we construct knowledge and how we think» (Bers, M.U., 2017, 10). Si le monde de l'éducation considère que les idées les plus importantes de Piaget concernent la théorie de développement, alors ce n'est pas le cas pour Papert qui était plutôt concentré sur son constructivisme, et son structuralisme. (Papert, S., 1988 ; cité dans Solomon, C. et al., 2020, 8). Focalisé sur les idées constructivistes qui considèrent l'apprentissage comme une reconstruction plutôt qu'une transmission de connaissances, Papert développe sa théorie constructionniste en mettant l'accent sur la manière d'apprendre, il considère que l'apprentissage est plus efficace lorsqu'il est intégré dans une activité que l'apprenant vit comme la construction d'un produit significatif : une œuvre d'art, une machine qui fonctionne, un rapport de recherche ou un programme informatique... (Papert, S. et al., 1986 ; cité dans Solomon, C. et al., 2020, 13). Dans le cadre de cette théorie, le langage de programmation Logo a été créé. C'est un langage simple que les enfants peuvent facilement maîtriser, comprendre et surtout utiliser. Le but est de profiter d'une programmation simple pour que les enfants puissent créer tout en étant autonomes (Papert, S. ; Jaillet, A., 2006, np). Papert considère la programmation comme une activité favorisant le développement d'une pensée procédurale, dans la mesure où « penser » revient à exécuter des procédures ou encore à appliquer des algorithmes (Crahay, M., 1987, 47). L'intérêt du langage Logo pour Papert est d'apprendre aux enfants à penser et à apprendre autrement, cela concerne davantage les idées fondamentales que les enfants peuvent découvrir ou s'approprier dans des situations informatisées que l'apprentissage de l'informatique pour elle-même.

En 2006, Jeannette Wing relance le débat autour de la pensée informatique avec une autre vision qui distingue la pensée informatique de l'apprentissage de la programmation. Elle précise que la programmation permet d'implémenter la pensée informatique, mais celle-ci ne s'y réduit pas, il s'agit bien d'une pensée, pas simplement de calcul mécanique au sens de routinier (Wing, J.,2006, 36). Selon elle « la pensée informatique décrit l'activité mentale dans la formulation d'un problème pour admettre une solution informatique par un homme ou une

machine, ou plus généralement, par des combinaisons d'hommes et de machines » et ne peut être réduite à l'enseignement de la programmation (Wing, J., 2010, 1).

Même s'il existe un consensus sur le fait que la pensée informatique est bien plus que la programmation, les recherches sur la pensée informatique se limitent souvent à des contextes de programmation, ce qui montre que la programmation reste un moyen incontournable pour développer la pensée informatique (Voogt, J. et al., 2015, 726 ; Buitrago Flórez, F. et al., 2017, 3 ; Drot-Delange, B. et al., 2019, 15).

3.2 Pensée computationnelle et transfert des compétences

Seymour Papert postule qu'à l'aide du langage de programmation Logo, les élèves peuvent développer des compétences transférables dans des contextes autres que la programmation, à l'intérieur et à l'extérieur de la classe (Voogt, J. et al., 2015, 717). Dans ce même ordre d'idée, Rogalski, J. (1987, 7) insiste sur l'apport de l'apprentissage de la programmation dans la formation des élèves à une pensée singulière liée à la résolution de problèmes dans d'autres disciplines. Cet aspect de la programmation comme outil de pensée implique une part importante de modélisation de situations-problèmes et même de transférer cette capacité de modéliser à d'autres domaines. Toutefois, les résultats des études dans ce sens n'étaient pas concluants. Pea, R. et al., (1985, 208) ont testé le transfert proche de la compétence « planification d'une tâche », avec deux groupes d'élèves, le premier groupe ayant passé une année à programmer avec Logo, le deuxième groupe (témoin) n'ayant aucune expérience de programmation, les résultats montrent qu'il n'y a pas de différences significatives entre les deux groupes au niveau des capacités de planification d'une tâche hors du contexte de la programmation. Ainsi les auteurs de cette étude affirment qu'il ne semble pas y avoir d'amélioration automatique des capacités de planification grâce à l'apprentissage de la programmation. Ces propos ont été confirmés par Baron, G.-L. (1990, 67) dans sa revue de littérature, en examinant plusieurs études dans ce sens, il souligne que les transferts de compétences acquises lors de l'activité de programmation ne sont pas prouvés lors d'expérimentations cherchant à les mesurer. Pourtant cela n'empêche qu'il existe aussi des études qui ont constaté le contraire, comme celle de Klahr et Carver qui ont constaté que les étudiants formés aux techniques de débogage pendant l'utilisation de LOGO ont transféré ces compétences dans un environnement non programmé (Klahr, D. et al., 1988, cité dans Voogt, J. et al., 2015, 718). Cette question de transfert est toujours d'actualité et les études dans ce sens sont encore peu concluantes, une méta-analyse récente par Scherer, R. et al., (2019, np) trouve des preuves d'un fort transfert proche, et d'un transfert lointain modéré à faible, " le transfert est plus susceptible de se produire dans des situations qui requièrent des compétences cognitives proches de la programmation ", cependant les auteurs reconnaissent les limites de leur corpus de recherche actuel et préconisent plus d'études pour pouvoir conclure.

3.3 Pensée computationnelle et compétences sous-jacentes

Différents auteurs s'accordent pour la nouvelle définition de la pensée computationnelle selon la vision de Wing, dans la mesure ou la pensée informatique permet l'apprentissage de certains concepts transversaux qui aident à la résolution de beaucoup de problèmes — dont la programmation, qui n'en est qu'un parmi d'autres (Drot-Delange, B. et al., 2019, 46). Toutefois, les recherches visant à circonscrire les concepts et les compétences qui engendrent la pensée informatique révèlent des résultats qui diffèrent d'un auteur à l'autre, ce qui

constitue un obstacle devant l'opérationnalisation de cette pensée dans des activités concrètes (Chen, G. et al., 2017, 14).

Cette prise de conscience de l'importance d'opérationnaliser la pensée informatique a eu lieu il y a une quinzaine d'année dans la littérature anglo-saxonne, grâce à l'association des enseignants d'informatique (CSTA) créée en 2004 aux États-Unis, association Informatique pour s'amuser (ComputerScience4Fun) créée en 2005 et Calcul à l'École (computing@school) créée en 2008 en Angleterre. Ainsi, nombreux pays anglo-saxons (États-Unis d'Amérique, Angleterre, Australie) ont pu mettre en place des curriculums intégrant des compétences en lien avec la pensée informatique (Busana, G. et al., 2020, 3). En 2011, la Société internationale pour la technologie dans l'éducation (ISTE) en collaboration avec la CSTA ont proposé une définition opérationnelle de la pensée informatique qui s'appuie sur six composantes :
(1) Formuler les problèmes d'une manière qui nous permet d'utiliser un ordinateur et d'autres outils pour aider à les résoudre ;
(2) Organiser et analyser logiquement les données ;
(3) Représenter les données par des abstractions telles que des modèles et des simulations ;
(4) Automatiser les solutions par la pensée algorithmique (une série d'étapes ordonnées) ;
(5) Identifier, analyser et mettre en œuvre des solutions possibles dans le but d'obtenir la combinaison d'étapes et de ressources la plus efficace et la plus efficiente ;
(6) généraliser et transférer ce processus de résolution de problèmes à une grande variété de problèmes (CSTA & ISTE, 2011, 1).
La question de l'opérationnalisation de la pensée informatique a été aussi étudié par Barr et Stephenson qui ont identifié un ensemble de capacités et concepts de base de la pensée computationnelle axé sur ce que les élèves feraient réellement, ces capacités comprennent : la conception des solutions aux problèmes (en utilisant l'abstraction, l'automatisation, la création d'algorithmes, la collecte et l'analyse de données); la mise en œuvre des conceptions (en programmant); l'expérimentation et le débogage; la modélisation, la simulation, l'analyse de systèmes ; la réflexion sur la pratique et la communication ; l'utilisation du vocabulaire ;la reconnaissance des abstractions et le passage d'un niveau d'abstraction à un autre; l'innovation, l'exploration et la créativité entre les disciplines; la résolution des problèmes en groupe ; et la mobilisation de stratégies d'apprentissage diverses (Barr, V. ; Stephenson, C., 2011, 51). En examinant plusieurs études Selby, C.C. (2013, 2-5) a pu extraire les différents termes liés à la pensée informatique apparu dans les recherches de 2006 à 2013 à savoir : le processus de pensée, l'abstraction, la décomposition, la pensée logique, la résolution de problèmes, la pensée algorithmique, l'évaluation, la conception de systèmes, le contenu informatique, la généralisation, l'automatisation, la modélisation, la simulation et la visualisation. Après avoir exclu les termes qui ne semblent pas avoir de consensus ou bien qui ne sont pas suffisamment définis dans la littérature, Selby ne garde que cinq concepts sur la base desquels il définit la pensée informatique comme un processus cognitif lié à la résolution de problème et qui reflète l'abstraction, la décomposition, l'algorithmique, l'évaluation, et la généralisation.
Du côté Européen, un référentiel intitulé « Brevet Informatique et Internet » a été instauré en 2000 et actualisé en 2001, puis remplacé par un cadre de compétences numériques qui a été proposé par des experts depuis 2013 et actualisé en 2019 sous l'intitulé « cadre référentiel des compétences numériques », cependant il ne distingue pas les compétences de la pensée informatique de manière explicite. Dans la continuité de la tendance à opérationnaliser les compétences de la pensée informatique, un autre projet soutenu par l'union Européenne est

lancé en 2018, intitulé pensée informatique et algorithmique dans l'enseignement fondamental (PIAF). Les chercheurs dans ce projet ont distingué six compétences propres à la pensée informatique, chacune étant décomposée en un ensemble de trois à sept sous compétences : C1 Définir des abstractions / généraliser - C2 Composer / décomposer une séquence d'actions - C3 Contrôler une séquence d'actions - C4 Évaluer des objets ou séquences d'actions - C5 Traduire des informations dans déférentes représentations - C6 Construire une séquence d'actions itérativement. (Busana, G. et al., 2020, 2-4).

3.4 L'évaluation de la pensée computationnelle

Il existe plusieurs approches pour évaluer la pensée computationnelle selon les compétences impliquées dans la conceptualisation adoptée par le chercheur. Dans une revue systématique par De Araujo et al., consacrée à l'identification des artefacts et instruments utilisés pour évaluer la pensée computationnelle, les résultats montrent que le cours de programmation est le contexte le plus utilisé pour développer et évaluer l'apprentissage de la pensée computationnelle. Par conséquence, le code est l'artefact le plus courant à utiliser, toutefois les auteurs de cette étude soulignent que cette approche devrait être discutée davantage du fait qu'elle limite l'évaluation à la présence ou l'absence d'une structure de programmation. Les questionnaires à choix multiples présentent aussi un instrument très courant dans la mesure du développement de la pensée computationnelle, ils sont simples à reproduire et à compiler les résultats, cependant le défi réside dans l'élaboration d'un questionnaire approprié au processus cognitifs à mesurer. Certaines études élaborent leurs propres questionnaires, alors que d'autres adoptent des tests connus comme le PISA. L'analyse des réponses aux exercices est également utilisée dans plusieurs études, cette approche permet d'identifier les erreurs commises par les apprenants et de mieux comprendre leur raisonnement et leur processus de pensée (De Araujo, A. L. S. O. et al., 2016, 7).

Dans la continuité des efforts visant à évaluer la pensée computationnelle, Moreno et al., ont présenté en 2015 un outil web intitulé Dr. Scratch qui permet aux étudiants et aux enseignants d'analyser et évaluer automatiquement les projets réalisés avec Scratch étant donné que c'est l'environnement le plus utilisé pour soutenir le développement de la pensée informatique au primaire et au secondaire. Dr. Scratch calcule automatiquement les scores pour l'abstraction, la logique, et le contrôle de flux, il fournit également un feedback instantané permettant d'améliorer le code. Pour évaluer l'efficacité de cet outil, les chercheurs ont organisé des ateliers avec des élèves de 10 à 14 ans. Les résultats montrent que grâce au rapport d'évaluation fourni par Dr. Scratch, les élèves ont pu améliorer leurs projets et augmenter leurs scores de pensée informatique (Moreno-León, J. et al., 2015, 20). Selon Romero, M. et al. (2017, 6), cet outil semble être plutôt adapté pour évaluer le niveau de maitrise technique de scratch, cependant il ne peut pas évaluer les composantes essentielles de la pensée informatique dans un contexte créatif. Dr. Scratch ne prend en compte que la complexité des programmes et non pas leur signification, puisque le programme analysé ne donne pas de preuves sur les processus de pensée ni sur la tâche demandée. Dans ce sens, Romero et al. estiment que les outils d'analyse automatique du code en général ne peuvent pas évaluer la créativité, la parcimonie et la pertinence d'un programme. En revanche, Romero et al. proposent d'évaluer la pensée informatique en tenant compte de la dimension créative de la programmation, suivant le modèle #5c21 qui fait référence à cinq compétences clés dans l'éducation au 21ième siècle à savoir : la pensée informatique, la créativité, la collaboration, la résolution de problèmes et la pensée critique. Dans ce cadre, la pensée informatique est basée sur six compétences opérationnelles décrites dans la figure1.

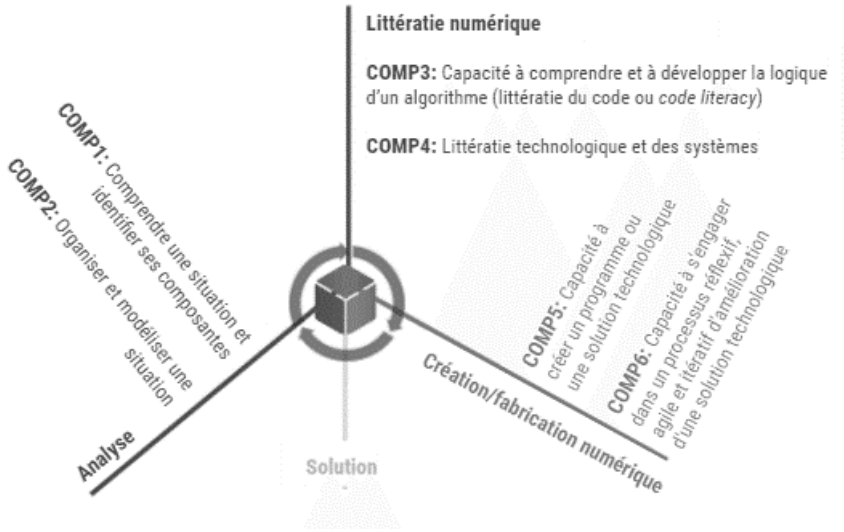

Figure 1 : les six composantes de la pensée informatique selon le modèle #5c21 (Lepage, A. ; Romero, M., 2017, np)

A partir de cette opérationnalisation, un protocole et un outil d'évaluation ont été développés: avant l'évaluation l'enseignant définit les observables à mesurer avec l'outil #5c21, ce dernier permet de réaliser un pré-test, un post-test et une évaluation à temps réel par l'enseignant ou une auto-évaluation par l'apprenant. Après une période l'enseignant peut générer des rapports montrant l'évolution des compétences de la pensée informatique chez les apprenants. La comparaison de l'outil #5c21 avec l'outil Dr.Scratch montre que l'analyse automatique des projets des participants donne des scores similaires en termes de complexité algorithmique, par contre l'analyse avec #5c21 montre une grande différence entre les projets en termes de performance de programmation créative. À la suite de cette étude les auteurs recommandent de combiner les outils d'évaluation automatique du code à l'évaluation de l'expertise humaine sur les aspects créatifs de la programmation. (Romero, M. et al., 2017, 8-12)

4. LA PROGRAMMATION COMME CADRE DE DÉVELOPPEMENT DE LA PENSÉE INFORMATIQUE : LES DIFFICULTÉES RENCONTRÉES, ET LES DISPOSITIFS D'APPRENTISSAGE UTILISÉS

En s'intéressant à l'enseignement et l'apprentissage de la programmation comme cadre de développement de la pensée informatique, la littérature met en évidence les difficultés d'apprentissage, les différences entre langages visuels et langages textuels, et notamment la question du passage de l'un à l'autre, avec les langages hybrides. La robotique pédagogique incrémente une dimension supplémentaire avec le déplacement physique d'un objet.

4.1 Difficultés de l'enseignement et l'apprentissage de la programmation

Les difficultés de l'enseignement et l'apprentissage de la programmation sont évoquées dans la littérature à travers plusieurs recherches. Selby, C.C. (2015, 81-85) adopte un cadre de référence basé sur deux taxonomies éducatives : la taxonomie de Bloom et la taxonomie SOLO, afin de déterminer la relation entre les processus cognitifs, la pédagogie de la programmation et les niveaux de difficultés perçus des compétences de la pensée computationnelle. Les données sont recueillies auprès des enseignants et sont analysées selon une approche de théorie ancrée. Après avoir attribué les compétences de la pensée computationnelle (l'évaluation, la conception d'algorithmes, l'abstraction, la décomposition

et la généralisation) aux niveaux du domaine cognitif de la taxonomie de Bloom (figure 3), l'analyse des données indique que la décomposition est perçue comme la compétence de pensée computationnelle la plus difficile à maîtriser, l'abstraction de la fonctionnalité est moins difficile que l'abstraction des données, mais les deux sont perçues comme difficiles. Cet ordre de difficulté est une inversion de la complexité cognitive prédite par le modèle de Bloom (figure 2).

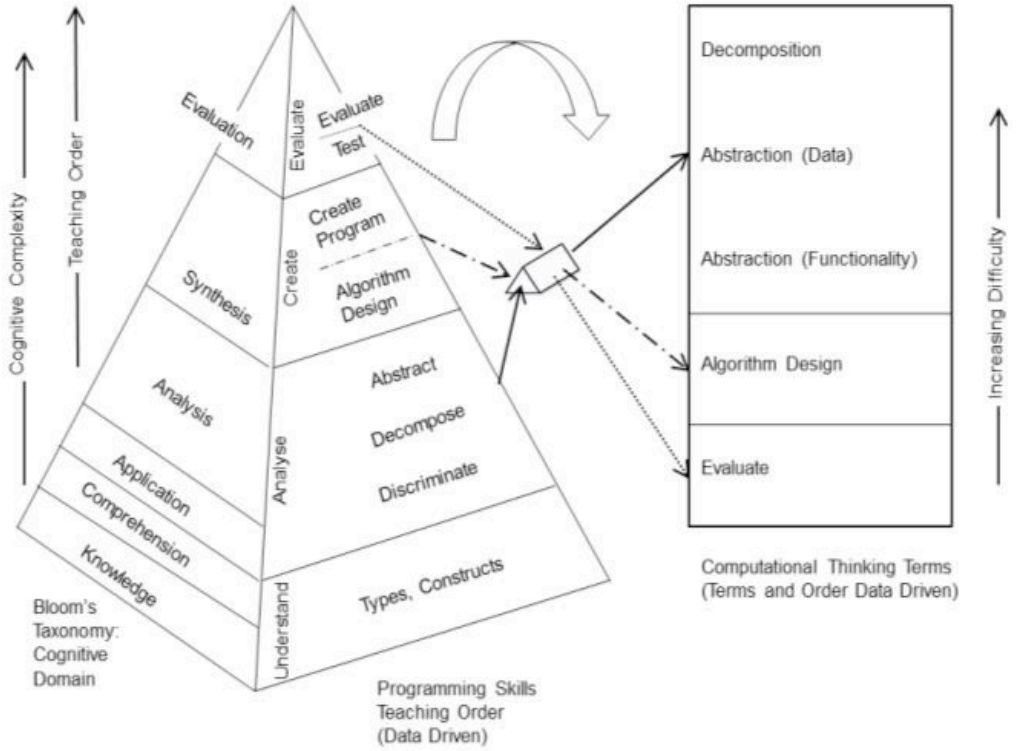

Figure 2: Model: computational thinking, pedagogy of programming, and Bloom's Taxonomy (Selby, C.C., 2015 : 85)

Une revue systématique réalisée par Medeiros, R. P. et al. (2019, 11) montre que les difficultés fréquentes constatées dans les études sont liées à la résolution de problème et les compétences mathématiques, suivie de la motivation et de l'engagement, puis des difficultés d'apprentissage de la syntaxe des langages de programmation. D'autres études ont cités le manque d'innovation dans les méthodes d'enseignement (Ouahbi, I. et al., 2015, 2), la faiblesse des capacités cognitives des étudiants (Mladenović, M. et al., 2018, 3) notamment dans l'enseignement primaire et secondaire, la complexité de l'activité de programmation (Dehnadi, S., 2009; cité dans Mladenović, M. et al., 2018, 5), les étudiants doivent d'abord apprendre des détails techniques entourant l'utilisation de l'environnement de développement et les règles syntaxiques du langage en plus de la construction d'algorithmes (Duguay, S., 2018 : 27; Gomes, A. ; Mendes, A., 2007, 5), cela demande la mémorisation d'un grand éventail d'informations et la mobilisation de plusieurs compétences en même temps ce qui impose une charge cognitive exigeante (Mow, I. T. C., 2008, 2).

Pour les novices en programmation, la focalisation sur la syntaxe d'un langage avant d'avoir un modèle mental de la construction et la structuration de l'algorithme amène les étudiants à appliquer une approche de programmation « ligne à ligne » plutôt qu'une approche significative et structurée, il en résulte un apprentissage superficiel qui ne conduit pas à appréhender la résolution de problèmes de manière efficace, même si les apprenants apprennent la syntaxe et la sémantique des instructions, il ne parviennent pas à les combiner pour avoir un programme valide par manque de modèles mentaux détaillés (Winslow, L. E.,

1996 ; cité dans Siti Rosminah M. D. et al., 2012, 2). Cela affecte la perception des élèves envers la programmation et leur capacité à programmer, comme indiqué dans l'étude de Ramalingam, V. et al. (2004, 4), l'expérience antérieure des élèves en programmation continue d'affecter la perception qu'ils ont de leurs capacités, même vers la fin du parcours universitaire, et le fait d'avoir développé un modèle mental solide et précis augmente le sentiment d'auto-efficacité des étudiants et affecte directement leur performance dans le cours.

4.2 Les langages de programmation visuels versus textuels

Le choix du langage de programmation et de l'environnement de développement pour débuter est une décision primordiale qui affecte les apprenants, leurs rapports aux savoir, et les stratégies d'enseignement et d'évaluation déployées (Duguay, S., 2018, 26). Une recherche action par Yadin, A. (2011, 75) affirme que les échecs dans les cours d'initiation à la programmation ne sont pas occasionnés nécessairement par la mauvaise compréhension des concepts algorithmiques présentés dans le cadre du cours, mais par un langage de programmation inapproprié. Dans ce sens, plusieurs environnements et langages ont été conçus spécifiquement pour l'apprentissage de la programmation permettant de réduire la complexité de l'environnement de travail et ses fonctionnalités (Duguay, S., 2018, 27). Nombreuses études mettent en œuvre le potentiel éducatif des environnements de programmation visuels et en particulier à blocs, Ouahbi, I. et al. (2015, 4) affirment que l'utilisation de l'environnement scratch selon une approche basée sur la création de jeux permet de stimuler la motivation et l'engagement des novices envers l'apprentissage de la programmation.

Selon Price, T. W. et Barnes, T (2015, 98) les environnements de programmation à bloc, comparés aux environnements textuels, améliorent les performances de programmation chez les novices, réduisent le temps de l'activité, et augmentent le taux de réalisation des objectifs. En outre cette étude ne montre pas les mécanismes et la manière dont ces interfaces à bloc affectent l'apprentissage de la programmation. Dans ce même sens une méta-analyse récente (2019) tente d'examiner l'effet des environnements à bloc versus les environnements textuels sur les résultats affectifs et cognitifs des apprenants, les résultats montrent que les interfaces à bloc sont plus attractives et plus faciles à utiliser, cependant on ignore quelles caractéristiques et possibilités de ces environnements ont une influence positive sur les résultats d'apprentissage cognitifs des élèves ? (Xu, Z. et al., 2019, 26).

Quant aux apprentissages des concepts de base, des études (Meerbaum-Salant, O. et al., 2010, 75 ; Weintrop, D. ; Wilensky, U., 2017, 10) montrent que la programmation par blocs entraîne une meilleure compréhension des concepts élémentaires de la programmation qu'un langage de programmation standard. En termes de résultats d'apprentissage Weintrop, D. et Wilensky, U. (2017, 18-21) ont comparé les deux modalités de programmation texte/blocs pour conclure que les étudiants dans les deux conditions ont amélioré leurs scores entre un pré-test et un post-test, mais les gains d'apprentissages ont été plus importants dans le cas de la programmation par bocs. Toutefois la portée de ces résultats est limitée par des contraintes liées au profil des participants qui sont tous des élèves brillants inscrits dans une école sélective.

Une étude menée par Mladenović, M. et al. (2018, 15) s'est penchée en particulier sur le concept de boucle en programmation, les résultats montrent que le type de langage de programmation influence l'apparition des idées fausses sur les boucles et que ces idées fausses sont minimisées pour les élèves du primaire et secondaire lors de l'utilisation du

langage de programmation basé sur des blocs Scratch, par rapport à l'utilisation de langages de programmation textuels comme Python ou Logo. Cependant certains concepts présentent encore des difficultés, l'étude de Meerbaum-Salant, O. et al. (2010, 75) cite « l'initialisation » « les variables » et « la concurrence », ces concepts semblent être plus abstraits que les autres concepts étudiés dans cette recherche, les auteurs expliquent que le problème réside peut-être dans le fait que ces trois concepts impliquent plusieurs instructions Scratch ce qui nécessite des capacités multi-structurelles et parfois même des capacités relationnelles , alors que les autres concepts peuvent être exprimés par une seule instruction et par conséquent se réalise par des capacités locales et uni-structurelles. Le même constat est établi par Grover, S. et Basu, S. (2017, 272) dans une étude qui montre que les difficultés conceptuelles pour comprendre et utiliser les blocs de construction clés des programmes tels que les variables, les boucles et les conditions persistent, malgré les efforts des concepteurs d'environnements comme Scratch qui ont seulement aidé les apprenants avec les aspects syntaxiques de la programmation, et non les aspects sémantiques / conceptuels (ni stratégiques) de la programmation.

Même si plusieurs recherches ont montré que la programmation basée sur les blocs est un concept très fort pour les cours d'introduction à la programmation, certains chercheurs pensent qu'elle est insuffisante pour aller loin, mais elle peut être considérée comme une passerelle vers l'apprentissage de la programmation textuelle (Noone, M. ; Mooney, A., 2018, 17). Du coté des attitudes des élèves, une étude de Braune, G. et Mühling, A. (2020, 7-8) montre que les élèves perçoivent les environnements graphiques comme n'ayant qu'un but éducatif spécifique et que les capacités de ces outils sont très limitées. De plus, ils ont une conception de la "vraie programmation" qui est en quelque sorte distincte de leurs leçons de programmation en classe. En effet les auteurs soulèvent la question : Comment la programmation peut-elle être enseignée d'une manière compréhensible sans perdre le lien avec le monde réel de l'étudiant ?

4.3 La transition d'un langage graphique à un langage textuel et les langages hybrides

Plusieurs études s'inquiètent de la manière d'aider les élèves à faire la transition à partir de langages graphiques aux langages textuels. Dans une étude de cas italienne, Giordano, D. et Maiorana, F. (2014, 557-562) ont mené une expérience d'introduction à la programmation pour des lycéens entre 14 et 16 ans, l'approche consistait à utiliser en premier un langage visuel à blocs pour se concentrer sur les concepts de base de la programmation et la résolution de problème, quelques semaines après le langage C a été introduit pour donner aux étudiants une expérience appropriée avec un langage textuel. Les résultats montrent que les élèves ont commis moins d'erreurs que ce qu'on pouvait s'attendre lors de la première exposition à un langage exigeant en syntaxe comme C. une autre étude utilisant un langage visuel en conjonction avec un langage textuel sous forme de pseudo code, montre que cette approche permet d'améliorer la confiance, l'indépendance et la résilience des élèves (Dorling, M. ; White, D., 2015, 6).

En considération des résultats encourageants à combiner les environnements visuels et textuels, plusieurs chercheurs ont tenté d'évaluer le concept des environnements hybrides. Dans une étude comparant Scratch et Pocket Code (environnement mobile hybride), des élèves de 13 à 17 ans ont utilisé les deux environnements pour réaliser une tâche qui consiste à manipuler des formules contenant des opérateurs arithmétiques, relationnels et logiques ainsi que des fonctions mathématiques, les résultats montrent que l'approche hybride était plus efficace que l'approche purement visuelle (Koitz, R. ; Slany, W., 2014, 26-29). Le même

constat a été établi par Kyfonidis, C. et al. (2017, 578-579) dans une étude qui compare cette fois l'approche hybride et l'approche purement textuelle, l'expérience consiste à comparer un groupe témoin utilisant le langage C et un groupe expérimental utilisant Block-C (environnement hybride), les auteurs montrent que Block-C favorise la résolution de problèmes, prévient les erreurs de syntaxe, et soutient l'auto-apprentissage de la programmation textuelle par la traduction du code basé sur les blocs en langage C, ce qui aide les apprenants à comprendre la correspondance entre les blocs et les instructions en langage C et leurs syntaxes. Les recherches ont évalué d'autres environnements hybrides comme BlockEditor (Matsuzawa, Y. et al., 2015, 189) qui permet une traduction directe entre les blocs et le langage Java, ainsi que l'environnement Patch (Robinson, W., 2016, 97-98) qui combine des éléments de Scratch et Python. Les résultats de ces recherches encouragent à utiliser l'approche hybride pour les novices en programmation afin de minimiser l'écart entre les langages visuels et les langages textuels et préparer les apprenants à la migration vers les langages professionnels.

4.4 La robotique pédagogique : programmation d'un objet physique versus numérique

La robotique pédagogique est étudiée depuis longtemps pour ses apports cognitifs dans les écoles maternelles avec des travaux pionniers issus directement du courant Logo et des approches constructionnistes créant un contexte de développement idéal pour les jeunes enfants (Papert, S., 1980). Dans ce sens, Misirli, A. et Komis, V. (2013, 1-4) ont mené une recherche portant sur l'utilisation des jouets programmables dans 7 classes de maternelle, 108 enfants âgés de 4 à 6 ans ont participé à cette étude qui consiste à utiliser un scénario en robotique pédagogique basé sur le Bee-Bot, un robot de sol programmable sans ordinateur. Le scénario visait à développer les compétences d'orientation et de direction des enfants en utilisant un système de référence extérieur. Les résultats ont montré que l'utilisation de la robotique pédagogique avait un impact positif sur le développement des compétences de haut niveau des enfants, telles que la résolution de problèmes, l'orientation spatiale et la sensibilisation aux schémas et aux angles, ainsi que sur leur motivation et leur engagement dans l'apprentissage.

Au niveau de l'école élémentaire, une étude par Allain, S. (2019, 79-80) sur deux classes de CM1 et CM2 montre que les robots de sol augmentent la motivation et l'engagement des apprenants, et offrent des perspectives intéressantes pour le développement de leur créativité. Cependant une posture d'accompagnement particulière est nécessaire pour permettre aux apprenants de développer les compétences visées.

Concernant l'apprentissage des concepts de la programmation Nonnon, P. (2002, 41) affirme que programmer un objet tangible permet d'appréhender les concepts étudiés d'une manière plus sensorielle, et donc favoriser l'acquisition de représentations structurantes et pertinentes de ces concepts, du fait que l'apprenant regarde directement et concrètement les résultats de sa programmation par les actions du robot contrairement à la manière traditionnelle ou l'apprenant manipule l'ordinateur ou un environnement informatisé en tant qu'artefact virtuel et intégré, et non pas un matériel embarqué, distinct, et extérieur à l'ordinateur (Nijimbere, C., 2014, 67-68). Bien que ces travaux montrent les avantages de la programmation physique (tangible), il existe aussi des études sur la programmation numérique (objet virtuel) qui ont donné des résultats similaires Major, L. et al. (2014, 34) ont testé une simulation de robot (Kebot) pour une introduction aux concepts de base de la programmation, les participants ont montré un engagement élevé et une grande motivation, et la plupart des tâches proposés ont été achevées à un niveau satisfaisant. Pour déterminer

si une situation est mieux adaptée que l'autre, une étude est conçue dans ce sens par Fessard, G. et al. L'étude compare des gains d'apprentissage lorsqu'un apprenant programme un objet tangible ou sa simulation en se focalisant sur l'apprentissage de trois concepts fondamentaux de la programmation : les variables, les structures conditionnelles, et les structures itératives. Les résultats indiquent qu'il n'y a pas de différence significative entre les deux situations, toutefois l'état de l'art reste très limité en termes de comparaison entre les deux situations au niveau des apprentissages, des stratégies utilisés par les apprenants, et l'évolution de la compréhension des concepts (Fessard, G. et al., 2019, 11-12).

D'autres recherches recommandent des approches combinant l'utilisation d'un environnement visuel à blocs avec un robot tangible. Une étude grecque propose l'utilisation de l'environnement « Enchanting » qui est une variation de Scratch, avec un kit robot Lego Mindstroms, les auteurs montrent que cette approche est bénéfique du fait qu'elle améliore les conditions de l'apprentissage en favorisant l'expérimentation, la création, et l'élaboration des stratégies (Orfanakis, V. ; Papadakis, S., 2014, 272). Une autre étude combinant Lego Mindstroms à App Inventor (application Android de programmation à blocs) montre une grande implication et engagement des élèves dans les activités de programmation proposées (Papadakis, S. ; Orfanakis, V., 2017, 201). En termes de gains d'apprentissage, les étudiants semblent comprendre facilement les concepts de programmation avec Mindstorms. Cependant certaines structures posent encore des difficultés comme while-do ou repeat-until, du fait que la structure répétitive est indirecte. Cette difficulté est constatée dans les recherches menées depuis plus de vingt ans, ce qui semble renforcer l'hypothèse que la difficulté dans ces structures peut être indépendante de l'environnement et du langage de programmation utilisé (Sartatzemi, M. et al., 2005, 511). Comme le précise Alimisis, D. (2013, 68), les gains d'apprentissage ne sont pas garantis par une simple introduction de la robotique en classe, malgré son grand potentiel éducatif, car plusieurs facteurs impactent le résultat, et particulièrement le curriculum.

Si les environnements à blocs combinés aux kits robotiques sont bénéfiques sur le plan motivationnel et créatif, alors sur le plan cognitif des travaux supplémentaires sont nécessaires pour évaluer comment la rétroaction physique d'un robot peut aider à la construction de modèles mentaux chez les élèves (Scott, M. J. et al., 2015, 5).

5. APPROCHES PÉDAGOGIQUES MOBILISÉES DANS LES COURS D'INITIATION À LA PROGRAMMATION

La littérature montre que le choix de l'environnement de programmation et de l'objet à programmer est crucial pour initier les apprenants à la programmation. Cependant le type d'activités proposées présente aussi un choix fondamental qui affecte l'engagement de l'apprenant et favorise son apprentissage (Chi, M. T. H. ; Wylie, R., 2014 ; Romero, M. ; Laferriere, T. ; Power, T. M., 2016 ; cité par Lepage, A. ; Romero, M., 2017, 3).

Malgré les progrès réalisés dans le domaine de la pédagogie et le développement de plusieurs environnements d'apprentissage de la programmation, les apprenants rencontrent toujours des difficultés d'apprentissage, comme le rappellent Buitrago Flórez, F. et al. (2017, 11) dans leur revue de littérature. Les auteurs suggèrent que le problème ne réside peut-être pas dans le langage utilisé pour enseigner la programmation, mais plutôt dans l'approche pédagogique adoptée et le niveau de soutien apporté tout au long du processus d'apprentissage

Le fondement des environnements d'apprentissage de la programmation s'inspire des travaux de Papert et sa vision constructionniste issue des théories pédagogiques de tradition plus ancienne comme le constructivisme de Kelly et de Piaget. Basée sur des idées clés comme « Learning by doing » ou « Learning by making », cette vision de l'apprentissage de la programmation exige des projets et des problèmes authentiques stimulants, pour engager les apprenants dans leur environnement d'apprentissage social, car l'acquisition des connaissances n'est pas une simple question de transmission, d'internalisation ou d'accumulation de savoirs, mais plutôt une question d'engagement actif de l'apprenant dans la construction des connaissances à partir de l'expérience concrète et des informations fournies par l'enseignant (Salomon, G. ; Perkins, D. N., 1996 cité dans Gaudiello, I. ; Zibetti, E., 2013, 23).
Pour améliorer les compétences des apprenants en programmation plusieurs approches d'enseignement ont été explorées, Scherer, R. et al. (2020, 11) montrent dans une méta-analyse que l'apprentissage hybride est le plus répandu dans les recherches consacrées à l'enseignement et l'apprentissage de la programmation, suivi par l'apprentissage par le jeu, puis les stratégies cognitives (comme les cartes conceptuelles), puis les activités collaboratives, ensuite vient la résolution de problèmes et enfin les stratégies de rétroaction.

L'introduction à la programmation par le jeu est une approche très utilisée selon Al-Bow, M. et al. (2009, 104). Deux variantes de cette approche sont distinguées par Muratet, M. et al. (2008, np) : (1) soit les étudiants développent leur propre jeu, (2) soit les étudiants apprennent en jouant à un jeu. Dans le cadre de la première variante Al-Bow, M. et al. (2009, 104-108) ont créé le projet P4Game pour susciter l'intérêt des étudiants du secondaire (14 à 15 ans) à apprendre la programmation dans cadre ludique, interdisciplinaire, et basé sur des projets. L'étude s'est déroulée dans un camp d'été durant 2 semaines en utilisant l'environnement Greenfoot (un logiciel de programmation de jeux vidéo 2D), 26 participants ont répondu à une enquête avant et après le jeu. Les résultats montrent une amélioration significative dans les connaissances en programmation, et une grande confiance en soi. D'autres études ont expérimenté la deuxième variante de l'apprentissage de la programmation par le jeu, nous citons à titre d'exemple l'étude de Debabi, W.; Bensebaa, T. (2016, 133-137) qui ont utilisé un jeu vidéo nommé AlgoGame conçu spécialement pour les novices afin de faciliter l'apprentissage des concepts fondamentaux de la programmation (les séquences, les boucles, les conditions, etc..) selon une approche de résolution de problème. Le jeu est sous forme d'un ensemble de missions, chacune correspond à un concept de la programmation, ou à une combinaison de concepts. Le joueur visualise la transformation de ses actions en algorithme dans la partie droite de l'écran, ce qui permet de concrétiser les concepts abstraits. Les auteurs ont testé ce jeu dans l'objectif d'enseigner l'algorithme de tri, 33 étudiants divisés en deux groupes ont participé à cette expérimentation. Le premier groupe (témoin) a été invité à écrire un algorithme de tri sans connaitre le jeu, seule une explication du principe de l'algorithme a été donnée. Le deuxième groupe (expérimental) a été invité à écrire l'algorithme de tri après avoir joué à AlgoGame durant 30 min. les résultats montrent que le groupe expérimental a eu une meilleure moyenne par rapport au groupe témoin.

Les activités de collaboration dans les cours d'initiation à la programmation ont été sujet de plusieurs études qui tentent d'améliorer l'apprentissage des compétences de la pensée informatique. Dans une revue systématique, Vihavainen, A. et al. (2014, 22)distinguent trois approches identifiées dans la littérature, à savoir : les activités de programmation en binômes,

les activités d'apprentissage en équipe, et les pratiques coopératives. Les auteurs constatent que les études basées sur la collaboration ou le soutien par les pairs ont généralement obtenu une grande amélioration dans les taux de réussite par rapport aux études basées sur d'autres approches. Toutefois la réussite des approches collaboratives dépend de certains attributs essentiels, Preston, D. (2006, 16-17) a identifié cinq attributs : (1) la tache commune, (2) l'apprentissage en petits groupes de 2 à 6 personnes, (3) le comportement coopératif, (4) l'interdépendance positive, (5) l'obligation et la responsabilité individuelle. L'auteur affirme que la programmation en binôme répond bien à ces attributs. Dans la programmation en binôme deux élèves travaillent en collaboration pour réaliser un programme sur un ordinateur. L'élève qui joue le rôle du "conducteur" est responsable de la tâche de codage. L'autre élève, appelé "observateur", recherche les éventuelles déficiences tactiques et stratégiques du programme afin de l'affiner (Williams, L.A.; Kessler, R.R., 2001, 7-8). Cette approche a donné des résultats prometteurs dans l'apprentissage de la programmation informatique, prouvés par l'expérience de Williams, L.A.; Kessler, R.R. (2001, 13-17). Les auteurs ont expérimenté la programmation par binôme dans une classe de 41 étudiants de 16 à 18 ans. Les participants ont été divisé en deux groupes, un groupe (expérimental) de 28 étudiants qui ont travaillé en binôme et un groupe (témoin) de 13 étudiants qui ont travaillé individuellement. Les résultats montrent que la programmation en binôme améliore la productivité et la qualité de la programmation par rapport à l'apprentissage individuel traditionnel. Elle a également réduit la charge de travail des enseignants, car les étudiants se tournaient souvent les uns vers les autres pour obtenir de l'aide. Ces résultats sont confirmés par Preston, D. (2006, 16), dans sa revue de littérature l'auteur constate que la pédagogie de programmation par binôme est efficace, les recherches examinées montrent qu'elle permet de produire des programmes de meilleure qualité, de réduire le temps nécessaire à la réalisation des programmes, de mieux comprendre du processus de programmation, d'augmenter le plaisir d'apprendre la programmation, de diminuer la dépendance au personnel enseignant, d'améliorer le taux de réussite des cours, et d'améliorer les performances à l'examen. Toutefois cette approche a pour inconvénient le risque que l'élève le plus fort fasse la majorité ou la totalité du travail, pour cela Preston, D. (2006, 19) recommande la nécessité d'observer les étudiants pendants leur travail en binôme afin d'encourager l'élève le plus faible à s'impliquer davantage dans la tâche.

Les bénéfices de l'apprentissage de la programmation basé sur la résolution des problèmes sont approuvés dans l'étude de Bai, H. et al. (2021, 5-10) qui ont comparé deux groupes d'élèves de la huitième année du collège, le premier groupe a suivi un cours de programmation Python selon une approche POL (Problem-Oriented Learning) alors que le deuxième groupe a adopté une approche LAP (Lecture-And-Practice). L'évaluation porte sur les concepts, les pratiques, et les perspectives de la pensée computationnelle. Les concepts évalués inclus les séquences, les boucles, les conditions, les données, les fonctions, et les opérateurs. Les pratiques inclus le test, le débogage, la réutilisation et le remixage. Quant aux perspectives évaluées dans cette étude, elles comprennent l'expression, la connexion, et le questionnement. Les résultats montrent que les élèves du premier groupe ont obtenu des résultats mieux que ceux du deuxième groupe en termes de concepts et perspectives de la pensée computationnelle, cependant en termes de pratiques la différence n'été pas significative.

La programmation est un outil de modélisation de connaissances d'un grand potentiel créatif et (méta)cognitif (Romero, M., 2016, 88). Cependant, Romero, M. et DeBlois, L. (2022, 4) rappellent « que certaines activités de programmation n'engagent pas les élèves dans des démarches de résolution de problèmes car elles consistent en une suite d'étapes prédéterminées, comme certains tutoriels de Code.org. Il s'agit dans ce cas d'activités de programmation guidée et non créative. Dans ce contexte, les élèves sont confrontés à des problèmes auxquels ils peuvent appliquer des connaissances acquises pour trouver une solution. D'autres activités de programmation permettent aux élèves de disposer d'une marge créative pour analyser et développer une solution originale par le biais d'un programme informatique de manière débranchée (unplugged computing), sur écran ou sur des artefacts programmables comme des robots pédagogiques. » Dans ce sens, il convient de rappeler que la dimension émotionnelle est à considérer dans la conception des situations d'apprentissage de manière à engager l'apprenant à relever un défi avec un sentiment de plaisir, ce que Papert appelle une situation de « Hard Fun » (Jaillet, A., 2019, 3). L'utilisation des environnements d'apprentissage de la programmation avec des méthodes pédagogiques qui ne sont pas propices à ces idées, ne permettrai pas de tirer profit du potentiel éducatif de ces environnements (Xu, Z. et al., 2019, 27), Car il est inutile, d'un point de vue pédagogique, d'introduire les technologies disponibles dans le système éducatif alors que les stratégies éducatives ne sont pas convenablement révisées (De Corte,E., 1996 cité dans Gaudiello, I. ; Zibetti, E., 2013, 34).

Dans la revue systématique de Xu, Z. et al. (2019, 27), les auteurs trouvent que la littérature a peu mis l'accent sur la description des stratégies pédagogiques utilisées pour intégrer la programmation visuelle (blocs) dans l'apprentissage non formel et formel. En général, les approches pédagogiques sollicitées dans le cadre scolaire s'inscrivent particulièrement dans une tradition constructiviste et socioconstructiviste de l'apprentissage comme l'approche de l'environnement Logo, alors qu'apparaissent de nouveaux robots de sol, l'usage d'objets tangibles doit être accompagné des situations – problèmes adéquates et des outils appropriés (Pekarova, J., 2008, 112 ; Spach, M., 2017, 41).

6. SYNTHÈSE DU TRAVAIL

Dans cette partie présentons une synthèse de l'état de l'art autour de l'enseignement et l'apprentissage de la programmation. Nous commençons par une comparaison des études incluses dans ce travail de manière à mettre en lumière leurs points de convergence et de divergence en termes de problématiques traitées et des méthodes adoptées. Et nous terminons par la présentation des perspectives de recherches futures et les éventuels questionnements qui en découlent.

6.1 Convergences et divergences des recherches incluses dans ce travail

Au niveau des problématiques qui ont été soulevées dans la littérature autour de l'enseignement et l'apprentissage de la programmation, plusieurs thématiques ont été abordées. Certaines études (Papert, 1980 ; Wing, 2006 ; Selby, 2013 ; Voogt, 2015 ; Buitrago, 2017 ; Drot-Delange, 2019) se sont penchées sur le concept de la pensée informatique et son lien étroit avec l'activité de programmation. Bien que les débats autour de cette question convergent vers le fait que la pensée informatique est bien plus que la programmation, la littérature montre que les recherches qui abordent la pensée informatique se limitent souvent à des contextes de programmation. Toutefois il n'existe pas de consensus sur la définition des

compétences impliquées dans la conceptualisation de la pensée informatique ce qui constitue un obstacle devant son opérationnalisation dans des activités concrètes (Chen, G. et al., 2017, 14). Cette problématique ouvre un autre champ d'investigation connexe qui se concentre sur la manière d'évaluer les compétences liées à la pensée informatique (De Araujo et al, 2016, 1). Diverses approches sont abordées par les chercheurs, certains ont adopté des approches quantitatives comme le questionnaire à choix multiples (De Araujo et al., 2016, 5) et le calcul automatique des scores en utilisant l'outil web Dr.Scratch (Moreno-León, J. et al., 2015, 1-23). Cependant ces approches semblent être limitées, d'une part il est difficile d'élaborer un questionnaire approprié au processus cognitif à mesurer, et d'autre part cette approche ne permet pas de comprendre le raisonnement de l'apprenant et son processus de pensée (De Araujo et al., 2016, 7). Quant aux outils d'évaluation automatiques, Romero, M. et al (2017, 6) reprochent le fait que ce genre d'outils ne prend pas en compte la signification du programme analysée, ni la tâche demandée, ni la dimension créative de la programmation. D'autres études ont adopté des approches qualitatives comme l'analyse des réponses aux exercices de programmation, les entretiens, les questionnaires ouverts, les journaux vidéo,… (De Araujo et al., 2016, 5). Toutefois il existe aussi des recherches qui combinent plusieurs approches et instruments d'évaluation pour couvrir les différents aspects de la pensée informatique. (De Araujo, A.L.S.O et al., 2016, 5 ; Lepage, A. ; Romero, M., 2017, 8-12). Tant qu'il n'y a pas de consensus sur les composantes de la pensée informatique, la question de l'évaluation reste ouverte : quelles compétences évaluer ? et comment ?

L'évaluation de la pensée informatique à travers la programmation a pour objectif principal de juger le progrès des individus, les programmes d'études, le matériel, et les pédagogies mobilisées (National Research Council, 2011). Cela permettrait de mettre en lumière les enjeux de l'enseignement et l'apprentissage de la programmation, et d'ouvrir la voie pour des questionnements qui s'imposent autour des difficultés fréquentes dans les cours d'introduction à la programmation, ainsi que les approches pédagogiques et les dispositifs d'apprentissages adaptés pour soutenir cet apprentissage, notamment avec l'émergence d'une multitude d'environnements de programmation. Les recherches ont abordé ces questionnements à travers des cadres de référence multidisciplinaires. D'un point de vue des sciences cognitives, la programmation est une activité complexe qui demande la mémorisation de plusieurs informations et la mobilisation de plusieurs compétences en même temps ce qui impose une charge cognitive exigeante pour les novices (Mow, I.T.C., 2008, 2) et en particulier pour les étudiants du primaire et du secondaire qui (par considération de leur âge) n'ont pas encore développé les capacités de réflexion suffisantes qui permettent de comprendre les concepts abstraits de la programmation et les appliquer pour créer des algorithmes concrets de résolution de problèmes (Mladenović, M. et al., 2018, 3), d'autres recherches indiquent que la focalisation sur la syntaxe du langage de programmation avant d'avoir un modèle mental de la construction des algorithmes amène les étudiants à un apprentissage superficiel qui ne permet pas de résoudre les problèmes de manière efficace (Siti Rosminah, M. D. et al., 2012, 2). Le manque de motivation et d'engagement présente aussi l'un des obstacles fréquents constaté dans plusieurs études (Medeiros, R. P. et al., 2019, 11).

Pour pallier ces difficultés, plusieurs études se sont penchées sur des problématiques liées aux environnements d'apprentissages de la programmation. D'un point de vue de l'ergonomie cognitive les environnements graphiques dites aussi à block permettent de diminuer

l'ampleur des difficultés rencontrées, d'une part ces environnements évitent la surcharge cognitive en éliminant les soucis de syntaxe, et d'autre part leur aspect ludique permet de stimuler la motivation et l'engagement des apprenants (Duguay, S., 2018, 26 ; Ouahbi, I. et al., 2015, 4).

Malgré la divergence des thématiques traitées et des méthodes utilisées dans les études examinées dans cette revue de littérature, nous remarquons qu'elles sont tout de même liées. Les problématiques de chaque thème finissent par ouvrir la voie pour de nouveaux champs d'exploration et de nouvelles thématiques connexes.

6.2 Perspectives de recherches

Les études consultées dans cette revue de littérature montrent que plusieurs perspectives de recherche restent à explorer. On ignore encore les caractéristiques et les possibilités des environnements graphiques qui ont une influence positive sur les résultats d'apprentissage, les effets à long terme de ces environnements sur la capacité des programmeurs novices à passer à des situations de programmation en mode texte (par exemple, des cours ultérieurs ou une carrière), et les différences individuelles (par exemple, la capacité de mémoire de travail, le sexe, l'âge) parmi les programmeurs novices qui correspondent le mieux aux conditions d'apprentissage fournies par les environnements d'apprentissage par blocs (Xu, Z. et al., 2019, 26). Une étude de Braune, G. et Mühling, A. (2020, 7-8) montre que les apprenants novices ont une conception de « la vraie programmation » distinctes de ce qu'ils perçoivent dans les environnements de programmation graphique. Ce constat soulève des questions comme : Pourquoi de nombreux étudiants novices en programmation informatique ont-ils une attitude défavorable à l'égard des environnements de programmation par blocs par rapport à leurs homologues en mode texte ? (Xu, Z. et al., 2019, 26). Comment la programmation peut-elle être enseignée d'une manière compréhensible sans perdre le lien avec le monde réel de l'étudiant ? (Braune, G. ; Mühling, A., 2020, 7-8). La question des prérequis et des représentations cognitives préexistant et évoluant chez les élèves ne semble pas avoir fait l'objet de beaucoup d'études et reste un champ important à explorer, en liaison avec les images mentales. (Grandbastien, 2020, 20)

Au-delà du débat autour des langages textuel et visuel, le pilotage des robots, et autres objets tangibles est une autre modalité d'initiation à la programmation qui mérite l'attention des chercheurs. Cependant l'état de l'art reste très limité en termes de comparaison entre la programmation d'un objet tangible et la programmation d'une simulation au niveau des apprentissages, des stratégies utilisés par les apprenants, et l'évolution de la compréhension des concepts (Fessard et al., 2019, 12)

7. TENDANCE DES RECHERCHES RECENTES NON PRISES EN COMPTE DANS CETTE REVUE

Après avoir analysé cinq articles récents publiés entre 2021 et 2022 portant sur l'enseignement et l'apprentissage de la programmation informatique, nous constatons que les sujets de recherche examinés dans cette étude sont toujours d'actualité. Les recherches se concentrent sur divers aspects, tels que l'utilisation de la robotique éducative et des outils pédagogiques tels que les robots de sol, l'identification des difficultés rencontrées par les apprenants dans les cours d'initiation à la programmation et par les futurs enseignants dans

l'enseignement de l'informatique, ainsi que l'élaboration de méthodes d'apprentissage et la conception de scenarii pédagogiques adaptés pour aider les apprenants à mieux comprendre les concepts de base de la programmation et à améliorer leur capacité à résoudre des problèmes.

Gratani, F. et al. (2021, 5-12) ont étudié l'impact de la robotique éducative sur le développement des compétences de résolution de problèmes dans une classe de quatrième année primaire italienne. Avec une approche interdisciplinaire et multimodale qui encourage la collaboration et le travail sur différents domaines de connaissances. La conception du parcours renforce l'alternance de certains méthodes d'apprentissage : théorie/pratique, leçon frontale/apprentissage coopératif, et dimension analogique/numérique. Les résultats montrent une amélioration significative des compétences de résolution des problèmes chez les élèves.

Une étude en Grèce par Nikolos, D. et al. (2021, 65-73) s'est penché sur le processus de débogage dans une nouvelle simulation d'environnement robotique qui offre des fonctionnalités de débogage telles que l'exécution rapide des commandes, la possibilité de supprimer une commande ou bien une trace. Deux enfants de 7 ans ont utilisé l'environnement et leurs activités ont été enregistrées. Les enfants devaient déplacer un robot simulé à travers un labyrinthe en utilisant des commandes de déplacement et de direction similaires à Logo. Les résultats montrent que enfants ont utilisé efficacement les fonctionnalités de débogage de l'environnement et ont résolu les problèmes qui leur étaient assignés. Ils ont suivi les stratégies de débogage telles que l'identification du bogue, le repérage du segment de code qui pose un problème et la correction du programme. L'instructeur a pu les aider dans le processus lorsque cela était nécessaire.

Dans une étude récente par Valorge, M. (2022, 49-60) menée dans une classe de CE1 en France qui utilise le robot pédagogique Ozobot pour enseigner la programmation informatique, les enseignants ont créé des scénarios et des supports pédagogiques pour aider les élèves à travailler en petits groupes et en autonomie en suivant une démarche d'investigation. L'auteure a utilisé une méthode d'analyse de tâches pour étudier l'activité de deux élèves travaillant avec le robot Ozobot. Elle a examiné les conditions qui ont facilité ou entravé leur activité d'apprentissage en utilisant les concepts d'information et d'affordance. Les résultats préliminaires ont montré que l'utilisation du robot Ozobot en classe offre des possibilités d'apprentissage intéressantes pour les élèves en leur fournissant des expériences pratiques de programmation et en favorisant leur engagement et leur motivation. Cependant, l'étude souligne également certains défis liés à l'utilisation de cet outil pédagogique, notamment la difficulté à trouver des scénarios pédagogiques appropriés pour les élèves et la nécessité de former les enseignants à l'utilisation de l'outil.

Bugmann, J. et al. (2022, 61-76) se sont penchés sur les difficultés d'enseignement et d'apprentissage de l'informatique, notamment la programmation et l'algorithmique au primaire. Les résultats de la recherche montrent que les futurs enseignants ont des besoins de formation en informatique pour être en mesure d'enseigner efficacement aux élèves. Les enseignants ont du mal à trouver du sens à ces activités et souffrent d'un manque de confiance en eux, ce qui se traduit par une maîtrise insuffisante des concepts à enseigner. Les élèves, quant à eux, éprouvent des difficultés à se décentrer par rapport à l'activité, à trouver du sens

et à transposer leurs apprentissages dans un autre contexte. Les futurs enseignants privilégient les activités sans écran pour les élèves de premier cycle, mais la recherche montre que cela ne suffit pas pour enseigner efficacement l'informatique.

En s'appuyant sur les instructions des théories de la charge cognitive et du transfert des apprentissages, Goletti, O. et al. (2022, 77-85) proposent une méthode d'apprentissage de la programmation avec Python basée sur la création d'exemples résolus avec des objectifs étiquetés. Cette méthode vise à aider les apprenants à mieux comprendre les concepts de base de la programmation et à développer leur capacité à résoudre des problèmes informatiques. Les résultats montrent que la création d'exemples résolus avec objectifs étiquetés peut être une méthode efficace pour enseigner la programmation avec Python et pour aider les apprenants à acquérir des compétences pratiques en résolution de problèmes informatiques

8. CONCLUSION

Les articles examinés dans cette revue de littérature convergent tous vers l'importance d'enseigner la pensée informatique aux élèves dès le plus jeune âge et mettent en évidence les bénéfices de cette pratique pour les élèves, notamment en termes de développement des compétences cognitives qui peuvent être transférables dans de nombreux domaines tels que la résolution de problèmes, la pensée critique, et la créativité. Néanmoins, il existe des divergences dans les recherches au niveau du contexte et la population ciblée, les approches pédagogiques adoptées, et les résultats obtenus.

Plusieurs approches pédagogiques sont présentées dans les recherches, telles que la création de jeux, la robotique éducative, l'utilisation d'un environnement de programmation textuel, visuel ou bien hybride, ou encore l'utilisation de jeux sérieux.

Au niveau des résultats obtenus, certaines études ont montré que l'enseignement de la programmation peut avoir des effets positifs sur les résultats scolaires, tandis que d'autres ont montré que les effets sur les résultats scolaires sont limités ou inexistants. De plus, les études sur les avantages à long terme de l'apprentissage de la programmation sont encore limitées. Ces divergences permettent d'ouvrir la voie pour d'autres questionnements à savoir : Si les étudiants apprennent mieux avec la programmation par blocs, alors quelles sont les caractéristiques et les possibilités de ces environnements qui ont une influence positive sur les résultats d'apprentissage ? Quels sont les effets à long terme des environnements de programmation par blocs sur la capacité des programmeurs novices à passer à des situations de programmation en mode texte (par exemple, des cours ultérieurs ou une carrière) ? Pourquoi de nombreux étudiants novices en programmation informatique ont-ils une attitude défavorable à l'égard des environnements de programmation par blocs par rapport à leurs homologues en mode texte ? Y a-t-il des différences individuelles (par exemple, la capacité de mémoire de travail, le sexe, l'âge) parmi les programmeurs novices qui correspondent le mieux aux conditions d'apprentissage fournies par les environnements d'apprentissage par blocs ? (Xu, Z. et al., 2019, 26).
Quant à la robotique tangible, certaines recherches ont montré son efficacité sur le plan affectif et cognitif, alors que d'autres trouvent que l'effet de la programmation d'un objet ou sa simulation ne présente pas de différence significative (Fessard, G. et al., 2019, 11), on

ignore encore les stratégies mises en place par les apprenants pour résoudre les exercices proposés dans chaque situation, et l'évolution de la compréhension des concepts étudiés à travers les actions réalisées par chaque apprenant.

Il convient également de noter que les recherches sur l'enseignement de la programmation à l'école ont souvent été menées dans des contextes différents, avec des populations différentes, ce qui peut rendre difficile la généralisation des résultats. Il est donc important de continuer à mener des recherches pour mieux comprendre les avantages et les limites de l'enseignement de la programmation à l'école.

En somme, bien que tous les articles reconnaissent l'importance de l'enseignement de la programmation à l'école, ils ont des objectifs et des approches différents pour y parvenir, ce qui reflète la complexité et la diversité du domaine de la programmation informatique et de l'enseignement de la pensée computationnelle.

9. RÉFÉRENCES BIBLIOGRAPHIQUES

Al-Bow, M.; Austin, D.; Edgington, J.; Fajardo, R.; Fishburn, J.; Lara, C.; Leutenegger, S.; Meyer, S. (2009). Using game creation for teaching computer programming to high school students and teachers. *ACM SIGCSE Bulletin*, 41(3), 104-108. https://doi.org/10.1145/1595496.1562913

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

Allain, S. (2019). Créativité, pensée informatique et robotique, quels processus mis en œuvre par les élèves ? 113. https://hal-univ-fcomte.archives-ouvertes.fr/hal-02325425/document

Bai, H.; Wang, X.; Zhao, L. (2021). Effects of the Problem-Oriented Learning Model on Middle School Students' Computational Thinking Skills in a Python Course. *Frontiers in Psychology*, 12, 771221. https://doi.org/10.3389/fpsyg.2021.771221

Baron, G.-L. (1990). Note de synthèse [L'informatique en éducation - Le cas de la France] : L'informatique en éducation - Le cas de la France. *Revue française de pédagogie*, 92(1), 57-77. https://doi.org/10.3406/rfp.1990.2474

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

Bers, M. U. (2017). The Seymour test : Powerful ideas in early childhood education. *International Journal of Child-Computer Interaction, 14, 10-14. https://doi.org/10.1016/j.ijcci.2017.06.004*

Braune, G.; Mühling, A. (2020). Learning to program : The gap between school world and everyday world. Proceedings of the 15th Workshop on Primary and Secondary Computing Education, 1-9. https://doi.org/10.1145/3421590.3421597

Bugmann, J. ; Chevalier, M. ; Pellet, J-P. ; Parriaux, G. (2022). Difficultés d'enseignement et d'apprentissage de la science informatique au primaire. L'informatique, objets d'enseignement et d'apprentissage. Quelles nouvelles perspectives pour la recherche ? May 2022, Le Mans, France. pp.61-76. ⟨hal-03697924⟩

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

Busana, G.; Denis, B.; Duflot-Kremer, M.; Higuet, S.; Kataja, L.; Kreis, Y.; Laduron, C.; Meyers, C.; Parmentier, Y.; Reuter, R.; Weinberger, A. (2020, février). PIAF : Développer la Pensée Informatique et Algorithmique dans l'enseignement Fondamental. *Didapro 8 – DidaSTIC – L'informatique, objets d'enseignements – enjeux épistémologiques, didactiques et de formation*. https://hal.archives-ouvertes.fr/hal-02463940

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

Crahay, M. (1987). Logo, un environnement propice à la pensée procédurale. *Revue française de pédagogie*, 80(1), 37-56. https://doi.org/10.3406/rfp.1987.1473

CSTA; ISTE. (2011). Computational_Thinking_Operational_Definition_ISTE. https://cdn.iste.org/www-root/Computational_Thinking_Operational_Definition_ISTE.pdf

De Araujo, A. L. S. O. ; Andrade, W. L. ; Serey Guerrero, D. D. (2016). A systematic mapping study on assessing computational thinking abilities. 2016 IEEE Frontiers in Education Conference (FIE), 1-9. https://doi.org/10.1109/FIE.2016.7757678

Debabi, W.; Bensebaa, T. (2016). USING SERIOUS GAME TO ENHANCE ALGORITHMIC LEARNING AND TEACHING. *Journal of E-Learning and Knowledge Society*, 12(2). https://doi.org/10.20368/1971-8829/1125

Department of Education UK (2013). National curriculum in England: Computing programmes of study: key stages 3 and 4. https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/239067/SECONDARY_national_curriculum_-_Computing.pdf

Dorling, M.; White, D. (2015). Scratch : A Way to Logo and Python. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 191-196. https://doi.org/10.1145/2676723.2677256

DROT-DELANGE, B. ; PELLET, J.-P. ; DELMAS-RIGOUTSOS, Y. ; BRUILLARD, R. (2019). Pensée informatique : Points de vue contrastés. *Rubrique de la Revue STICEF*, Volume 26, numéro 1, 2019, DOI :10.23709/sticef.26.1.1

Duguay, S. (2018). Conception d'une grille d'analyse des langages de programmation utilisés en algorithmique dans le programme Techniques de l'informatique. 142. http://hdl.handle.net/11143/14037

Grandbastien, M. (2020). Quelle actualité pour les questions abordées lors du premier colloque de didactique de l'informatique (1988) ? L'informatique, objets d'enseignements enjeux épistémologiques, didactiques et de formation. Colloque DIDAPRO 8 -DIDASTIC. hal-02474983

Gratani, F.; Giannandrea, L.; Renieri, A.; Annessi, M. (2021). Fostering Students' Problem-Solving Skills Through Educational Robotics in Primary School. In M. Malvezzi, D. Alimisis, & M. Moro (Éds.), *Education in & with Robotics to Foster 21st-Century Skills* (Vol. 982, p. 3-14). Springer International Publishing. https://doi.org/10.1007/978-3-030-77022-8_1

Fessard, G.; Wang, P. ; Renna, I. (2019). Objet Tangible ou Simulation Numérique : Deux Situations Équivalentes pour l'Apprentissage de la Programmation *? Environnements Informatiques pour l'Apprentissage Humain*, Jun 2019, Paris, France. https://hal.archives-ouvertes.fr/hal-02156174/document

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

Giordano, D. ; Maiorana, F. (2014). Use of cutting edge educational tools for an initial programming course. *2014 IEEE Global Engineering Education Conference (EDUCON)*, 556-563. https://doi.org/10.1109/EDUCON.2014.6826147

Goletti, O. ; De Pierpont, F. ; Mens, K. (2022). Création d'exemples résolus avec objectifs étiquetés pour l'apprentissage de la programmation avec Python. *L'informatique, objets d'enseignement et d'apprentissage. Quelles nouvelles perspectives pour la recherche ?* May 2022, Le Mans, France. Pp.77-85. ⟨hal-03697932⟩

Gomes, A.; Mendes, A. (2007). Learning to program—Difficulties and solutions. *International Conference on Engineering Education, Coimbra*, Portugal. http://icee2007.dei.uc.pt/proceedings/papers/411.pdf

Grover, S.; Basu, S. (2017). Measuring Student Learning in Introductory Block-Based Programming : Examining Misconceptions of Loops, Variables, and Boolean Logic. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 267-272. https://doi.org/10.1145/3017680.3017723

Hart, C., (2009). *Doing a literature review: Releasing the social science research imagination*, LA/London, Sage.

Jaillet, A. (2019). Cerveau, émotions et bonheurs : Apprendre par le « Hard Fun ». *Tréma*, 52. https://doi.org/10.4000/trema.5340

Koitz, R.; Slany, W. (2014). Empirical Comparison of Visual to Hybrid Formula Manipulation in Educational Programming Languages for Teenagers*. Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools*, 21-30. https://doi.org/10.1145/2688204.2688209

Kyfonidis, C., Moumoutzis, N. ; Christodoulakis, S. (2017). Block-C : A block-based programming teaching tool to facilitate introductory C programming courses. *2017 IEEE Global Engineering Education Conference (EDUCON)*, 570-579. https://doi.org/10.1109/EDUCON.2017.7942903

Lepage, A. ; Romero, M. (2017). Évaluation par compétences d'activités de programmation créative avec l'outil #5c21. In *Actes du colloque CIRTA 2017* (Vol. 1). UQAM, Québec : CRIRES. http://alepage.net/wp-content/uploads/2020/08/CIRTA2017-PI-Evaluation-LepageRomero2017-R07.pdf

Major, L.; Kyriacou, T.; Brereton, P. (2014). The effectiveness of simulated robots for supporting the learning of introductory programming : A multi-case case study. *Computer Science Education*, 24(2-3), 193-228. https://doi.org/10.1080/08993408.2014.963362

Matsuzawa, Y. ; Ohata, T. ; Sugiura, M. ; Sakai, S. (2015). Language Migration in non-CS Introductory Programming through Mutual Language Translation Environment. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 185-190. https://doi.org/10.1145/2676723.2677230

Medeiros, R. P.; Ramalho, G. L.; Falcao, T. P. (2019). A Systematic Literature Review on Teaching and Learning Introductory Programming in Higher Education. *IEEE Transactions on Education*, 62(2), 77-90. https://doi.org/10.1109/TE.2018.2864133

Meerbaum-Salant, O.; Armoni, M.; Ben-Ari, M. (Moti). (2010). Learning computer science concepts with Scratch. *Proceedings of the Sixth International Workshop on Computing Education Research* - ICER '10, 69. https://doi.org/10.1145/1839594.1839607

Misirli, A. ; Komis, V. (2013). Construire les notions de l'orientation et de la direction à l'aide des jouets programmables : une étude de cas dans des écoles maternelles en Grèce.

Colloque international éTIC – Les TICE à l'école primaire A l'université de Limoges, Faculté des Lettres et des Sciences Humaines. https://doi.org/10.4000/books.septentrion.15138

Mladenović, M.; Boljat, I.; Žanko, Ž. (2018). Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level. *Education and Information Technologies*, 23(4), 1483-1500. https://doi.org/10.1007/s10639-017-9673-3

Moreno-León, J.; Robles, G.; Román-González, M. (2015). Dr. Scratch : Automatic Analysis of Scratch Projects to Assess and Foster Computational Thinking. *RED-Revista de Educación a Distancia.* 46. 1-23. https://www.um.es/ead/red/46/moreno_robles.pdf

Mow, I. T. C. (2008). Issues and Difficulties in Teaching Novice Computer Programming. In M. Iskander (Éd.), *Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education*. 199-204. Springer Netherlands. https://doi.org/10.1007/978-1-4020-8739-4_36

Muratet, M. ; Torguet, P. ; Jessel, J.-P. ; Vallet, F. (2008). Vers un jeu sérieux pour enseigner la programmation. AFRV 2008, Oct 2008, Bordeaux, France. https://core.ac.uk/download/pdf/50530308.pdf

Nijimbere, C. (2014). Apprendre l'informatique par la programmation des robots : Cas de Nao. *Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation*, 21(1), 63-109. https://doi.org/10.3406/stice.2014.1091

Nikolos, D., Misirli, A. ; Komis, V. (2021). Children's Debugging Processes and Strategies with a Simulated Robot : A Case Study. In M. Malvezzi, D. Alimisis, & M. Moro (Éds.), *Education in & with Robotics to Foster 21st-Century Skills.* (982). 64-74. Springer International Publishing. https://doi.org/10.1007/978-3-030-77022-8_6

Nonnon, P. (2002). Robotique pédagogique et formation de base en science et technologie. *Aster*, 34(34, p. 213). https://doi.org/10.4267/2042/8787

Noone, M.; Mooney, A. (2018). Visual and textual programming languages : A systematic review of the literature. *Journal of Computers in Education*, 5(2), 149-174. https://doi.org/10.1007/s40692-018-0101-5

Orfanakis, V.; Papadakis, S. (2014). A new programming environment for teaching programming. A first acquaintance with Enchanting. *international virtual Scientific Conference*. https://www.academia.edu/7576980/A_new_programming_environment_for_teaching_programming_A_first_acquaintance_with_Enchanting

Ouahbi, I. ; Darhmaoui, H. ; Kaddari, F. ; Elachqar, A. ; Lahmine, S. (2015). Vers un enseignement de la programmation compatible avec la culture vidéoludique des élèves au Maroc. 7ème Conférence sur les Environnements Informatiques pour l'Apprentissage Humain (EIAH 2015), 405-407. https://hal.archives-ouvertes.fr/hal-01413631/document

Papadakis, S.; Orfanakis, V. (2017). The Combined Use of Lego Mindstorms NXT and App Inventor for Teaching Novice Programmers. In D. Alimisis, M. Moro, & E. Menegatti (Éds.), *Educational Robotics in the Makers Era*. (560). 193-204. Springer International Publishing. https://doi.org/10.1007/978-3-319-55553-9_15

Papert, S. (1980). *Mindstorms : Children, computers, and powerful ideas*. Basic Books. http://worrydream.com/refs/Papert%20-%20Mindstorms%201st%20ed.pdf

Papert, S. ; Jaillet, A. (2006, octobre 5). Un défi de taille pour l'école. *Les Cahiers pédagogiques*. https://www.cahiers-pedagogiques.com/un-defi-de-taille-pour-l-ecole/

Pea, R. D. ; Kurland, D. M.; Hawkins, J. (1985). LOGO and the Development of Thinking Skills.

In Chen, M. ; Paisley, W. *Children and Microcomputers: Research on the Newest Medium, Sage*.193-317. https://telearn.archives-ouvertes.fr/hal-00190534/document

Pekarova, J. (2008). Using a Programmable Toy at Preschool Age : Why and How?. Workshop Proceedings of SIMPAR 2008. Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS. Venice(Italy) 2008 November. 112-121 https://www.terecop.eu/downloads/simbar2008/pekarova.pdf

Preston, D. (2006). Using collaborative learning research to enhance pair programming *pedagogy. ACM SIGITE Newsletter*. 3(1). 16-21. https://doi.org/10.1145/1113378.1113381

Price, T. W.; Barnes, T. (2015). Comparing Textual and Block Interfaces in a Novice Programming Environment. Proceedings of the Eleventh Annual International Conference on International Computing Education Research - ICER '15, 91-99. https://doi.org/10.1145/2787622.2787712

Ramalingam, V.; LaBelle, D.; Wiedenbeck, S. (2004). Self-efficacy and mental models in learning to program. Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education - ITiCSE '04, 171. https://doi.org/10.1145/1007996.1008042

Robinson, W. (2016). From Scratch to Patch : Easing the Blocks-Text Transition. Proceedings of the 11th Workshop in Primary and Secondary Computing Education, 96-99. https://doi.org/10.1145/2978249.2978265

Rogalski, J. (1987). Acquisition de savoirs et de savoir-faire en informatique. *Cahier de didactique des mathématiques*. (43). https://publimath.univ-irem.fr/numerisation/PS/IPS87004/IPS87004.pdf

Romero, M. (2016). De l'apprentissage procédural de la programmation à l'intégration interdisciplinaire de la programmation créative. Formation et profession : *revue scientifique internationale en éducation*, 24(1), 87-89. https://doi.org/10.18162/fp.2016.a92

Romero, M. ; Lepage, A. ; Lille, B. (2017). Computational thinking development through creative programming in higher education*. International Journal of Educational Technology in Higher Education*, 14(1), 42. https://doi.org/10.1186/s41239-017-0080-z

Romero, M. ; DeBlois, L. (2022). Analyse du processus de construction de connaissances dans des activités de programmation à l'école. *Canadian Journal of Science, Mathematics and Technology Education.* https://doi.org/10.1007/s42330-022-00210-9

Sartatzemi, M. ; Dagdilelis, V. ; Kagani, K. (2005). Teaching Programming with Robots : A Case Study on Greek Secondary Education. In P. Bozanis & E. N. Houstis (Éds.), *Advances in Informatics* (Vol. 3746, p. 502-512). Springer Berlin Heidelberg. https://doi.org/10.1007/11573036_47

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), 764-792. https://doi.org/10.1037/edu0000314

Scherer, R. ; Siddiq, F. ; Sánchez Viveros, B. (2020). A meta-analysis of teaching and learning computer programming : Effective instructional approaches and conditions. *Computers in Human Behavior*. (109). https://doi.org/10.1016/j.chb.2020.106349

Scott, M. J.; Counsell, S.; Lauria, S.; Swift, S.; Tucker, A.; Shepperd, M.; Ghinea, G. (2015). Enhancing Practice and Achievement in Introductory Programming With a Robot Olympics. *IEEE Transactions on Education*, 58(4), 249-254. https://doi.org/10.1109/TE.2014.2382567

Selby, C. C. (2013). Computational Thinking : The Developing Definition. University of Southampton (E-prints) 6pp. https://core.ac.uk/download/pdf/17189251.pdf

Selby, C. C. (2015). Relationships : Computational thinking, pedagogy of programming, and Bloom's Taxonomy. Proceedings of the Workshop in Primary and Secondary Computing Education, 80-87. https://doi.org/10.1145/2818314.2818315

Siti Rosminah, M. D. ; Zamzuri, A. M. A. (2012). Difficulties in learning programming : Views of students. 1st International Conference oo Current Issues in Education, ICCIE2012At: Yogyakarta, Indonesia. https://doi.org/10.13140/2.1.1055.7441

Solomon, C.; Harvey, B.; Kahn, K.; Lieberman, H.; Miller, M. L.; Minsky, M.; Papert, A.; Silverman, B. (2020). History of Logo. Proceedings of the ACM on Programming Languages, 4(HOPL), 1-66. https://doi.org/10.1145/3386329

Spach, M. (2017). Activités robotiques à l'école primaire et apprentissage de concepts informatiques : Quelle place du scénario pédagogique ? Les limites du co-apprentissage. https://tel.archives-ouvertes.fr/tel-02271924

Valorge, M. (2022) Possibilités d'agir d'un environnement d'apprentissage de la programmation informatique à l'école primaire. *L'informatique, objets d'enseignement et d'apprentissage. Quelles nouvelles perspectives pour la recherche ?* May 2022, Le Mans, France. pp.49-60. ⟨hal-03697911⟩

Vihavainen, A.; Airaksinen, J.; Watson, C. (2014). A systematic review of approaches for teaching introductory programming and their influence on success. Proceedings of the Tenth Annual Conference on International Computing Education Research - ICER '14, 19-26. https://doi.org/10.1145/2632320.2632349

Voogt, J.; Fisser, P.; Good, J.; Mishra, P.; Yadav, A. (2015). Computational thinking in compulsory education : Towards an agenda for research and practice. *Education and Information Technologies*, 20(4), 715-728. https://doi.org/10.1007/s10639-015-9412-6

Weintrop, D.; Wilensky, U. (2017). Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. *ACM Transactions on Computing Education*, 18(1), 1-25. https://doi.org/10.1145/3089799

Williams, L. A.; Kessler, R. R. (2001). Experiments with Industry's "Pair-Programming" Model in the Computer Science Classroom. *Computer Science Education*, 11(1), 7-20. https://doi.org/10.1076/csed.11.1.7.3846

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33. https://doi.org/10.1145/1118178.1118215

Wing, J. M. (2010). Computational Thinking : What and Why? Unpublished Manuscript, Pittsburgh, PA: Computer Science Department, Carnegie Mellon University. http://www.cs.cmu.edu/~CompThink/papers/TheLinkWing.pdf

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

Yadin, A. (2011). Reducing the dropout rate in an introductory programming course. *ACM Inroads*, 2(4), 71-76. https://doi.org/10.1145/2038876.2038894

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

10. ANNEXE

Les références consultées et les articles inclus dans le corpus d'études :

Articles inclus dans le corpus d'études	Autres références consultés
Al-Bow, M.; Austin, D.; Edgington, J.; Fajardo, R.; Fishburn, J.; Lara, C.; Leutenegger, S.; Meyer, S. (2009). Using game creation for teaching computer programming to high school students and teachers. ACM SIGCSE Bulletin, 41(3), 104-108. https://doi.org/10.1145/1595496.1562913	Alimisis, D. (2013). Educational robotics : Open questions and new challenges. 9.
Allain, S. (2019). Créativité, pensée informatique et robotique, quels processus mis en œuvre par les élèves ? 113. https://hal-univ-fcomte.archives-ouvertes.fr/hal-02325425/document	Baron, G.-L. (1990). Note de synthèse [L'informatique en éducation - Le cas de la France] : L'informatique en éducation - Le cas de la France. Revue française de pédagogie, 92(1), 57-77. https://doi.org/10.3406/rfp.1990.2474
Bai, H.; Wang, X.; Zhao, L. (2021). Effects of the Problem-Oriented Learning Model on Middle School Students' Computational Thinking Skills in a Python Course. Frontiers in Psychology, 12, 771221. https://doi.org/10.3389/fpsyg.2021.771221	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
Braune, G.; Mühling, A. (2020). Learning to program : The gap between school world and everyday world. Proceedings of the 15th Workshop on Primary and Secondary Computing Education, 1-9. https://doi.org/10.1145/3421590.3421597	Bers, M. U. (2017). The Seymour test : Powerful ideas in early childhood education. International Journal of Child-Computer Interaction, 14, 10-14. https://doi.org/10.1016/j.ijcci.2017.06.004
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	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
Debabi, W.; Bensebaa, T. (2016). USING SERIOUS GAME TO ENHANCE ALGORITHMIC LEARNING AND TEACHING. 12(2), 14. https://doi.org/10.20368/1971-8829/1125	Busana, G.; Denis, B.; Duflot-Kremer, M.; Higuet, S.; Kataja, L.; Kreis, Y.; Laduron, C.; Meyers, C.; Parmentier, Y.; Reuter, R.; Weinberger, A. (2020, février). PIAF : Développer la Pensée Informatique et Algorithmique dans l'enseignement Fondamental. Didapro 8 – DidaSTIC – L'informatique, objets d'enseignements – enjeux épistémologiques, didactiques et de formation. https://hal.archives-ouvertes.fr/hal-02463940
Dorling, M.; White, D. (2015). Scratch : A Way to Logo and Python. Proceedings of the 46th ACM Technical Symposium on Computer Science Education, 191-196. https://doi.org/10.1145/2676723.2677256	Crahay, M. (1987). Logo, un environnement propice à la pensée procédurale. Revue française de pédagogie, 80(1), 37-56. https://doi.org/10.3406/rfp.1987.1473
Fessard, G.; Wang, P. ; Renna, I. (2019). Objet Tangible ou Simulation Numérique : Deux Situations Équivalentes pour l'Apprentissage de la Programmation ? 13. https://hal.archives-ouvertes.fr/hal-02156174/document	CSTA; ISTE. (2011). Computational_Thinking_Operational_Definition_ISTE. https://cdn.iste.org/www-root/Computational_Thinking_Operational_Definition_ISTE.pdf
Giordano, D. ; Maiorana, F. (2014). Use of cutting edge educational tools for an initial programming course. 2014 IEEE Global Engineering Education Conference (EDUCON), 556-563. https://doi.org/10.1109/EDUCON.2014.6826147	De Araujo, A. L. S. O. ; Andrade, W. L. ; Serey Guerrero, D. D. (2016). A systematic mapping study on assessing computational thinking abilities. 2016 IEEE Frontiers in Education Conference (FIE), 1-9. https://doi.org/10.1109/FIE.2016.7757678
Grover, S.; Basu, S. (2017). Measuring Student Learning in Introductory Block-Based Programming : Examining Misconceptions of Loops, Variables, and Boolean Logic. Proceedings of the 2017	Department of Education UK (2013). National curriculum in England: Computing programmes of study: key stages 3 and 4. https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_

ACM SIGCSE Technical Symposium on Computer Science Education, 267-272. https://doi.org/10.1145/3017680.3017723

Koitz, R.; Slany, W. (2014). Empirical Comparison of Visual to Hybrid Formula Manipulation in Educational Programming Languages for Teenagers. Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools, 21-30. https://doi.org/10.1145/2688204.2688209

Kyfonidis, C., Moumoutzis, N. ; Christodoulakis, S. (2017). Block-C : A block-based programming teaching tool to facilitate introductory C programming courses. 2017 IEEE Global Engineering Education Conference (EDUCON), 570-579. https://doi.org/10.1109/EDUCON.2017.7942903

Major, L.; Kyriacou, T.; Brereton, P. (2014). The effectiveness of simulated robots for supporting the learning of introductory programming : A multi-case case study. Computer Science Education, 24(2-3), 193-228. https://doi.org/10.1080/08993408.2014.963362

Matsuzawa, Y. ; Ohata, T. ; Sugiura, M. ; Sakai, S. (2015). Language Migration in non-CS Introductory Programming through Mutual Language Translation Environment. Proceedings of the 46th ACM Technical Symposium on Computer Science Education, 185-190. https://doi.org/10.1145/2676723.2677230

Meerbaum-Salant, O.; Armoni, M.; Ben-Ari, M. (Moti). (2010). Learning computer science concepts with Scratch. Proceedings of the Sixth International Workshop on Computing Education Research - ICER '10, 69. https://doi.org/10.1145/1839594.1839607

Misirli, A. ; Komis, V. (2013). Construire les notions de l'orientation et de la direction à l'aide des jouets programmables : une étude de cas dans des écoles maternelles en Grèce. Colloque international éTIC – Les TICE à l'école primaire A l'université de Limoges, Faculté des Lettres et des Sciences Humaines. https://doi.org/10.4000/books.septentrion.15138

Mladenović, M.; Boljat, I.; Žanko, Ž. (2018). Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level. Education and Information Technologies, 23(4), 1483-1500. https://doi.org/10.1007/s10639-017-9673-3

Moreno-León, J.; Robles, G.; Román-González, M. (2015). Dr. Scratch : Automatic Analysis of Scratch Projects to Assess and Foster Computational Thinking. 23. https://www.um.es/ead/red/46/moreno_robles.pdf

data/file/239067/SECONDARY_national_curriculum_-_Computing.pdf

DROT-DELANGE, B. ; PELLET, J.-P. ; DELMAS-RIGOUTSOS, Y. ; BRUILLARD, R. (2019). Pensée informatique : Points de vue contrastés. https://doi.org/10.23709/STICEF.26.1.1

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

Gomes, A.; Mendes, A. (2007). Learning to program—Difficulties and solutions. International Conference on Engineering Education, Coimbra, Portugal. http://icee2007.dei.uc.pt/proceedings/papers/411.pdf

Hart, C., (2009). Doing a literature review: Releasing the social science research imagination, LA/London, Sage.

Jaillet, A. (2019). Cerveau, émotions et bonheurs : Apprendre par le « Hard Fun ». Tréma, 52. https://doi.org/10.4000/trema.5340

Lepage, A. ; Romero, M. (2017). Évaluation par compétences d'activités de programmation créative avec l'outil #5c2. 5. http://alepage.net/wp-content/uploads/2020/08/CIRTA2017-PI-Evaluation-LepageRomero2017-R07.pdf

Medeiros, R. P.; Ramalho, G. L.; Falcao, T. P. (2019). A Systematic Literature Review on Teaching and Learning Introductory Programming in Higher Education. IEEE Transactions on Education, 62(2), 77-90. https://doi.org/10.1109/TE.2018.2864133

Mow, I. T. C. (2008). Issues and Difficulties in Teaching Novice Computer Programming. In M. Iskander (Éd.), Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education (p. 199-204). Springer Netherlands. https://doi.org/10.1007/978-1-4020-8739-4_36

Nonnon, P. (2002). Robotique pédagogique et formation de base en science et technologie. Aster, 34(34, p. 213). https://doi.org/10.4267/2042/8787

Noone, M.; Mooney, A. (2018). Visual and textual programming languages : A systematic review of the literature. Journal of Computers in Education, 5(2), 149-174. https://doi.org/10.1007/s40692-018-0101-5

Orfanakis, V.; Papadakis, S. (2014). A new programming environment for teaching programming. A first acquaintance with Enchanting. international virtual Scientific Conference. https://www.academia.edu/7576980/A_new_programming_environment_for_teaching_programming_A_first_acquaintance_with_Enchanting

Papert, S. (1980). Mindstorms : Children, computers, and powerful ideas. Basic Books.

Muratet, M. ; Torguet, P. ; Jessel, J.-P. ; Vallet, F. (2008). Vers un jeu sérieux pour enseigner la programmation. 6. https://core.ac.uk/download/pdf/50530308.pdf

Nijimbere, C. (2014). Apprendre l'informatique par la programmation des robots : Cas de Nao. Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation, 21(1), 63-109. https://doi.org/10.3406/stice.2014.1091

Ouahbi, I. ; Darhmaoui, H. ; Kaddari, F. ; Elachqar, A. ; Lahmine, S. (2015). Vers un enseignement de la programmation compatible avec la culture vidéoludique des élèves au Maroc. 7ème Conférence sur les Environnements Informatiques pour l'Apprentissage Humain (EIAH 2015), 405-407. https://hal.archives-ouvertes.fr/hal-01413631/document

Papadakis, S.; Orfanakis, V. (2017). The Combined Use of Lego Mindstorms NXT and App Inventor for Teaching Novice Programmers. In D. Alimisis, M. Moro, & E. Menegatti (Éds.), Educational Robotics in the Makers Era (Vol. 560, p. 193-204). Springer International Publishing. https://doi.org/10.1007/978-3-319-55553-9_15

Pea, R. D. ; Kurland, D. M.; Hawkins, J. (1985). LOGO and the Development of Thinking Skills. 23. https://telearn.archives-ouvertes.fr/hal-00190534/document

Pekarova, J. (2008). Using a Programmable Toy at Preschool Age : Why and How? 10. https://www.terecop.eu/downloads/simbar2008/pekarova.pdf

Price, T. W.; Barnes, T. (2015). Comparing Textual and Block Interfaces in a Novice Programming Environment. Proceedings of the Eleventh Annual International Conference on International Computing Education Research - ICER '15, 91-99. https://doi.org/10.1145/2787622.2787712

Ramalingam, V.; LaBelle, D.; Wiedenbeck, S. (2004). Self-efficacy and mental models in learning to program. Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education - ITiCSE '04, 171. https://doi.org/10.1145/1007996.1008042

Romero, M. ; Lepage, A. ; Lille, B. (2017). Computational thinking development through creative programming in higher education. International Journal of Educational Technology in Higher Education, 14(1), 42. https://doi.org/10.1186/s41239-017-0080-z

Romero, M. ; DeBlois, L. (2022). Analyse du processus de construction de connaissances dans des activités de programmation à l'école. Canadian Journal of Science, Mathematics and Technology Education. https://doi.org/10.1007/s42330-022-00210-9

http://worrydream.com/refs/Papert%20-%20Mindstorms%201st%20ed.pdf

Papert, S. ; Jaillet, A. (2006, octobre 5). Un défi de taille pour l'école. Les Cahiers pédagogiques. https://www.cahiers-pedagogiques.com/un-defi-de-taille-pour-l-ecole/

Preston, D. (2006). Using collaborative learning research to enhance pair programming pedagogy. ACM SIGITE Newsletter, 3(1), 16-21. https://doi.org/10.1145/1113378.1113381

Robinson, W. (2016). From Scratch to Patch : Easing the Blocks-Text Transition. Proceedings of the 11th Workshop in Primary and Secondary Computing Education, 96-99. https://doi.org/10.1145/2978249.2978265

Rogalski, J. (1987). Acquisition de savoirs et de savoir-faire en informatique. 43. https://publimath.univ-irem.fr/numerisation/PS/IPS87004/IPS87004.pdf

Romero, M. (2016). De l'apprentissage procédural de la programmation à l'intégration interdisciplinaire de la programmation créative. Formation et profession : revue scientifique internationale en éducation, 24(1), 87-89. https://doi.org/10.18162/fp.2016.a92

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), 764-792. https://doi.org/10.1037/edu0000314

Scherer, R. ; Siddiq, F. ; Sánchez Viveros, B. (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

Selby, C. C. (2013). Computational Thinking : The Developing Definition. 6. https://core.ac.uk/download/pdf/17189251.pdf

Solomon, C.; Harvey, B.; Kahn, K.; Lieberman, H.; Miller, M. L.; Minsky, M.; Papert, A.; Silverman, B. (2020). History of Logo. Proceedings of the ACM on Programming Languages, 4(HOPL), 1-66. https://doi.org/10.1145/3386329

Spach, M. (2017). Activités robotiques à l'école primaire et apprentissage de concepts informatiques : Quelle place du scénario pédagogique ? Les limites du co-apprentissage. 354. https://tel.archives-ouvertes.fr/tel-02271924

Vihavainen, A.; Airaksinen, J.; Watson, C. (2014). A systematic review of approaches for teaching introductory programming and their influence on success. Proceedings of the Tenth Annual Conference on International Computing Education Research - ICER '14, 19-26. https://doi.org/10.1145/2632320.2632349

Sartatzemi, M. ; Dagdilelis, V. ; Kagani, K. (2005). Teaching Programming with Robots : A Case Study on Greek Secondary Education. In P. Bozanis & E. N. Houstis (Éds.), Advances in Informatics (Vol. 3746, p. 502-512). Springer Berlin Heidelberg. https://doi.org/10.1007/11573036_47

Scott, M. J.; Counsell, S.; Lauria, S.; Swift, S.; Tucker, A.; Shepperd, M.; Ghinea, G. (2015). Enhancing Practice and Achievement in Introductory Programming With a Robot Olympics. IEEE Transactions on Education, 58(4), 249-254. https://doi.org/10.1109/TE.2014.2382567

Selby, C. C. (2015). Relationships : Computational thinking, pedagogy of programming, and Bloom's Taxonomy. Proceedings of the Workshop in Primary and Secondary Computing Education, 80-87. https://doi.org/10.1145/2818314.2818315

Siti Rosminah, M. D. ; Zamzuri, A. M. A. (2012). Difficulties in learning programming : Views of students. https://doi.org/10.13140/2.1.1055.7441

Weintrop, D.; Wilensky, U. (2017). Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. ACM Transactions on Computing Education, 18(1), 1-25. https://doi.org/10.1145/3089799

Williams, L. A.; Kessler, R. R. (2001). Experiments with Industry's "Pair-Programming" Model in the Computer Science Classroom. Computer Science Education, 11(1), 7-20. https://doi.org/10.1076/csed.11.1.7.3846

Yadin, A. (2011). Reducing the dropout rate in an introductory programming course. ACM Inroads, 2(4), 71-76. https://doi.org/10.1145/2038876.2038894

Voogt, J.; Fisser, P.; Good, J.; Mishra, P.; Yadav, A. (2015). Computational thinking in compulsory education : Towards an agenda for research and practice. Education and Information Technologies, 20(4), 715-728. https://doi.org/10.1007/s10639-015-9412-6

Wing, J. M. (2006). Computational thinking. Communications of the ACM, 49(3), 33. https://doi.org/10.1145/1118178.1118215

Wing, J. M. (2010). Computational Thinking : What and Why? http://www.cs.cmu.edu/~CompThink/papers/TheLinkWing.pdf

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

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