Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC

Olivier Goletti, Kim Mens

DIDAPRO 10

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC

30 janvier-01 février 2024 Louvain-la-Neuve, Belgique

https://didapro.org/10/

édités par Olivier GOLETTI, Kim MENS

Le colloque DIDAPRO 10 a été organisé par L'UCLouvain avec l'aide de l'UGent et de l'UNamur

Avec le soutien de

Préface

Editorial des actes du colloque Didapro 10

En tant qu'organisateurs de l'édition du 10ème anniversaire de la série de colloques Didapro, qui s'est tenue à l'université UCLouvain dans la ville universitaire de Louvain-la-Neuve, Belgique, du 30 janvier au 1er février 2024, nous avons l'honneur de vous présenter les actes de ce colloque. L'intégralité des articles publiés dans ces actes est disponible en open accès ouvert en ligne dans une archive ouverte pluridisciplinaire HAL.
Après les deux éditions précédentes en France (Didapro 9 au Mans en 2022 et Didapro 8 à Lille en 2020) et celle d'avant en Suisse (Didapro 7 à Lausanne), Didapro était de retour en Belgique en 2024 (après Didapro 6 qui a été organisé en janvier 2016 par l'université de Namur).
Le colloque a proposé un programme très attractif incluant de nouvelles initiatives introduites à l'occasion de cette dernière édition. En particulier, lors de la matinée du premier jour du colloque, nous avons organisé, pour la première fois, une journée dédiée aux jeunes chercheurs et chercheuses dans le domaine de l'enseignement et l'apprentissage de l'informatique ou des domaines connexes. Malgré le fait que le colloque était majoritairement francophone, cette session a été délibérément organisée en anglais, afin de pouvoir inclure des collègues de Flandre et des pays voisins.
Nous avons également accueilli un conférencier et deux conférencières invitées qui ont donné des exposés inspirants. Nous avons organisé une session de présentation de posters l'après-midi du premier jour, où les participants au colloque, ainsi que les doctorants de la journée jeunes chercheurs et chercheuses et certains de nos étudiants de master en sciences informatiques et sciences de l'ingénieur en informatique ont eu l'occasion de présenter leurs travaux aux autres participants du colloque à travers des posters affichés dans le hall de la conférence. Des présentations scientifiques sur une variété de thèmes ont suivi le deuxième jour et la première moitié du troisième jour du colloque. L'après-midi du troisième jour, nous avons organisé des ateliers davantage dédiés aux enseignants, bien qu'ils étaient également ouverts aux participants et participantes du colloque et ont principalement été fréquentés par ceux-ci.
Enfin, pour encourager une participation active, il y avait de nombreuses opportunités de réseautage pour les participants et participantes à la conférence, et entre chercheurs ou chercheuses et praticiens ou praticiennes : nous avons prévu amplement d'espace et de temps pour des discussions pendant les pauses café, le gala du colloque à la Louvain House, un événement de dégustation de bières (ou une alternative non alcoolique) et dans les soirées après la conférence dans la ville de Louvain-la-Neuve.

Historique

Avant de détailler davantage le déroulement et le contenu de cette dixième édition de Didapro, nous souhaiterions partager avec vous quelques mots concernant son historique, établis avec l'aide de Georges-Louis Baron, un des initiateurs et donc « la mémoire » de cette série de colloques.
Même avant l'existence de Didapro, dès le début des années 1980 il y avait des rencontres francophones Logo, concernant plutôt l'enseignement primaire. Ensuite, un premier colloque de didactique de l'informatique a été organisé en 1988 à l'Université Paris 5 René

Descartes. Il a été suivi d'éditions à Namur en 1991, à Sion en Suisse en 1992, à Québec en 1994, à Monastir en Tunisie en 1996.
Mais on n'enseignait plus à cette époque l'informatique dans le second degré en France ni dans les autres pays francophones et la communauté constituée depuis 1988 s'est donc dispersée, car on ne peut faire de la recherche sur la didactique d'une discipline que si elle est enseignée.
C'est seulement à partir de 2003 qu'une nouvelle série de colloques a été relancée à l'initiative de Georges-Louis Baron et Eric Bruillard. Cette série était appelé Didapro, prêtant plus largement attention aux enjeux DIDActiques des outils informatiques qu'on avait d'abord appelé « PROgiciels » (en commençant par les traitements de textes, tableurs...) et qui étaient utilisés dans toutes les disciplines au collège et au lycée.
Aujourd'hui, le colloque porte plus généralement sur les questions de l'enseignement et l'apprentissage de l'informatique de l'école primaire à l'université, des ressources et dispositifs pour l'enseignement de l'informatique et des politiques publiques et curricula. Il contribue à rassembler une communauté constituée d'enseignants-chercheurs, d'enseignants du primaire, secondaire et supérieur.

Didapro 10

Revenant sur l'édition 2024 du colloque, nous avons accueilli environs presque 70 participants, dont une poignée était inscrite uniquement pour le premier jour et une petite dizaine pour le dernier jour seulement. La grande majorité des inscrits a assisté à l'intégralité du colloque.
Parmi les participants, nous avons recensé une dizaine de Suisses, plus de vingt Belges et plus de trente Français, ainsi que quelques participants d'autres pays tels que les Pays-Bas et la Grèce. Nous avons également observé une parité entre participants masculins et féminins à la conférence.
Dans les paragraphes suivants, nous aborderons les différents types de contributions soumises et présentées lors du colloque.

Journée jeunes chercheurs et chercheuses

La matinée jeunes chercheurs et chercheuses, organisée à l'intention des doctorants, a été tenue en anglais afin de permettre une participation plus internationale. Les contributions ont donc été soumises et présentées en anglais. L'organisation de cette matinée était dans les mains du Prof. Francis Wyffels de l'université de Gand. Cinq jeunes doctorants et doctorants ont été acceptés pour présenter leurs recherches en cours lors de cet événement. Un court article sur leur travail a été inclus dans ces actes du colloque (Chapitre II, page 108).
Après avoir présenté leurs recherches aux autres doctorants et échangé des commentaires, ils ont participé à une session de brainstorming pour travailler sur des sujets de recherche conjoints potentiels. Durant la session de l'après-midi, ils ont également eu l'occasion d'exposer leur travail aux autres participants du colloque lors de la session de posters.

Posters

Nous avons sollicité des contributions de type « poster », décrivant des travaux récemment complétés ou encore en cours, des applications concrètes, ou illustrant des pratiques correspondant aux thématiques du colloque. Un article plus long a également été reclassifié

en poster, ainsi qu'un atelier. Pour chaque poster, un résumé de maximum 2 pages a été inclus dans les actes du colloque (Chapitre III, page 136). Les participants ont également eu l'option de téléverser une version PDF de leur poster. En plus de ces 7 posters, les participants à la journée jeunes chercheurs et chercheuses ont également présenté un poster. En total, 13 posters ont donc été affichés.

Colloque scientifique

L'après-midi de la première journée, la deuxième journée ainsi que la matinée de la troisième journée ont été consacrées au colloque scientifique. Les présentations scientifiques ont été regroupé par thématique. Les différents thématiques retenus étaient la classification et l'analyse par référentiel, le contexte et les besoins dans l'enseignement, l'interdisciplinarité et la diversité, la robotique à l'école primaire, et finalement l'apprentissage de la programmation et de l'algorithmique. Pour chaque thématique, entre 2 et 4 articles ont été présentés, modéré par un expert dans le domaine de la didactique de l'informatique. Vous trouverez les versions révisées de ces articles sur base des commentaires des relecteurs, plus loin dans ces actes.
Sur 18 contributions de type "papier long" reçues, 10 ont été acceptées (Chapitre I, page 1) et 5 avec uniquement une publication de résumé dont une n'a finalement pas été acceptée par absence d'orateur, donc seulement 4 dans ces actes (Chapitre IV, page 155). Sur les 10 articles longs, cinq contributions ont demandé des révisions majeures (et donc une ultime relecture) et les cinq autres seulement des modifications mineures. De ces 14 présentations, 5 ont donné lieu à une présentation courte et 9 une présentation longue. Une des 18 contributions n'a pas été accepté pour présentation mais a été requalifiée en poster. Pour les deux contributions restantes refusées, une était incomplète et une hors contexte.

Table ronde

Devant l'augmentation des tailles des cohortes dans le supérieur ou l'arrivée massive de l'informatique à l'école, plusieurs outils ont été développés pour aider à l'enseignement de l'informatique. Entre les parcours d'apprentissage automatisés, les tests ponctuels de masse et les exerciseurs plus ou moins riches en types d'exercices et en type de retours, ces outils numériques se multiplient. Quelles en sont les caractéristiques (exercices ou évaluation, qualité du feedback, QCM, ...), les avantages (opportunités de recherche, passage à l'échelle, évaluation identique pour tous, ...) et les inconvénients (peu de différentiation, peu de résistance à la triche et aux outils de génération de code, ...) ? Quel serait l'outil idéal ?
Lors de la table ronde, faisant partie du colloque scientifique, le pour et le contre de l'automatisation des parcours d'apprentissage et des évaluations, ont été débattus par trois intervenants internationaux : Fahima Djelil, Maîtresse de Conférences à IMT Atlantique ; Vanda Luengo, Professeure à Sorbonne Université et Engin Bumbacher, Professeur en technologie de l'éducation à la HEP Vaud. Les échanges entre intervenants et public ont été modérés par Olivier Goletti et ont concerné les trois thématiques suivantes préalablement concertées avec les intervenants :

- L'exploration d'une application adaptative pour évaluer les compétences en informatique ;
- Les aspects de feedbacks/rétroactions, les retours en boucle courte/immédiate versus en boucle longue ;

— La cohérence technique et d'utilisation, et l'interopérabilité entre ces outils d'enseignement.

Présentations invitées

La première oratrice invitée était Felienne Hermans, professeure de Computer Science Education à la Vrije Universiteit Amsterdam. Un jour par semaine, elle est aussi enseignante d'informatique dans le secondaire supérieur dans le programme Codasium du Lyceum Kralingen aux Pays-Bas. Felienne est la créatrice du langage de programmation Hedy, un langage progressif et multilingue pensé pour enseigner. Elle est l'autrice de "The Programmer's Brain : What every programmer needs to know about cognition, Simon and Schuster, 2021", un livre qui aide les programmeurs à comprendre comment leur cerveau fonctionne et comment l'utiliser plus efficacement. En 2021, elle a reçu le prix national de recherche en TIC des Pays-Bas (Dutch Prize for ICT research). Elle a aussi une rubrique technologique deux fois par semaine sur BNR, une radio Néerlandaise. Son discours en anglais portait sur "Gradual Programming with Hedy", et parlait de ses expériences lors du développement de ce nouveau langage didactique pour apprendre à programmer.

Notre deuxième orateur invité était Francis Wyffels, qui a également organisé la journée pour les jeunes chercheurs et chercheuses. Francis Wyffels est professeur à l'université de Gand dans le groupe de recherche IDLab. Il possède une expertise dans les domaines de l'apprentissage automatique et de la robotique, avec un accent sur le contrôle des robots, les réseaux de neurones, l'apprentissage par renforcement et l'apprentissage profond. Francis Wyffels croit également fermement en l'importance des activités de diffusion et d'enseignement de l'informatique pour lesquelles il a reçu plusieurs prix. Il est également président de Dwengo (une organisation sans but lucratif), à travers laquelle il a mené plusieurs projets avec des enfants, des élèves et des écoles, afin d'assurer une perspective plus réaliste de ce que la robotique peut accomplir et ainsi réduire la peur de la robotique. Ces projets éducatifs constituent également la base de la recherche didactique sur l'enseignement de l'informatique réalisée dans son laboratoire. Dans son discours, il a parlé de ces perspectives et de ce que sont et ne sont pas l'IA et la robotique.

Notre troisième oratrice invitée était Julie Henry de l'Université de Namur. Julie Henry est à la fois chimiste, techno-pédagogue et docteure en didactique de l'informatique. Elle occupe un poste de cheffe de projet STEM à l'Université de Namur. Ses projets de recherche sont souvent interdisciplinaires, reliant les sciences humaines et sociales, les sciences fondamentales, les sciences appliquées et l'art. Elle est passionnée par la didactique de l'informatique, en particulier la didactique de la programmation et la didactique des TIC, et la problématique de genre en informatique. Elle est co-autrice de l'ouvrage "Éduquer au numérique. 12 clés pour comprendre l'informatique", Henry Julie et Fanny Boraita, Politeia 2012. Son exposé portait sur "Réinventer l'enseignement de l'informatique" et faisait partie de la dernière journée du colloque avec un focus sur les enseignants.

Journée « enseignants » et ateliers

Lors de l'édition du colloque Didapro 7, à Lausanne, une première journée d'ouverture a été initiée à destination des praticiens et praticiennes, enseignants et enseignantes, et formateurs et formatrices, en poste ou en formation initiale. Lors de cette journée des ateliers ont été organisés qui visaient à faire découvrir des activités, des ressources ou des

actions en lien avec l'enseignement et l'apprentissage de l'informatique (au sens large). Lors des éditions suivantes de Didapro, des journées d'ouverture similaires avec entre autres des ateliers ont été organisées à Didapro 8 à Lille, à Didapro 9 au Mans et donc aussi lors de cette dernière édition de Didapro 10 à Louvain-la-Neuve.
Grâce aux sponsors de la conférence, la « journée des enseignants » était gratuite pour les enseignants et les formateurs. L'existence de cette journée, à côté de la partie plus « scientifique », est un des points fort du colloque. La diversité et la richesse des ateliers ont été un des temps forts des colloques Didapro et constituent un des éléments contribuant au développement et à l'animation d'une communauté de pratiques de personnes intéressées par l'enseignement de l'informatique et la formation des enseignant · e · s à la didactique de l'informatique.
Lors du conseil scientifique de Didapro 9, la question de la valorisation des ateliers et notamment de l'accessibilité à l'issue du colloque des éléments qui s'y sont partagés a été évoquée. Il a alors été proposé de réaliser un numéro thématique portant sur les ateliers dont l'objectif serait de rendre accessible à un cercle plus large les ressources et actions présentées. Comme pour Didapro 9, pour l'édition 2024 du colloque Didapro 10, on a offert aux organisateurs des ateliers la possibilité de publier le contenu de leur atelier dans un numéro thématique en ligne de la revue Adjectif, une revue d'interface entre des chercheurs et des praticiens en éducation et formation de l'informatique et le STIC.
Les actes du colloque Didapro 10 ne contiennent donc que les titres, contributeurs et contributrices et un résumé de ces ateliers (Chapitre V, page 160), mais pour consulter leur contenu vous êtes invité à aller consulter le numéro thématique d'Adjectif destiné aux ateliers de Didapro 10 (https://adjectif.net/spip.php?article614).

Remerciements

Nous souhaitons exprimer notre profonde gratitude à tous ceux qui ont contribué à la réussite de ce colloque. Tout d'abord, nos sincères remerciements vont à nos divers sponsors, dont le soutien financier a été essentiel pour organiser cet événement de haute qualité :

- Le secteur des sciences et des technologies (SST) de l'UCLouvain ;
- L'entité Digital Wallonia de l'Agence du Numérique ;
- Le Fonds National de Recherche Scientifique (FNRS) ;
- L'institut de recherche ICTEAM (Information and Communication Technology, Electronics and Applied Mathematics) de l'UCLouvain ;
- Le pôle de recherche INGI (Ingénierie Informatique) ;
- Les entreprises Raincode et Isabel Group ;
- L'école doctorale GRASCOMP du FNRS.

Pour leur engagement, soutien, collaboration et aide précieuse nous remercions également :

- Nos co-organisateurs, Prof. Francis Wyffels de l'UGent et Prof. Bruno Dumas de l'UNamur.
- Vanessa Maons, notre secrétaire de département, pour son travail logistique indispensable, ainsi que Damien Cremer, responsable des paiements, et Camille Chatelle, pour son aide logistique en début de préparation.
- Julien Liénard, assistant de recherche en génie logiciel appliqué à l'enseignement de l'informatique, pour son aide précieuse sur la logistique locale et la finalisation de ces actes de conférence.
- Fanny Liénard, pour la refonte du logo Didapro et la création de la bannière du colloque.

— Nos administrateurs système, pour la création des identifiants réseau pour les participants et d'autres interventions techniques.
— Le personnel du CRM, pour leur aide avec le module d'inscription et de paiement.
— Les assistants de recherche, pour leur soutien durant le colloque.
— L'université et son personnel technique, pour la mise à disposition des auditoriums.
— Les 28 membres du comité scientifique, pour leur travail rigoureux de lecture et d'analyse qui a permis la sélection et l'amélioration des contributions.
— Les organisateurs et organisatrices des éditions précédentes de Didapro, pour leurs conseils et leur aide.
— Tous les contributeurs et contributrices à la journée doctorale, aux sessions de posters, aux sessions scientifiques et aux ateliers.
— Enfin, et surtout, tous les participants de la conférence, sans qui cet événement n'aurait pas été le succès qu'il a été.

Cette conférence a été un véritable effort collectif, et le soutien de toutes ces personnes a été le pilier de son succès. Merci à tous pour votre engagement, votre passion et votre dévouement.

Conclusion

Avec plus de 70 participants aux différentes parties du colloque, et la richesse et diversité des différentes types d'activités et des contributions, on peut dire que cette dixième édition du colloque fut un grand succès, dont nous espérons que ces actes permettent à ceux qui n'étaient pas présent de goûter un peu. A bientôt dans une prochaine édition de Didapro.

Kim Mens & Olivier Goletti, février 2024

Comités

Comité d'organisation

— Prof. Kim Mens, UCLouvain, ICTEAM/INGI
— Olivier Goletti, UCLouvain, ICTEAM/INGI
— Prof. Bruno Dumas, UNamur, NADI/PReCISE
— Prof. Francis Wyffels, UGent, IDLab/imec

Comité scientifique

— Baron Georges-Louis, Université Paris Descartes
— Béziat Jacques, Université de Caen Normandie
— Boulc'h Laetitia, Université Paris Descartes
— Brandt-Pomares Pascale, Aix-Marseilles Université
— Broisin Julien, Université de Toulouse
— Bruillard Eric, ENS Paris-Saclay
— Caron Pierre-André, Université de Lille
— Danquigny Thierry, Université de Lille
— Declercq Christophe, Université de la Réunion
— Desmontils Emmanuel, Université de Nantes
— Drot-Delange Béatrice, Université Clermont Auvergne
— Fluckiger Cédric, Université de Lille
— Goletti Olivier, Université Catholique de Louvain
— Grandbastien Monique, Université Poincaré, Nancy
— Henry Julie, Université de Namur
— Komis Vassilis, Univeristé de Patras
— Mens Kim, Université catholique de Louvain
— Muratet Mathieu, INSHEA
— Nogry Sandra, Université de Cergy-Pontoise
— Parriaux Gabriel, HEP Vaud
— Pellet Jean-Philippe, HEP Vaud
— Peter Yvan, Université de Lille
— Renault Valérie, Le Mans Université
— Secq Yann, Université de Lille
— Séjourné Arnauld, INSPÉ de l'académie de Nantes
— Voulgre Emmanuelle, Université Paris Descartes
— Wyffels Francis, Universiteit Gent
— Yessad Amel, Sorbonne Université

Présentations invitées

Felienne Hermans, professeure de Computer Science Education à la Vrije Universiteit Amsterdam

Gradual Programming with Hedy

Hedy est un langage de programattion progressif pour faciliter l'apprentissage de la programmation. Hedy utilise plusieurs niveaux de langage différents.
Au niveau 1, il n'y a presque pas de syntaxe. Par exemple, pour imprimer un message, on écrit :

```
print hello Louvain!
```

À chaque niveau, de la syntaxe et des concepts sont rajoutés, jusqu'à ce que les enfants utilisent un sous-ensemble de Python au niveau 18 avec conditions, fonctions, boucles, variables et listes. L'approche par niveau signifie que les apprenants n'ont pas à apprendre tout la syntaxe en un coup. Hedy est prévu pour les enfants qui veulent commencer la programmation textuelle mais pour qui commencer avec Python directement serait encore trop complexe. Hedy est open source, tourne dans le navigateur, est gratuit et disponible dans 47 (!) langues différentes (notamment anglais, français, néerlandais, espagnol, chinois, arabe et hindi).
Hedy a été lancé début 2020 et compte plus de 350.000 utilisateurs mensuels (vous pouvez le tester sur hedy.org !).
Dans cet exposé, Felienne va expliquer la pédagogie qu'utilise Hedy mais aussi quelques aspects techniques de Hedy. Par exemple, un ensemble de grammaries de plus en plus complexes plutôt qu'une seule grammaire amène son lot de challenge en terme de design du langage.

Francis Wyffels, Computer Science professor at Universiteit Gent

The robot teacher isn't there yet, but we can teach about it.

In the past decade, the digitisation surge, driven by AI breakthroughs, has introduced new socio-economic challenges. Recognising the imperative for tomorrow's innovators and users to understand and shape these technologies, we emphasise the necessity for context-driven, interdisciplinary STEM education. "The World of Sophie" addresses this need through innovative teaching materials that equip high school students with a comprehensive grasp of AI principles, engaging them across STEM fields. This project integrates disciplines such as geography, biology, chemistry, mathematics, and computer science, using themes such as climate change to develop relevant and authentic STEM learning experiences. By tailoring recent scientific research for secondary education, the initiative navigates challenges like curriculum alignment, accessibility for beginner programmers, and the demystification of AI concepts, including convolutional neural networks. Students thereby learn to apply theoretical knowledge in novel contexts, facilitating the transfer of insights to real-world issues. This approach aligns with educational standards and prepares students for future problem-solving, fostering a comprehensive and practical educational. The teaching materials are open source and can be found at http://aiopschool.be/

Julie Henry, PhD in CER, cheffe de projet STEM, Université de Namur

Réinventer l'enseignement de l'informatique

À l'ère numérique, l'évolution effrénée de la technologie, la pénurie de diplômés en numérique et les disparités de genre dans les filières informatiques posent des défis complexes à résoudre. En Belgique francophone, la solution passe par une réintroduction de l'informatique dès 8 ans. Une belle initiative qui voit se dresser un obstacle majeur : le manque de formation des enseignants.
Et si, plutôt que de tenter d'en faire des experts, nous repensions l'enseignement de l'informatique pour que celui-ci leur soit accessible, tout en répondant aux besoins de la société ? Il s'agirait d'adopter une approche holistique, multiréférentielle, visant à développer la pensée critique, la résolution de problèmes, la collaboration et la créativité, mais aussi à faire comprendre comment l'informatique sert d'autres domaines tels que les Sciences et les Arts.

Table des matières

Première partie

Articles

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Apprentissage de la programmation, contexte numérique des enseignants d'informatique genevois

Eva Dechaux[1][0000-0001-8579-1398] and Sébastien Jolivet[2][0000-0003-3915-8465]

[1] IUFE, Université de Genève, Suisse
[2] IUFE & TECFA, Université de Genève, Suisse
eva.dechaux@unige.ch ; sebastien.jolivet@unige.ch

Abstract. Cet article s'inscrit dans le contexte d'une étude compréhensive des situations d'enseignement de l'informatique. Plus précisément nous identifions les différents contextes numériques (langage étudié ; IDE ; environnement numérique d'apprentissage) existants dans l'enseignement secondaire genevois, quelques usages associés et certains des facteurs ayant mené à leur mise en place. Cette étude est basée sur l'analyse d'une enquête réalisée auprès des enseignants d'informatique du secondaire du canton de Genève. Nous mettons en évidence la diversité des contextes numériques existants et analysons les différents facteurs menant à leur construction.

Keywords: apprentissage de la programmation, contexte numérique, enquête, secondaire 2 à Genève.

1 Introduction et problématique

Pour favoriser les premiers apprentissages en lien avec la science informatique au primaire et au début de l'enseignement secondaire (élèves de 3 à 14 ans), en particulier autour de la programmation, diverses modalités ont été développées et promues ces dernières années allant de l'informatique dite débranchée à l'utilisation de robots et cela a notamment donné lieu au développement d'environnements de programmation par blocs de type Scratch. Dans le canton de Genève, lors du passage au secondaire II (élèves de 15 à 18 ans), il est plutôt de mise d'utiliser des langages qui sont effectivement exploités dans le monde professionnel. Si les premiers apprentissages sont généralement réalisés dans le paradigme de la programmation impérative, cela ne détermine pas de manière univoque les choix de langage de programmation et d'environnements associés. Dans le canton de Genève, et plus largement en Suisse Romande, il n'y a pas de préconisations officielles relatives au choix du langage, ni même d'orientation indiquée (voir par exemple dans les curriculums produits par la Direction Générale de l'Enseignement Secondaire II (DGES II, 2021)). De plus, l'absence d'une épreuve de fin d'études commune à l'ensemble des établissements, favorise une grande liberté dans les options choisies. Il peut donc y avoir des différences significatives d'un établissement à l'autre, et parfois même entre enseignants au sein d'un établissement. Or, comme cela est étudié par exemple dans (Branthôme,

Volume 1, pages 2–10.

2022; Khazaei & Jackson, 2002; White & Sivitanides, 2005) le choix du langage de programmation et/ou du paradigme n'est pas sans impact sur les apprentissages, il est donc pertinent de dresser un état des lieux de l'existant.

Si dans cet article nous ne proposons pas une étude permettant de comparer la situation entre différents cantons ou pays, il est intéressant de noter qu'en France le choix concernant les langages est beaucoup moins libre. Ainsi, les choix de Scratch (langage utilisé notamment aux épreuves du brevet des collèges français en mathématiques) pour le secondaire 1, et de Python pour le secondaire 2[1], sont quasi imposés.

En plus de celui du ou des langages travaillés, d'autres choix se posent aux enseignants ; en particulier celui du ou des environnements de développement intégré (IDE) et d'exploiter un environnement numérique d'apprentissage (ENA) de la programmation. Dans la suite de l'article nous appellerons *contexte numérique d'apprentissage de la programmation*, noté *contexte numérique* par simplicité, l'ensemble constitué du ou des langages, du ou des IDE, du ou des ENA, exploités par un enseignant. Le contexte numérique global d'un enseignant comprend potentiellement aussi d'autres éléments (ENT, système de pilotage de salle…) que nous ne prenons pas en compte dans ce travail car non spécifiquement liés à l'apprentissage de la programmation.

La description des choix réalisés par les enseignants dans le canton de Genève est donc le premier objectif de ce travail et donne lieu à la question :

- (QR1) Dans le contexte genevois, qui est propice à l'existence d'une diversité, quels sont les choix des enseignants en matière de langage, IDE et ENA ? Quels sont les usages associés ?

Par ailleurs, dans (Vandeput & Henry, 2018) les auteurs pointent le manque de réflexion didactique des enseignants dans le choix des outils. Qu'en est-il dans le canton de Genève ? Les déterminants des choix sont-ils les mêmes pour le langage, l'IDE et les ENA ? Ces déterminants sont-ils plutôt pragmatiques, fonctionnels ou didactiques ? Par pragmatique nous entendons les facteurs du type choix imposé au niveau de l'établissement, seul environnement installé, etc. ; par fonctionnel nous entendons les facteurs liés par exemple aux modalités d'accès (en ligne ou nécessitant une installation), la langue de l'environnement, etc ; par didactique nous entendons les facteurs liés à un effet, au moins supposé, sur l'apprentissage.

Ceci nous amène à notre deuxième question :

- (QR2) Quels facteurs influent sur la construction par les enseignants de leur contexte numérique d'apprentissage ?

Pour dresser cet état des lieux, nous avons commencé par construire un inventaire des ressources existantes puis nous avons diffusé un questionnaire à destination des enseignants d'informatique du canton de Genève pour identifier choix et facteurs

[1] Les programmes de SNT et de NSI (lycée français) stipulent qu'il faut utiliser "un langage simple d'usage, interprété, concis, libre et gratuit, multiplateforme, largement répandu, riche de bibliothèques adaptées et bénéficiant d'une vaste communauté d'auteurs dans le monde éducatif. (…) le langage retenu est Python (…)" (MEN, 2021, p. 3)

influents sur ces choix. Dans cet article nous présentons le questionnaire et l'analyse des réponses. Pour conclure, nous discutons les premiers résultats obtenus et proposons quelques prolongements à cette étude.

2 Méthode

Pour obtenir des réponses à nos deux questions, nous avons réalisé une enquête en direction des enseignants d'informatique du secondaire 2, du canton de Genève. Cette enquête a pris la forme d'un questionnaire LimeSurvey diffusé en ligne. Pour construire ce questionnaire, afin d'investiguer la QR1, nous avons tout d'abord construit des listes de langages, IDE et ENA. Ces listes ont été élaborées à l'aide de nos observations de terrain, notre expertise du domaine et la littérature professionnelle (de nombreux sites proposent des listes, selon différents critères, d'IDE pour tel ou tel langage). La présence systématique d'une réponse « autre », avec champ à compléter, permet de pallier d'éventuels manques. Pour disposer de quelques éléments permettant une compréhension plus fine qu'une simple liste d'éléments nous avons aussi intégré des questions permettant d'obtenir des informations sur les usages associés (fréquence, modalités, etc.).

Pour obtenir des éléments de réponse à la QR2, nous avons associé, à chaque question relative au choix d'un élément du contexte numérique, une question sur les facteurs qui ont orienté ce choix. Pour faciliter le traitement des réponses nous avons proposé des questions à choix multiples dans une liste plutôt que des réponses ouvertes. Pour contraindre les répondants à se positionner nous avons limité le nombre de choix possibles à trois. Les listes de réponses proposées permettent d'exprimer des raisons pragmatiques (choix de l'établissement), fonctionnelles (multiplateforme ou accès en ligne par exemple) ou didactiques (présence d'une fonctionnalité particulière). A nouveau la réponse « autre » permet d'étendre les choix proposés *a priori*. Le questionnaire est trop long pour être présenté intégralement ici mais est disponible en ligne en format PDF (lien d'accès au questionnaire).

Pour des raisons de pratiques institutionnelles propres à l'organisation de l'enseignement scolaire secondaire genevois, le questionnaire n'a pas été diffusé directement à l'ensemble des enseignants mais aux responsables de la discipline informatique dans les 11 établissements du secondaire 2 à Genève, avec la demande de relayer aux enseignants d'informatique de leur établissement. Nous n'avons pas accès à l'effectif exact mais en estimant à une moyenne de 5 enseignants par établissement cela représente environ 55 répondants potentiels.

Nous avons obtenu 24 réponses dont 13 complètes. L'anonymat et la méthode de diffusion du questionnaire ne nous permettant pas de savoir si les réponses partielles étaient redondantes ou non avec des réponses complètes nous n'avons traité que ces dernières. Nous présentons le résultat de ce traitement dans la section suivante.

3 Résultats

Nous présentons dans un premier temps les éléments permettant de répondre à notre QR1 en décrivant les divers contextes numériques identifiés dans les réponses et quelques éléments sur les usages associés. La section suivante permet de revenir sur la QR2 au travers de l'analyse des critères de choix menant à la construction de ces contextes numériques.

3.1 Contextes numériques et usages

Concernant les langages, tous les enseignants qui ont répondu déclarent utiliser Python. Pour sept d'entre eux c'est le seul travaillé, alors que pour les six autres, au moins un autre langage est utilisé durant l'année. Dans ce cas-là il s'agit systématiquement d'utilisations successives, aucun ne déclare travailler deux langages en parallèle.

Aucune utilisation de C, C++, Java ou TygerJython n'est signalée. Nos observations sur le terrain nous ont cependant permis de constater des utilisations de TygerJython. Ce sont donc au moins sept langages différents qui sont utilisés (**Figure 1**).

Pour les IDE, Thonny[2] et Replit[3] sont les deux IDE très majoritairement utilisés avec les élèves (resp. 11 et 10 utilisateurs dont huit qui utilisent les deux). Le seul autre IDE, cité dans une réponse, est Access, en complément avec Replit et Thonny.

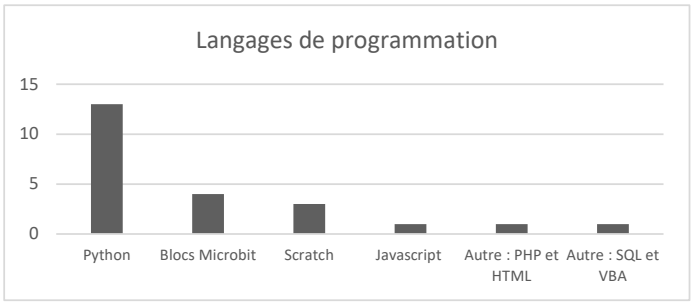


Fig. 1. Langages de programmation utilisés en classe (effectifs d'utilisateurs en ordonnée)

En complément du ou des couples (langage de programmation ; IDE), neuf enseignants déclarent utiliser un environnement numérique d'apprentissage avec leurs élèves en classe (question C2). Dans cette question, les enseignants devaient associer à chaque ENA une fréquence d'utilisation évaluée en nombre de périodes sur l'année. La ***Figure 2*** présente les résultats de cette question. Sur les onze ENA que nous avons proposés comme réponses possibles, sept sont utilisés par au moins deux enseignants et quatre par aucun (question C2). Seul AlgoPython est déclaré comme faisant l'objet

[2] https://thonny.org/
[3] https://replit.com/site/ide

d'un usage fréquent (deux enseignants), quatre ne le sont que très occasionnellement (de 1 à 5 fois dans l'année). Sur cette évaluation de la fréquence, on peut noter un certain décalage entre les réponses à la question C2 et celles obtenues à la question C8 qui demandait aux enseignants de préciser si l'utilisation de l'ENA, lors du travail sur la programmation est occasionnelle, régulière ou systématique puisque pour cette question les effectifs pour chaque alternative sont respectivement 5 ; 3 et 1.

Quand il y a un ENA utilisé, en complément de l'évaluation quantitative de sa place, nous avons questionné les intentions que les enseignants associent à leur usage. L'ENA est essentiellement un élément auxiliaire dans la pratique de l'enseignant, mais il est intégré au travail du savoir. Plus précisément, on distingue deux intentions principales attribuées aux ENA : *faire découvrir certaines notions* (six sur neuf) et *permettre aux élèves de s'entraîner sur des notions déjà travaillées* (sept sur neuf). Un enseignant a répondu que toutes les notions sont travaillées au travers de l'ENA, tandis que deux d'entre eux précisent qu'il ne s'agit que d'une activité visant à occuper les élèves les plus rapides, qui n'est pas exploitée ultérieurement.

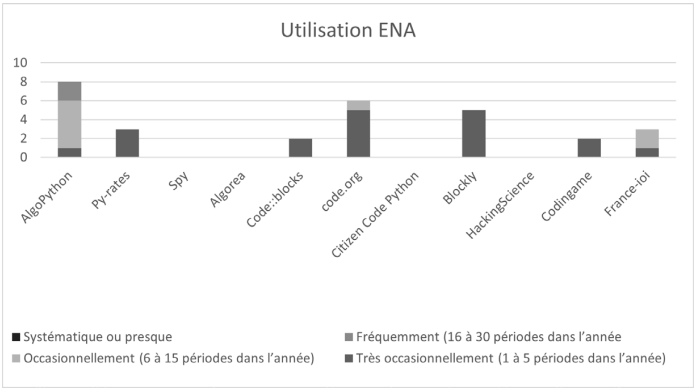


Fig. 2. ENA utilisés et fréquence d'utilisation

On relève également que personne n'utilise l'ENA dans un but évaluatif. La question formulée ne distinguait pas l'évaluation purement formative et l'évaluation à visée sommative, il est donc possible que ce soit cette deuxième forme qui ait été considérée pour formuler les réponses, il y a sinon une certaine contradiction avec la volonté annoncée de suivre le travail des élèves.

Cependant, les pratiques sont plus partagées concernant la place de l'ENA par rapport à l'IDE. Ainsi trois enseignants utilisent en alternance IDE et ENA ; cinq d'abord l'ENA puis l'IDE, ce qui confère une fonction plutôt introductive à l'ENA, et un enseignant qui les présente comme indépendants.

3.2 Facteurs impactant la construction des contextes numériques

Le premier élément définissant le contexte numérique est le langage de programmation. Celui-ci influence largement ceux des IDE et des ENA. Pour le langage il s'agit, pour la totalité des enseignants, de se conformer à des contraintes externes (choix d'équipe, épreuves communes ou contraintes matérielles), pour moins de la moitié d'entre eux, cinq sur treize, ces contraintes rejoignent leur choix personnel.

On constate aussi que, bien que le choix ne soit pas personnel, la grande majorité se dit satisfaite ou très satisfaite du choix puisque seuls trois enseignants se disent peu ou très peu satisfaits du choix de Python. Même s'il est majoritairement positif, le regard porté sur l'utilisation de Python n'est pas exempt de critique. Deux limites sont pointées, la première est la difficulté de rendre les programmes attrayants pour les élèves contrairement à un langage de programmation plus visuel (cette limite est parfois contournée par l'utilisation de la bibliothèque Pygame qui permet de disposer d'un environnement graphique riche). La deuxième limite signalée est relative aux éléments de programmation masqués par un langage de haut niveau comme Python, notamment sur les types des variables. *A contrario*, pour d'autres, cet aspect est cité comme un avantage, car cela rend Python plus accessible pour les élèves. Enfin, l'importante communauté d'utilisateurs et les nombreuses ressources à disposition sont des points positifs de Python selon plusieurs enseignants.

Table 1 : facteurs guidant le choix de l'IDE

Facteur	Effectif
Facilité d'accès dans l'établissement	8
Facilité d'utilisation (sauvegarde)	8
Facilité accès hors de la classe	7
Utilisation hors ligne	4
Exécution pas-à-pas	2
Choix imposé par l'établissement	1
Possibilité de paramétrer la langue de l'IDE	1
Paramétrer l'IDE	1
Autre : possibilité d'installer nouveaux packages	1
Autre : transition directe entre la classe et la maison	1

Si le premier facteur de choix de l'IDE est évidemment sa compatibilité avec le langage travaillé, leur diversité[4] nécessite de réaliser un ou des choix. Dans la table 1, nous présentons les différents facteurs de choix relatifs aux IDE, par ordre décroissant d'importance relativement au nombre de fois où ils ont été cités. On peut noter qu'il s'agit essentiellement de facteurs liés à des aspects fonctionnels (accès, sauvegarde…)

[4] Par exemple sur cette page https://www.commentcoder.com/ide-python/ il y a 11 IDE identifiés pour développer en Python

et que les dimensions qui pourraient donner lieu à une exploitation didactique (exécution pas-à-pas, paramétrages de l'IDE) ne sont citées que de manière marginale.

Concernant les raisons exprimées pour le choix d'un ENA il y a, à nouveau, une diversité importante comme le montre la Table 2 (trois réponses possibles au maximum, non ordonnées). Celle qui est identifiée comme importante par le plus grand nombre d'enseignants (sept sur neuf) est la *possibilité de suivre le travail des élèves*. Il serait intéressant de pouvoir mettre en relation cette déclaration avec l'exploitation des outils de suivi de l'activité proposés par les ENA ; est-elle effective ou non, selon quelles modalités ?

Table 2 : facteurs guidant le choix de l'ENA

Facteur	Effectif
Possibilité de suivi pour l'enseignant (tableau de bord…)	7
La scénarisation de l'apprentissage proposée par l'ENA (ordre des notions…)	4
Langue(s) de l'environnement	3
Présences de rétroactions	3
ENA accessible hors la classe	3
Type d'environnement : jeu sérieux (SG), exerciseur…	2
Présences d'aides (rappels de cours, exemples…)	2
Accès en ligne	2
Possibilité de paramétrer (parcours, accès conditionnel à des ressources)	1
Possibilité d'installation locale	0
Protection des données personnelles	0

Même si la scénarisation de l'apprentissage proposée par l'ENA arrive en deuxième position avec 4 réponses, il est notable qu'aucun des facteurs que l'on peut considérer comme relatifs à la place des savoirs dans l'ENA (scénarisation de l'apprentissage portée par l'ENA ; présences de rétroactions et/ou d'aides) n'est majoritaire. A la différence du choix de l'IDE, ce sont plutôt des facteurs de nature didactique qui semblent guider le choix des ENA.

Enfin, on pourra s'interroger sur le fait que la "protection des données personnelles" n'est citée comme élément important pour le choix d'un ENA par aucun enseignant. Ceci questionne à la fois le rapport que les enseignants entretiennent à cette question, et le lien qu'ils opèrent ou non entre leurs pratiques et l'enseignement qu'ils délivrent puisque la question des données personnelles fait partie des éléments auxquels ils doivent sensibiliser leurs élèves.

3.3 Quelques éléments de synthèse sur les réponses à QR1 et QR2

Le constat qui synthétise l'ensemble de ces résultats est celui de *diversité*.

- Diversité importante des contextes (langage, IDE, ENA) qui définissent les conditions d'apprentissage de la programmation pour les élèves, même si Python est un incontournable[5], articulé ou non avec d'autres éléments.
- Diversité des facteurs qui influencent la construction de ces contextes. On peut cependant noter la place relativement faible accordée à des éléments qui relèvent du didactique dans les réponses recueillies, comme si cette dimension était une préoccupation indépendante du contexte numérique d'exercice.
- Diversité des usages déclarés, notamment sur la place et les objectifs associés aux ENA.

4 Conclusion et perspectives

Dans cette contribution, nous avons dressé un panorama des contextes numériques créés par les enseignants dans le canton de Genève pour l'apprentissage de la programmation. Nous nous inscrivons dans une approche compréhensive des conditions d'enseignement de l'informatique au niveau scolaire, notamment dans le prolongement du travail présenté dans (Jolivet et al., à paraitre) qui se focalise sur la formation initiale des enseignants d'informatique en Suisse Romande.

Cette première enquête met en évidence une grande diversité des contextes numériques d'enseignement, tant pour les moyens utilisés que pour les facteurs qui influent sur les choix des enseignants. Même si le nombre de réponses récoltées est relativement modeste, nous faisons l'hypothèse qu'il est assez représentatif de la réalité genevoise, au regard du nombre d'établissements et d'enseignants. Si un nombre plus important de réponses avait sans doute permis de faire émerger encore quelques facteurs supplémentaires de diversité le constat ne serait pas remis en cause. Cependant, il serait pertinent de pouvoir disposer d'un nombre plus important de réponses pour affiner l'importance relative des différents critères menant à la construction de ces contextes numériques.

Il est aussi possible d'enrichir l'enquête en intégrant les éléments liés à l'enseignement de la programmation qui ne sont pas pris en compte dans le contexte numérique, par exemple les temps réalisés en « mode débranché » et les moyens d'enseignement non numériques exploités (brochures, cahiers d'exercices, etc.).

Par ailleurs, dans le prolongement d'une telle enquête, il serait intéressant de pouvoir confronter les déclarations à la réalité des pratiques, par exemple au moyen d'observations *in situ*. L'accès au terrain que nous donne notre fonction de formateurs ne nous a pas permis d'identifier d'écarts évidents *a priori*, mais cela mériterait d'être confirmé.

D'autre part, cette étude a été menée dans un contexte institutionnel qui favorise l'existence de cette diversité et l'on constate que les enseignants s'en emparent. Il serait intéressant de l'élargir à d'autres institutions d'enseignement de la programmation, par exemple d'autres cantons suisses ou d'autres pays. Par exemple dans le con-

[5] Ce résultat de l'enquête est sans doute à modérer un peu puisque nous avons, depuis la passation du questionnaire, identifié un établissement dans lequel l'apprentissage de la programmation est mené uniquement au travers du triplet JavaScript, HTML et CSS

texte du lycée français, le choix du langage de programmation est contraint mais il serait intéressant d'étudier si les autres éléments du contexte numérique sont variés. La description et la compréhension de ces différents contextes étant sans doute une base nécessaire, sans être exclusive, à la compréhension d'éventuelles différences observées sur l'apprentissage de l'informatique entre institutions. Il s'agirait ainsi d'aborder la question fondamentale de l'étude des effets sur les apprentissages de ces différents contextes.

Références

1. Branthôme, M. (2022). Pyrates : A Serious Game Designed to Support the Transition from Block-Based to Text-Based Programming. In I. Hilliger, P. J. Muñoz-Merino, T. De Laet, A. Ortega-Arranz, & T. Farrell (Éds.), *Educating for a New Future : Making Sense of Technology-Enhanced Learning Adoption* (Vol. 13450, p. 31-44). Springer International Publishing. https://doi.org/10.1007/978-3-031-16290-9_3
2. DGES II. (2021). *Programmes des disciplines enseignées dans la filière gymnasiale au Collège de Genève valable pour les élèves scolarisés dès 2021* (p. 160-165) [Programmes de la filière Gymnasiale]. https://www.ge.ch/document/programmes-disciplines-enseignees-dans-filiere-gymnasiale-au-college-geneve-valable-eleves-scolarises-2021
3. Jolivet, S., Wang, P., & Dechaux, E. (2024). Un retour d'expérience pour identifier constats et problématiques relatifs à la formation des enseignants d'informatique. *Revue RADIX*, *1*.
4. Khazaei, B., & Jackson, M. (2002). Is there any difference in novice comprehension of a small program written in the event-driven and object-oriented styles? *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments*, 19-26. https://ieeexplore.ieee.org/abstract/document/1046336/
5. MEN. (2021). *Programme de NSI de 1ère générale*. Bulletin Officiel de l'Education Nationale. https://eduscol.education.fr/document/30007/download
6. Vandeput, É., & Henry, J. (2018). Apprendre à programmer Comment les enseignants justifient-ils le choix d'un outil didactique ? In G. Parriaux, J.-P. Pellet, G.-L. Baron, & V. Komis (Éds.), *De 0 à 1 ou l'heure de l'informatique à l'école* (p. 325-342). Peter Lang. https://library.oapen.org/bitstream/handle/20.500.12657/47395/9783034333245.pdf#page=327
7. White, G., & Sivitanides, M. (2005). Cognitive Differences Between Procedural Programming and Object Oriented Programming. *Information Technology and Management*, *6*(4), 333-350. https://doi.org/10.1007/s10799-005-3899-2

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Déséquilibre d'engagement filles-garçons dans une activité débranchée de groupe en science informatique

Yong Xin / Lylia Lam[1], Gaëlle Molinari[1,2] et Yann Secq[3]

[1] TECFA, Université de Genève, Suisse
[2] UniDistance, Suisse
[3] LEARN Center, École Polytechnique Fédérale de Lausanne, Suisse

Résumé. Le déséquilibre de genre en informatique est identifié depuis quelques décennies et persiste malgré les actions en œuvre. Depuis la parution du *CS Unplugged*, les activités d'informatique débranchées se sont développées et reposent souvent sur le travail de groupe. Notre étude vise à étudier l'impact d'une activité débranchée sur l'engagement des filles et des garçons en cours de Science Informatique. A travers deux séries d'observations en classe et une séance de co-conception entre enseignantes et équipe de recherche, nous avons travaillé sur la scénarisation en tant que piste d'actions afin de favoriser l'engagement des filles, en réponse au déficit d'intérêt souvent observé au collège dans le domaine de l'informatique. Nos résultats se basent sur l'analyse de comportements verbaux et non-verbaux de groupes d'élèves de 12 à 13 ans lors d'une activité débranchée, et permettent de souligner le potentiel de celle-ci pour l'engagement de chacun·e. Ils révèlent aussi la complexité des facteurs régissant l'engagement.

Mots-clés: Déséquilibre de genre en informatique, activités débranchées, engagement dans les activités de groupe, initiation à la Science Informatique.

1 Introduction

1.1 Déséquilibre de genre en contexte scolaire et en Informatique

Selon les données récentes de l'Office Fédéral de la Statistique, seulement 10 à 15 % des personnes étudiant ou travaillant en informatique en Suisse sont des femmes. Cette disparité, reflétée à l'échelle mondiale, soulève des préoccupations quant aux inégalités économiques et sociales futures, exacerbées par une pénurie croissante de main-d'œuvre qualifiée dans ce secteur (Denner & Campe, 2018).

Les causes de ce déséquilibre entre hommes et femmes dans le domaine informatique sont multifactorielles. Historiquement, en Suisse comme en France, la représentation des femmes dans ce domaine a fluctué, connaissant une baisse après une période initiale difficile, suivie d'une augmentation dans les années 1970, stimulée par les premiers mouvements et lois féministes (Perrot, 2004). Des études récentes indiquent un désintérêt croissant des filles pour l'informatique durant la scolarité obligatoire et secondaire, qui peut être attribuable à une perception négative de leurs compétences en informatique, malgré des aptitudes souvent équivalentes ou supérieures à celles de leurs homologues masculins (Pirttinen et al., 2020). La qualité

pédagogique des cours et le soutien motivationnel des enseignant·e·s sont également cruciaux pour engager les filles en informatique. L'étude de Vieira et Couto (2020) indique que les filles perçoivent souvent ces cours comme ennuyeux, excessivement complexes et déconnectés de leurs intérêts lesquels s'orientent vers des applications centrées sur l'humain et son développement.

La sociologie, en particulier les études de genre, offre un éclairage sur ces déséquilibres. Le concept de « doing gender » (West & Zimmerman, 1987) souligne que le genre est un construit social, façonné par nos interactions quotidiennes. Cette perspective révèle que l'expérience en classe varie selon des facteurs externes, y compris socio-économiques et genrés, influençant la participation et la motivation des élèves.

Une tendance observée dans la scolarité obligatoire est la disproportion de l'attention accordée par les enseignant·e·s aux garçons par rapport aux filles. Cette dynamique peut s'expliquer par une tendance des enseignant·e·s à solliciter davantage les garçons, soit pour les canaliser, soit pour maintenir leur concentration, comme le suggère Howe (1997). Les filles, souvent perçues comme plus studieuses et disciplinées, reçoivent moins d'attention, ce qui peut influencer négativement leurs performances selon la théorie des prophéties auto-réalisatrices de Pygmalion et du Golem (Jahan & Mehrafzoon, 2019). De plus, les enseignant·e·s ont tendance à complimenter les garçons pour leurs réussites et à critiquer plus sévèrement les filles pour leurs erreurs (Seifert & Sutton, 2019), contribuant ainsi à une dévalorisation des compétences des filles.

L'environnement d'apprentissage scolaire, en reflétant et perpétuant les dynamiques sociales existantes (Collet, 2019), crée un « curriculum caché » qui favorise une participation active des garçons et pousse les filles à rester en retrait. Cette dynamique, observée dans diverses disciplines, renforce les inégalités de genre et affecte l'apprentissage des élèves, les garçons apprenant à s'imposer dans l'espace public et les filles à travailler de manière autonome, sans développer la compétence de se mettre en valeur dans un groupe. Dans le cadre des travaux de groupe, un déséquilibre similaire est observé, menaçant autant les échanges verbaux que non-verbaux. Les recherches indiquent que les garçons dominent souvent les interactions de groupe, entraînant une asymétrie de participation en leur faveur. L'étude d'Aguillon et al. (2020) a montré que les garçons étaient plus engagés que les filles dans les interactions de classe, répondant plus souvent et de manière volontaire dans le cadre des discussions en petits groupes. Cette tendance s'est répétée sur deux semestres, indiquant une domination masculine dans la participation en classe au-delà de leur proportion numérique. De plus, dans les programmes de robotique en milieu scolaire, les garçons se concentrent davantage sur les aspects opérationnels tandis que les filles portent plus d'attention à la dynamique de groupe, renforçant ainsi les stéréotypes de genre et limitant les opportunités d'apprentissage collaboratif équilibré (Ardito et al., 2020).

Face à la réintroduction de l'informatique dans les programmes scolaires obligatoires, il est essentiel de reconnaître et d'adresser ces déséquilibres pour intégrer des stratégies pédagogiques favorisant une participation équitable des filles et des garçons. Cela s'aligne sur les objectifs d'équité en éducation de l'OCDE (2013), qui visent à garantir que des circonstances telles que le genre, l'origine ethnique ou le milieu familial ne soient pas des obstacles à la réalisation du potentiel éducatif de

chaque individu. Une attention particulière doit être accordée à la pédagogie développée au secondaire 1 (élèves de 11 à 16 ans environ), période critique où les différences de représentations et d'intérêt pour l'informatique se développent fortement.

1.2 Des activités débranchées pour l'enseignement de la Science Informatique

Les activités dites « débranchées » jouent un rôle crucial dans l'enseignement de la Science Informatique (S.I.), en permettant aux élèves de découvrir pas à pas des concepts complexes, sans recourir à des outils technologiques comme les ordinateurs, tablettes ou robots. Par exemple, un jeu de tri physique peut enseigner les principes des algorithmes de tri, illustrant de manière tangible les concepts abstraits. Ces activités offrent une alternative accessible, tant sur le plan matériel qu'économique, et s'avèrent durables (Bell et al., 2009).

En contexte scolaire, ces activités proposent une approche pédagogique renforçant l'interaction directe entre enseignants et élèves. Elles intègrent des aspects kinesthésiques et ludiques pour faciliter la mobilisation des compétences en résolution de problèmes, tout en profitant des avantages du travail de groupe. Un exemple marquant est l'activité de construction d'un réseau en classe, où les élèves apprennent la communication et le fonctionnement des réseaux informatiques par le biais d'une simulation interactive. Ces activités débranchées sont particulièrement efficaces pour enseigner des concepts de base de l'informatique, en offrant une compréhension plus intuitive et engageante. Elles permettent de contourner les distractions et les complications potentielles liées à l'utilisation des machines, focalisant l'attention des élèves sur les principes sous-jacents de l'informatique. De plus, elles développent des compétences transversales telles que le travail d'équipe, la pensée critique et la créativité, essentielles à la formation des futurs informaticien·e·s.

Néanmoins, l'approche débranchée ne convient pas à tous les publics, et les connexions entre les activités débranchées et les compétences techniques en informatique peuvent être floues pour les apprenant·e·s. Selon Chen et al. (2023), les activités débranchées améliorent significativement les compétences en pensée computationnelle lorsqu'elles sont adaptées à l'âge des élèves et mises en œuvre sur une durée limitée pour conserver leur engagement. Par ailleurs, Alayrangues et al. (2017) soulignent que bien que ces activités permettent d'aborder de manière ludique des problèmes complexes et de favoriser la démarche scientifique chez les enfants et adolescents, elles ont leurs limites en termes de compétences informatiques techniques. Pour des compétences avancées, notamment en programmation, il est nécessaire de compléter ces activités par des expériences pratiques sur des machines, en utilisant des langages de programmation comme Scratch ou Python. Alayrangues et al. (2017) mettent en évidence que l'informatique débranchée, tout en offrant une introduction riche à la science informatique, doit être équilibrée avec des activités pratiques pour former non seulement des informaticien·e·s, mais aussi des citoyen·e·s capables de comprendre les enjeux informatiques de leur société. Ce juste équilibre est crucial pour assurer un apprentissage informatique complet et pertinent.

1.3 Questions de recherche

Notre recherche s'inscrit dans le contexte des déséquilibres de genre en informatique, exacerbés par les dynamiques scolaires et les interactions au sein des travaux de

groupe. Nous nous concentrons sur les activités débranchées en science informatique, qui servent à introduire les élèves à la pensée algorithmique par le biais de la résolution de problèmes en groupe, sans l'utilisation de la technologie.

L'objectif de notre étude est d'explorer les différences d'engagement (Q1) et de désengagement (Q2) entre filles et garçons dans le cadre de ces activités. Nous envisageons que ces activités, en raison de leur format de groupe, peuvent révéler des dynamiques relationnelles complexes, y compris des jeux de pouvoir et de positionnement social (Langenhove & Harré, 1994). Notre but est de comprendre comment ces dynamiques interagissent avec le genre, en examinant les variations de ces jeux de pouvoir entre filles et garçons (Q3), dans le contexte général de l'introduction à la pensée computationnelle.

Pour répondre à ces questions, nous avons mené une étude empirique de terrain en contexte réel, impliquant des enseignantes de secondaire 1 et leurs élèves. Des données vidéo ont été recueillies en classe, au cours d'une activité débranchée de groupe en sciences informatiques, avant et après une séance de co-conception avec les enseignantes. Cette méthode nous permet d'analyser les différences d'engagement et les dynamiques de groupe en fonction du genre, offrant ainsi des insights précieux sur l'impact des déséquilibres de genre dans l'enseignement de l'informatique.

2 Dispositif de recherche en milieu écologique

Notre approche repose sur la Recherche Orientée par la Conception (ROC ; e.g., Sanchez & Monod-Ansaldi, 2015), qui se concentre sur la création de dispositifs pédagogiques en contexte authentique. Ainsi, notre étude vise à co-concevoir, en collaboration avec les enseignantes impliquées, un scénario d'apprentissage visant à encourager un engagement optimal et équitable des filles et des garçons dans une activité débranchée.

Trois enseignantes, Mélanie, Catherine et Inés (prénoms d'emprunt), se sont portées volontaires pour participer à notre étude, permettant l'accès à un terrain de recherche réparti sur trois écoles du Canton de Vaud (Suisse). La population associée à l'étude concerne 54 à 58 élèves (dont 28 à 29 filles) de 12-13 ans distribué·e·s en 3 classes du cycle 3 (niveau secondaire). Des groupes de 4-5 élèves ont été formés à la convenance soit des enseignantes (classes de Mélanie et Catherine), soit des élèves (classe d'Inès avec néanmoins la consigne de créer des groupes mixtes).

Le processus de collecte de données s'est déroulé en trois étapes distinctes, comme illustré dans la Figure 1. La première étape a consisté en une observation en classe, suivie d'une séance de co-conception impliquant les enseignantes et l'équipe de recherche. La troisième et dernière étape a été une seconde observation en classe.

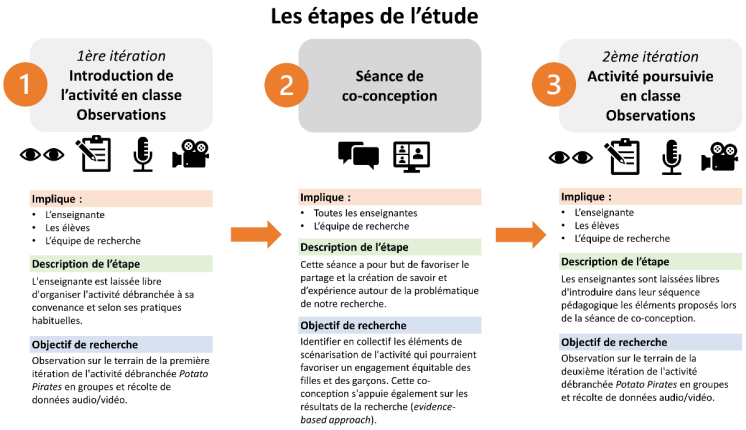


Figure 1. Les trois étapes de collecte de données.

Dans les deux séances d'observation (étapes 1 et 3), les enseignantes ont initié leurs classes à 'Potato Pirates', un jeu de cartes sérieux et compétitif, conçu pour développer des compétences algorithmiques. Dans ce jeu, chaque élève est le capitaine d'une flotte de navires pirates et doit élaborer des stratégies pour couler les flottes adverses. Cela se fait à travers la conception de courts algorithmes, où les élèves utilisent la décrémentation de variables (représentant le nombre de pirates sur un navire) et emploient des structures de contrôle classiques telles que boucles et instructions conditionnelles. Ces concepts, fondamentaux en programmation, sont abordés de manière intuitive, permettant aux élèves d'exploiter l'imbrication de structures de contrôle pour maximiser les dommages infligés aux navires ennemis.

Les élèves étaient répartis en îlot de 4 à 5 participant·e·s avec un dictaphone sur la table. Un des groupes était en plus filmé par une caméra localisée sur leur table. Une dernière caméra avec un plan plus large captait l'ensemble de la classe sauf une table hors champs avec les élèves ne souhaitant pas être filmé·e·s. Lors de la première séance d'observation, les enseignantes ont mis en œuvre l'activité débranchée en suivant leurs méthodes habituelles, sans aucune intervention de notre part.

La séance de co-conception subséquente a réuni les trois enseignantes concernées ainsi que les 3 chercheur·se·s avec un déroulement en quatre phases : (1) une collecte initiale des impressions de chaque participant·e sur la question des disparités de genre en informatique, (2) la présentation des résultats clés d'une revue de littérature sur le même sujet, (3) le partage des points de vue des enseignantes concernant ces résultats de recherche, et une discussion sur l'intégration de ces résultats dans leurs pratiques pédagogiques, (4) l'élaboration de stratégies pour améliorer le scénario de l'activité débranchée, dans le but de stimuler l'engagement des élèves et, plus spécifiquement, de promouvoir une participation équilibrée entre filles et garçons. Deux modifications majeures ont été adoptées : l'introduction de rôles explicites au sein des groupes et la cessation de l'activité dès qu'un·e participant·e est éliminé·e. Au niveau des objectifs pédagogiques de l'activité, seule la boucle à compteur (*for*) était présente lors de la

première séance tandis que la boucle à évènement (*while*) a été ajoutée pour la seconde séance. Lors de la troisième étape, cette version révisée de l'activité débranchée a été à nouveau présentée aux mêmes élèves, en utilisant le même protocole d'observation.

3 Cadre d'analyse et principaux résultats

3.1 Grille de classification des comportements

Afin de répondre à nos objectifs de recherche, nous avons concentré notre étude sur deux ensembles de données (étapes 1 et 3) provenant d'une classe unique, celle dirigée par Mélanie. Nous avons spécifiquement examiné les données vidéo des groupes filmés (celui de l'étape 1 et celui de l'étape 2), choisissant cette approche pour éviter les ambiguïtés d'interprétation liées aux attitudes non verbales, difficilement discernables à partir des enregistrements audio. Il est important de noter que les deux groupes filmés n'étaient pas identiques. Nous avons manuellement transcrit et codé les interactions de ces deux groupes en utilisant une grille de classification des comportements que nous avons développée spécifiquement pour cette étude. Cette grille s'inspire de la typologie de Bales (1959), distinguant les comportements liés à la tâche des comportements socio-émotionnels, et d'une échelle mesurant les dimensions cognitives, émotionnelles, comportementales et sociales de l'engagement dans les forums de discussion (Nleme-Ze & Molinari, 2022). Notre grille met l'accent sur la distinction entre les comportements d'engagement et de désengagement dans la tâche. En ce qui concerne les comportements engagés, nous avons porté une attention particulière à ceux associés aux déséquilibres de « pouvoir », comme défini par Dookie (2015), où une hiérarchie, qu'elle soit momentanée ou continue, se manifeste entre les participants. Cette approche est basée sur l'hypothèse que les différences d'engagement entre filles et garçons seraient plus évidentes lors de ces interactions impliquant un positionnement social. Nous avons identifié deux types de comportements de positionnement social : un lié à l'application des règles du jeu, et l'autre à l'exercice de l'autorité sur les autres membres du groupe. L'analyse des données a été réalisée avec le logiciel ATLAS.ti.

Table 2. Extrait de la grille de codage utilisée pour classer les comportements observés

Code	Définition	Exemple
Engagement : Autorité	*Prise de position d'autorité de la part d'un·e élève sur un·e autre.*	*Simon à Léa: Bon, mets toutes tes cartes alors.*
Engagement : Règles du jeu	*Prise de position d'autorité dans le but de faire observer les règles du jeu.*	*Simon à Léa: Tire 2 cartes.*
Désengagement : Décrochage	*Comportement en lien avec l'activité, mais qui manifeste un désengagement actif.*	*Daphnée à tous: *mets sa tête entre ses bras sur la table* Moi, je m'ennuie.*
Désengagement : Hors-sujet	*Comportement caractérisé par l'absence de rapport avec l'activité observée.*	*Léa à S: C'est moi où c'est les seuls dans la classe à muer ?*

3.2 **Principaux résultats**

Résultats généraux. Dans la séance 1, le groupe observé impliquait 5 élèves, 2 filles (Léa et Martine) et 3 garçons (Paul, Simon et Yanis). Au total, le groupe a manifesté 787 comportements verbaux et non-verbaux. 45% des comportements ont été produits par les filles, 55% par les garçons. Les 10% supplémentaires liés à l'activité des garçons s'expliquent par le fait qu'ils sont majoritaires dans le groupe.

Dans la séance 2, le groupe observé impliquait également 5 élèves, mais la distribution était légèrement différente avec 3 filles (Daphnée, Léa et Zoé) et 2 garçons (Nico et Simon). En comparaison à la séance 1, un nombre plus important de comportements a été observé ($N = 1153$). 57% de ces comportements ont été attribués aux filles contre 43% par les garçons. Les 14% supplémentaires liés à l'activité des filles s'expliquent par le fait qu'elles sont majoritaires dans le groupe.

(a) Qui s'adresse à qui au sein du groupe ?

Dans la séance 1, les filles (minoritaires) se sont d'abord adressées aux garçons ($N = 150$) puis au groupe entier ($N = 103$) et dans une moindre mesure aux autres filles ($N = 58$). En revanche, les garçons (majoritaires) se sont adressés presque autant à leurs pairs masculins ($N = 150$) que féminins ($N = 133$) puis de façon moindre au groupe ($N = 73$).

Durant la séance 2, les résultats sont légèrement différents. Les filles (majoritaires) se sont d'abord adressées à leurs collègues masculins ($N = 209$) puis féminins ($N = 147$), et dans une moindre mesure au groupe entier ($N = 124$). Les garçons (minoritaires) se sont adressés majoritairement aux filles ($N = 229$) et dans une moindre mesure à leurs pairs masculins ($N = 93$) et au groupe dans sa globalité ($N = 76$).

Table 3. Distribution des interactions au sein du groupe entre les 2 séances

		Séance 1	Séance 2	Total
Filles s'adressant	Aux filles	$N = 58$	$N = 147$	$N = 205$
	Aux garçons	$\mathbf{N = 150}$	$\mathbf{N = 209}$	$\mathbf{N = 359}$
	Au groupe	$N = 103$	$N = 124$	$N = 227$
Garçons s'adressant	Aux filles	$\mathbf{N = 133}$	$\mathbf{N = 229}$	$\mathbf{N = 362}$
	Aux garçons	$N = 150$	$N = 93$	$N = 243$
	Au groupe	$N = 73$	$N = 76$	$N = 149$

Comme indiqué dans le Tableau 3, globalement, les filles se sont adressées plus fréquemment à leurs collègues masculins que féminins, et ce qu'elles soient minoritaires (séance 1) ou majoritaires (séance 2). C'est l'inverse qui est observé pour les garçons. En d'autres termes, les interactions sont majoritairement mixtes. Un résultat intéressant montre que les filles se sont adressées plus fréquemment au groupe dans sa globalité en comparaison avec les garçons.

(b) Différences filles-garçons en termes de comportements engagés

8

Table 4. Proportion de comportements engagés chez les filles et les garçons entre les 2 séances

	Pouvoir/règles du jeu		Pouvoir/autorité		Engagement dans la tâche		Total
	S1	S2	S1	S2	S1	S2	
Filles	6 soit 3 par fille	7 soit 2.3 par fille	**13 soit 6.5 par fille**	19 soit 6.3 par fille	**296 soit 148 par fille**	449 soit 149.7 par fille	790
Garçons	**14 soit 4.7 par garçon**	**29 soit 14.5 par garçon**	17 soit 5.7 par garçon	**22 soit 11 par garçon**	397 soit 132.33 par garçon	**397 soit 198.5 par garçon**	**876**

Le Tableau 4 montre un déséquilibre filles-garçons en matière d'engagement dans la tâche avec un nombre total de comportements engagés plus important pour les garçons que pour les filles. Les garçons produisent plus de comportements de type « faire observer les règles du jeu » que les filles et ce quelle que soit la session observée. Alors qu'elles sont minoritaires, les filles dominent les garçons en termes de nombre de comportements de type « engagement dans la tâche » et « soumettre son autorité » dans la session 1, tandis que c'est l'inverse qui se manifeste dans la session 2 alors que les garçons sont en minorité. Cela suggère que ce n'est pas tant le genre, mais plutôt le statut de minorité qui pourrait influencer l'engagement et les comportements d'autorité.

(c) Différences filles-garçons en termes de comportements désengagés

Comme l'indique le Tableau 5, le nombre de comportements désengagés est massivement plus important chez les filles que chez les garçons. Contrairement à l'attendu, le nombre de comportements désengagés augmente entre les deux sessions, pour les filles comme pour les garçons, et cela pourrait s'expliquer par le fait que c'est la même activité débranchée qui est proposée deux fois aux élèves. Chez les filles, le désengagement se manifeste sous la forme de comportements de décrochage dans la session 1, et sous la forme de discussions « hors-sujet » dans la session 2. Un nombre très faible de comportements de désengagement est observé chez les garçons dans la session 1, tandis qu'ils ont tendance à se désengager de la tâche par le biais de discussions « hors-sujet » dans la session 2.

Table 5. Proportion de comportements désengagés chez les filles et les garçons aux 2 séances

	Décrochage		Hors-sujet		Total
	S1	S2	S1	S2	
Filles	**31 soit 15.5 par fille**	**65 soit 21.67 par fille**	**10 soit 5 par fille**	**122 soit 40.67 par fille**	**228**
Garçons	2 soit 0.67 par garçon	15 soit 7.5 par garçon	1 soit 0.33 par garçon	31 soit 15.5 par garçon	49

Lors de la session 2, parmi les discussions « hors-sujet », il est à noter que des conversations inappropriées à connotation sexuelle, initiées par des filles et impliquant un garçon (Figure 2), ont occupé une place significative dans le déroulement du jeu. Ces échanges ont contribué à un désengagement plus prononcé au sein du groupe.

37:59 **Nico à Simon:** T'imagines, là, il y en a encore une plus grosse [d'attaque].
37:59 **Daphnée à Léa:** Petit cochon ! [N]
38:01 **Léa, Daphnée:** **gloussent et rigolent**
38:02 **Zoé à Léa:** C'est à toi Léa hein.
38:06 **Daphnée à tous:** **fait des bruits de grognements de cochon**
38:04 **Nico à tous:** Oh non...**exaspéré**
38:07 **Léa à Nico:** **imite Nico** Oh non! Tu [???] avec nous ?
38:09 **Simon à Léa:** Tire 2 cartes.
38:09 **Daphnée à Nico:** Beuuuh ! **rigole**
38:13 **Léa à Nico:** Tu veux voir mon petit...**rigole**
38:13 **Nico à Léa:** Non. Non. Jamais.
38:14 **Daphnée à Nico:** Tu veux voir mon chat? **glousse** Tu veux voir mon chaton?
38:15 **Nico à Simon:** Viens, on change de groupe.

Figure 2. Quand l'intime s'immisce dans la tâche.

Le dialogue représenté dans la Figure 3 met en lumière une autre dimension de l'interaction entre les conversations personnelles et l'engagement dans la tâche. Ce dialogue s'inscrit dans un phénomène déjà documenté par Depoilly (2017), à savoir, un « effet de dissimulation » dans la transgression scolaire chez les filles. Cet effet permet aux filles de naviguer habilement entre la conformité aux règles de la classe et des interactions sociales plus informelles. Bien qu'elles mènent des discussions non liées à la tâche, elles semblent néanmoins suivre le jeu et y être pleinement investies.

Une autre interprétation suggère que cette dissimulation pourrait être une réponse aux stéréotypes de genre. Les filles pourraient adopter des comportements qui semblent conformes aux attentes stéréotypées, comme discuter de sujets considérés comme « superficiels », tout en étant en réalité engagées dans la tâche. Ce comportement pourrait créer ce que nous appelons un « faux négatif » dans l'observation de leur engagement, où elles semblent désengagées alors qu'elles sont en fait actives mais de manière moins visible ou moins conforme aux attentes traditionnelles.

Cette complexité suggère que les différences apparentes de désengagement entre filles et garçons doivent être interprétées avec prudence. En effet, ces différences peuvent masquer des dynamiques sociales complexes, influencées par des facteurs tels que le genre et les attentes sociales, qui se manifestent de manière nuancée.

48:39 **Daphnée à Nico:** Si, je l'ai attaqué et elle- [Léa]
48:39 **Léa à Nico:** Elle m'a pris trois trucs ! Elle m'a pris trois trucs ! J'ai même assez pour ne pas aller à la piscine.
48:44 **Nico à Simon:** C'est une [???], elles ne s'attaquent pas les deux. **agite la main entre Léa et [???]**
48:45 **Simon à Nico:** C'est bon, c'est pour aller [???].
48:46 **Nico à Simon:** Ce qui me marque [???].
48:46 **Daphnée à Léa:** **caresse les cheveux de Léa** Tu vas quand même aller à la piscine avec nous !
48:47 **Léa à Daphnée:** Oui oui. On s'en bat des pots de fleurs.
48:49 **Simon à Daphnée:** Allez Daphnée.
48:51 **Nico à Daphnée:** C'est à toi hein !
48:52 **Daphnée à Nico:** Oui baby c'est à moi.

Figure 3. Exemple de dissimulation dans la transgression scolaire.

10

4 Conclusion et perspectives

Cette étude se focalise sur les différences d'engagement entre les filles et les garçons dans une activité débranchée en science informatique, réalisée en groupes mixtes au niveau secondaire. En analysant les comportements d'engagement et de désengagement, ainsi que sur les jeux de pouvoir au sein des groupes, nous avons cherché à identifier les disparités liées au genre et à améliorer les scénarios pédagogiques pour un engagement plus équilibré. Cette démarche s'est appuyée sur une collaboration étroite avec les enseignantes pour co-élaborer des stratégies visant à optimiser l'engagement des élèves dans ce type d'activités. Notre analyse a combiné l'observation des comportements manifestes et l'interprétation des interactions sociales plus nuancées, en tenant compte des défis de mesurer l'engagement dans un contexte éducatif.

Les principaux résultats que nous observons sont les suivants :

- Le niveau de participation des élèves, mesuré en termes de comportements verbaux et non-verbaux, semble être influencé par leur statut de majorité ou de minorité dans le groupe, plutôt que par leur genre.
- Les interactions au sein des groupes sont majoritairement mixtes, mais les filles ont tendance à s'adresser plus fréquemment au groupe dans son ensemble que les garçons. Cette observation pourrait indiquer que les filles ont une orientation plus collective dans la résolution de tâches, tandis que les garçons pourraient avoir une approche plus individuelle.
- Bien que les garçons montrent globalement un engagement plus élevé dans la tâche, le statut de majorité ou de minorité dans le groupe semble être un facteur clé. Lorsqu'elles sont en minorité, les filles montrent un engagement plus fort et plus de comportements d'autorité ; le même phénomène est observé chez les garçons lorsqu'ils sont en minorité. Cela suggère que le statut de majorité ou de minorité peut avoir une influence plus grande sur l'engagement et les jeux de pouvoir que le genre lui-même.
- Le désengagement n'est pas uniforme et peut être influencé par des facteurs complexes tels que le genre, les stéréotypes et les dynamiques de groupe. Les filles, bien qu'apparemment plus désengagées que les garçons, peuvent en réalité être investies dans la tâche, ce qui nécessite une interprétation plus nuancée des mesures d'engagement. Ainsi, le comportement de dissimulation dans la transgression scolaire voire dans la provocation à l'égard des garçons pourrait être une réponse aux stéréotypes de genre, où elles semblent conformes aux attentes sociales tout en restant engagées dans la tâche.
- L'introduction des rôles n'a pas impacté significativement l'engagement des élèves et le rôle spécifique visant à conserver l'attaque la plus efficiente afin d'insister sur les structures de contrôle et leur imbrication n'a pas été mobilisé par les enseignantes lors de la phase d'institutionnalisation post-activité.

Nous pouvons donc conclure que les différences observées en matière d'engagement et de désengagement dans l'activité débranchée ne semblent pas principalement liées au genre, mais plutôt au statut de majorité ou de minorité au sein du groupe. Cela indique que l'activité débranchée ne met pas nécessairement en évidence des disparités de genre, mais révèle plutôt des dynamiques de groupe complexes. De façon intéressante, les résultats montrent que le désengagement est un phénomène

nuancé, où les filles peuvent sembler désengagées en raison de stéréotypes de genre auxquels elles tentent de se conformer, tout en restant néanmoins actives dans la tâche.

Enfin, malgré les ajustements convenus avec les enseignantes lors de la séance de co-conception, notamment l'assignation de rôles spécifiques aux élèves pour atténuer les comportements influencés par les stéréotypes de genre, nos résultats indiquent une augmentation des comportements désengagés lors de la deuxième session. Ce résultat pourrait être attribué en partie à l'absence d'évaluation des performances des élèves dans cette activité débranchée, une situation également observée pour d'autres activités en science informatique dans le Canton de Vaud. L'absence d'évaluation formative et/ou sommative pourrait avoir limité l'investissement des élèves, soulignant ainsi l'importance que peuvent revêtir les mécanismes de reconnaissance et de validation des compétences dans l'engagement des apprenant·e·s. Cette interprétation s'aligne avec d'autres recherches dans les domaines STIM (science, technologie, ingénierie et mathématiques) qui montrent l'importance du *feedback* des enseignant·e·s, en l'occurrence sur le processus d'apprentissage, pour renforcer le sentiment d'auto-efficacité et développer un état d'esprit de croissance (*growth mindset*) chez les filles (Paechter et al., 2020).

Une autre explication des résultats observés pourrait être liée aux défis rencontrés par les enseignant·e·s dans l'orchestration efficace de l'activité de groupe. Ceci souligne la nécessité d'une recherche collaborative orientée par la conception en vue d'un meilleur accompagnement des enseignant·e·s pour adresser les complexités liées à l'interaction entre les spécificités de l'informatique en tant que discipline didactique, de l'activité débranchée utilisée pour l'enseigner, des dynamiques de groupe et de l'impact de divers facteurs, y compris le genre, sur l'engagement des élèves. Notre étude est une première étape vers un programme de recherche plus large qui vise à fournir aux enseignant·e·s des outils et des connaissances pour mieux comprendre et appréhender ces complexités. En particulier, cela implique de développer une compréhension approfondie des spécificités de l'informatique débranchée, de son potentiel en tant qu'outil pédagogique pour surmonter les stéréotypes de genre, et des stratégies pour scénariser, planifier et orchestrer efficacement ces activités dans un environnement d'apprentissage collaboratif équitable, inclusif et engageant.

Remerciements: nous tenons à remercier les enseignantes qui se sont portées volontaires pour réaliser cette étude et se sont impliquées sur l'ensemble des phases de ce travail. Sans leur implication et leurs préoccupations partagées, cette étude n'existerait pas, les auteur·e·s tiennent sincèrement à les remercier pour tout cela.

Références

Aguillon, S. M., Siegmund, G. F., Petipas, R. H., Drake, A. G., Cotner, S., & Ballen, C. J. (2020). Gender differences in student participation in an active-learning classroom. *CBE—Life Sciences Education*, *19*(2), ar12.

Alonso, M. T., Barba-Sánchez, V., López Bonal, M. T., & Macià, H. (2021). Two Perspectives on the Gender Gap in Computer Engineering: From Secondary School to Higher Education. *Sustainability*, *13*(18), 10445.

Alayrangues, S., Peltier, S., & Signac, L. (2017). Informatique débranchée: Construire sa pensée informatique sans ordinateur. In *Colloque Mathématiques en Cycle 3 IREM de Poitiers* (pp. 216-226).

Ardito, G., Czerkawski, B., & Scollins, L. (2020). Learning computational thinking together: Effects of gender differences in collaborative middle school robotics program. *TechTrends*, *64*(3), 373-387.

Bales, R. F. (1950). A set of categories for the analysis of small group interaction. American *Sociological Review*, *15*(2), 257-263.

Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology*, *13*(1), 20-29.

Chen, P., Yang, D., Metwally, A. H. S., Lavonen, J., & Wang, X. (2023). Fostering computational thinking through unplugged activities: A systematic literature review and meta-analysis. *International Journal of STEM Education*, *10*(1), 1-25.

Collet, I. (2015). Faire vite et surtout le faire savoir. Les interactions verbales en classe sous l'influence du genre. *Revue internationale d'ethnographie*, (4), 6-22.

Collet, I. (2019). *Les oubliées du numérique*. Le passeur.

Denner, J., & Campe, S. (2018). Equity and inclusion in computer science education. Dans S. Sentance (Éd.), *Computer science education in school: Perspectives on teaching and learning* (pp. 189–206). Bloomsbury.

Depoilly, S. (2017). Les filles de milieux populaires et l'école : de la docilité aux arts de la ruse. Dans H. Buisson-Fenet (Éd.), *École des filles, école des femmes: L'institution scolaire face aux parcours, normes et rôles professionnels sexués* (pp. 119-129). De Boeck Supérieur.

Dookie, L. (2015). Examining marginalizing acts of social positioning in mathematical group work: Towards a better understanding of how microaggressions and stereotype threat unfold in intergroup interactions. University of Toronto (Canada).

Jahan, F., & Mehrafzoon, D. (2019). Effectiveness of pygmalion effect-based education of teachers on the students' self-efficacy and academic engagement. *Iranian Journal of Learning & Memory*, *1*(4), 17-22.

Langenhove, L. V., & Harré, R. (1994). Cultural stereotypes and positioning theory. *Journal for the Theory of Social Behaviour*, *24*(4), 359-372.

Nleme Ze, Y. S., & Molinari, G. (2022). Développement et validation psychométrique d'une échelle de mesure de l'engagement des apprenants dans les forums de discussion des MOOC. *Distances et médiations des savoirs. Distance and Mediation of Knowledge*, *40*(40).

Paechter, M., Luttenberger, S., & Ertl, B. (2020, August). Distributing Feedback Wisely to Empower Girls in STEM. In Frontiers in Education (Vol. 5, p. 141). Frontiers Media SA.

Perrot, M. (2004). *Quelle mixité pour l'école ?*. CNDP Albin Michel.

Pirttinen, N., Hellas, A., Haaranen, L., & Duran, R. (2020, October). Study Major, Gender, and Confidence Gap: Effects on Experience, Performance, and Self-Efficacy in Introductory Programming. In 2020 IEEE Frontiers in Education Conference (FIE) (pp. 1-7).

Sanchez, É., & Monod-Ansaldi R. (2015). Recherche collaborative orientée par la conception. Un paradigme méthodologique pour prendre en compte la complexité des situations d'enseignement-apprentissage. *Éducation & didactique*, *9*(2), 73-94.

Seifert, K., & Sutton, R. (2019). Gender differences in the classroom. Student diversity. Educational Psychology.

Vieira, A. S., & Couto, M. J. V. (2020). Gender differences as influence factors to choose computer science as a professional career option. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, *15*(3), 205-210.

West, C., & Zimmerman, D. H. (1987). Doing gender. *Gender & society*, *1*(2), 125-151.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Informatique et durabilité, une difficile transposition didactique

Baptiste de Goer[1,3], Micha Hersch[2], and Sophie Quinton[1]

[1] Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, F-38000 Grenoble, France
[2] Haute Ecole Pédagogique Vaud, 1007 Lausanne, Suisse
[3] Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, F-38000 Grenoble, France

Résumé Dans le contexte actuel de changement climatique, une demande forte d'enseigner à l'école les enjeux de durabilité liés au numérique émerge de la société et des institutions. Or, la transposition didactique de ces sujets dans les cours d'informatique soulève un certain nombre de difficultés. Dans un premier temps, ce papier présente un aperçu des savoirs scientifiques sur la question et des pratiques d'enseignement académique associées, puis met en avant la difficile articulation entre l'informatique comme discipline scientifique et les questions de durabilité. Dans un second temps, inspirés de réflexions issues de l'éthique de la technologie, nous évoquons la possibilité de réancrer l'informatique dans le contexte économique, social, politique, mais également matériel, de son développement et de ses applications ; ce qui constituerait une évolution importante de l'informatique aussi bien au niveau scolaire qu'académique.

Keywords: Durabilité · Transposition didactique · Pratiques émergentes · Enseignement secondaire

1 Introduction

Alors que les effets du changement climatique se font sentir de plus en plus intensément [6], que la perte de la biodiversité se poursuit de façon alarmante [25] et que d'autres limites planétaires sont dépassées [36], une mobilisation des différents secteurs de la société, nécessaire pour inverser la tendance, se met en place. L'école, de par son rôle central dans le façonnement des valeurs, des savoirs et des comportements, est perçue comme un instrument important pour affronter ces défis, ce qui a mené à l'inclusion des questions de durabilité dans les programmes scolaires, sous la forme de l'éducation au développement durable [13]. Plus récemment, la question des enjeux environnementaux liés aux technologies numériques a également émergé dans une partie des programmes scolaires. La protection de la santé, du bien-être et de l'environnement est une préoccupation transversale de l'éducation au numérique, autant au niveau de l'Europe via son cadre de compétences numériques (*DigComp* [11]), qu'en France via son *Cadre de Référence des Compétences Numériques* (CRCN) [32]. Dans ce contexte, le monde francophone commence à voir émerger des travaux portant notamment

Volume 1, pages 23–32.

sur l'éducation à la sobriété numérique [15,5]. Cette préoccupation apparaît également dans les programmes disciplinaires, par exemple en Suisse francophone, où les programmes d'informatique du secondaire supérieur demandent de traiter des enjeux sociaux du numérique, dont la question de son impact environnemental. De même, cette problématique figure dans le programme d'informatique de la formation générale pour adultes du Québec, ainsi que dans les programmes de *Sciences Numériques et Technologie* (SNT) en France parmi les « impacts sur les pratiques humaines ».

Il existe ainsi une demande institutionnelle quant à l'intégration des enjeux de durabilité au sein des cours d'informatique, dont les enseignantes et enseignants, et plus largement la discipline scolaire elle-même, vont devoir se saisir. Y répondre soulève bien entendu des questions d'ordre didactique, le sujet à intégrer étant généralement perçu comme externe à l'informatique en tant que science. Dans cet article, nous évoquons quelques-unes des difficultés rencontrées et proposons comme sources d'inspiration pour y faire face des approches issues de l'enseignement supérieur.

2 Quelques enjeux de la transposition didactique

Dans sa conception classique [12], la didactique définit la *transposition didactique* comme le processus par lequel des savoirs et des pratiques d'enseignement académiques sont adaptés et transformés en savoirs et pratiques scolaires. Cette transposition a lieu deux fois : une fois dans le milieu extra-scolaire, par exemple dans le cadre de la rédaction des programmes et des manuels scolaires (transposition dite externe) ; puis une autre fois en classe par les enseignants et enseignantes (transposition dite interne). Si cette transposition didactique s'applique bien pour la science informatique, son application pour les questions de durabilité en lien avec le numérique est plus compliquée, et ce pour plusieurs raisons.

2.1 Des savoirs scientifiques non stabilisés

Une première difficulté relève du fait que les savoirs scientifiques autour des questions qui relient durabilité et numérique ne sont pas encore stabilisés (voir [38] pour un rapide aperçu de l'état actuel des connaissances, des inconnues et des controverses sur le sujet).

Ce manque de maturité des savoirs scientifiques s'explique tout d'abord par le fait que les travaux académiques sur les impacts environnementaux du numérique sont historiquement l'œuvre de communautés de recherche distinctes qui étudient des problématiques spécifiques [24]. Citons pour exemple l'*informatique environnementale*, qui cherche à accompagner la construction de politiques environnementale adéquates en se servant de la puissance de calcul des ordinateurs ; ou encore le *Green IT*, qui vise en particulier à améliorer l'efficacité énergétique du matériel informatique.

Or, une approche générale des enjeux de durabilité liés au numérique nécessiterait un vaste travail interdisciplinaire, qui touche à des domaines allant des *Sciences de la Vie et de la Terre* (SVT), nécessaires pour apporter une compréhension fine des enjeux de durabilité, jusqu'aux *Sciences and Technology Studies* (STS), qui mettent en lumière les liens complexes existant entre un outil technique comme le numérique, les sciences informatiques qui le sous-tendent, et la société qui le développe et s'en trouve transformée [10].

En effet, les enjeux de durabilité des outils numériques ne se limitent pas aux *effets directs*[4] de ces derniers sur l'environnement, lesquels correspondent à l'*empreinte environnementale* des équipements numériques, généralement calculée à l'aide de l'*Analyse de Cycle de Vie* (ACV). Il faut également considérer les *effets indirects*, qui comprennent les changements induits par les applications de la technologie, d'une part dans les processus de production, et d'autre part dans les comportements des individus ; ainsi que les modifications structurelles de l'économie et de la société amenées par le numérique.

L'étude des effets directs du numérique est rendue difficile par le fait qu'une réalité matérielle complexe est couverte par la notion de « *numérique* ». Cette complexité résulte de l'omniprésence du numérique dans tous les secteurs économiques et sociaux, du caractère international de ses chaînes de conception, de production et de consommation, et de la rapidité de son évolution. Ainsi, si un certain consensus existe pour affirmer que les émissions de gaz à effet de serre liées au cycle de vie des équipements numériques représentent entre 1.5% et 4% du total des émissions globales d'origine humaine [9,20], l'évolution future de ces émissions reste controversée. En effet, certaines études estiment que les impacts globaux du secteur numérique suivent une tendance exponentielle, tandis que d'autres études argumentent qu'ils seraient en cours de stabilisation. Par ailleurs, la plupart des questions liées aux métaux, à l'eau, et aux pollutions restent ouvertes.

De leur côté, les effets indirects sont particulièrement difficiles à évaluer. En effet, la transition numérique bouleverse l'ensemble de notre système socio-économique sans qu'il soit possible de lui attribuer formellement et quantitativement la responsabilité des impacts environnementaux et sociaux qui en résultent. Les différents types d'effets rebond, c'est-à-dire l'augmentation des usages induite par des gains en efficacité imputables aux technologies numériques, en sont l'illustration la plus marquante.

Les enjeux de durabilité liés au numérique représentent donc un domaine de recherche interdisciplinaire émergent, où de nombreuses incertitudes et controverses demeurent sur la situation actuelle, sur les tendances futures, et sur les opportunités et les risques que le numérique représente pour l'avenir.

4. Nous utilisons ici, parmi les typologies existantes, le modèle *LES* [24] en faisant le lien entre cette classification et la terminologie couramment employée par ailleurs.

2.2 Des pratiques d'enseignement académiques en construction

La seconde difficulté provient du fait que les pratiques académiques intégrant les questions de durabilité à l'informatique sont encore en construction et que peu de personnes y ont déjà été exposées. Elles n'ont donc pas acquis un statut de référence auquel pourraient se raccrocher les acteurs et actrices de l'école, comme c'est le cas pour d'autres pratiques académiques telles que la programmation ou le développement des algorithmes. Cependant, un certain nombre d'enseignants et d'enseignantes en informatique intègrent déjà les impacts environnementaux du numérique dans leurs cours. Une vaste revue de littérature sur le sujet (que nous détaillons dans les paragraphes suivants) est compilée chez [35], tandis que [29] résume une partie des initiatives françaises. Les retours d'expérience analysés sont très variés.

Tout d'abord, les sujets traités ainsi que les organisations pédagogiques et les objectifs d'apprentissage diffèrent en fonction des cas. La revue de littérature distingue ainsi trois grandes catégories de sujets abordés : les sujets purement informatiques, les sujets uniquement liés à la durabilité (changement climatique, dimensions de la durabilité, éthique et impact social), et les sujets à l'interface entre les deux (ACV, Green IT, Green by IT). Elle identifie également des méthodes d'apprentissage variées (classiques, par le jeu, par la résolution de problèmes), et différentes manières d'aborder une discussion interdisciplinaire (groupes d'élèves de disciplines mixtes, interventions de spécialistes externes), auxquelles s'ajoutent une variété d'objectifs d'apprentissage disciplinaires (impact positif et négatif du numérique sur l'environnement) et transversaux (parmi lesquels la pensée systémique, l'esprit critique, la communication, et la créativité). Enfin, les cours étudiés diffèrent de par leur contexte (cursus avant tout centrés sur l'informatique ou sur la durabilité), mais également de par leur but éducatif (la formation des élèves ou leur émancipation).

Cette diversité dans les approches reflète en partie le caractère non stabilisé des savoirs concernant les enjeux de durabilité liés au numérique. En effet, les enseignements ne s'appuient pas sur la même définition de la durabilité, et mettent en avant des visions différentes de l'interaction entre informatique et durabilité : incrémentale (réduction des effets directs des systèmes numériques), permissive (capacité des systèmes numériques à être une solution aux enjeux de durabilité) et disruptive (remise en question des normes et pratiques de la technologie).

Notons que la revue de littérature met en avant le manque de recherche en didactique pour analyser et consolider ces initiatives locales.

2.3 Une articulation à penser entre informatique et durabilité

Une discipline académique est souvent présentée comme un ensemble cohérent de savoirs et de pratiques centrés autour d'objets et de phénomènes spécifiques traduisant une vision du monde [8], et portés par une communauté de personnes et d'institutions. L'informatique s'est construite historiquement en dehors de considérations environnementales, et les questions de durabilité sont largement perçues comme étant externes à cette discipline, comme l'illustre le

paysage de la recherche présenté dans la section 2.1. La question se pose alors de la manière de faire rentrer les enjeux de durabilité au sein de la discipline informatique, afin de proposer une articulation cohérente qui puisse être transposée en classe.

Face à cette difficulté, l'intégration prend souvent la forme d'une juxtaposition, où les problématiques de pollution numérique sont abordées sans rapport direct avec le reste du cours. Cette juxtaposition se retrouve dans divers programmes et manuels tels que Modulo en Suisse [37], ou les sections « Impact sur les pratiques humaines » du programme de SNT en France. Or, c'est en se mettant au service d'intérêts scientifiques, économiques et politiques que l'informatique exerce une grande partie de son pouvoir d'agir. Cette juxtaposition ne semble donc pas permettre une éducation à la durabilité qui saurait offrir des perspectives à la fois globales et complexes aux élèves [14].

Pour aller au-delà de la simple juxtaposition, il est tentant de proposer une articulation centrée sur les réponses que peut apporter l'informatique aux défis écologiques qui s'annoncent. Cela peut inclure par exemple la réduction des effets directs du numérique (approche *Green IT*) ou la contribution du numérique à une société décarbonée (approche *Green by IT*). Ces approches sont particulièrement attirantes de par la nature de l'informatique, une discipline orientée vers la résolution de problèmes. Elles ne peuvent toutefois pas être suffisantes, d'une part car le potentiel du numérique pour la durabilité est loin d'être évident [38] ; et d'autre part car il est probable que le récit de la technologie perçue comme uniquement pourvoyeuse de solutions s'en retrouve renforcé. En effet, la résolution de problèmes est particulièrement centrale dans les pratiques pédagogiques de l'informatique, ce qui illustre et participe certainement à cette essentialisation de l'informatique en tant que solution. Contribuer à ce récit entrerait en contradiction avec une partie des compétences attendues en durabilité par la Commission Européenne (par exemple la pensée critique et systémique, ou la capacité à envisager des avenirs alternatifs) [16].

Cette discussion reflète les débats qui animent depuis quelques années le monde de l'enseignement supérieur et de la recherche et s'illustre dans la diversité des enseignements proposés, comme indiqué dans la section 2.2. Ces débats mettent en lumière d'une part le caractère répandu de l'adhésion à la thèse de la neutralité de la technique, et d'autre part la difficulté qu'a l'informatique à se percevoir elle-même comme une source potentielle de problèmes.

Or, ces questionnements sur les valeurs portées (ou non) par l'informatique animent depuis déjà plusieurs décennies certaines communautés de recherche au niveau international. C'est le cas de la recherche à l'interface entre l'informatique et l'éthique de la technologie. Les réflexions autour de la manière d'intégrer l'éthique dans l'enseignement de l'informatique sont donc susceptibles de fournir des éléments utiles en vue d'une articulation entre informatique et durabilité.

3 S'inspirer de l'éthique de la technologie

Les réflexions portant sur l'intégration de l'éthique dans les formations en informatique existent depuis les années 70 aux États-Unis [33]. Avec le temps, les pratiques correspondantes ont évolué en une multitude de formes sous-tendant différents objectifs d'apprentissage [34], qui se traduisent par une variété de sujets traités [19]. Dans cette diversité, les directions prises par certaines approches récentes s'inspirant majoritairement des STS nous semblent intéressantes à mentionner, notamment celles se revendiquant de la justice sociale.

3.1 Une discipline qui s'inscrit dans un contexte

Les approches centrées sur la justice sociale partent du constat qu'a lieu une invisibilisation du contexte social et culturel au sein duquel la discipline informatique s'est développée [39,27,30].

Parmi les éléments contribuant à cette vision décontextualisée de l'informatique, on peut mentionner sa propension à se présenter comme neutre, uniquement technique et détachée de son identité disciplinaire, qui se traduit entre autre (comme évoqué dans la section 2.3) par une vision essentiellement positive de la technologie et la focalisation sur la résolution de problèmes de manière calculatoire. Or cette identité disciplinaire (et les valeurs qui la sous-tendent) entre en interaction, voire en contradiction, avec d'autres formes d'identité des élèves, notamment citoyennes. Cela a pour conséquence une difficulté pour l'informatique à inclure certaines minorités, et plus généralement à prendre en compte une diversité de points de vue et de valeurs.

Cette tendance est appuyée par le manque de mise en lumière de certains éléments de son histoire, tels que la place des femmes à ses débuts, et est maintenue par le manque de dialogue horizontal des informaticiens et informaticiennes avec d'une part la société civile, et d'autre part les nombreuses autres disciplines nécessaires à la compréhension des enjeux de société liés au numérique [27,39,31].

3.2 Quelques pistes de réflexion

À partir de ce constat, il n'est pas étonnant que les solutions proposées visent à réancrer la discipline informatique dans un contexte social. Certaines approches portent une attention particulière à l'intégration de minorités dans les classes d'informatique, lesquels rendent leurs cours plus inclusifs en travaillant main dans la main avec des personnes issues de différents milieux sociaux et culturels [26]. D'autres approches cherchent à visibiliser les enjeux de pouvoir en lien avec différentes technologies [39], ou à mettre en avant les problématiques sociales pouvant être reliées aux différentes notions d'informatique (par exemple autour du concept de *biais algorithmique*) [18,17,23]. Un certain nombre de plateformes en ligne proposent des contenus (en anglais) allant dans ce sens (par exemple [2,4,1]). Enfin, d'autres travaux cherchent à étendre l'étude des algorithmes à leurs applications sociales, élargissant ainsi le domaine de recherche de l'informatique [22,7].

Pour revenir aux enjeux environnementaux, la recontextualisation de l'informatique dans son contexte social paraît importante pour mieux comprendre les effets indirects du numérique sur l'environnement. Une piste complémentaire consisterait à rematérialiser l'informatique, par exemple en partant d'un objet physique tel qu'un ordinateur ou un téléphone portable et en examinant ses composants. Il est alors possible de remonter de manière approximative la chaîne de production de l'objet jusqu'à l'extraction des matières premières. Cela permet d'évoquer les ressources consommées, mais également les impacts environnementaux, sociaux et économiques de l'industrie minière et de celle des semi-conducteurs. Certaines ressources en ce sens commencent à être développées (p. ex [3]), mais leur intégration dans les cursus d'informatique n'est pas évidente, et le risque est de ne pas réussir à produire un lien cohérent avec l'informatique en tant que discipline. Une pédagogie qui s'affranchirait quelque peu de la résolution de problème et proposerait d'autres modes d'exercisation pourrait sans doute contribuer à une articulation plus cohérente entre informatique et durabilité. En ce sens, les sciences de l'éducation qui s'intéressent à l'enseignement des *Questions Socialement Vives*, avec des méthodes pédagogiques adaptées au travail sur des sujets controversés [28], pourraient s'avérer être un axe de recherche inspirant.

4 Conclusion

Ce bref survol des considérations et propositions issues de l'éthique de la technologie suggère qu'une piste intéressante pour aborder les enjeux de durabilité consiste à réancrer l'informatique dans les contextes de son développement et de son déploiement, notamment matériel. Cela passerait sans doute par la prise en considération des aspects économiques, politiques et sociaux qui orientent l'évolution et l'application de l'informatique ; mais également par le recours accru à l'interdisciplinarité et aux STS ; et pose donc la question de la formation du corps enseignant, généralement lui-même issu de formations mono-disciplinaires. Ainsi, l'urgence écologique met à mal le modèle classique descendant de la transposition didactique, à l'instar d'autres situations où la discipline scolaire a précédé la discipline académique [21]. Cela contraint les acteurs et actrices de l'enseignement secondaire, privés de savoirs et pratiques de références stabilisés, à expérimenter afin de proposer des enseignements pertinents en matière d'éducation à la durabilité en lien avec l'informatique. Parallèlement, le monde académique se saisit également à sa manière de ces enjeux, remettant en question certaines de ses pratiques de recherche et d'enseignement. Une collaboration entre les domaines de l'enseignement secondaire et universitaire aiderait certainement à converger vers des savoirs et pratiques scientifiquement fondés et socialement pertinents dans le but de faire émerger une nouvelle génération mieux à même de se saisir des enjeux de durabilité et d'affronter les défis présents et à venir.

Références

1. Do abstractions have politics ?, https ://kevinl.info/do-abstractions-have-politics/, visité le 13 décembre 2023
2. EthicalCS, https ://ethicalcs.github.io/, visité le 13 décembre 2023
3. Modulo - Cartographie du numérique, https ://enseigner.modulo-info.ch/enjx2/activ/carto_numerique.html, visité le 20 octobre 2023
4. Responsible Computing, https ://www.internetruleslab.com/responsible-computing, visité le 13 décembre 2023
5. Adopter une éducation à la sobriété numérique : Présentation du GTnum #ÆSON (2023), https ://edunumrech.hypotheses.org/8846
6. Adler, C., Wester, P., Bhatt, I., Huggel, C., Insarov, G., Morecroft, M., Muccione, V., Prakash, A. : Cross-Chapter Paper 5 : Mountains, pp. 2273–2318. Cambridge University Press, Cambridge, UK and New York, USA (2022). https://doi.org/10.1017/9781009325844.022.2273
7. de Arruda Falcão, J.P., de Lemos Meira, S.R., Ramalho, G.L. : Algorithmic pragmatism : First steps. In : 2021 IEEE International Symposium on Technology and Society (ISTAS). pp. 1–8. IEEE (2021). https://doi.org/10.1109/ISTAS52410.2021.9629190
8. Astolfi, J.P. : La saveur des savoirs : disciplines et plaisir d'apprendre. ESF (2008). https://doi.org/10.14375/NP.9782710126782
9. Bieser, J.C., Hintemann, R., Hilty, L.M., Beucker, S. : A review of assessments of the greenhouse gas footprint and abatement potential of information and communication technology. Environmental Impact Assessment Review **99**, 107033 (2023). https://doi.org/10.1016/j.eiar.2022.107033
10. Bonneuil, C. : Sciences, techniques et société (2013). https://doi.org/10.3917/dec.bonne.2013.01
11. Centre., E.C.J.R. : DigComp 2.2, The Digital Competence framework for citizens : with new examples of knowledge, skills and attitudes. Publications Office (2022). https://doi.org/10.2760/490274, https ://data.europa.eu/doi/10.2760/490274
12. Chevallard, Y. : Pourquoi la transposition didactique. Actes du Séminaire de didactique et de pédagogie des mathématiques de l'IMAG pp. 167–194 (1982)
13. Considère, S., Tutiaux-Guillon, N. : L'éducation au développement durable : entre «éducation à» et disciplines scolaires. Recherches en didactiques **15**(1), 111–133 (2013). https://doi.org/10.3917/rdid.015.0111
14. Curnier, D. : Vers une école éco-logique. Le Bord de l'eau (2021)
15. Descamps, S., Temperman, G., Lièvre, B.D. : Vers une éducation à la sobriété numérique. Humanités numériques (5) (2022). https://doi.org/10.4000/revuehn.2858, https ://journals.openedition.org/revuehn/2858
16. European Commission. Joint Research Centre. : GreenComp, Le cadre européen des compétences en matière de durabilité. Publications Office (2022). https://doi.org/10.2760/17791, https ://data.europa.eu/doi/10.2760/17791
17. Ferreira, R., Vardi, M.Y. : Deep tech ethics : An approach to teaching social justice in computer science. In : Proceedings of the 52nd ACM Technical Symposium on Computer Science Education. pp. 1041–1047 (2021). https://doi.org/10.1145/3408877.3432449

18. Fiesler, C., Friske, M., Garrett, N., Muzny, F., Smith, J.J., Zietz, J. : Integrating ethics into introductory programming classes. In : Proceedings of the 52nd ACM Technical Symposium on Computer Science Education. pp. 1027–1033 (2021). https://doi.org/10.1145/3408877.3432510

19. Fiesler, C., Garrett, N., Beard, N. : What do we teach when we teach tech ethics ? a syllabi analysis. In : Proceedings of the 51st ACM technical symposium on computer science education. pp. 289–295 (2020). https://doi.org/10.1145/3328778.3366825

20. Freitag, C., Berners-Lee, M., Widdicks, K., Knowles, B., Blair, G.S., Friday, A. : The real climate and transformative impact of ict : A critique of estimates, trends, and regulations. Patterns **2**(9), 100340 (2021). https://doi.org/10.1016/j.patter.2021.100340

21. Goodson, I. : Becoming an academic subject : Patterns of explanation and evolution. British journal of sociology of education **2**(2), 163–180 (1981). https://doi.org/10.1080/0142569810020203

22. Green, B., Viljoen, S. : Algorithmic realism : expanding the boundaries of algorithmic thought. In : Proceedings of the 2020 conference on fairness, accountability, and transparency. pp. 19–31 (2020). https://doi.org/10.1145/3351095.3372840

23. Grosz, B.J., Grant, D.G., Vredenburgh, K., Behrends, J., Hu, L., Simmons, A., Waldo, J. : Embedded ethics : integrating ethics across cs education. Communications of the ACM **62**(8), 54–61 (2019). https://doi.org/10.1145/3330794

24. Hilty, L., Aebischer, B. : ICT for Sustainability : An Emerging Research Field, vol. 310, pp. 3–36 (01 2015). https://doi.org/10.1007/978-3-319-09228-7_1

25. IPBES : Global assessment report on biodiversity and ecosystem services of the Intergovernmental Science-Policy Platform on Biodiversity and Ecosystem Services. Zenodo, IPBES secretariat, Bonn, Germany (2019). https://doi.org/doi.org/10.5281/zenodo.3831673

26. Lachney, M., Bennett, A.G., Eglash, R., Yadav, A., Moudgalya, S. : Teaching in an open village : a case study on culturally responsive computing in compulsory education. Computer Science Education **31**(4), 462–488 (2021). https://doi.org/10.1080/08993408.2021.1874228, https ://www.tandfonline.com/doi/full/10.1080/08993408.2021.1874228

27. Lachney, M., Ryoo, J., Santo, R. : Introduction to the special section on justice-centered computing education, part 1 (2021). https://doi.org/10.1145/3477981

28. Legardez, A. : Enseigner des questions socialement vives. quelques points de repères. In : L'école à l'épreuve de l'actualité. Enseigner les questions vives, vol. 110, pp. 19–32. ESF Paris (2006)

29. Ligozat, A.L., Marquet, K., Bugeau, A., Lefevre, J., Boulet, P., Bouveret, S., Marquet, P., Ridoux, O., Michel, O. : How to integrate environmental challenges in computing curricula ? In : Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1. pp. 899–905 (2022). https://doi.org/doi.org/10.1145/3478431.3499280

30. Lin, K. : Do abstractions have politics ? toward a more critical algorithm analysis. In : 2021 Conference on Research in Equitable and Sustained Participation in Engineering, Computing, and Technology (RESPECT). pp. 1–5. IEEE (2021). https://doi.org/10.1109/RESPECT51740.2021.9620635

31. Lin, K. : Cs education for the socially-just worlds we need : The case for justice-centered approaches to cs in higher education. In : Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1. pp. 265–271 (2022). https://doi.org/10.1145/3478431.3499291
32. Ministère de l'éducation nationale et de la jeunesse, Ministère de l'enseignement supérieur, de la recherche et de l'innovation, Ministère des outre-mer : Décret n° 2019-919 du 30 août 2019 relatif au développement des compétences numériques dans l'enseignement scolaire, dans l'enseignement supérieur et par la formation continue, et au cadre de référence des compétences numériques (aout 2019), https ://www.legifrance.gouv.fr/jorf/id/JORFTEXT000039005162
33. Nielsen, N.R. : Social responsibility and computer education. ACM SIGCSE Bulletin **4**(1), 90–96 (1972). https://doi.org/10.1145/873684.873708
34. Paltiel, M., Cheong, M., Coghlan, S., Lederman, R. : Teaching digital ethics in information systems. ACIS 2022 Proceedings (25) (2022), https ://aisel.aisnet.org/acis2022/25/
35. Peters, A.K., Capilla, R., Coroamă, V., Heldal, R., Lago, P., Leifler, O., Moreira, A., Fernandes, J., Penzenstadler, B., Porras, J., et al. : Sustainability in computing education : A systematic literature review. arXiv preprint arXiv :2305.10369 (2023). https://doi.org/10.48550/arXiv.2305.10369
36. Richardson, K., Steffen, W., Lucht, W., Bendtsen, J., Cornell, S.E., Donges, J.F., Drüke, M., Fetzer, I., Bala, G., von Bloh, W., et al. : Earth beyond six of nine planetary boundaries. Science Advances **9**(37), eadh2458 (2023). https://doi.org/10.1126/sciadv.adh2458
37. Petreska von Ritter, P., Da Silva, D., Edelmann, R., Farenc, N., Haussauer, V., Holzer, R., Pellet, J.P., Hersch, M. : Modulo, des moyens d'enseignement de l'informatique à visée participative. Petit x (2023), https ://hal.science/hal-04186753, in press
38. Roussilhe, G., Ligozat, A.L., Quinton, S. : A long road ahead : a review of the state of knowledge of the environmental effects of digitization. Current Opinion in Environmental Sustainability **62**, 101296 (2023). https://doi.org/10.1016/j.cosust.2023.101296
39. Vakil, S., Higgs, J. : It's about power. Communications of the ACM **62**(3), 31–33 (2019). https://doi.org/0.1145/3306617

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Modèles de mémoire pour l'enseignement de la programmation

Léo Exibard[1 [0000−0003−0318−1217]], Nadime Francis[1 [0009−0009−4531−7435]], Antoine Meyer[1 [0000−0003−4513−4347]], and Marie Van Den Bogaard[1 [0009−0007−2070−1196]]

LIGM, CNRS, Univ Gustave Eiffel, F77454 Marne-la-Vallée, France
{leo.exibard,nadime.francis,antoine.meyer,
marie.van-den-bogaard}@univ-eiffel.fr

Résumé Dans cette communication, nous proposons une réflexion sur la construction de représentations et de modèles mentaux du fonctionnement de la mémoire dans le cadre d'un enseignement d'initiation à la programmation, et de ses liens avec l'acquisition des concepts de variable et d'affectation. Après avoir donné plusieurs exemples de modèles de mémoire envisageables dans le discours enseignant, nous choisissons comme cas d'étude l'apprentissage du langage Python, prescrit par les programmes officiels du lycée en France et très utilisé en premier cycle d'enseignement supérieur. Nous présentons les principales caractéristiques du modèle de mémoire de Python, en particulier dans son implémentation de référence (CPython). Ces observations nous amènent à formuler des hypothèses quant aux conséquences possibles du choix d'un modèle de mémoire sur l'apprentissage de la programmation.

Keywords: Didactique de l'informatique · Sémantique des langages de programmation · Machine notionnelle · Diagramme mémoire

1 Introduction

L'apprentissage de la programmation, qu'il concerne la découverte d'un premier langage ou qu'il s'appuie sur des connaissances antérieures, suppose l'appropriation conjointe de règles de syntaxe et de sémantique (comme souligné par [6]). Dans sa thèse, Nguyen [9, partie B] livre une réflexion sur l'impact possible de la présence (explicite ou non) ou de l'absence dans le propos didactique d'une *machine de référence*, en prenant appui sur plusieurs traités et sur un corpus de manuels scolaires. Il insiste aussi sur l'émergence progressive dans l'histoire de l'informatique de la notion de *mémoire effaçable*, qui sous-tend les conceptions actuelles les plus courantes sur les notions de variable et d'affectation. Dans un article proposant, dès 1985, un panorama de quelques enjeux de recherche en didactique de l'informatique, Rogalski [10] souligne l'importance de la prise en compte des représentations et modèles « spontanés » des apprenantes [1] quant aux concepts fondamentaux liés à l'algorithmique et à la programmation.

1. Nous choisissons arbitrairement d'employer dans ce texte le genre grammatical féminin partout où il sera question de personnes génériques.

Volume 1, pages 33–43.

Parallèlement au propos de la communauté didactique, les langages de programmation employés dans l'enseignement ont connu une évolution certaine. Logo et Turbo Pascal, largement utilisés dans les années 1980 – 1990 (respectivement à l'école élémentaire et dans les classes préparatoires), ont aujourd'hui essentiellement disparu de l'enseignement public français. Dans les collèges, à la faveur de l'introduction de l'informatique dans les programmes de mathématiques et de technologie en 2016, le langage visuel Scratch, héritier de Logo, s'impose progressivement. Entre 2009 et 2019 le lycée a vu apparaître des contenus informatiques, d'abord dans les programmes de mathématiques puis sous la forme de matières spécifiques (comme « Informatique et Sciences du Numérique » (ISN) en terminale en 2012). Dans cette période ont été créés plusieurs langages « pédagogiques » tentant de s'adapter aux recommandations officielles (voir [7] pour une discussion de l'influence possible d'un tel langage). En 2019 le langage Python, déjà populaire dans le monde professionnel et à l'université (où il tend à remplacer le C, le Java simplifié ou encore le Scheme comme premier langage) devient le langage de référence dès la classe de seconde en mathématiques, et pour la nouvelle spécialité Numérique et Sciences Informatiques (NSI).

Face à l'évolution des langages, la question du choix d'un modèle d'exécution ainsi que de son degré d'explicitation nous semble donc importante. Dans ce travail exploratoire, nous nous intéressons spécifiquement au modèle de mémoire qui sous-tend la sémantique du langage Python. Nous portons une attention particulière sur les concepts de variable, valeur et type, sur ceux de référence mémoire, d'*aliasing* et de mutabilité, sur les différentes zones de mémoire (pile, tas) et leur fonction et sur la sémantique des différentes formes d'affectation (y compris le passage de paramètres). Sans volonté normative, nous souhaitons contribuer à décrire les enjeux du choix d'un modèle de mémoire « de travail » par l'enseignante et formulons des hypothèses quant à ses conséquences possibles sur les apprentissages. Nous nous posons en particulier la question du niveau de détail des modèles, de leur efficacité pour la résolution des tâches proposées, de leur validité vis-à-vis de ce qu'on pourrait qualifier de « modèle de référence », et de leur impact possible sur les apprentissages ultérieurs.

Nos réflexions sont liées au concept de *machine notionnelle*, défini par du Boulay et al. [4] comme « le modèle idéalisé de l'ordinateur induit par les structures d'un langage de programmation » et exploré depuis par de nombreux travaux. Ces machines posent la question d'un nécessaire compromis entre complétude (permettre d'expliquer les comportements des programmes), cohérence (permettre d'établir des prédictions conformes à la sémantique du langage) et simplicité (niveau de détail et d'abstraction les rendant accessibles aux apprenants). Dans [2], après un intéressant survol bibliographique, les auteurs insistent sur la distinction entre une machine notionnelle proprement dite et les outils de visualisation associés (même si ces deux aspects sont liés). Des travaux proches sur les diagrammes mémoire [3] mettent en lumière les enjeux du choix et de l'adaptation d'une représentation de l'état mémoire selon les objectifs d'enseignement poursuivis. Dans cet article, nous nous appuyons sur les représentations produites par l'outil Python Tutor [5], non pour les étudier pour elles-mêmes

mais comme simple illustration des machines notionnelles sous-jacentes. Nous nous appuyons aussi (pour l'instant de manière informelle) sur l'idée de *conception* proposée par Vergnaud [11] puis intégrée par Balacheff au modèle cK¢[1], repris par Modeste [8] dans son étude du concept d'algorithme, qui pourra servir à une analyse plus complète dans la poursuite de ce travail.

Le reste de cet article est organisé comme suit. La section 2 présente plusieurs variantes de modèles de mémoire « de travail » possibles, indépendamment d'un langage particulier. La section 3 discute les conséquences possibles du choix de chacun d'eux sur l'enseignement de Python. Enfin, la section 4 propose une synthèse de nos observations et quelques pistes de recherche. L'annexe A définit brièvement quelques termes liés à notre questionnement.

2 Modèles de mémoire pour l'enseignement

Nous distinguons ici trois catégories de « modèles de travail » couramment utilisés dans l'apprentissage de la programmation, chacune admettant un certain nombre de variantes ou d'hybridations. Ces modèles reflètent avec un degré de fidélité plus ou moins grand la *sémantique* du langage étudié, c'est-à-dire le comportement (documenté ou observé) des programmes écrits dans ce langage. Ils emploient des métaphores supposées faciliter l'apprentissage d'un langage, en particulier des concepts de variable et d'affectation. Nous illustrons ces modèles à l'aide de visualisations générées par l'outil Online Python Tutor [2] [5].

2.1 Modèles à boîtes

Cette première catégorie de modèles s'appuie sur l'idée que la mémoire est structurée sous forme de « boîtes » ou « tiroirs » que l'on peut désigner, ouvrir ou fermer afin d'accéder à leur contenu. Selon les descriptions, on pourra insister sur le caractère consécutif des boîtes, leur numérotation, leur capacité, etc. Un tel exemple apparaît dans l'un des manuels analysés par Nguyen [3][9, p. 36] :

> Dans notre modèle de l'ordinateur, chaque variable est associée à une boîte de stockage. Une étiquette du nom de variable est collée sur la boîte. Dans la boîte se trouve un papier sur lequel est écrit la présente valeur de la variable. (...) Chaque boîte possède un couvercle que l'on peut ouvrir pour affecter une nouvelle valeur à la variable. De plus, il y une fenêtre grâce à [laquelle] on peut lire cette valeur sans la modifier.

On notera qu'il n'est pas fait mention dans cet extrait de la notion de *type*, bien que cela soit susceptible d'apparaître dans la suite du manuel. L'adjonction de cette notion imposerait de statuer sur l'existence de plusieurs sortes de boîtes de capacités variées (au sens de la quantité d'information qu'elles peuvent contenir), et sur le genre d'objets qu'il est permis d'y stocker. Il est possible d'interpréter cette description de deux manières différentes.

2. Disponible sur https://pythontutor.com, captures effectuées le 13/10/2023.

3. Le manuel M2, qui est le seul du corpus étudié à fournir une description de machine destinée à exécuter les programmes.

Boîtes à étiquettes fixes. Premièrement, on peut supposer que chaque variable possède sa boîte propre, créée lorsque la variable est déclarée ou initialisée pour la première fois, et qu'une étiquette reste associée à la même boîte tout au long de sa durée de vie. Dans ce cas, la sémantique probable de l'affectation (disons `x = expr` avec `expr` une expression quelconque) consiste premièrement à construire (ou à localiser s'il existe déjà) l'objet `obj` résultant de l'évaluation de `expr`, puis à recopier `obj` dans la boîte associée à `x`. Le mode de passage de paramètre y correspondant naturellement serait un passage par valeur, chaque appel de fonction créant sur la pile des boîtes correspondant à ses paramètres et variables locales. La figure 1 montre une visualisation possible pour un tel modèle.

Boîtes à étiquettes déplaçables. La seconde interprétation suppose que l'étiquette représentant une variable puisse être repositionnée sur une boîte différente. Lors d'une affectation, l'objet `obj` est créé en mémoire (ou localisé), et l'étiquette `x` est apposée à la boîte le contenant. La boîte précédemment désignée par `x` devient alors inaccessible, à moins qu'elle ne soit encore étiquetée par une autre variable. Dans ce type de modèle, il n'est pas évident de déterminer un mode de passage de paramètre cohérent : sans recopie des objets, comment par exemple distinguer les paramètres de fonctions des variables déjà existantes ?

2.2 Modèles à fils et épingles

Dans cette catégorie de modèles, on postule que l'accès aux objets est assuré par l'intermédiaire de références (Cf. annexe A), représentées par des flèches ou « fils » reliant un nom (par exemple une variable ou un attribut d'objet) à l'objet qu'il désigne. Les variables sont recensées dans une zone ne contenant aucun objet, typiquement, un espace de noms stocké dans la pile d'appels, et les objets sont tous alloués dans le tas. Selon les cas, un type peut être associé à chaque variable ou attribut (typage statique), ou seulement à l'objet désigné (typage dynamique). Une variable désignant un objet est reliée par l'intermédiaire d'un fil (« épinglée ») à l'emplacement mémoire contenant les données de l'objet. Les objets composites ou objets de type conteneur (par exemple les tableaux) sont également reliés aux objets qu'ils contiennent par le même procédé.

Dans ce modèle comme dans le modèle à étiquettes déplaçables, l'affectation `x = expr` crée ou localise en premier lieu dans le tas l'objet `obj` issu de l'évaluation de `expr`. Ensuite, l'étiquette `x` (après avoir été créée le cas échéant) est reliée par un fil à l'emplacement mémoire le contenant. Le mode de passage de paramètres correspondant est le passage par référence, et l'affectation d'un attribut à un objet ou d'un élément à un tableau suit le même principe. Il est en outre aisé de regrouper clairement l'ensemble des étiquettes (variables) appartenant à un même contexte d'exécution ou à une même portée grâce à la pile : variables globales, variables locales à un appel de fonction, etc. La figure 2 montre l'exécution du programme précédent dans un modèle de ce type.

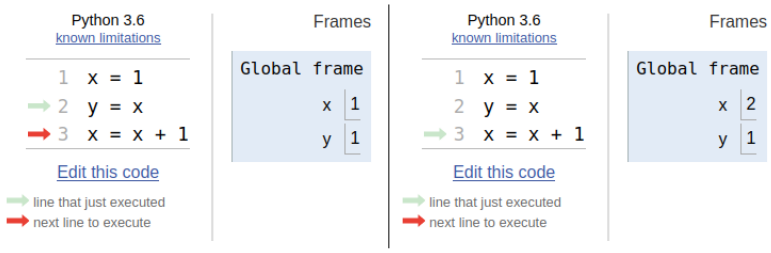


Figure 1. Représentation de type « boîtes à étiquettes fixes » dans Python Tutor. Ici, `x` et `y` désignent deux boîtes indépendantes, même après l'instruction `y = x`. La zone *Frames* représente l'état de la pile, le cadre bleu l'espace de noms global.

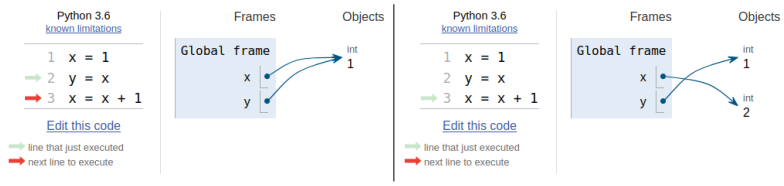


Figure 2. Représentation de type « fils » dans Python Tutor. La zone *Objects* représente le tas. Chaque objet y est représenté par sa valeur surmontée de son type. Ici, `x` et `y` désignent la même boîte après l'instruction `y = x` (aliasing), et `x = x + 1` provoque la création d'un nouvel objet en mémoire, laissant `y` inchangée.

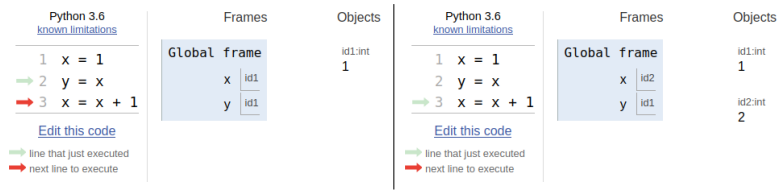


Figure 3. Représentation de type « adresses » dans Python Tutor. La sémantique est semblable au déroulement représenté en Figure 2.

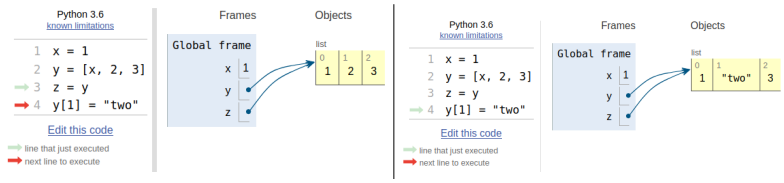


Figure 4. Représentation mixte dans Python Tutor. Les entiers et chaînes de caractères sont stockées par valeur, les listes par référence. Cet exemple illustre également l'effet de la modification d'un objet mutable en présence d'aliasing (une modification de la liste désignée par `y` est visible depuis la variable `z`).

2.3 Modèles à adresses

Dans cette catégorie de modèles, on distingue deux types de valeurs : les valeurs d'objets (ou données) et les adresses (ou emplacements mémoire). Le principe d'indirection présent dans les modèles à fils est maintenu, mais il est rendu plus explicite : lors d'une affectation `x = expr`, l'objet `obj` issu de l'évaluation de `expr` est créé ou localisé en mémoire, et son *adresse* est déterminée. Celle-ci est alors stockée dans une zone ad-hoc (par exemple sur la pile), qu'on peut voir comme un registre associant à chaque variable une adresse, ou une collection de boîtes contenant l'adresse de l'objet désigné par chaque variable. La figure 3 montre l'exécution du programme précédent selon un tel modèle.

2.4 Modèles mixtes

Il est possible d'envisager des modèles mêlant certaines des caractéristiques des modèles à boîtes, à fils ou à adresses. Au risque d'une plus grande complexité, ceci peut permettre de rendre plus fidèlement compte de la sémantique de référence du langage étudié. L'une de ces variantes consiste à représenter les objets de types « primitifs » (comme les nombres entiers ou les caractères) comme stockés dans des boîtes à étiquettes fixes, et de désigner les objets composites (y compris conteneurs) par l'intermédiaire de fils ou d'adresses (c'est-à-dire par référence). Une visualisation de ce type est représentée dans la figure 4.

Ce type de modèle mixte est par exemple assez proche de la sémantique réelle de langages comme Java, voire comme le C où les deux mécanismes existent. Dans ce dernier cas, une adaptation supplémentaire est requise pour tenir compte du fait que les adresses mémoires peuvent être manipulées directement par la programmeuse au même titre que tout autre objet.

3 Discussion : le cas de Python

Python est un langage interprété, dont l'implémentation de référence (écrite en C) est CPython. Dans ce langage, toutes les données d'un programme (y compris fonctions, classes, modules, etc.) sont des objets [4]alloués sur le tas. Chaque objet possède un type et un identifiant assimilable à une adresse mémoire (tous deux invariants), et une valeur potentiellement modifiable (objets mutables). Les adresses peuvent être consultées mais ne sont jamais manipulées directement par la programmeuse. L'allocation et la libération de mémoire sont prises en charge par l'interpréteur. Ce modèle de mémoire est *homogène* au sens où il ne repose que sur un seul mécanisme, celui de référence, contrairement à des langages comme C ou Java. Variables et attributs d'objets sont des références vers d'autres objets, et le passage de paramètres à une fonction s'effectue par référence. Ainsi, Python repose fortement sur le phénomène d'aliasing, dont la documentation indique qu'« [i]l n'apparaît pas au premier coup d'œil en Python

4. Documentation Python – 3. Modèle de données. https://docs.python.org/fr/3/reference/datamodel.html, consulté le 16/10/2023.

et [qu']il peut être ignoré tant qu'on travaille avec des types de base immu[t]ables (nombres, chaînes, n-uplets) », mais prévient que « les alias peuvent produire des effets surprenants » dans le cas contraire [5].

En raison de ces caractéristiques, il semble que les modèles à boîte (à étiquettes fixes ou déplaçables) présentent des domaines de validité trop restreints pour représenter fidèlement l'ensemble des concepts enseignés dans la plupart des cours d'initiation à Python. En effet, les progressions habituelles sont susceptibles d'aborder relativement tôt les questions relatives à la structuration des espaces de noms (variables globales *vs.* variables locales, phénomène de masquage), au passage de paramètres mutables (typiquement des listes ou dictionnaires) et à leur éventuelle modification par le corps d'une fonction, ou encore aux phénomènes d'aliasing fréquemment rencontrés dans la construction de listes imbriquées. De même, il peut être malaisé dans ces modèles d'expliquer par des métaphores satisfaisantes le fonctionnement des objets de taille arbitraire (comme les entiers de Python ou les conteneurs imbriqués), sauf à admettre qu'il est possible de « faire tenir » une quantité illimitée d'information dans une zone mémoire finie, ce qui peut entrer en contradiction avec l'acquisition d'autres concepts de l'informatique comme le codage binaire des données.

A contrario, la plupart des caractéristiques de Python évoquées ci-dessus peuvent être prises en charge par les modèles à base de fils ou d'adresses. On remarque que ces modèles mettent nettement en évidence la notion d'aliasing dans l'explication et la visualisation de la sémantique de programmes même très simples, et induit une séparation claire entre pile et tas, noms (sources de flèches) et objets (désignés par des flèches).

Cependant, les modèles à fils peuvent sembler moins concrets (les objets semblant « flotter » dans une mémoire amorphe), et ne permettent pas aisément de porter un discours sur l'aspect contigu des emplacements mémoire. Les modèles à adresses introduisent quant à eux une notion d'indirection relativement complexe, et présentent un potentiel risque de confusion entre valeurs d'adresses et valeurs d'objets. En outre, ces deux types de modèles sont susceptibles d'imposer à l'enseignante ainsi qu'aux apprenantes déjà exposées à un autre modèle de mémoire une remise en question plus profonde de conceptions antérieures. La question générale de déterminer si de tels modèles peuvent être globalement perçus par les apprenantes comme plus complexes qu'un modèle à boîtes reste ouverte, et peut justifier la mise en place d'un travail expérimental.

4 Conclusion et perspectives

Nous avons proposé une discussion sur les domaines de validité de quelques modèles de mémoire « de travail » dans le cas du langage Python. Nous considérons que ces observations restent valables pour l'enseignement d'autres langages de programmation, au sens où elles mettent l'accent sur l'importance du choix d'un modèle de mémoire adapté à la sémantique du langage choisi et aux

5. Tutoriel Python – 9. Classes. https://docs.python.org/fr/3/tutorial/classes.html, consulté le 16/10/2023.

concepts à enseigner, quels qu'ils soient. Il nous semblerait utile d'approfondir ce début d'analyse *a priori* à l'aide d'un cadre théorique rigoureux, tel que celui proposé par Balacheff [1], notamment en termes de validité, cohérence et efficacité des conceptions sous-tendues par chaque modèle. On portera une attention particulière à ce que les auteurs nomment « structures de contrôle », qui permettent d'une part d'attester de la non-contradiction de la conception par rapport à la réalité, et de l'état (résolu ou non) d'un problème. Dans le contexte de la programmation, l'interpréteur offre des moyens de contrôle potentiellement immédiats, qu'il conviendrait d'analyser avec soin.

Nous faisons l'hypothèse que le choix d'un modèle de mémoire de la part de l'enseignante est susceptible d'entraîner des effets didactiques importants sur la formation de conceptions liées à la sémantique du langage ainsi qu'aux objets fondamentaux de l'informatique. Une première piste d'expérimentation permettant de mettre à l'épreuve cette hypothèse serait de relever dans les discours d'enseignantes, dans les ressources qu'elles utilisent et dans les productions de leurs élèves ou étudiantes des indices de conceptualisations liées au modèle de mémoire. L'analyse de ce corpus pourrait permettre d'approfondir notre connaissance des modèles de travail utilisés en classe, et de tenter de mettre en évidence leur effet possible sur les conceptualisations des apprenantes et sur leur acquisition des savoirs et savoir-faire visés.

D'autres apprentissages, parallèles ou ultérieurs, par exemple sur la représentation et le codage des données, sur l'acquisition de langages de programmation de caractéristiques différentes, sur l'analyse d'algorithmes et de programmes (notamment leur complexité) ou encore sur la conception de compilateurs ou d'interpréteurs, nous semblent requérir une perception raisonnablement précise de la structure de la mémoire et de ses usages les plus courants. Nous faisons l'hypothèse que la manière dont ces notions sont initialement rencontrées et « fréquentées » peut favoriser ou au contraire retarder certains des ces apprentissages. On peut s'intéresser également à la question de la pérennité de chaque modèle au fil des apprentissages, et à celle de leur dépassement progressif.

Références

1. Balacheff, N., Margolinas, C. : cK¢. Modèle de connaissances pour le calcul de situations didactiques. In : Balises en didactique des mathématiques : Cours de la 12e école d'été de didactique des mathémathiques, pp. 1–32. Recherches en Didactique des Mathématiques, La Pensée Sauvage (2005)
2. Dickson, P.E., Brown, N.C.C., Becker, B.A. : Engage Against the Machine : Rise of the Notional Machines as Effective Pedagogical Devices. In : Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education. pp. 159–165. ACM, Trondheim Norway (Jun 2020)
3. Dickson, P.E., Dragon, T. : A Memory Diagram for All Seasons. In : Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1. pp. 150–156. ACM, Virtual Event Germany (Jun 2021)

4. du Boulay, B., O'Shea, T., Monk, J. : The black box inside the glass box : presenting computing concepts to novices. International Journal of Man-Machine Studies **14**(3), 237–249 (Apr 1981)
5. Guo, P.J. : Online Python Tutor : Embeddable Web-Based Program Visualization for CS Education. In : Proceeding of the 44th ACM technical symposium on Computer science education. pp. 579–584. SIGCSE '13, Association for Computing Machinery, New York, NY, USA (Mar 2013)
6. Hoc, J.M., Nguyen-Xuan, A. : Chapter 2.3 - Language Semantics, Mental Models and Analogy. In : Hoc, J.M., Green, T.R.G., Samurçay, R., Gilmore, D.J. (eds.) Psychology of Programming, pp. 139–156. Academic Press, London (Jan 1990)
7. Meyer, A., Modeste, S. : Rôle d'un logiciel dans la transposition didactique du concept d'algorithme : le cas du logiciel AlgoBox en France et des programmes du lycée entre 2009 et 2019. Cahiers d'histoire du Cnam **vol.15**(1), pp. 101 (2022)
8. Modeste, S. : Prendre en compte l'épistémologie de l'algorithme. Quels apports d'un modèle de conceptions ? Quelle transposition didactique ? Recherches en didactique des mathématiques **38**(1), 1–15 (2021)
9. Nguyen, C.T. : Etude didactique de l'introduction d'éléments d'algorithmique et de programmation dans l'enseignement mathématique secondaire à l'aide de la calculatrice. Ph.D. thesis, Université Joseph-Fourier - Grenoble I (Dec 2005)
10. Rogalski, J. : Alphabétisation informatique. Bulletin de l'APMEP **347**, 61–74 (1985)
11. Vergnaud, G. : La théorie des champs conceptuels. Recherches en Didactique des Mathématiques **10**(2.3), 133–170 (1990)

A Définitions

On définit ici de manière informelle quelques concepts de référence utiles. Ces définitions sont indicatives, elles admettent des variantes et des exceptions selon les contextes. Nous nous gardons volontairement de les détailler outre mesure.

Objet, valeur et type. On appellera *objet* (dans un sens général, et non au sens de la programmation orientée objet) tout littéral ou plus généralement toute entité résultant de l'évaluation d'une expression. Chaque objet possède a priori une *valeur* (les données dont il est constitué) et un *type* qui peut déterminer, entre autres, les opérations qu'on peut lui appliquer, et potentiellement la manière dont ses données seront stockées en mémoire (ce dont la programmeuse n'a pas nécessairement connaissance).

Variable. Ce terme désigne généralement une entité permettant de désigner un objet par un nom à un moment donné de l'exécution d'un programme. Le détail du fonctionnement et des caractéristiques des variables dépend fortement du langage de programmation considéré.

Espace de noms. Dans la terminologie propre à certains langages, on appelle ainsi les « répertoires » de noms accessibles à un instant donné de l'exécution. Généralement, plusieurs espaces de noms co-existent (global, locaux, propres à un objet, etc.). Leur gestion relève des mécanismes de *portée* et de *résolution de noms*, qui diffèrent selon les langages.

Politique de typage. On distingue généralement (outre les langages non ou faiblement typés) les langages *statiquement typés* (comme Java ou C), dans lesquels les vérifications de type sont effectuées à la compilation, des langages *dynamiquement typés*, où cette vérification est faite à l'exécution.

Adresse, référence, pointeur. Une *adresse* est une donnée (généralement un nombre entier) permettant d'identifier de manière unique un emplacement mémoire[6]. Dans plusieurs langages, il est possible de désigner des objets de manière indirecte par le biais de leur adresse. On parle de *références* lorsque cette « indirection » est assurée sans contrôle de la programmeuse (comme en Java ou en Python), et de *pointeurs* lorsqu'un langage permet une manipulation directe de l'adresse (comme en C).

Pile d'appel. La *pile d'appel* désigne une zone de mémoire consacrée à la gestion et à la bonne exécution des appels de fonctions (potentiellement imbriqués). Elle a pour rôle le maintien du flot de contrôle correct d'un programme lors d'appels de fonctions, l'identification des paramètres reçus ainsi que la transmission du résultat d'un appel au contexte englobant.

Tas. Le *tas* est la zone de mémoire généralement consacrée au stockage des valeurs. Selon les cas, ceci peut concerner les valeurs désignées par des variables globales, les valeurs stockées dans des zones de mémoire allouées dynamiquement, voire toutes les valeurs utilisées par un programme.

6. On ne rentrera pas ici dans la distinction entre adresses réelles et virtuelles.

Politique de passage de paramètres. Lorsqu'une fonction reçoit des paramètres, on parle de *passage par valeur* quand la valeur de chaque paramètre est recopiée dans une zone de mémoire locale (par exemple la pile), et de *passage par référence* lorsque seule son adresse est transmise. Dans ce cas, différents mécanismes (par exemple en C ou C++) peuvent permettre d'autoriser ou d'interdire la modification des valeurs reçues.

Aliasing. C'est le fait pour un objet d'être désigné par plusieurs variables ou plusieurs noms (attributs d'objets, éléments de tableaux...). Ceci peut être source d'efficacité (passage par référence d'un objet volumineux à une fonction, partage de données dans la programmation orientée objet) ou source d'erreur (modification inopinée d'un objet partagé).

Mutabilité. Cet anglicisme désigne le fait, pour un objet, d'être modifiable sur place en mémoire. En présence d'aliasing, ceci peut provoquer des comportements inattendus. En programmation fonctionnelle dite *pure*, on interdit parfois totalement la manipulation de valeurs mutables.

D'autres concepts sont liés à notre questionnement et mériteraient un traitement plus approfondi. Nous ne pouvons tous les détailler faute de place.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Retours d'expériences et analyse de besoins sur l'introduction de l'enseignement obligatoire de l'informatique au niveau fédéral en Suisse

Murièle Jacquier[1] et Alain Sandoz[2]

[1] IUFE - Université de Genève - 1211 Genève – Suisse
muriele.jacquier@unige.ch
[2] IIUN - Université de Neuchâtel - 2000 Neuchâtel – Suisse
alain.sandoz@unine.ch

Résumé. Jusqu'en 2018, l'informatique était une branche optionnelle au secondaire 2 en Suisse. Chaque canton organisait son enseignement selon ses propres dispositions et agenda politique, tout en restant conforme aux dispositions nationales et à celles définies par les conférences régionales (alémanique ou latine) dont il était membre. L'introduction de l'informatique comme discipline obligatoire (DO) au niveau fédéral a conduit à harmoniser et réhausser les exigences, notamment en termes de plan d'études et de qualification formelle des enseignants en informatique. Ceci pose plusieurs problèmes : 1) la dotation en enseignants formellement qualifiés pour cet enseignement ; 2) leur mise à niveau académique ; 3) l'adaptation de leur enseignement aux nouvelles exigences ; et 4) la provision d'outils pédagogiques adaptés. L'article étudie les difficultés rencontrées par les enseignants et des solutions développées par eux en Suisse romande à l'aide de deux études de terrain. La première adresse le problème 3) et s'est déroulée dans le canton de Genève où les enseignants confrontés à la nouvelle situation ont été questionnés l'année suivant l'introduction de la DO informatique. La seconde, dans la partie Nord-Est de la Romandie, où un groupe d'enseignants suivant une mise à niveau (2) a produit collectivement un ensemble de scénarios pédagogiques documentés, couvrant une partie du nouveau plan d'études et pouvant être partagés et adaptés selon les besoins des membres du groupe et de leurs collègues.

Mots-clés : enseignement de l'informatique, discipline obligatoire, plan d'études, scénarios pédagogiques.

1. Introduction

Jusqu'en 2018, l'informatique n'était pas une *discipline obligatoire* dans l'enseignement au niveau secondaire 2[1] en Suisse. Des enjeux politiques et économiques, ainsi que des craintes soulevées par son introduction, tissaient un arrière-plan sur lequel des initiatives locales, c'est-à-dire cantonales dans le contexte suisse, tentaient de maintenir l'école au même rythme de numérisation que le reste de la société. L'informatique était une *branche à option*, organisée selon des modalités déterminées par les cantons. Elle pouvait être enseignée par des professeurs ayant complété leur formation pédagogique dans une Haute école sans être titulaires d'un Master en science informatique. Ce dispositif laissait une grande liberté aux enseignants pour le choix de leurs sujets, de leurs méthodes et de leurs supports. La plupart des élèves inscrits à cette branche optionnelle avaient par défaut un intérêt pour l'informatique, ce qui pouvait faciliter le travail des enseignants.

L'informatique comme *discipline obligatoire* (la DO informatique) au secondaire 2 est apparue plus tard en Suisse que chez certains de ses voisins, par exemple en 2012 en France avec l'ISN et en 2002, puis 2019, en Allemagne avec *DigitalPakt Schule*. La qualification récente de l'informatique comme DO a nécessité, d'une part de former en un temps relativement court des enseignants pour cette nouvelle discipline, et d'autre part de fournir à ces derniers les moyens, ressources et outils pour la mettre en œuvre. Quatre problèmes sont apparus à ce moment :

(1) par suite de l'augmentation du nombre d'élèves devant recevoir une formation en informatique, il faut compléter la dotation en enseignants aptes à la fournir ;
(2) pour ceux qui l'enseignaient déjà sans être titulaires d'un Master dans la discipline, il faut déterminer des modalités transitoires et définir des programmes de formation leur permettant d'acquérir la formation de base et le titre académique correspondant à la nouvelle qualification de discipline obligatoire de l'informatique ;
(3) pour tous les enseignants, il faut adapter les programmes dispensés jusqu'alors aux nouveaux objectifs définis au niveau fédéral et déclinés dans les cantons ;

[1] pour des élèves en fin d'études secondaires ayant entre 15 et 19 ans, dans des *collèges*, des *gymnases*, ou des *lycées*, selon la désignation cantonale. Nous utilisons *collège* qui est utilisé à Genève. Quelques jalons à propos de l'introduction de la DO informatique et des méandres du système fédéral suisse sont fournis dans l'annexe.

(4) finalement, il faut fournir aux collèges et aux enseignants les moyens de dispenser ce nouvel enseignement.

Ces problèmes sont toujours en cours de résolution dans la plupart des cantons. Concernant le deuxième, les Instituts de formation des enseignants et les Hautes écoles pédagogiques ont ouvert des filières en didactique de l'informatique ou renforcé celles qui existaient. Pour les professeurs déjà nommés dans une autre discipline d'enseignement, des compléments tels que le CAS informatique de BEJUNE [1] ou des programmes tels que GymInf [2] soutenu pour son volet francophone par les universités de Berne, Fribourg, Genève, et Neuchâtel, et par l'EPFL, ont vu le jour dès la rentrée académique 2020. En parallèle à ces dispositifs élaborés dans un contexte de formation continue pour répondre au problème (2), les autres problèmes demeurent, et notamment, les problèmes (3) et (4).

Le présent article apporte un éclairage sur le besoin, pour les enseignants de la nouvelle discipline obligatoire informatique, de disposer d'outils leur permettant de concevoir, puis de mettre en œuvre dans leurs classes, des scénarios pédagogiques qui correspondent à la fois aux nouvelles directives *et* à leurs (nouvelles) classes. Ce besoin est double : d'abord que des scénarios adéquats *existent*, puis qu'ils puissent *être diffusés*, *réutilisés*, et *adaptés* aux situations individuelles par les enseignants concernés, sans que ces derniers n'aient à reprendre chaque sujet à zéro, qui plus est dans un contexte déterminé aléatoirement selon la classe, le collège, le canton, etc.

Il se base sur deux études de cas. La première s'est déroulée dans le canton de Genève, au collège, où la première auteure a conduit une enquête auprès de ses collègues et auprès des établissements, pour comprendre comment la nouvelle DO a été appréhendée au cours de la première année de son existence. La seconde est l'étude *ex post* du matériel pédagogique produit par une classe du CAS-HEP au cours de ses quatre semestres. Chaque semestre était sanctionné par le rendu d'un projet individuel consistant en un scénario pédagogique produit par l'étudiant/e et testé dans sa (ses) classe(s). Une quarantaine de scénarios ont été créés et mis en œuvre, présentés avec un retour d'expérience et discutés, et mis à la disposition de leurs collègues par ces étudiants. L'étude de ces contenus est instructive pour mesurer combien leur diffusion et la discussion dans le collectif peut être utile. De surcroit, le retour d'expérience demandé aux étudiants à la fin de leur formation donne le pendant aux incertitudes qui ressortent de l'étude Genevoise.

Les auteurs ont tenu les rôles suivants dans le contexte décrit ci-dessus. La première enseigne la physique, les mathématiques et l'informatique au collège Claparède à Genève. Elle est chargée d'enseignement à l'IUFE dans le groupe de didactique de la physique. Elle suit sur le terrain les futurs enseignants de physique et elle est en charge d'un atelier de didactique et du séminaire MITIC. Elle a suivi la formation GymInf et sa contribution à cet article constitue une partie de son travail de Master (problème 2 ci-dessus). Le second auteur a été chargé d'enseignement dans le programme GymInf en même temps que le coordinateur pour l'Université de Neuchâtel du CAS-HEP. Il en a défini avec son *alter ego* de la HEP-BEJUNE le programme d'études et le mode d'apprentissage en classe inversée basé sur la production de scénarios pédagogiques [3].

La section suivante présente quelques caractéristiques de la discipline informatique en tant qu'objet d'étude de la présente contribution. La section 3 présente les deux études et leurs résultats. Elle est suivie d'une discussion (section 4) et d'une conclusion (section 5). Des éléments choisis jalonnant l'émergence de l'informatique dans l'enseignement secondaire en Suisse sont fournis dans l'annexe avec quelques références en ligne.

2. Caractéristiques de la discipline obligatoire informatique

La place accordée à l'enseignement de l'informatique dans les *plans d'études* est hétérogène et changeante, ce qui résulte de ses multiples rôles et dimensions [4]. On retiendra ici, les trois approches complémentaires de Bruillard [5] : les algorithmes et le traitement automatisé dans le cycle classique données/traitement/résultats (la pensée computationnelle) ; l'interaction continue avec des machines et des artefacts sémiotiques (les outils informatiques) ; la participation à des interactions sociales avec des agents humains et non humains via les réseaux (l'éducation au numérique). A ceci s'ajoutent d'une part l'informatique, en tant que « discipline carrefour » qui offre un cadre idéal pour l'interdisciplinarité, et de l'autre la confusion entre l'informatique comme objet d'études et comme outil pour l'enseignement d'autres disciplines.

On retrouve les 3 approches et le cadre interdisciplinaire dans les thématiques des plans d'études de la DO :

- la pensée computationnelle se retrouve dans la thématique *Algorithmique et programmation*, présente dans tous les plans d'études cantonaux, et même dans les deux années d'enseignement de plusieurs cantons ;
- les outils informatiques sont traités dans les thématiques *Informations et données* (ou *Représentation de l'information*), *Architecture*, et *Réseaux ;*
- l'éducation au numérique se fait dans la thématique *Informatique et société* présente explicitement dans quatre plans d'études sur sept cantons de langue française (dont trois, Berne, Fribourg et Valais sont bilingues) ;

- l'interdisciplinarité est mentionnée dans tous les plans d'études et apparaît comme thématique dans deux d'entre eux.

Deux caractéristiques de la DO informatique sont communes à ces plans d'études : la diversité des domaines d'application de l'informatique et l'existence de contenus relevant des sciences humaines et sociales. Cela met en question la formation des enseignants : les enseignants ont-ils la formation académique et didactique adéquate pour enseigner ces contenus ? Une appropriation récente et volumineuse de contenus qui doivent être enseignés et transmis permet-elle d'opérer des choix pertinents ? D'autre part, quelle place donner aux contenus issus des sciences sociales ? L'informatique est intégrée aux branches scientifiques dans le cursus au collège. Mais développer une approche systémique et transmettre aux élèves les enjeux de l'informatique semblent indispensables pour leur permettre de faire des choix éclairés sur le numérique dans la société.

Au-delà de ces questionnements, on peut se pencher sur l'origine de ces caractéristiques. Le nombre important de domaines étudiés peut s'expliquer par le fait que l'informatique est une discipline relativement jeune et introduite tardivement dans le cursus scolaire. L'existence de contenus relevant de sciences humaines et sociales révèle que l'informatique est une discipline carrefour, mais aussi le fait que ces contenus ne sont pas présents dans les disciplines liées. Peut-être les enseignants ne se sentent-ils pas responsables ou légitimes pour les donner.

Une analyse comparative des plans d'études (PE) de la DO des cantons de langue française révèle :

- la mention de connaissances de base ou de connaissances fondamentales à acquérir, sans autre précision, ce qui laisse la place à l'interprétation (seul le PE du canton de Neuchâtel ne les mentionne pas) ;
- des compétences psychosociales à développer, notamment celle de *savoir résoudre des problèmes* ;
- la mise en œuvre de formes sociales de travail avec la *pratique*, le *projet* et le *travail de groupe*.

La principale différence en termes de contenus se situe au niveau des technologies de l'information et de la communication. Dans certains PE cantonaux, les compétences ou bases en TIC sont explicitement mentionnées. Dans d'autres, seul l'usage responsable des TIC apparaît dans les plans d'études.

L'enseignement de la DO informatique s'effectue sur deux ans, sauf pour le canton de Bern (sur 3 ans) (voir le tableau 1). Le volume horaire global est une caractéristique commune : pour Berne, Fribourg, Valais et Vaud il est de 4 heures, à Genève, Neuchâtel et dans le Jura de 3h. La plupart des cantons font le choix de répartir cet enseignement sur la 1ère et la 2ème année, sauf le Valais (2ème et 4ème années) et Berne (de la 1ère à la 3ème années).

Tableau 1. Place de la DO dans les plans d'étude des cantons romands (Genève, Berne, Jura, Neuchâtel, Fribourg, Valais, Vaud)

Canton	Ge	Be	Ju	Ne	Fr	Vs	Vd
Nombre d'années	2	3	2	2	2	2	2
1ère année	2h	1h	2h	2h	2h	0h	2h
2ème année	1h	2h	1h	1h	2h	2h	2h
3ème année	0h	1h	0h	0h	0h	0h	0h
4ème année	0h	0h	-	-	0h	2h	0h
Total	3h	4h	3h	3h	4h	4h	4h

Concernant la mise en œuvre de la DO informatique, l'enseignant n'a pas pleinement la main sur deux éléments : d'une part l'évaluation ; et d'autre part les outils, l'environnement technique et les logiciels. Pour l'évaluation, il y a plusieurs biais : l'équipement informatique des établissements (il n'y a pas toujours un ordinateur par élève) et les règlements des examens (propres à chaque collège dans un cadre défini au niveau cantonal). L'évaluation se fait souvent par écrit, donc sans possibilité d'adopter une démarche essais-erreur. Ceci va à l'encontre de ce qui est préconisé, par exemple, dans le PE genevois. L'évaluation par projet et/ou par groupe (recommandée par exemple dans ce PE), ainsi que l'évaluation de compétences, de savoir-faire ou de savoir-être, est plus complexe, notamment en termes de grille d'évaluation, et cela nécessite une adaptation de la part des enseignants.

Pour ce qui est des outils, de l'environnement et des logiciels, dont le choix est parfois imposé à l'enseignant, ils sont cruciaux en termes de charge cognitive. Des stratégies doivent être développées pour que cette dernière soit réduite, de sorte que les outils et l'environnement informatique ne deviennent pas un obstacle aux apprentissages.

En conclusion, l'enseignant d'informatique a plusieurs défis didactiques à relever. Il doit créer des situations et mettre en œuvre des approches interdisciplinaires comme modalités de travail dans son enseignement. Il doit veiller à ne pas seulement enseigner l'informatique comme un outil. Il doit maîtriser de nombreux contenus disciplinaires de la science informatique et ce dans des domaines d'application variés. Enfin, il doit adapter son enseignement pour prendre en compte non seulement le niveau de connaissances et la motivation des élèves, mais également les contraintes technologiques et institutionnelles, en particulier en ce qui concerne les modalités d'évaluation.

3. Présentation des deux études et de leurs résultats[2]

Ce travail repose sur une enquête de terrain réalisée auprès des enseignants d'informatique du canton de Genève (§3.1) et sur une étude des résultats produits par des enseignants en informatique spécialisés dans d'autres disciplines et en formation dans le CAS-HEP BEJUNE (Berne francophone, Jura, Neuchâtel) en vue de l'introduction de la nouvelle DO (§3.2). Les caractéristiques des deux études sont résumées dans le Tableau 2.

Tableau 2. Caractéristiques des deux études

Étude	**1**	**2**
Public	Enseignant/es de la DO informatique	
	Diplômé/es (MSc) ou en cours de formation dans le programme GymInf	En cours de formation dans le CAS HEP-BEJUNE
Cantons	Genève	Berne (francophone), Jura, Neuchâtel
Nombre	18 (répondant/es) / 64 (enseignant/es)	10 (ayant complété) / 12 (au départ)
Forme	Questionnaire	Classe inversée Mise en pratique et évaluation
Durée	Quelques semaines	4 semestres
Période	2023	2020-2022

3.1 Enquête de terrain dans le canton de Genève (Étude 1)

L'enquête de terrain dans le canton de Genève est constituée d'un questionnaire destiné aux enseignants (9 pages) et d'un questionnaire par établissement (5 pages). Le premier a été envoyé à tous les enseignants d'informatique au secondaire 2 du canton de Genève, via les groupes informatiques dans leurs établissements, puis, nominalement dans un second temps (avril et juin 2023). Le temps estimé pour le remplir est de 45 minutes. Il comprend une partie « profil personnel » qui permet d'avoir une vue d'ensemble sur les conditions d'exercice du métier d'enseignant, l'expérience et la formation académique et pédagogique de l'enseignant et une partie portant sur le PE genevois qui adresse : 1) le plan d'études ; 2) le ressenti de l'enseignant sur ce qui a été facile ou difficile à enseigner ; 3) le ressenti de l'enseignant sur ce qui a été facile ou difficile pour les élèves à apprendre ; 4) la progression annuelle ; et 5) les difficultés rencontrées et les ressources utilisées (logicielles et matériel).

Le questionnaire d'établissement a été envoyé en février 2023. Il comprend trois parties (1ère et 2e années, et savoir-faire transversaux) et demande pour chaque thématique ou savoir-faire les langages de programmation, les matériels, les logiciels et les environnements d'apprentissage utilisés.

Sur 64 enseignants d'informatique au secondaire 2, 18 ont répondu au questionnaire (28%). La moyenne d'âge est 47 ans. La majorité travaille à 80% ou plus, dans un seul établissement, enseigne ou a enseigné dans une autre discipline (les mathématiques pour 2/3 des répondants), et est titulaire d'un diplôme d'enseignement de l'informatique ou est en train de l'obtenir. Tous les enseignants ayant répondu ont un diplôme d'enseignement pour le secondaire, et au minimum un Master (40% ont un doctorat). Ils ont de nombreuses années d'expérience dans l'enseignement, mais très peu dans l'enseignement de l'informatique. Tous les répondants avaient consulté le PE avant de commencer à enseigner (une exception l'a consulté après) ; quinze enseignaient l'informatique l'année précédente ; parmi eux, onze avaient traité tous les objectifs du PE.

Sur l'enseignement, le ressenti est contrasté pour les thématiques *Réseaux* et *Algorithmique et programmation* en ce qui concerne les contenus, l'évaluation ou la coopération avec les collègues. Pour l'apprentissage, les résultats sont partagés pour *Réseaux* et *Information et données*, mais *Algorithmique et programmation* apparaît nettement plus *difficile pour les élèves*. 55% des répondants ont respecté leur planification contre 45% qui n'ont pas pu la respecter, 75% ont planifié en collaboration avec les collègues (échanges, discussions, réunions de groupe) et 50% en s'appuyant sur le PE. Les raisons données pour expliquer la difficulté à respecter la planification sont le manque de temps, les cours annulés, un programme trop ambitieux et détaillé, des prérequis absents chez les élèves, leurs difficultés, un faible engagement et un manque d'investissement, la difficulté de faire travailler les élèves sur la programmation en dehors de la classe.

[2] Les données brutes de l'Étude 1 pourraient être mises à disposition de la recherche sous réserve d'être rendues anonymes. Les auteurs remercient les participants du CAS-HEP pour la contribution de leurs travaux à cette étude. Les données brutes de l'Étude 2 pourraient être rendues disponibles pour la recherche sous réserve 1) de leur anonymisation et 2) de l'accord des auteurs quant à l'utilisation de ce qu'ils ont produit. Certains résultats contiennent l'enregistrement vidéo de scénarios de robotique et sont sujets à des restrictions d'usage au vu de l'âge des élèves qui y figurent.

Le 5ème point avait trait à la préparation des séquences d'enseignement. Le Tableau 3 rapporte des difficultés rencontrées. Des difficultés ressortent aux niveaux didactique (*difficulté d'enseigner les contenus du PE* et *difficultés que posent ces contenus aux élèves*) et technique (l'*enseignant n'est pas libre* d'installer les logiciels qu'il souhaite, et *les logiciels mis à disposition ne sont pas adéquats*).

Pour la mise en œuvre, quatre souhaits ressortent : l'alternance théorie/pratique, une approche plus ludique ou utilitaire, la réduction du temps passé sur les langages par blocs au profit des langages textuels, la programmation débranchée. Concernant les ressources, trois ressortent : les cours personnels, les cours des collègues et le site https://modulo-info.ch. Les ressources françaises sont également bien utilisées.

Tableau 3. Difficultés rencontrées par les enseignants

Types de difficultés	Quelques éléments mentionnés	Nombre
Aucune	« aucune », « pas de contraintes »	3
Temps	« manque de temps pour préparer les séquences », « pas assez de temps pour préparer »	3
Ressources	« pas de ressources » , « difficultés à trouver les ressources », « ressources non adaptées »	3
Maîtrise des logiciels/outils par les enseignants	« Filius à maitriser »	1
Maîtrise des contenus à enseigner	« mes lacunes en réseaux », « les bases de données »	3
Didactique	« comment enseigner », « trop abstrait », « trop compliqué », « adapter le contenu au niveau des élèves » « séries d'exercices progressives et efficaces », « méconnaissance des préconceptions des élèves», « difficultés des élèves non anticipées », « manque d'expérience », « comment partager du code », « comment leur apprendre à utiliser le clavier »	7
Matériel	« défaillant », « qui ne marchait pas », « inconfortables », « non disponible », « soit opérationnel »	3
Logiciels	« impossible d'installer les logiciels », « pas d'IDE », « logiciels non confortables non design »	4
Organisation spatio-temporelle	« disposition de la salle », « élèves de 2e qu'on voit 2h toutes les 2 semaines »	3
Motivation	« motivation des élèves », « mauvaise image de la discipline »	3
Évaluation		1
Peur de l'inconnu		1

Les difficultés rencontrées par les élèves du point de vue de leurs enseignants se situent à deux niveaux : sur l'hétérogénéité entre élèves (et entre enseignants), d'une part et sur les contenus à enseigner, d'autre part.

Concernant l'hétérogénéité entre élèves et entre enseignants, les enseignants soulignent une « scission entre des élèves qui réussissent bien et quelques élèves qui sont complètement à la rue », des « exigences différentes au sein d'un même établissement » (ce qui déstabilise les élèves), « des élèves qui remettent en question la discipline », des élèves qui découvrent l'ordinateur, des disparités et un désavantage pour les élèves qui n'ont pas d'ordinateur à la maison.

Algorithmique et programmation arrive en tête des contenus posant des difficultés aux élèves (selon les enseignants, à nuancer toutefois la thématique occupant une part importante -environ 60% en temps- dans le PE genevois), suivi de la thématique *Réseaux*. Les difficultés principales sont la conception d'un programme ou d'un algorithme, la correction d'un programme, et le suivi pas à pas d'un programme ou d'un algorithme. Le manque de maîtrise des concepts mathématiques, l'abstraction, la modélisation, un temps trop long passé sur la programmation par blocs et la nouveauté (les élèves n'ont jamais programmé auparavant) sont pour les enseignants à l'origine de ces difficultés. Enfin pour *Réseaux*, l'analogie avec le réseau postal, la thématique éloignée des préoccupations des élèves et l'enseignement magistral sont les raisons avancées par les enseignants pour expliquer ces difficultés.

Il faudra disposer de données transversales, sur une période de quelques années, pour confirmer ou infirmer, à l'échelle du canton, cette perception des enseignants que les contenus sont difficiles à appréhender pour les élèves.

Sur 10 collèges que compte le canton de Genève, 4 ont répondu au questionnaire pour les établissements. Ils engagent les logiciels, matériels et supports communs suivants : Python3, Thonny Python, comme IDE, Filius, comme logiciel de simulation sur les réseaux informatiques, SQlite, comme système de base de données. Les vidéos ou capsules vidéos, les épaves d'ordinateurs, et Micro:bit sont aussi des supports utilisés.

3.2 Étude du CAS HEP-BEJUNE (Étude 2)

Le CAS informatique de la HEP-BEJUNE s'est déroulé entre août 2020 et octobre 2022. Il a permis à 10 enseignants d'informatique au secondaire 2 (sur 12 candidats au début du programme) de compléter leur formation académique et de pouvoir continuer à enseigner la DO informatique dans l'espace BEJUNE. La partie théorique de la formation était dispensée par des professeurs d'informatique de l'Université de Neuchâtel. La partie pratique, sur laquelle les étudiants étaient évalués pour obtenir leurs crédits, s'est déroulée selon un modèle collaboratif de classe inversée :

- les étudiants devaient produire à la fin de chaque semestre un scénario pédagogique (appelé séquence pédagogique, dans le cadre du programme) dans le contexte de la nouvelle DO informatique, et notamment en conformité avec les objectifs du PE cantonal correspondant, pour l'une au moins de leurs classes ;
- les séquences ainsi produites étaient mises à disposition de tous les étudiants, via une plateforme de gestion de contenus sur internet [6] ;
- les séquences étaient présentées et discutées en classe, testées par les étudiants dans leurs enseignements, partagées avec les collègues, et pour certaines réutilisées par eux ;
- les deux coordinateurs du programme évaluaient les séquences sur la base de critères prédéfinis, ainsi que deux autres participants pour chaque séquence, avec les mêmes critères, qui retournaient leur évaluation au candidat concerné et aux coordinateurs comme partie intégrante de leur travail pratique.

La majorité des candidats sont professeur de mathématiques, deux sont professeur de géographie et un professeur de droit. Au début de chaque semestre une discussion générale permet de fixer les sujets des séquences afin d'éviter des redondances et de garantir une large variété thématique par semestre et sur l'ensemble du programme. Au cours des quatre semestres, les sujets choisis évoluent d'une relative généralité (désinformation, harcèlement en ligne, internet, énergie/environnement) à des sujets de plus en plus consistants (bases de données, réseaux, robotique), avec en arrière-plan un intérêt persistant pour les algorithmes et la programmation.

A l'issue de cette formation, un retour réflexif a été demandé aux étudiants. Les séquences pédagogiques, ainsi que les retours réflexifs, ont été mis à disposition par les étudiants du CAS et par le cadre de la formation afin d'être étudiés pour cet article.

L'analyse de chaque séquence a porté sur 1) le contenu et l'évaluation ; 2) les modalités d'apprentissage et les stratégies pédagogiques ; et 3) le contexte d'apprentissage et la possibilité de la partager et de la réutiliser. L'analyse des retours réflexifs des participants à la fin de leur dernier semestre révèle :

- le développement et/ou l'approfondissement des connaissances en informatique, comprenant la découverte et la mise en pratique de certains types de logiciels (niveau des connaissances) ;
- le développement et/ou l'approfondissement des connaissances sur l'enseignement de l'informatique, son contexte historique, ainsi que les champs d'activité et de recherche en informatique (niveau culturel) ;
- l'engagement, la motivation, et l'intérêt des élèves ; l'identification des notions à leur présenter ; et les représentations développées par eux, ainsi que l'identification de leurs difficultés (niveau didactique) ;
- les modalités de travail en classe et la gestion des problèmes techniques (niveau de la mise en œuvre) ;
- le vécu, et en en particulier les connaissances et la maîtrise dans la discipline (niveau métier).

4. Discussion

La scénarisation pédagogique a évolué, passant d'une approche centrée sur l'enseignant à une approche centrée sur l'apprenant. Les modèles [7][8] aident à structurer les scénarios, à organiser les activités et à assurer la cohérence entre les objectifs pédagogiques, les contenus et les méthodes d'enseignement. L'avènement des technologies numériques a influencé comment les scénarios pédagogiques sont conçus, livrés, voire partagés [3], avec notamment l'utilisation de plateformes d'apprentissage en ligne, de simulation et de réalité virtuelle et augmentée ou de jeux sérieux. Un cadre conceptuel général (indépendant d'un langage de programmation ou d'un paradigme particulier) et applicable à tous niveaux d'enseignement [9], une fois complété par des techniques ou des méthodes spécifiques [10], permet d'adapter la mise en œuvre du concept de scénario pédagogique à l'enseignement de la discipline informatique.

L'Étude 1, de nature quantitative, porte sur la DO informatique dans un grand canton romand et permet d'avoir une vision transversale sur les qualités et le ressenti des enseignants en rapport avec le plan d'étude et ses objectifs. Les réponses aux différentes questions sont contrastées, notamment lorsqu'elles sont projetées sur les thèmes abordés. A ce moment spécial de la DO (l'année suivant son introduction), elle montre, dans une certaine mesure qui pourrait être précisée par une seconde étude plus poussée, que les difficultés sont liées en partie aux parcours précédemment accomplis par les enseignants dans leurs carrières. Les principales thématiques sur lesquels ils butent (ou non), ainsi que leurs élèves (ou non), soit *Réseaux*, *Algorithmique et programmation* et *Information et*

données sont des révélateurs de leurs limites actuelles, des seuils difficiles à passer. Il est aujourd'hui facile à un enseignant ayant été ingénieur réseaux d'enseigner les réseaux, alors que c'est compliqué pour un enseignant n'ayant pas eu, ou créé, dans son parcours académique et/ou son expérience professionnelle un lien avec ce domaine informatique. A ceci semblent s'ajouter des contraintes supplémentaires liées aux établissements, comme l'évaluation, la coopération entre collègues, les règlements et le matériel. On relève également des difficultés partagées au niveau élève : les concepts mathématiques sous-jacents qui demeurent une difficulté pour les élèves, le décalage entre le contenu à enseigner et les besoins des élèves dans certaines thématiques (*e.g.*, *ordinateur*, *machine physique* et *réseaux*), les élèves ne percevant pas l'intérêt de ces enseignements et les enseignants n'étant pas, de leur côté, toujours certains de ce qu'ils doivent enseigner.

Cette dernière remarque ressort aussi de l'Étude 2, non plus transversale, mais bien verticale, qui considère un petit groupe, assez homogène cette fois-ci pour ce qui est de leur parcours d'enseignants en cours de rattrapage formatif. Même si ces enseignants sont tous expérimentés (dans d'autres disciplines), on reconnaît dans le corpus des scénarios produits les signes d'une discipline nouvellement abordée : les objectifs d'apprentissage y sont peu catégorisés, traduisant une focale sur le savoir encore dénuée d'approche holistique, alors même que la motivation, l'intérêt et la curiosité des élèves sont mentionnés, comme difficulté ou comme volonté de l'enseignant. Les enseignants ne hiérarchisent pas explicitement les objectifs en termes de charge cognitive pour les élèves et n'abordent pas le niveau taxonomique des objectifs d'apprentissage cognitifs, alors qu'ils relèvent les difficultés des élèves avec certains savoirs. Si les liens entre objectifs d'apprentissage et contenus d'une part, et contenus et activités d'autre part, sont bien présents, ceux entre les objectifs d'apprentissage et les activités ne sont pas clairs, posant la question de l'alignement pédagogique.

Les retours réflexifs des participants au CAS-BEJUNE, quant à eux, fournissent de précieuses indications qualitatives sur leurs besoins, leurs apprentissages (lors de la formation), les difficultés qu'ils ont rencontrées et leurs questionnements.

Les enseignants rencontrent des difficultés pour s'approprier un contenu disciplinaire qui va au-delà des contenus à enseigner et qui leur permettrait de se sentir plus à l'aise devant les élèves. Ils ont des problèmes à identifier les ressources utiles dans le cadre de leur enseignement (soit par manque de ressources, comme c'est ressenti par les enseignants de Genève, soit au contraire par une surabondance comme dans le cas du CAS-BEJUNE). Le choix des logiciels et des matériels leur est parfois imposé, ce qui les limite et leur ajoute des contraintes s'ils ne les connaissent pas. Dans la mise en œuvre, ils sont confrontés à la gestion des problèmes techniques en classe, à l'accès inégal des élèves aux TIC, et à un manque de motivation et d'intérêt ressenti. Finalement, au niveau didactique, ils se questionnent sur les notions essentielles à transmettre, jusqu'où aller, et comment les transmettre de la meilleure manière aux élèves.

Pour résumer, en termes de besoins, il faudrait créer des espaces de ressources auxquels les enseignants auraient accès (Étude 1) et pourraient contribuer (Étude 2) ; définir des critères permettant de caractériser ces ressources afin d'en faciliter la recherche et le choix pour les enseignants ; et proposer des relations et des articulations entre les notions figurant dans les plans d'études et les stratégies, les méthodes et les ressources pour les mettre en œuvre.

5. Conclusion

Les deux approches, quantitative et transversale dans l'Étude 1, qualitative et verticale dans l'Étude 2, fournissent des informations sur les besoins des enseignants en rapport avec la discipline informatique devenue discipline obligatoire au secondaire 2. Si les résultats de la première peuvent aider à définir des stratégies et des politiques homogènes pour le déploiement de la DO au niveau d'un canton, le scénario pédagogique, comme aide à l'enseignement (de l'informatique aux élèves) et à l'apprentissage (de cet enseignement par les participants) s'est révélé au cours des deux années qu'a duré le CAS-BEJUNE être un outil précieux pour les enseignants. Il a permis aux étudiants à la fois de formuler des propositions, d'en discuter avec leurs collègues, d'apprendre ce qui se faisait par d'autres, d'en reprendre des éléments pertinents, de réagir et de s'améliorer au cours des semestres, et surtout de se libérer de leurs doutes en échangeant avec leurs pair(e)s. Il leur a été utile comme base de planification (des ressources nécessaires, dont le temps), de la préparation et de l'exécution de leurs leçons, mais surtout comme base de réflexion et comme base d'expérience, là où ces deux dernières bases manquaient pour la DO.

Ensemble, les deux études montrent qu'il existe un besoin réel d'outils pour accompagner l'introduction de la DO informatique dans les classes du secondaire 2 en Suisse. L'article identifie l'un d'entre eux : le scénario pédagogique, comme outil de travail pour les enseignants, certes, mais il n'y a là rien de nouveau. Mais surtout comme outil permettant la conception des nouveaux enseignements requis par l'augmentation soudaine des exigences en relation avec l'informatique et son corps enseignant. Le besoin est double. Il faut que des scénarios existent, mais il faut aussi que leur construction, leur gestion et leur mise à disposition permettent aux enseignants

de les trouver, les réutiliser et de les adapter, en fonction des besoins et des ressources disponibles, à leurs situations particulières.

Références

1. https://www.hep-BEJUNE.ch/fr/Formations-continues/Formations-postgrades/CAS/Enseigner-par-le-numerique/CAS-Enseigner-par-le-numerique.html (consulté le 11 août 2023).
2. https://www.unifr.ch/gyminf/fr/ (consulté le 11 août 2023).
3. Zahnd, J., Hurter, F., Vallat, P.-O. : Intégration des TICE dans l'apprentissage, une approche influencée par les objets pédagogiques, Actes de la Recherche No.4 de la HEP–BEJUNE, Bienne, Suisse (2005)
4. Lang, B. : L'informatique : Science, Techniques et Outils, INRIA (1998), https://edutice.hal.science/edutice-00685338/file/a1101f.htm (consulté le 18 août 2023).
5. Bruillard, É. : Acteurs et territoires de l'éducation à l'information : un point de vue « informatique. Dans Chapron F., Delamotte, E. : L'éducation à la culture informationnelle. Villeurbanne : Presses de l'ENSSIB, p. 68-75, (2010)
6. https://graasp.org
7. Brunet, O. : Un modèle de scénarisation pédagogique en vue d'un enseignement de la science informatique à l'école élémentaire dans une optique curriculaire : un travail mené dans le cadre du projet de recherche ANR IE-CARE. Education. (2019) https://dumas.ccsd.cnrs.fr/dumas-03167844
8. Emin, V. : Modélisation dirigée par les intentions pour la conception, le partage et la réutilisation de scénarios pédagogiques, Thèse de doctorat, Lab. Informatique de Grenoble, Université Joseph Fourier, http://tel.archives-ouvertes.fr/tel-00545553/fr/ (2010)
9. Hazzan, O., Lapidot, T., Ragonis, N. : Guide to Teaching Computer Science - An Activity-Based Approach, Springer (2011).
10. Hermans, F. : The Programmer's Brain- What every programmer needs to know about cognition, Manning. (2021)
==========
11. https://www.ne.ch/autorites/GC/objets/Documents/Rapports/2001/01004_CE.pdf (consulté le 19 août 2023)
12. https://www.ne.ch/autorites/GC/objets/Documents/Rapports/2004/04043_CE.pdf (consulté le 19 août 2023)
13. https://www.ne.ch/legislation-jurisprudence/pubfo/ArrRegCE/Documents/2011/FO39_04_ACE_DECS_Ratification_PlanEtudeRomand.pdf (consulté le 19 août 2023)
14. https://ge.ch/grandconseil/data/texte/PL11477.pdf (consulté le 19 août 2023)
15. Concordat Harmos: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwjwnqOQ5-iAAxVbg_0HHcQjCHEQFnoECA8QAQ&url=https%3A%2F%2Fedudoc.ch%2Frecord%2F96778%2Ffiles%2Fharmos-konkordat_f.pdf&usg=AOvVaw1TrdtK3A-ByFFuYzDbkd6-&opi=89978449 (consulté le 19 août 2023)
16. https://owl-ge.ch/IMG/pdf/PlanDirMITIC_10_110215.pdf (consulté le 19 août 2023)
17. https://edudoc.ch/record/131918?ln=fr (consulté le 19 août 2023)
18. https://www.ge.ch/document/12518/telecharger (consulté le 19 août 2023)

Annexe : l'informatique dans l'enseignement en Suisse et l'émergence de la DO

Dans les années 1970, quelques tentatives avaient vu le jour pour intégrer l'enseignement de la programmation à l'école [5]. Toutefois, ce n'est que suite aux développements de l'informatique dans les autres domaines (sciences, économie, et même politique) que l'introduction de l'informatique à l'école a commencé à devenir un thème.

Par exemple, entre 2001 et 2004, le projet « ICT 01-04 » [11], doté d'un budget de 11'900'000CHF et accepté à l'unanimité moins une abstention par le Grand conseil (GC) du canton de Neuchâtel (un canton peuplé alors de 165'500 habitants), équipe toutes les classes d'ordinateurs personnels et toutes les écoles de réseaux informatiques interconnectés (le Réseau pédagogique neuchâtelois) et connectés à Internet. Le volet logiciel en sera malheureusement abandonné et l'initiative, pourtant marquée comme un succès par un second vote du GC en 2006 [12] tombera lentement en désuétude. En 2007, le projet de loi PL10177, présenté par le Conseil d'État au GC genevois et ouvrant un crédit d'investissement de 30'850'000CHF pour le développement de l'administration cantonale en ligne (AeL), définit la dixième et dernière prestation proposée comme l'« Espace école en ligne », la première étant les « Impôts en ligne ». Le PL10177 sera bouclé par le GC en 2014 [13].

L'équipement des écoles en matériel informatique et leur connexion à Internet suscite des débats et des discussions entre les enseignants, mais aussi entre les parents d'élèves. Les questionnements portent aussi bien sur des problématiques spécifiques (la manière de mettre en place un programme d'enseignement efficace et adapté aux besoins des élèves, la formation des enseignants), que sur des problématiques plus générales (protection de la vie privée, fracture numérique entre les élèves qui ont accès à la technologie et ceux qui n'y ont pas accès, les coûts de l'équipement, les risques d'addiction aux écrans). Dans les deux exemples cantonaux ci-dessous, les approches et la compréhension de l'informatisation de l'École divergent entre le politique (maître des finances), l'administration (maîtresse de la technique à travers ses services informatiques) et les services de l'enseignement (sans véritable pouvoir, sinon celui de la connaissance des besoins et de ce qui se passe chez nos voisins).

Pendant ce temps, les régions linguistiques (alémanique d'un côté et latine de l'autre) mettent en place leurs premiers plans d'étude communs. Le plan d'études romand (PER) est introduit progressivement à partir de 2010 [14], alors que le plan d'études alémanique (Lehrplan 21) est toujours en cours d'introduction (élaboré entre 2010 et 2014 sous l'égide de la Conférence des directeurs de l'instruction publique de Suisse alémanique). Comment envisager dès lors qu'une nouvelle discipline, l'informatique, soit introduite en parallèle à des travaux d'harmonisation cantonale de plans d'étude et à l'introduction d'un concordat fédéral sur l'école obligatoire [15] qui suscitent une fronde politique dans plusieurs cantons ?
Quelques repères montrent le temps passé et les sujets abordés, avant l'introduction de la DO :

- entre 2011 et 2016, un plan directeur intitulé « Enseigner et apprendre à l'ère numérique » est construit par le Département de l'instruction publique du canton de Genève [16] ;
- en 2017, la Suisse met en place un plan national pour l'informatique à l'école ;
- en septembre 2017, un postulat du groupe libéral-radical est accepté au GC *neuchâtelois* (voir ci-dessus) pour une meilleure intégration du numérique à l'école ;
- approuvé au niveau fédéral en 2017 et entré en vigueur en août 2018, le plan d'études cadre Informatique pour les écoles de maturité remplace celui de 2008 [17] ;
- en 2018, le PER est révisé pour intégrer les nouveaux apprentissages relevant de l'éducation numérique pour la scolarité obligatoire ;
- en juin 2018, la Conférence suisse des directeurs cantonaux de l'instruction publique (CDIP) adopte une Stratégie nationale sur la numérisation dans le domaine de l'éducation ;
- fin 2018, la CIIP valide un Plan d'action en faveur de l'éducation numérique pour la Romandie et le Tessin ;
- en 2018, "L'école au service de la citoyenneté numérique" [18] est présenté par le DIP de Genève ;
- en 2019, les députés de la commission des travaux du GC genevois *rejettent* un projet de loi visant à équiper les établissements de matériel informatique ;
- en mars 2019 à Genève, une première version du Rapport du Conseil d'État au GC à l'appui d'un projet de décret portant octroi d'un crédit d'engagement pour le programme Éducation numérique s'appliquant aux écoles obligatoires et post obligatoires est mis en consultation, validé et soumis au Grand Conseil ;
- à la rentrée 2021, la science informatique est ajoutée au programme du Collège et de l'École de culture générale (enseignement secondaire 2) du canton de Genève ;
- en 2022, les députés de la commission des travaux du GC genevois *rejettent à nouveau* un projet de loi visant à équiper les établissements de matériel informatique ;
- à la rentrée 2022 enfin, dans le cadre du déploiement numérique du PER, le nouveau programme cantonal est généralisé en 9^{e} année à Genève et déployé à titre expérimental en classes de 10^{e} et 11^{e} années.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Un référentiel de compétences en programmation pour identifier les difficultés des débutants et différencier les activités

Sophie Chane-Lune, Christophe Declercq et Sébastien Hoarau

Laboratoire d'Informatique et de Mathématiques, Université de La Réunion

Résumé Nous proposons un référentiel de compétences en programmation croisant compétences de la pensée informatique, et notions informatiques au programme du lycée en France, dans l'objectif de mieux identifier les difficultés des programmeurs débutants, en particulier par rapport aux compétences d'abstraction et de généralisation mises en œuvre par l'usage de fonctions informatiques.

Nous proposons une série d'activités élémentaires alignées avec les compétences définies et en déduisons un test destiné à identifier ces difficultés chez les programmeurs débutants.

Nous présentons ensuite les résultats de deux cohortes d'élèves et d'étudiants à ce test et discutons ces résultats. Nous en déduisons des nécessités de remédiation et proposons des activités diversifiées permettant de travailler les compétences non maitrisées par les élèves.

1 Introduction

L'introduction de l'informatique en tant que discipline au lycée en France en 2019 s'est effectuée avec d'une part la mise en place de programmes et d'horaires dédiés et d'autre part la formation continue puis initiale d'enseignants pour la discipline. Les moyens d'enseignement ont suivi avec des manuels, une activité intense d'élaboration entre pairs [3] et l'arrivée de sujets d'épreuves avec leur effet normalisateur sur les pratiques enseignantes.

Les praticiens ont alors redécouvert les difficultés des programmeurs débutants, pourtant largement documentées par la recherche à l'époque de l'option informatique des lycées dans les années 1980 [11,10], à savoir la difficulté des notions de variable et de fonction, l'intérêt de méthodes de programmation, la complexité intrinsèque des dispositifs d'exécution, le statut particulier des variables logiques... Une revue des travaux de cette époque a été effectuée plus récemment par Lagrange et Rogalski [6]. Les enseignants ont aussi pu découvrir, dans le contexte nouveau créé par l'introduction de Scratch au collège en 2016, la délicate transition d'un langage de programmation par blocs à un langage textuel (Python).

Les chercheurs ont rapidement mis en évidence la faible diversité des activités proposées aux élèves dans les moyens d'enseignement - voir en particulier l'analyse de [5] concernant les tâches figurant dans les activités des manuels ou dans

Volume 1, pages 53–63.

les EIAH. Des formes d'activités originales, proposant d'analyser les programmes avant d'en écrire, ont déjà été proposées par Goletti et Mens [4].

Les praticiens regrettaient au même moment la normalisation en partie provoquée par la forme imposée des sujets d'épreuves pratiques au baccalauréat à savoir l'activité de type feuille blanche (écrire un programme qui...) ou le programme à trous.

C'est dans ce contexte que notre recherche a débuté en tentant d'explorer d'autres formes d'activités pour pallier aux difficultés constatées des élèves. Lors d'un premier travail, nous avons commencé à utiliser l'approche par compétences [1], popularisée en informatique par les travaux de Wing [13], pour analyser les compétences en jeu dans différentes situations.

2 Vers un référentiel de compétences en programmation adapté aux programmes du lycée en France

Nous avons initialement fondé notre réflexion sur la définition opérationnelle de la pensée informatique proposée par Selby et Woolard [12] à savoir la capacité à penser en termes d'abstraction, de décomposition, à penser « algorithmiquement », et à penser en termes d'évaluation et de généralisation. Nous avions déjà proposé de formuler ces compétences en français par des verbes [2] et de préciser la « pensée algorithmique » en utilisant le verbe *anticiper*. Anticiper est en effet le principal obstacle didactique rencontré par les programmeurs débutants : « Une propriété difficile à intégrer [...] est le caractère différé d'une exécution du programme » [9]. Anticiper les étapes successives d'un traitement et les différents cas à envisager, c'est tout l'art de programmer. Nous avions alors cinq compétences génériques pour décrire la pensée informatique : évaluer, anticiper, décomposer, généraliser et abstraire.

Nous avons ensuite proposé d'ajouter une sixième compétence générique aux précédentes : la compétence *modéliser*. Cette compétence est largement partagée avec d'autres sciences dont en particulier les mathématiques, mais elle revêt une signification particulière en informatique quand il s'agit de choisir l'information à représenter comme donnée d'un problème ou résultat. Cette compétence avait déjà été énoncée par Wing : « It is choosing an appropriate representation for a problem or *modeling* the relevant aspects of a problem to make it tractable ». La compétence *modéliser* avait cependant été écartée dans la définition de Selby et Woolard au prétexte que les modèles sont utilisés par les compétences de la pensée informatique mais ne la définissent pas. Nous pensons au contraire qu'il est utile de distinguer la compétence *modéliser* de la compétence *abstraire* à laquelle elle pourrait être rattachée, parce que nous avons pu observer que des élèves maîtrisent l'une et pas l'autre.

Prenons le problème du calcul de la méridienne avec un sextant et une montre qui permet à un marin de calculer sa longitude par mesure de l'heure de culmination du soleil. Modéliser, consiste à choisir les variables pour représenter les données du problème, à savoir le temps de la montre en heures, minutes et secondes. Il y a bien sûr des abstractions le plus souvent implicites : dans ce calcul,

on peut faire abstraction de l'âge du capitaine. On pourrait aussi proposer une abstraction de données, consistant à représenter un temps par une structure de données abstraite. Pour des débutants en programmation, savoir *modéliser* des données par quelques variables élémentaires est une compétence assez accessible. La compétence *abstraire* demande quant à elle un apprentissage spécifique et beaucoup plus long, d'où l'intérêt de distinguer ces deux compétences.

Les compétences *évaluer* et *anticiper* permettent déjà de diversifier les activités. *Evaluer* un programme, c'est la capacité à lui attribuer mentalement une valeur (résultat, type...). L'activité part donc d'un programme fourni. *Anticiper*, c'est décrire à l'avance ce que devra faire la machine, c'est-à-dire prévoir par un programme, l'enchaînement séquentiel, répétitif ou conditionnel des instructions, avant même le début de son exécution. L'activité part donc d'une feuille blanche. Nos premières expériences [1] ont montré qu'il était plus facile pour les élèves d'apprendre à lire un programme (compétence *évaluer*) que d'apprendre à en écrire (compétence *anticiper*).

La maîtrise de la compétence *modéliser* est un préalable à toute activité d'écriture de programme. En l'explicitant, on déduira des activités préliminaires où la tâche de l'élève consiste à choisir des variables où des structures de données pour représenter l'information disponible ou à calculer.

Les compétences *décomposer*, *généraliser* et *abstraire* sont des compétences cognitives de plus haut niveau. Ce sont des compétences méthodologiques qui vont être sollicitées lors de la réécriture d'un programme vers une version plus générale ou plus abstraite, ou lors de l'analyse descendante d'un problème (compétence *décomposer*).

Le préambule commun aux programmes d'informatique au lycée en France [8] fait explicitement référence à ces compétences :

- *analyser et modéliser un problème en termes de flux et de traitement d'informations ;*
- *décomposer un problème en sous-problèmes, reconnaître des situations déjà analysées et réutiliser des solutions ;*
- *concevoir des solutions algorithmiques ;*
- ...
- *développer des capacités d'abstraction et de généralisation.*

Cependant la suite des programmes n'y fait plus référence et est organisée par contenus, dans lesquels sont notées des capacités attendues.

La difficulté, pour l'enseignant, à mettre en œuvre une approche par compétences est alors de distinguer pour chaque compétence, les différents contextes où elle peut intervenir. Par exemple, la compétence *généraliser* est invoquée dans une activité consistant à passer d'une instance à une classe de problèmes. Au niveau élémentaire cela peut se faire en remplaçant une constante par une variable. Cette compétence est aussi mise en oeuvre lors du remplacement d'un fragment de code par une fonction ou plus tard en utilisant la notion de classe en programmation objet. Il n'est donc pas réaliste de dire qu'un élève sait *généraliser*, sans préciser le contexte dans lequel intervient cette généralisation.

2.1 Construction d'un référentiel croisant compétences et notions au programme

Notre proposition est de croiser une approche basée sur les compétences génériques de la pensée informatique avec une approche basée sur les compétences *programmer avec des notions particulières*. Notre méthode a consisté à réécrire l'ensemble des capacités attendues des programmes d'informatique au lycée relatives au domaine de la programmation, en utilisant uniquement les six verbes choisis. Cela nous a amené à déboubler en plusieurs compétences élémentaires quand une capacité attendue au programme masquait plusieurs compétences, ou parfois à regrouper plusieurs capacités en une seule compétence élémentaire. Ce travail a été mené de mars à octobre 2023 lors de séances de travail régulières entre les co-auteurs (8 séances de 3h) avec recherche de consensus pour tous les choix effectués.

La figure 1 montre cette catégorisation pour les notions de programmation au programme de la classe de seconde en sciences numériques et technologie (SNT).

Compétence	Évaluer	Modéliser	Anticiper	Décomposer	Généraliser	Abstraire
Programmer avec des expressions	Évaluer la valeur et le type (entier, chaîne ou booléen) d'une expression comportant variables, constantes et opérateurs	Modéliser les informations disponibles/à calculer par des variables de type élémentaire (entier, chaîne et booléen)	Anticiper l'écriture d'une expression comportant variables, constantes et opérateurs	Décomposer une expression complexe en identifiant une ou des variables intermédiaires	Généraliser une expression en remplaçant une constante par une variable	
Programmer avec des affectations et des séquences	Évaluer les valeurs des variables, à l'exécution d'un programme comportant une séquence d'affectations	Modéliser les informations à calculer par une ou des variables intermédiaires	Anticiper l'écriture d'un traitement séquentiel pour obtenir un résultat spécifié, la méthode étant donnée	Décomposer un traitement complexe en identifiant une ou des variables intermédiaires		
Programmer avec des constructions répétitives	Évaluer les valeurs des variables, à l'exécution d'un programme comportant une construction répétitive	Modéliser par une ou des variables les informations à modifier à chaque répétition	Anticiper l'écriture d'un traitement répétitif pour obtenir un résultat spécifié, la méthode étant donnée	Décomposer un traitement complexe en identifiant une répétition	Généraliser un traitement séquentiel en le remplaçant par un traitement répétitif	
Programmer avec des constructions alternatives ou conditionnelles	Évaluer les valeurs des variables, à l'exécution d'un programme comportant une ou des constructions alternatives ou conditionnelles	Modéliser par une variable booléenne la condition d'une alternative ou d'une conditionnelle	Anticiper l'écriture d'un traitement alternatif ou conditionnel pour obtenir un résultat spécifié, la méthode étant donnée	Décomposer un traitement complexe en plusieurs cas en utilisant des constructions alternatives ou conditionnelles		
Programmer avec des fonctions	Évaluer le résultat de l'appel d'une fonction avec des paramètres donnés	Modéliser un traitement par une fonction en spécifiant ses paramètres	Anticiper l'écriture du corps d'une fonction	Décomposer un traitement complexe en utilisant la composition de fonctions	Généraliser un traitement, par la définition d'une fonction avec paramètres ou par l'ajout d'un paramètre à une fonction	Abstraire un traitement par une définition de fonction avec paramètres et commentaire

Figure 1. Référentiel pour la classe de seconde

Au croisement d'une colonne et d'une ligne, figure une capacité ou compétence élémentaire correspondant à la compétence générique de la colonne, contextualisée par la ou les notions en jeu dans la ligne courante.

On retrouve ainsi en particulier au croisement de *généraliser* et de *programmer avec des fonctions*, la compétence élémentaire *généraliser un traitement par la définition d'une fonction avec paramètres*. Les cases restant grisées correspondent à des croisements auxquels nous n'avons pas trouvé d'intérêt spécifique. Les compétences *généraliser* et *abstraire* ne sont pas systématiquement sollicitées pour toutes les notions de programmation car elles sont intrinsèquement liées aux notions en jeu. L'abstraction regroupe l'abstraction de données qui permet d'encapsuler des informations, l'abstraction par des fonctions qui permet d'encapsuler des traitements, et l'abstraction par les classes en programmation objet qui conjugue abstraction de données et abstraction par des fonctions. La généralisation regroupe la généralisation par l'ajout de variables ou paramètres et

la généralisation de motifs qui consiste à inférer un traitement répétitif à partir d'un motif séquentiel répété.

Ce travail est en cours pour l'ensemble des programmes du lycée de la seconde à la terminale dans le cadre du travail doctoral de S. Chane-Lune. Il consiste à identifier pour chaque contexte de programmation lié à l'introduction de nouvelles notions, les possibles croisements avec les compétences génériques de la pensée informatique.

2.2 Aligner activités et compétences

Une de nos hypothèses de recherche est de postuler que l'on peut pour chaque compétence élémentaire identifier une activité élémentaire permettant de tester en contexte cette compétence.

La figure 2 regroupe pour les compétences citées en figure 1, des propositions d'activités permettant de les tester.

Compétence	Évaluer	Modéliser	Anticiper	Décomposer	Généraliser	Abstraire
Programmer avec des expressions	Quel est le type et la valeur de l'expression suivante ?	Choisir une ou des variables pour représenter ...	Ecrire une expression dépendant de ... permettant de calculer ...	Réécrire l'expression suivante en nommant ... la partie entre parenthèses	Réécrire l'expression suivante en remplaçant la valeur ... par une notation plus générale	
Programmer avec des affectations et des séquences	Quelle est la valeur de la variable ... à la fin de l'exécution du programme Python suivant ?	Choisir une ou des variables pour mémoriser les résultats intermédiaires utiles pour calculer ...	Ecrire un programme Python permettant de calculer... / Ecrire un programme Python permettant d'afficher ...	Décomposer le problème suivant en utilisant des variables intermédiaires		
Programmer avec des constructions répétitives	Quels sont les messages affichés à l'exécution du programme Python suivant ? / Quelle est la valeur de la variable ... à la fin de l'exécution du programme Python suivant ?	Choisir une ou des variables à modifier itérativement pour calculer...	Ecrire un programme Python permettant de calculer ... par la méthode suivante ... / Ecrire un programme Python permettant d'afficher ...	Décomposer le problème suivant en identifiant un traitement pouvant être répété plusieurs fois	Modifier le programme Python suivant en utilisant une répétition, tout en conservant les mêmes affichages à l'exécution / le même résultat	
Programmer avec des constructions alternatives ou conditionnelles	Quels sont les messages affichés à l'exécution du programme Python suivant ? / Quelle est la valeur de la variable ... à la fin de l'exécution du programme Python suivant ?	Choisir et nommer une variable pour mémoriser le respect de la condition ... dans le problème suivant ...	Ecrire un programme Python permettant de calculer ... par la méthode suivante ... / Ecrire un programme Python permettant d'afficher ...	Décomposer le problème suivant en distinguant les cas nécessitant un traitement particulier		
Programmer avec des fonctions	Quelle est la valeur à l'évaluation de l'expression ... avec la définition de fonction Python suivante ?	Spécifier le nom, les paramètres d'entrée et le type de résultat à renvoyer pour une fonction destinée à calculer...	Ecrire une fonction Python ... , qui étant donnés ... , renvoie ...	Décomposer le problème suivant en identifiant deux parties pouvant chacune être programmées par une fonction	Dans le programme Python suivant, remplacer les instructions écrites plusieurs fois, en définissant et utilisant une fonction	Décrire le traitement effectué par la fonction suivante en faisant abstraction de la méthode utilisée

Figure 2. Activités génériques par compétences

Il s'agit, dans cette proposition, d'énoncés génériques d'activités devant être complétés par un programme ou un problème particulier. Par exemple l'activité 1 ci-dessous est une instance possible d'activité pour la compétence élémentaire : *évaluer les valeurs des variables, à l'exécution d'un programme comportant une séquence d'affectations*. On appelera dans la suite question, toute activité qui sera incluse ultérieurement dans un test.

Question 1. Quelles sont les valeurs des variables `a` et `b` à la fin de l'exécution du programme Python suivant ?

```
a = 42
b = 56
a = b - a
b = b - a
a = a + b
```

L'activité suivante est une variante concernant la même compétence.

Question 2. Quelle est la valeur de la variable `bissextile` à la fin de l'exécution du programme Python suivant ?

```
par4 = 2024 % 4 == 0
par100 = 2024 % 100 == 0
par1000 = 2024 % 1000 == 0
bissextile = par4 and not par100 or par1000
```

Concernant les compétences de programmation au programme de la classe de seconde, nous avons ainsi pu aligner notre référentiel de compétences avec une proposition d'activités élémentaires ciblant chacune l'apprentissage d'une compétence particulière. La réponse est moins évidente concernant les compétences en jeu dans la suite des programmes du lycée où des interdépendances apparaissent, ce qui rend plus difficile la conception d'activités élémentaires spécifiques.

2.3 Proposition d'un test de compétences en programmation

Notre proposition d'un test de compétences en programmation a pour objectif d'identifier les difficultés et de proposer des activités de remédiation. Il ne s'agit pas d'évaluer les élèves, ce qui nécessiterait un travail préliminaire d'alignement entre les objectifs de l'enseignement, les activités proposés et les modalités d'évaluation.

Notre proposition d'évaluation diagnostique a pour seule ambition de repérer les difficultés récurrentes des élèves et de proposer des activités de remédiation adaptées.

Nous avons ainsi pu fonder une proposition de test de compétences en programmation sur la base des 24 énoncés génériques du tableau de la figure 2. Nous avons aussi souhaité diversifier pour chaque activité élémentaire, le type d'informations manipulées et la nature du programme (interactif ou calculatoire). Nous avons alors obtenu un test comportant plus de cinquante questions couvrant l'ensemble des compétences.

Dans la pratique, il n'a pas été jugé possible par les enseignants expérimentateurs de mettre en œuvre le test complet, ce qui aurait pris un temps supérieur à deux heures pour les élèves. Dans l'attente du développement d'une méthodologie de choix de questions permettant d'assurer une couverture suffisante à la fois des compétences et des types de données et de problèmes, tout en conservant un nombre de questions raisonnable, il a été convenu de laisser les expérimentateurs choisir leurs questions pour la première expérimentation.

Les résultats qui suivent portent donc sur des extraits de ce test, adaptés aux contextes d'expérimentation. La première expérimentation a été conduite auprès d'une classe de première NSI, dont les élèves avaient tous suivis l'enseignement de seconde l'année précédente. La seconde expérimentation a été menée avec une promotion de licence première année d'informatique et de mathématiques, qui avaient eux aussi tous suivi l'enseignement de seconde trois ans auparavant.

3 Expérimentation et premiers résultats

3.1 Expérimentation avec une classe de première spécialité NSI

Une version allégée du test a été proposée aux 49 élèves de première NSI du lycée Mémona Hintermann-Afféjee (Saint-Denis de La Réunion), en début d'année scolaire 2023. Ce test a été découpé en deux parties et chacune d'elle a été réalisée à une semaine d'intervalle. La première partie du test comportait 11 questions et portait sur les compétences *évaluer* (5 questions dont la question 3), *anticiper* (4 questions), *modéliser* (une question) et *généraliser* (question 4) dans le contexte de la programmation élémentaire (sans fonctions).

Question 3. Quelle est la valeur de l'expression Python suivante ?

```
2 + 2 == 4 or 2 + 2 == 5
```

Question 4. Modifier le programme Python suivant en utilisant une répétition, tout en conservant les mêmes affichages à l'exécution.

```
print("1 ...")
print("2 ...")
print("3 ...")
print("Partez !")
```

La deuxième partie du test portait sur le contexte des fonctions et comptait 5 questions, chacune d'elle portant sur l'une des compétences suivantes : *évaluer des fonctions*, *modéliser une fonction*, *anticiper l'écriture d'une fonction*, *généraliser* et *abstraire avec des fonctions*.

Les résultats sont médiocres en moyenne. Les seules questions avec un taux de réussite supérieur à 50 % sont les questions d'évaluation d'expressions de type entier ou chaîne. Toutes les autres questions ont un taux de réussite inférieur à 25 %.

La figure 3 montre la répartition des scores obtenus sur la cohorte de première pour les deux parties du test.

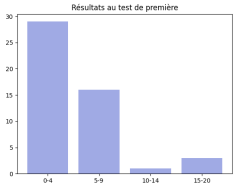


Figure 3. Répartition des scores pour les élèves de première

3.2 Expérimentation avec une promotion de licence première année

Le test qui a été administré aux étudiants de l'université une semaine après le test au lycée, avait un objectif différent : mesurer ce qu'il reste des enseignements d'informatique en seconde pour des primo-entrants à l'université n'ayant pour moitié pas poursuivi la spécialité NSI au lycée jusqu'en terminale. La variable déterminante dans ce contexte est le fait d'avoir ou pas suivi l'enseignement NSI en classe de première et terminale.

Les contraintes matérielles de passation du test en amphi le jour de la réunion de rentrée ont contraint à choisir un questionnaire à réponse numérique, réalisé sur papier et lisible par lecture optique. Ce choix permettait de réduire le temps de correction et d'obtenir rapidement de premières analyses. Cela a aussi limité les possibilités de tester différentes compétences et a amené à tester exclusivement la compétence *évaluer*. Au vu des résultats de la première expérimentation avec la classe de première, où seules ces questions ont donné lieu à des réponses majoritairement correctes, cela nous a semblé cependant utile pour obtenir une première photographie partielle des compétences acquises par cette promotion.

La promotion de première année de licence de l'université de La Réunion comportait 118 étudiants d'informatique et 43 étudiants de mathématiques. Les questions avec réponses numériques (entre 1 et 31) ont été encodées par les étudiants sur la feuille réponse en noircissant les cases correspondant au code en base 2 de la réponse. Le tableau de codage était donné sur l'énoncé. Les résultats des étudiants d'informatique ont été différenciés selon la variable : *a suivi NSI en première et en terminale*, catégorie qui comporte 51 étudiants sur 118.

Les 20 questions proposées portaient toutes sur la compétence *évaluer*, dont 5 sur *évaluer un programme avec des affectations et des séquences*, 6 sur *évaluer un programme avec des constructions répétitives*, 4 sur *évaluer un programme avec des constructions alternatives ou conditionnelles* et 5 sur *évaluer un programme avec des fonctions*. La figure 4 montre la répartition des scores pour ce test. Une analyse d'indépendance par la méthode du χ^2 confirme que la répartition des scores n'est pas indépendante de la variable *a suivi NSI en première et en terminale* avec une erreur de l'ordre de 10^{-6}.

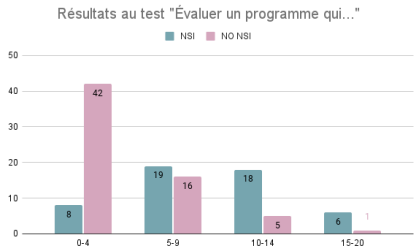


Figure 4. Répartition des scores pour les étudiants d'informatique

3.3 Analyse des résultats

On s'intéresse maintenant à l'analyse en détail de quelques questions caractéristiques de difficultés rencontrées à la fois chez les lycéens et les étudiants.

Les questions d'évaluation dont les scores moyens sont les moins bons sont ceux qui portent sur des booléens, ce qui confirme des résultats connus [6] : « les difficultés liées aux conditions (valeurs logiques de type booléen) sont sous-estimées ». Une interprétation possible du très fort taux d'erreur à la question 3, tient au conflit entre les pratiques des élèves en mathématiques et en informatique. En mathématiques les lycéens apprennent à n'écrire que des propositions vraies. Ceci peut entrer en conflit avec l'écriture en informatique d'expressions booléennes valant potentiellement faux. Cette interprétation est uniquement basée sur les verbatims d'élèves : « Madame, il y a une erreur dans la question 3 » et mériterait d'être approfondie par une étude qualitative des représentations des élèves.

Concernant la question 1, la mauvaise réponse `28` très fréquente illustre une conception erronée de la sémantique de l'affection au sein d'une séquence d'instructions. L'expression `b - a` semble être calculée une fois pour toutes avec les valeurs initiales (`42` et `56`) pour produire la valeur `14`. Cette valeur est utilisée ensuite dans les deux nouvelles affectations de `a` et `b`, expliquant la réponse `28` pour la valeur de la variable `a`. Ce qui est étonnant ici, c'est que les valeurs initiales de `a` et `b` soient utilisées dans les deux premières affectations, puis que leurs nouvelles valeurs soient enfin prises en compte dans la dernière. On peut supposer l'influence réductrice d'un savoir en calcul algébrique, le principe de substitution des variables, qui n'est pas transposable en programmation impérative, et qui est utilisé ici de manière erratique.

Les questions portant sur l'évaluation d'un programme comportant une répétition révèlent à nouveau un usage abusif du principe de substitution au moment de modifier une variable de type accumulateur. Des réponses d'élèves et d'étudiants, semblent montrer qu'ils substituent partout le nom d'une variable par sa valeur initiale avant de commencer à simuler l'exécution de la boucle.

Les questions portant sur la généralisation et l'abstraction par les fonctions n'ont été traitées par aucun élève de première. Ceci peut interroger sachant que l'écriture des fonctions informatiques figure bien à la fois au programme de mathématiques et de SNT en classe de seconde. Une piste d'explication pourrait venir des moyens d'enseignement utilisés : la quasi-totalité des activités portant sur les fonctions au sens informatique, dans les manuels de mathématiques de seconde, sont de la forme : *écrire une fonction Python ..., qui étant donnés ..., renvoie ...*. Ce type d'activité sollicite la compétence *anticiper l'écriture du corps d'une fonction*. Les élèves n'ont donc pas eu l'occasion de s'exercer à prendre l'initiative de définir une fonction pour *généraliser* ou *abstraire* un traitement. Ces compétences étaient, de fait, prises en charge par leur enseignant. L'explication peut venir aussi du curriculum de seconde, qui à la différence de celui de première et de terminale, ne précise pas l'objectif de développer les capacités d'abstraction et de généralisation.

4 Conclusion

L'élaboration d'un référentiel de compétences en programmation et d'un catalogue d'activités alignées sur ce référentiel nous a permis d'analyser finement quelques difficultés rencontrées par des programmeurs débutants. Cela nous permet de recommander la diversification d'activités sur la base des énoncés génériques proposés pour développer les compétences les moins souvent travaillées. En particulier les activités permettant de travailler la compétence *évaluer* nous semblent un préalable aux activités d'écriture de code. Nous proposons aussi de développer des activités de conception permettant de développer d'abord la compétence *modéliser* puis la compétence *décomposer* ainsi que des activités de réécriture permettant de développer les compétences *généraliser* et *abstraire*.

Les mauvais résultats globaux à nos tests sont à nuancer par la taille de nos échantillons et l'analyse des contextes d'enseignement. Le rapport du conseil supérieur des programmes [7] a en effet souligné pour l'enseignement de SNT que « la partie scientifique et technique relative aux fondements du numérique et aux activités de programmation est souvent délaissée », ce qui pourrait expliquer les faibles compétences en programmation à l'entrée en première. Le constat que ces compétences restent fragiles à l'entrée à l'université y compris pour certains élèves ayant suivi la spécialité NSI reste à investiguer à plus grande échelle. Il convient aussi d'être prudent concernant l'interprétation des résultats d'un test focalisé sur des compétences élémentaires, alors que les programmes d'enseignement préconisent un apprentissage par projet et proposent des situations de programmation parfois très complexes : programmation dynamique, algorithme de Boyer-Moore...On peut cependant s'interroger sur la possibilité d'enseigner de telles situations complexes, quand les compétences élémentaires sur les notions de base ne sont pas maîtrisées par les élèves.

Les prolongements de ce travail se déclinent sur plusieurs axes : la consolidation du référentiel, son usage pour élaborer des tests et son usage pour catégoriser des activités. Concernant la phase de consolidation, seules les cinq premières lignes du référentiel, concernant les compétences abordées en classe de seconde, ont été présentées ici et analysées. Notre référentiel global pour le lycée en comporte vingt et est en cours de finalisation. Chaque compétence élémentaire doit encore être confrontée à la possibilité de lui associer une activité spécifique. Ces activités doivent ensuite être testées en classe, soit dans le cadre de tests, soit dans le cadre d'activités d'apprentissage ou de remédiation.

Le développement de tests spécifiques de compétences pour la classe de première et de terminale, adossés à ce référentiel, est pour nous une perspective logique pour continuer à investiguer les difficultés rencontrées par les élèves et proposer des activités diversifiées et des remédiations adaptées à chaque niveau.

L'usage de notre référentiel pour catégoriser les activités de programmation complexes proposées aux élèves, lors des épreuves pratiques du baccalauréat ou dans les espaces collaboratifs de partage d'activités entre enseignants, constitue aussi une perspective de recherche intéressante, dans l'objectif de tenter de corréler les difficultés rencontrées par les élèves dans ces activités complexes avec celles identifiées lors de nos tests de compétences.

Références

1. Chane-Lune, S., Declercq, C., Hoarau, S. : Expérimentation du pair-programming en spécialité NSI en classe de première pour l'acquisition de compétences en programmation. In : Broisin, J., Declercq, C., Fluckiger, C., Parmentier, Y., Peter, Y., Secq, Y. (eds.) Atelier " Apprendre la Pensée Informatique de la Maternelle à l'Université ", dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH). pp. 25–32. Brest, France (2023), https ://hal.science/hal-04144211
2. Declercq, C. : Didactique de l'informatique : une formation nécessaire. Sticef **28**(3) (Apr 2022), http ://sticef.org/num/vol2021/28.3.8.declercq/28.3.8.declercq.htm
3. Djeballah, S., Comte, M.H., Fourny, M., Hoarau, S., Juton, A., Khaneboubi, M., Lagarrigue, A., Massart, T., Poulmaire, C., Prince, V., Renouf, S., Viéville, T., Vincent, J.M. : Un espace de formation francophone des enseignants, dédié à l'apprentissage de l'informatique, dans le secondaire. https ://adjectif.net (Sep 2023)
4. Goletti, O., Pierpont, F.d., Mens, K. : Création d'exemples résolus avec objectifs étiquetés pour l'apprentissage de la programmation avec Python (May 2022), https ://hal.archives-ouvertes.fr/hal-03697932
5. Jolivet, S., Dechaux, E., Gobard, A.C., Wang, P. : Description et analyse de trois EIAH d'apprentissage de Python. In : Broisin, J., Declercq, C., Fluckiger, C., Parmentier, Y., Peter, Y., Secq, Y. (eds.) Atelier " Apprendre la Pensée Informatique de la Maternelle à l'Université ", dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH). pp. 9–16. Brest, France (2023), https ://hal.science/hal-04144212
6. Lagrange, J.B., Rogalski, J. : Savoirs, concepts et situations dans les premiers apprentissages en programmation et en algorithmique. Annales de Didactiques et de Sciences Cognitives (Jul 2017), https ://hal.archives-ouvertes.fr/hal-01740442, publisher : Institut de Recherche sur l'Enseignement des Mathématiques de Paris (IREM)
7. Ministère de l'Education Nationale et de la Jeunesse : Avis du conseil supérieur des programmes sur la contribution du numérique à la transmission des savoirs et à l'amélioration des pratiques pédagogiques (juin 2022)
8. Ministère de l'Éducation nationale de la Jeunesse et des Sports : Programmes et ressources en numérique et sciences informatiques - voie G (2019), https ://eduscol.education.fr/2068/programmes-et-ressources-en-numerique-et-sciences-informatiques-voie-g
9. Rogalski, J., Samurçay, R. : Les problèmes cognitifs rencontrés par des elèves de l'enseignement secondaire dans l'apprentissage de l'informatique. European Journal of Psychology of Education (1986)
10. Rogalski, J., Samurçay, R., Hoc, J. : L'apprentissage des méthodes de programmation comme méthodes de résolution de problème. Le Travail Humain **51** (1988)
11. Samurçay, R. : Signification et fonctionnement du concept de variable informatique chez des eleves debutants. Educational Studies in Mathematics (1985)
12. Selby, C., Woollard, J. : Computational thinking : the developing definition. In : SIGCSE (2014)
13. Wing, J.M. : Computational Thinking. Communications of the ACM **49** (2006)

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Une analyse des affordances technologiques des robots éducatifs pour l'école primaire

Vassilis Komis[1] Anastasia Misirli[1] et Stavroula Karagiannopoulou[1]

[1] University of Patras, Rio, Patras, 26504, GREECE
komis@upatras.gr amisirli@upatras.gr

Abstract. Les dernières années, on constate une prolifération des robots avec des affordances et des fonctionnalités hétérogènes, permettant de diversifier les activités pédagogiques et d'envisager le développement de compétences variées. Cet article propose une analyse des affordances technologiques des robots destinés à l'école primaire à partir d'une taxonomie inspirée de la taxonomie de Catlin et al., en 2019. Cette taxonomie est validée par une revue systématique de l'offre des robots éducatifs et ludiques effectuée sur le site « amazon.com ». Nos résultats, portés sur cent (100) robots recensés, montrent une forte variabilité des affordances technologiques de ces robots par rapport aux trois types (kits de robotique, robots préconstruits et robots sociaux) et aux dix classes (robot kits, maker kits, robots modulaires, robots de sol, bras robotiques, walking robots, drones, jouets robots, humanoïdes, robots animaux) de notre taxonomie.

Keywords: Robotique éducative, Taxonomie des robots, Affordances.

1 Introduction

La robotique éducative, c'est-à-dire le domaine qui permet, entre autres, de construire un robot et le programmer à des fins pédagogiques, occupe depuis des années une place importante au sein de la recherche en éducation ainsi que dans la pratique éducative (Misirli & Komis, 2023). Pour certains, la robotique éducative est une approche pédagogique permettant de développer des compétences technologiques et numériques, la pensée informatique et de pratiquer la programmation informatique. En ce sens, étudier les différents aspects de la robotique éducative s'inscrirait dans la problématique développée au sein de la didactique de l'informatique. Pour d'autres, la robotique éducative est principalement une approche transversale pour favoriser le développement des connaissances dans plusieurs disciplines et des compétences de haut niveau. Apprendre la robotique ou apprendre par la robotique constitue donc deux volets complémentaires qui favorisent les usages des interfaces techniques, tels les robots, pour permettre aux apprenants de les manipuler et d'expérimenter, dans une approche d'apprentissage par investigation (Komis & Misirli, 2011 ; Theodoropoulou et al., 2023).

Dernièrement, les recherches en robotique éducative ont connu un nouvel essor, mais plusieurs questionnements subsistent. Un de plus intéressant consiste à identifier les conditions essentielles à une intégration efficace de la robotique éducative aux

Volume 1, pages 64–75.

curricula. Il s'agit d'une question globale qui comporte plusieurs sous-questions : à partir de quel niveau scolaire peut-on commencer ? Sous quelles formes institutionnelles et sous quelles modes disciplinaires (intégration aux disciplines existantes ou nouvelle spécialisation disciplinaire) ? Quels types d'instrumentation (logicielles, matérielle, dispositifs techniques, etc.). Quelles évaluations ?

La question de l'instrumentation matérielle s'avère très importante et doit être élucidée en amont, car les activités robotiques sont, dans leur majorité, des activités hautement instrumentées. La diversité des outils existants et la pluralité de leurs fonctionnalités doivent préoccuper les chercheurs du domaine. Sans une analyse approfondie des affordances technologiques des robots, il est difficile d'intégrer ces objets dans la prescription curriculaire et de proposer des activités pédagogiques adéquates (Komis et al. 2017). Dans ce contexte, cet article analyse les affordances technologiques des robots éducatifs à partir d'une taxonomie des robots adaptée pour l'école primaire.

2 Une taxonomie des robots éducatifs pour l'école primaire

Une taxonomie, en tant que pratique scientifique, est un système de classification, souvent en forme hiérarchique, dans laquelle les choses ou les concepts sont organisées en groupes. Elle se réfère également aux principes qui sous-tendent une telle classification. Catlin et al. (2019a) ont proposé une taxonomie des robots éducatifs dont les deux premiers niveaux comportent trois *types* et treize *classes* de robots (Figure 1).

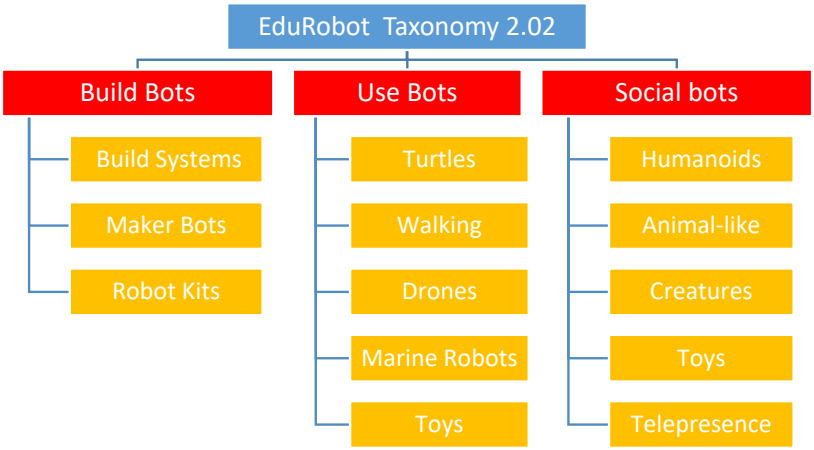


Fig. 1. La taxonomie Edurobot 2.02 (Catlin et al., 2019b).

Le *Type* (premier niveau de la taxonomie) concerne un groupe de robots éducatifs défini par rapport à son principal usage dans une situation pédagogique : construire un robot, programmer et utiliser un robot et interagir avec un robot autonome. La *Classe* (deuxième niveau de la taxonomie) concerne un ensemble clairement identifiable de robots

appartenant au même type (Catlin et al., 2019b) et partageant plusieurs principaux caractéristiques. Deux classes du même type se différencient sur un ou plusieurs caractéristiques. Par exemple, turtles et walking robots sont des robots prêts à utiliser mais la locomotion se fait dans le premier cas avec des roues et dans le second cas avec des pieds. Les différents *types* et les différentes *classes* des robots sont définis par la suite.

Nous avons adapté (Komis & Misirli, 2023) cette taxonomie à l'école primaire (Figure 2) en ajoutant un quatrième *type*, appelé « Robots virtuels », dans lequel nous avons intégré tous les outils de programmation informatique de type Logo, les microlangages et les simulateurs de robots. La robotique virtuelle est basée sur l'utilisation d'environnements virtuels pour la programmation des robots. Dans ce cas, il n'est pas nécessaire d'avoir le robot physique ou le kit de développement pour expérimenter. La programmation de robots virtuels est particulièrement utile pour le travail éducatif et pour l'apprentissage de la robotique à l'école primaire. Ce *type*, ajouté pour des raisons d'exhaustivité, n'est pas utilisé dans le travail en cours. Notre taxonomie comporte une *classe* supplémentaire dans les kits de construction (la classe « *Impression 3D* »). En revanche, nous n'avons pas gardé la classe « *Marine Robots* ». De plus, les classes « *Robots humanoïdes* » et « *Human-like* » ont été fusionnées. Cette taxonomie, comme nous le verrons par la suite, a été validée à partir des données recensées dans la base des données « amazon.com » de la société Amazon.

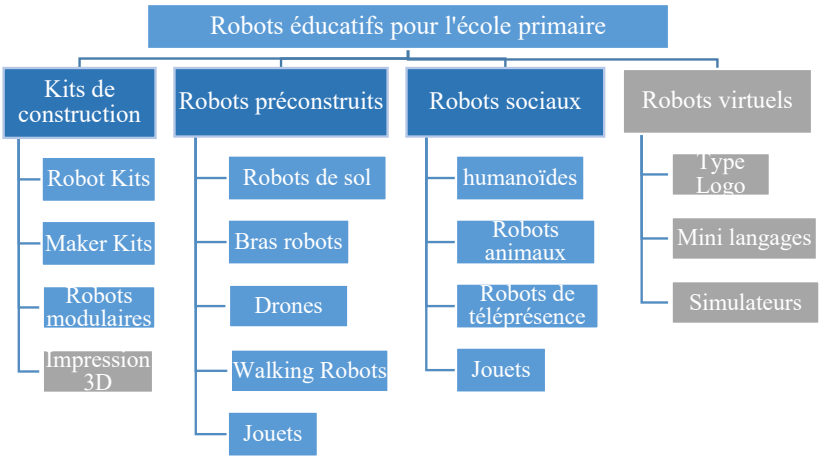


Fig. 2. La taxonomie « Robots éducatifs pour l'école primaire ». Les types et classes non considérés dans cette étude sont grisés.

2.1 Kits de construction

Les « kits de construction » sont des dispositifs flexibles permettant aux élèves de construire des robots en utilisant des composants de base. Ils comprennent une série de pièces mécaniques, de capteurs, d'actionneurs et de microcontrôleurs, ce qui permet aux élèves d'assembler des robots selon leurs préférences en matière de conception et

de fonctionnement. Le processus de construction de robots à partir de composants de base favorise l'apprentissage pratique, la résolution de problèmes et la créativité (Sullivan et al., 2013). En construisant des robots, les élèves acquièrent des connaissances en mécanique, en électronique et en codage. Les « kits de construction » de robots peuvent inclure des composants de base qui peuvent être assemblés pour créer diverses structures, divers mécanismes et systèmes, tels que *Lego mindstorms, Lego Wedo, GIGO - Kids First Coding & Robotics*, (Lee et al., 2013) et des composants modulaires qui créent des systèmes avec de multiples articulations et degrés de liberté, tels que *Cubelets*. Les kits de construction de robots incluent également la création des systèmes de contrôle, tels que *Micro:bit*, ou *Arduino* qui comprennent souvent des capteurs, des actionneurs et des microcontrôleurs qui permettent aux utilisateurs de construire des systèmes automatisés (Balogh, 2010). Enfin, dans les kits de construction, nous avons inclus l'« *Impression 3D* ».

2.2 Robots préconstruits

Les « robots préconstruits » ou robots prêts à utiliser (Catlin et al., 2019b) sont des systèmes robotiques entièrement assemblés et prêts à l'emploi avec lesquels les élèves peuvent interagir sans avoir à les construire. Ces robots sont dotés de capteurs et de systèmes de contrôle intégrés, ce qui constitue un point de départ facile pour les élèves qui ne sont pas forcément familiarisés avec les processus de fabrication techniques (Sullivan & Bers, 2016). Leur affordance de base et le déplacement dans l'espace à deux ou à trois dimensions qui est principalement géré par la programmation.

Les robots préconstruits, tels que *Roamer, Bee-bot, Blue-Bot, Probot, Thymio, Dash and Dot, EaRL Coding Robot, Ozobot, Botley, Edison,* trouvent des applications dans les contextes éducatifs d'initiation et d'approfondissement. Dans les cours d'introduction, ils servent d'outils attrayants pour initier les élèves aux concepts de programmation sans les submerger de complexités techniques. Dans des contextes plus avancés, les robots préconstruits peuvent être dotés de capteurs supplémentaires ou de comportements modifiés, ce qui permet aux étudiants d'explorer des concepts de programmation et de contrôle de plus haut niveau. Les robots préconstruits sont des systèmes robotiques mobiles (ou à composants mobiles, tels les bras robotiques) conçus pour naviguer et interagir avec l'environnement. Ils sont de conception très variée, allant des robots à roues et à chenilles aux modèles humanoïdes ou quadrupèdes et aux drones. Ces robots offrent aux élèves la possibilité de s'engager dans des applications robotiques du monde réel. En interagissant avec ces robots, les élèves explorent des concepts tels que la locomotion, la navigation, l'évitement d'obstacles et l'intégration de capteurs (Sullivan & Bers, 2016). Ces interactions renforcent leur compréhension de la mécanique, de l'électronique et de la programmation, tout en leur permettant d'expérimenter des opérations autonomes et télécommandées.

2.3 Robots sociaux

Les robots sociaux sont conçus pour interagir avec les humains sur le plan social et émotionnel. Ils sont également préconstruits mais leurs affordances sont très différentes par rapport aux robots du type précédent. Ces robots sont dotés de capacités telles que

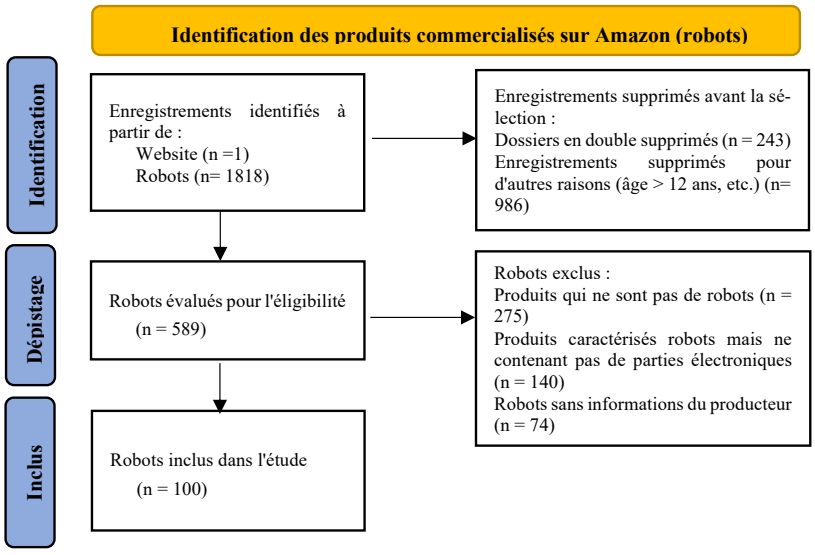
la reconnaissance vocale, le traitement du langage naturel et la reconnaissance faciale, ce qui leur permet de comprendre les émotions, les gestes et les modèles de discours humains et d'y répondre (Catlin et al, 2012).

Dans les contextes éducatifs, les robots sociaux contribuent au développement de la communication et des compétences interpersonnelles. Ils peuvent servir de compagnons interactifs d'apprentissage des langues, facilitant la pratique de la langue et la conversation. En outre, ils sont utilisés dans l'enseignement spécialisé pour aider les élèves à faire face aux défis sociaux et émotionnels en leur fournissant une plateforme sans jugement pour pratiquer les interactions sociales. Les robots sociaux humanoïdes sont des systèmes robotiques conçus pour reproduire les comportements, les interactions et l'apparence des humains. Ces robots présentent souvent des caractéristiques anthropomorphiques, telles que des visages et des gestes humains, comme *Pepper* et *Nao*. Dans les environnements éducatifs et sociaux, les robots sociaux humanoïdes servent de compagnons interactifs, engageant des conversations avec les utilisateurs, fournissant des informations et réagissant aux émotions.

3 Méthodologie

L'objectif de ce travail est l'analyse des affordances technologiques (Gaver, 1991; Norman, 1999) des robots « commerciaux » pour l'école primaire à partir d'une taxonomie comportant trois types et douze classes de robots (figure 2). Pour l'analyse, nous avons effectué une revue systématique, selon la méthode PRISMA (Page et al., 2021).

Table 1. Identification des robots commercialisés sur Amazon (méthode PRISMA).



Cette revue ne porte pas sur des articles, mais elle se réfère directement sur des produits commerciaux recensés sur le site principal de la société Amazon. Il s'agit donc d'une cartographie industrielle (industrial mapping) qui comporte les produits distribués à large échelle, sans tenir compte des producteurs indépendants des robots éducatifs qui ne commercialisent pas leurs produits à partir du site www.amazon.com. La requête a été effectuée du 03 au 19 septembre 2023 et la méthode est présentée dans la table 1.

Nous avons gardé pour l'analyse cent (100) robots « commerciaux ». Les mots-clés utilisés sont : « *educational coding robot , educational walking robot, educational robot arm, educational drone, educational robot kit, mechanic robotic kit, controller board kit, building block system, educational social robot, interactive robot, educational humanoid robot, interactive animal toy, interactive electronic robot* ».

4 Résultats

4.1 Description générale des robots

La figure 3, indique que la plupart des robots ne sont dans le marché que depuis peu années. Une partie significative (22 robots) n'a pas de date de commercialisation, mais il est quasi certain qu'il s'agit des produits commercialisés les dernières années.

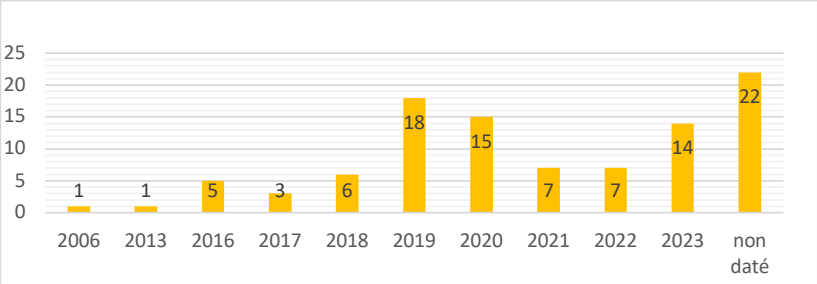


Fig. 3. Date de commercialisation des robots.

Le type le plus courant est celui du robot préconstruit (54 occurrences). En revanche, peu de robots (13 occurrences) sont des robots sociaux (Figure 4). Néanmoins, le tiers de robots (33 occurrences) concerne des kits de construction. Il s'agit d'un résultat intéressant : une partie significative des robots offre des affordances variées permettant d'envisager des activités multiples, tant au niveau de la construction (ce qui n'est pas possible avec les autres classes) qu'au niveau de la programmation.

L'analyse des données concernant les classes de robots montre une variabilité intéressante (Figure 4). Les « robots kits » est la modalité la plus courante (20 occurrences) suivie par la modalité « jouets » dans le type « robots préconstruits » (17 occurrences). Il est à noter que les classes « robots humanoïdes » et « robots de télé présence » n'ont pas été représentées dans les données recensées. Par la littérature, nous savons que ces deux classes de robots existent dans le primaire (par exemple le robot *Nao* est utilisé

dans des situations d'enseignement à distance pour des élèves ne pouvant être scolarisés totalement ou partiellement dans un établissement). Celles-ci sont commercialisées par des sociétés indépendantes, c'est pourquoi nous ne les avons pas détectées dans notre corpus d'analyse.

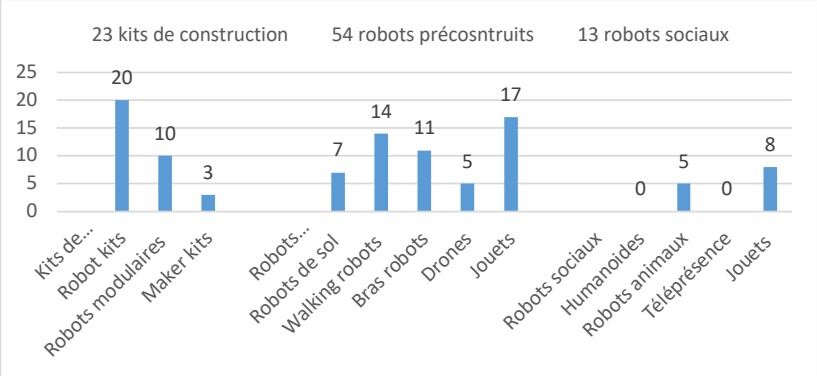


Fig. 4. Types et classes de robots.

4.2 Affordances technologiques

Le terme d'*affordance* est fréquemment utilisé dans le domaine du numérique pour concevoir des interfaces homme – machine ou pour savoir si ces technologies peuvent être utilisées pour faciliter des approches particulières de la pratique éducative (Conole et Dyke, 2004). Le terme est devenu courant dans la littérature à partir des travaux de Gibson qui lui associe « toutes les possibilités d'actions sur un objet » (Gibson, 1977). Selon Gibson, les affordances reflètent les relations possibles entre les acteurs et les objets : ce sont des propriétés du monde physique ou technique. Cette définition a ensuite été restreinte par Norman aux seules possibilités dont l'acteur est conscient. En effet, les affordances spécifient l'éventail des activités possibles, mais elles sont peu utiles si elles ne sont pas perceptibles pour les utilisateurs (Norman, 1999).

De son côté, Gaver définit les affordances en tant que propriétés du monde qui sont compatibles et pertinentes pour l'interaction des personnes. Lorsque les affordances sont perceptibles, elles offrent un lien entre la perception et l'action (Gaver, 1991). Notre approche s'inspire de la définition de Norman (2013), car nous considérons que les affordances des dispositifs robotiques en éducation ne sont pas toujours appréhendées par leurs utilisateurs (enseignants et élèves).

L'affordance qui concerne la *locomotion robotique* est étudiée à l'aide de sept (7) modalités proposées par Catlin et al. (2019b) pour la décrire. « Roues » est la modalité la plus courante, suivie par « Statique » et « Marcher ». Un résultat intéressant est que le tiers des robots étudiés est statique, c'est-à-dire n'effectue aucun mouvement.

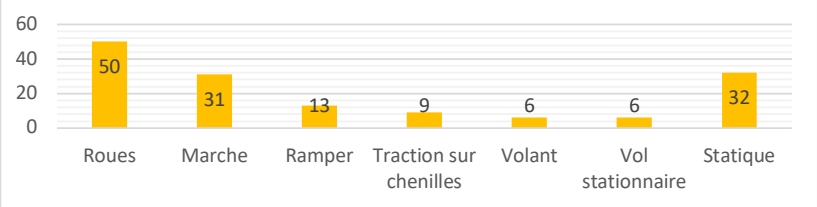


Fig. 5. Locomotion robotique.

La *communication* entre le robot et son environnement physique, technologique et humain se fait avec une multitude de dispositifs. Onze (11) modalités nous aident à étudier cette affordance (Figure 6). Les cinq premières concernent la communication entre le robot et l'élève : « haut-parleurs » (50 occurrences) et « clavier » (30 occurrences) sont les modalités les plus courantes de cette sous-catégorie, suivies par « Microphones », « caméras » et « écrans ». La communication entre le robot et son environnement physique se fait par des « capteurs » (41 occurrences) et la communication avec d'autres dispositifs technologiques est effectuée avec des « câbles », du « wi-fi », du « bluetooth » et de l'« infrarouge ». Certains robots disposent des fonctionnalités variées de communication.

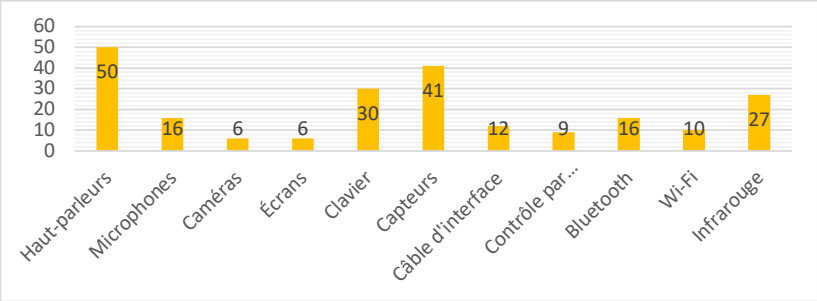


Fig. 6. Communication.

L'affordance *Commande et contrôle* (Figure 7) concerne les moyens avec lesquels l'élève interagit avec le robot. Neuf (9) modalités, proposées par Catlin et al. (2019b) sont utilisées pour étudier cette affordance. La « Télécommande » (33 occurrences) et la « Technologie mobile » (30 occurrences), à savoir des téléphones et des tablettes pour contrôler le robot, sont les modalités les plus courantes.

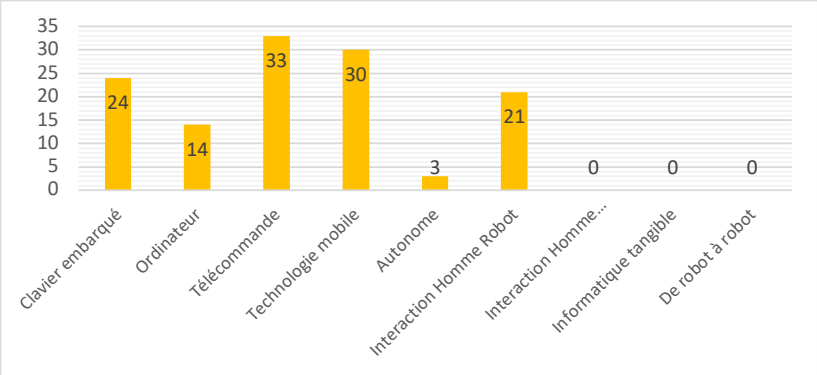


Fig. 7. Commande et contrôle.

Ensuite, on trouve le « Clavier embarqué » (24 occurrences) et l'«Interaction Homme - Robot » (interaction directe entre l'élève et le robot, comme avec des gants interactifs, console radio et joysticks) (21 occurrences). Une partie de robots (10 occurrences) permettent le contrôle à partir d'un « Ordinateur ». Trois robots sont « Autonomes », c'est-à-dire qu'ils réagissent à leur environnement que les élèves peuvent manipuler. Nous n'avons pas trouvé de robots permettant une « Interaction homme-ordinateur » (l'ordinateur sert de médiateur à l'interaction entre l'élève et le robot, par exemple, contrôle de l'écran tactile), une « Informatique tangible » (c'est-à-dire une utilisation d'objets tangibles pour contrôler le robot) et une interaction « Robot à robot » (l'utilisation de robots pour contrôler d'autres robots).

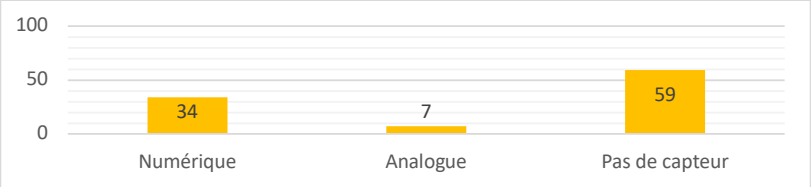


Fig. 8. Capteurs.

Les *capteurs* d'un robot sont les dispositifs qui le renseignent sur son environnement. Nous avons utilisé trois modalités caractérisant cette affordance (Figure 8) : capteurs numériques, analogiques et inexistence de capteurs. Les capteurs numériques se déclenchent à des valeurs spécifiques et signalent l'événement au robot, par exemple les capteurs tactiles et capteurs de proximité numériques (Catlin et al., 2019b). Plus d'un tiers de robots analysés dispose des capteurs numériques. Les capteurs analogiques transmettent au robot des informations complexes, comme des caméras et des boussoles électroniques (Catlin et al., 2019b). Peu de robots (sept sur cent analysés) disposent de ce type de capteur. Il est à noter qu'une majorité de robots (57 occurrences) ne dispose

pas de capteur. Ces robots ne perçoivent pas leur environnement. Des robots très utilisés aux écoles, comme *BeeBot* ou *Valiant T*urtle, ne disposent pas de capteur. Pour Catlin et collaborateurs (Catlin et al., 2019a) le manque de cette affordance n'est pas très important : « *Certaines personnes ne croient pas qu'une machine puisse être qualifiée de robot si elle n'est pas équipée de capteurs. Cela ne s'applique pas aux robots éducatifs. Si vous considérez l'ensemble du système de commande, il inclut l'élève qui observe ce que fait le robot et modifie son programme si nécessaire.* »

5 Discussion – Conclusion

Une affordance est une relation entre les propriétés d'un objet et les capacités de l'agent qui détermine comment l'objet peut être utilisé (Norman, 2013). Dans ce cadre, les affordances définissent les actions possibles d'un agent sur son environnement naturel et technologique. Comme le souligne Gaver, explorer les affordances permet de se concentrer sur les forces et les faiblesses des technologies en ce qui concerne les possibilités qu'elles offrent aux personnes susceptibles de les utiliser (Gaver, 1991). L'étude des affordances, par conséquent, est très importante dans le champ des technologies numériques en éducation et en didactique de l'informatique (Komis et al., 2021). Plus précisément, la connaissance des affordances des robots éducatifs par les enseignants s'avère nécessaire pour mieux intégrer ces dispositifs dans des situations éducatives (Kalmpourtztis et Romero, 2022). Ce travail s'inscrit dans cette perspective.

L'objectif de cet article est l'étude des affordances technologiques des robots éducatifs et ludiques pour l'école primaire à partir d'une taxonomie de ces robots (Komis et Misirli, 2023). Un prolongement de cette étude nous permettrait, dans un deuxième temps, d'étudier les affordances pédagogiques et sociales des robots éducatifs dans le but de proposer des interventions éducatives adéquates (Komis et al., 2017). Les affordances étudiées dans le présent travail, au sens des affordances réelles et non des affordances perçues (Norman, 1999), sont assez proches des fonctionnalités des robots examinés. Il est à noter que ces deux concepts (affordances réelles et fonctionnalités) sont étroitement liés, mais ont des significations distinctes. Par exemple, au niveau des fonctionnalités, la locomotion serait plutôt décrite à partir des moteurs et d'autres caractéristiques (actionneurs) qui sont internes au système robotique.

Pour l'analyse des affordances des robots éducatifs nous avons effectué une revue systématique, selon la méthode PRISMA (Page et al., 2021). Nos résultats, portant sur cent (100) robots éducatifs et ludiques recensés sur le site www.amazon.com, d'une part, fournissent des éléments significatifs pour valider la taxonomie proposée (Komis et Misirli, 2023) et d'autre part, montrent une forte variabilité des affordances technologiques de ces robots. Cette variabilité concerne tant les trois types (kits de robotique, robots préconstruits et robots sociaux) que les dix classes (robot kits, maker kits, robots modulaires, robots de sol, bras robotiques, walking robots, drones, robots jouets, humanoïdes, robots animaux) de taxonomie de Komis et Misirli (Komis et Misirli, 2023). En ce qui concerne les types de robots, la plupart sont des robots préconstruits, un tiers sont de kits de construction et peu de robots sont des robots sociaux. Les classes les plus courantes de robots sont les jouets et les kits de construction. Nous n'avons pas

trouvé de robots humanoïdes et de robots de téléprésence. Ce manque est probablement lié à l'âge des élèves (un de critères de sélection était l'âge inférieur à douze ans).

La locomotion robotique se fait par des roues et en marche tandis qu'une partie significative de robots ne sont pas capables de mouvement (robots statiques). Le contrôle et la commande sont principalement effectués via des claviers intégrés, par télécommande ou par technologies mobiles. Une partie essentielle de robots ne dispose pas de capteurs. L'absence de cette affordance signifie que les élèves qui programment le robot doivent simuler les capteurs pour que le robot interagisse à son environnement. Enfin, la communication entre le robot et son environnement physique, technologique et humain est variée et multiple.

Notre étude est effectuée sur la base de données du site amazon.com et par conséquent nous n'avons pas pu recenser de robots produits par plusieurs constructeurs indépendants. Cette démarche limite la portée de nos résultats, c'est pourquoi il est nécessaire d'élargir la cartographie industrielle en utilisant des sources variées.

References

Balogh, R. Educational Robotic Platform based on Arduino (2010)

Catlin, D., Kandlhofer, M., Cabibihan, JJ., Angel-Fernandez, J., Holmquist, S., Csizmadia, A.P. EduRobot Taxonomy. In: Daniela, L. (eds) Smart Learning with Educational Robotics. Springer, Cham (2019a)

Catlin, D., Kandlhofer, M., Holmquist, S., Csizmadia, A. P., Angel-Fernandez, J., & Cabibihan, J. J. Robots for Education: Online EduRobot Taxonomy (2019b), last accessed 2023/10/15

Conole, G. & Dyke, M. What are the affordances of information and communication technologies?, ALT-J, **12**(2), 113-124 (2004)

Gaver, W.W. Technology affordances. In Proceedings of CHI 91, pp. 79-84. New York: ACM (1991)

Kalmpourtzis, G. & Romero, M. An affordance-based framework for the design and analysis of learning activities in playful educational robotics contexts. Interactive Learning Environments (2022)

Komis, V. & Misirli, A. Une classification des outils robotiques pour l'école primaire, In éTIC 5 Colloque international : « École et TIC» on Proceedings. Caen, France (2023)

Komis, V. & Misirli, A. Robotique pédagogique et concepts préliminaires de la programmation à l'école maternelle : Une étude de cas basée sur le jouet programmable Bee-Bot. In: Proceedings DIDAPRO 4, pp. 271–284. New Technologies Editions, Greece (2011)

Komis, V., Karachristos, C., Mourta, D., Sgoura, K., Misirli, A., Jaillet, A. Smart Toys in Early Childhood and Primary Education: A Systematic Review of Technological and Educational Affordances. Applied Sciences, **11(**18), 8653 (2021)

Komis, V., Romero, M., Misirli, A. A Scenario-Based Approach for Designing Educational Robotics Activities for Co-creative Problem Solving. In: Alimisis, D., Moro, M., Menegatti, E. (eds) Educational Robotics in the Makers Era. Edurobotics 2016 2016. Advances in Intelligent Systems and Computing, vol 560. Springer, Cham. (2017)

Lee, J.J., Knox, B., Breazeal, C. Modeling the dynamics of nonverbal behavior on interpersonal trust for human-robot interactions. In Proceedings Spring Symposium Association Advances in Artificial Intelligence, pp. 46–47. Palo Alto, CA: AAAI 2013

Misirli, A., & Komis, V. Robotics and programming concepts in early childhood education: A conceptual framework for designing educational scenarios. In C. Karagiannidis, P. Politis, & I. Karasavvidis (Eds.), Research on e-learning and ICT in education technological, pedagogical and instructional perspectives. New York: Springer Science+Business Media. (2015)

Misirli, A. & Komis, V. Computational Thinking in early childhood education: The impact of programming a tangible robot on developing debugging knowledge. Special Issue: 'Computational Thinking in Early Childhood', Early Childhood Research Quarterly, **65**, 4th Quarter 2023, 139-158 (2023)
Norman, D.A. Affordances, conventions and design. Interactions **6**(3), 38 – 43 (1999)
Norman, D.A. The Design of Everyday Things. Basic Books, (2013)
Page, M.J., Moher, D., Bossuyt P.M., Boutron I., Hoffmann, T. C., Mulrow, C. D., et al. PRISMA 2020 explanation and elaboration: updated guidance and exemplars for reporting systematic reviews. The British Medical Journal, **372** (160) (2021)
Sullivan, A. & Bers, M. U. Girls, boys, and bots: Gender differences in young children's performance on robotics and programming tasks. Journal of Information Technology Education: Innovations in Practice, **15**, 145-165 (2016)
Sullivan, A., Kazakoff, E.R., & Bers, M.U. The Wheels on the Bot Go Round and Round: Robotics Curriculum in Pre-Kindergarten. Journal of Information Technology Education: Innovations in Practice, **12**, 203-219 (2013)
Theodoropoulou, I., Lavidas, K. & Komis, V. Results and prospects from the utilization of Educational Robotics in Greek Schools. Technology, Knowledge and Learning **28**, 225–240 (2023)

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Une séquence d'informatique débranché comportant une dimension recherche pour l'apprentissage des algorithmes de tri *

Maryna Rafalska [1] et Patrick Gibel [2]

[1] LINE, Université Côte d'Azur, France
maryna.rafalska@univ-cotedazur.fr
[2] LAB-E3D, Université de Bordeaux, France
patrick.gibel@u-bordeaux.fr

Résumé. Dans ce papier, nous présentons une séquence d'informatique débranché qui vise la découverte par les élèves de la spécialité Numérique et Sciences Informatiques (NSI) des algorithmes de tri ; ces derniers occupent une place importante dans les programmes. La recherche actuelle met en lumière et étaye les potentialités didactiques de l'activité « Tri de cartes » au lycée ce qui permet de prolonger et d'adapter les études antérieures menées à l'école primaire. En outre, dans ce travail nous étudions la possibilité du passage du développement des méthodes générales de rangement par les élèves à l'élaboration et l'analyse des algorithmes de tri associés du point de vue de leur complexité ainsi que les raisonnements des élèves associés. Pour cela nous faisons la référence à la Théorie des Situations didactiques comme cadre théorique et l'ingénierie didactique comme méthodologie de recherche. Cet écrit vise à présenter les premiers résultats de l'analyse des expérimentations de la séquence dans plusieurs classes de lycée français en l'illustrant par trois cas le travail des élèves sur le calcul du nombre minimal de coups nécessaires pour effectuer le rangement en fonction du nombre d'objets.

Mots clés : informatique débranché, algorithmes de tri, jeu, raisonnement, didactique, lycée.

1 Introduction

La spécialité Numérique et Sciences Informatiques (NSI) a été introduite au lycée en France en 2019 en classe de Première et Terminale. Ce nouvel enseignement pose plusieurs questions d'ordre épistémologique, didactique et pédagogique. Une des questions qui émerge concerne les activités et les ressources favorisant l'apprentissage des concepts du programme. Dans ce contexte, ce travail cherche à apporter des solutions

* Cette recherche a été partiellement soutenue par le projet ASMODEE, ANR-21-SSMS-0001.

concrètes pour le terrain et la formation des enseignants en se basant sur les résultats de recherche.

Un des thèmes proposés dans les programmes 2019 de NSI concerne les algorithmes de tri qui comprend des algorithmes “classiques” (comme, par exemple, le tri par insertion ou le tri par sélection) ainsi que leurs coûts dans le pire cas et la preuve de la correction et de la terminaison. En même temps, les observations menées au lycée montrent que l’enseignement de ces algorithmes est réalisé généralement par la présentation des étapes de chaque algorithme de tri par le professeur, suivi par leur programmation. Ainsi, il existe la possibilité d’une acquisition de ces algorithmes uniquement d’un point de vue « purement technique » sans que les élèves n’accèdent aux raisons de savoir et à l’analyse approfondie des méthodes de tri. De plus, cette approche peut amener à une représentation des classes de NSI comme un espace réservé à la programmation [7]. Dans cet écrit nous proposons une méthode alternative pour introduire des algorithmes de tri auprès des élèves en utilisant une séquence d’informatique débranché comportant une dimension recherche. L’hypothèse à la base de ce choix est que l’utilisation de ce type de situation permet de restituer le sens des concepts en jeu et d’engager l’ensemble des élèves dans une activité algorithmique riche du point de vue de sa nature ce qui, en conséquence, va favoriser leurs apprentissages.

Dans ce travail, nous nous appuierons sur les recherches antérieures [10] qui ont permis d’établir la potentialité de la situation « Tri de cartes » pour faire découvrir des procédures assimilables à des algorithmes de tri à l’école primaire. Nous avons un objectif double. Premièrement, mettre les potentialités didactiques de cette situation à l’épreuve au lycée. Deuxièmement, étudier la possibilité du passage du développement des méthodes générales de rangement par les élèves à l’analyse des algorithmes de tri associés (notamment, du point de vue de leur complexité) ainsi que les raisonnements des élèves associés.

Nous commençons par la présentation des aspects théoriques sur lesquels se base notre recherche et la méthodologie utilisée. Ensuite, nous présenterons la situation évoquée ainsi que son analyse a priori. Nous poursuivrons par la description des informations concernant des expérimentations réalisées et présenterons quelques résultats. Enfin, nous finissons par les conclusions et des perspectives du travail.

2 Cadre théorique et méthodologie

Dans cette étude nous nous basons sur la Théorie des situations didactiques (TSD) développée par Brousseau [4] et les travaux ultérieurs [5, 6]. La séquence a été conçue par les auteurs comme une séquence comportant une dimension recherche dans le sens où elle inclut une phase de recherche autonome par les élèves de la solution d’un problème, suivi par une présentation des solutions devant d’autres élèves ainsi que devant le professeur et se concluant par une institutionnalisation des savoirs par le professeur [3]. Le problème « Tri de cartes » proposé aux élèves (dont la description est donnée dans la section suivante) est basé sur le problème de tri d’une liste non vide. Ce dernier est un problème fondamental pour l’algorithme selon les critères correspondants pro-

posés par Modeste [8] et, ainsi, peut être utilisé pour la construction de situations d'enseignement et d'apprentissage (ce qui justifie notre choix). A partir du problème général « Tri de carte » et grâce au choix des valeurs des variables didactiques nous proposons une suite de jeux successifs qui vise à amener les élèves à la construction d'algorithmes de tri et à leur analyse. Plus précisément, le choix de variables didactiques a été fait de telle manière qu'au fur et au mesure que les élèves avancent dans les jeux, ils peuvent d'abord construire des suites d'actions, puis des procédures (suite d'actions dont les élèves peuvent rendre compte) et éventuellement des méthodes générales (programmes d'actions finalisés indépendants des conditions). Du point de vue algorithmique, au fil des jeux, les élèves passent des algorithmes instanciés (permettant effectuer le rangement uniquement pour certains placements initiaux) aux algorithmes de tri génériques (qui fonctionnent pour n'importe quel placement des cartes au départ). Il est notamment attendu que les élèves puissent retrouver les algorithmes de tri « classiques » comme, par exemple, le tri à bulle, le tri par sélection, le tri par insertion, etc.

Les jeux successifs ont été conçus comme des situations adidactiques d'action dans lesquelles l'élève au travers des interactions avec le milieu construit des connaissances implicites. En plus de ces situations d'action, la phase de recherche conduit à une phase de formulation des stratégies développées par les élèves dans laquelle l'élève transmet sous une forme quelconque à un autre sujet les connaissances en question (ce qui les rends explicites) et à la situation de validation dans laquelle les élèves établissent la validité de ces connaissances.

La situation « Tri de cartes » s'inscrit dans l'approche débranchée de l'enseignement de l'informatique (connue sous le nom de CS Unplugged) qui vise à initier les apprenants aux concepts clés de cette discipline sans utiliser les outils numériques [2]. La situation élaborée reprend la plupart des caractéristiques des activités débranchées proposées par [9], notamment par l'utilisation d'objets tangibles plutôt qu'un ordinateur et par l'engagement des élèves dans une activité basée sur les jeux.

Notre méthode est celle de l'ingénierie didactique qui comprend des étapes suivantes : les analyses préalables, la conception et l'analyse a priori des situations didactiques de l'ingénierie, l'expérimentation et enfin la phase de l'analyse a posteriori et de l'évaluation [1]. La confrontation de l'analyse a priori et l'analyse a posteriori vise à valider des hypothèses engagées dans la recherche ou proposer des modifications de l'ingénierie. Nous présentons les éléments de ces analyses dans les sections suivantes.

3 Présentation du problème « Tri de carte » et des jeux

Le problème « Tri de cartes » est formulé de la façon suivante :

Étant donné m grilles contenant n cartes faces cachées choisies au hasard parmi N cartes (n<N), trouver une(des) méthode(s) qui permet(tent) de trier simultanément les cartes de chacune des grilles en utilisant (au plus) k coups. Un coup correspond à l'enchaînement des trois opérations de base (« retourner deux cartes », « les comparer et les mettre dans l'ordre », « reposer les cartes, face cachée, sur les cases vides »).

Comme cela a été déjà indiqué dans la section précédente, les choix des valeurs des variables didactiques *n*, *N*, *m*, *k* génère la suite des jeux successifs suivants.

« Battre les 13 cartes de la même couleur. Choisir au hasard 10 cartes parmi ces 13 cartes et les placer face cachée sur une grille avec 10 cases (Fig.1). Dans chaque jeu ci-dessous trier les cartes en utilisant uniquement les trois opérations suivantes :

- retourner deux cartes ;
- comparer les deux cartes et les mettre dans l'ordre ;
- reposer les cartes, face cachée, sur les cases vides.

Jeu 1 (individuel). Effectuer les actions que vous pensez nécessaires pour trier les cartes. Le nombre de coups n'est pas limité. La partie est gagnée si les cartes sont triées.

Jeu 2 (en binôme). Donner les instructions, correspondant aux trois opérations autorisées, à votre camarade qui ne peut que formellement exécuter les commandes sans vous montrer les cartes retournées. Quand vous pensez que les cartes sont triées, dites « stop ». Retournez les cartes pour vérifier. La partie est gagnée si les cartes sont triées.

Jeu 3 (en groupe). Donner les instructions, correspondant aux trois opérations autorisées, à trois (ou plus) camarades qui n'exécutent que formellement les commandes en même temps sans vous montrer les cartes retournées. Quand vous pensez que les cartes sont triées, dites « stop ». Retournez les cartes sur toutes les grilles pour vérifier. La partie est gagnée si les cartes de tous les camarades sont triées.

Jeu 4 (en groupe ou en classe entière). Donner les instructions pour trier les cartes de vos camarades en même temps selon des règles du jeu 3 avec le moins de comparaisons possibles. Quand vous pensez que les cartes sont triées, dites « stop ». La partie est gagnée si les cartes de tous sont triées en utilisant moins de coups que les autres joueurs. »

Fig. 1.

3.1 Analyse a priori du point de vue des stratégies susceptibles à apparaître dans les jeux

Dans le jeu 1, il n'est pas attendu l'apparition de stratégies génériques mais plutôt que les élèves élaborent des suites d'actions qui se reposent en grande partie sur la mémorisation des valeurs des cartes et leurs positions afin d'effectuer le tri. Ainsi, le but de ce jeu est plutôt l'appropriation des règles du jeu par les élèves. En particulier, ils doivent comprendre les trois opérations autorisées et « apercevoir » la possibilité de gagner

dans le jeu (le fait qu'il existe une suite d'actions (de permutations) qui permette à partir d'une suite donnée des cartes d'obtenir une suite triée).

Dans le jeu 2, le joueur formule la suite d'actions en désignant les cartes sur lesquelles opérer, il ne voit plus les cartes, mais il peut observer si les cartes ont été échangées ou non après l'observation de son camarade qui effectue les actions dictées. Il lui est toutefois difficile de mémoriser les positions des cartes échangées, ce qui incite à chercher des stratégies plus efficaces. Des stratégies très variées sont susceptibles d'apparaître. On peut, notamment, envisager plusieurs stratégies basant sur l'information de la présence ou de l'absence d'un échange de cartes lors d'une comparaison. Cette information peut être utilisée dans deux buts : pour savoir quand il faut dire « stop » (stratégies 1, 2 et 3 ci-dessous) ou pour déterminer dans la suite d'actions le choix suivant de cartes à comparer (stratégie 4 ci-dessous). L'élève, par exemple, peut indiquer à chaque fois au camarade les cartes choisies par hasard pour qu'il effectue les opérations évoquées et dire « stop » quand il remarque qu'au bout d'un moment il n'y a plus d'échanges effectués (stratégie 1). Sinon, l'élève qui joue peut mettre en place un procédé systématique de comparaison des cartes qui permettra de contrôler les échanges faits. Par exemple, il peut demander au camarade de comparer les cartes consécutives deux à deux en partant de la première position vers la dernière et en répétant le procédé jusqu'à ce qu'il n'y ait plus d'échanges (stratégie 2). Il est possible aussi que l'élève puisse demander d'effectuer les comparaisons des cartes consécutives deux à deux en alternant le sens (par exemple, de la première position vers la dernière puis de la dernière vers la première) jusqu'à ce qu'il n'y ait plus d'échanges (stratégie 3). Une autre stratégie qui peut apparaître consiste à création d'un sous-ensemble des cartes avec l'ordre local installé (du nombre 1 pour le début du procédé) qui à chaque étape augmente par l'insertion d'une carte de la partie non triée (stratégie 4). Selon cette stratégie, l'élève commence par comparer les cartes consécutives deux à deux en partant de la première position vers la dernière jusqu'au premier échange des cartes ou jusqu'à la dernière position (dans le cas où les cartes sont déjà triées). Dans le premier cas, il poursuit par le changement de sens de comparaison en demandant à comparer les cartes consécutives deux par deux en suivant le sens inverse jusqu'à ce qu'il n'y ait plus d'échange de cartes. Puis, il reprend la comparaison à partir de la carte où le changement de sens avait été effectué.

Parmi des stratégies où l'information sur l'échange n'est pas utilisée, on peut envisager la suivante. L'élève demande à comparer d'abord la première carte à toutes les autres cartes, ensuite la deuxième avec les autres, etc. (stratégie 5). Cela conduira à placer d'abord la plus petite carte à la première position, ensuite la « bonne » carte sur la deuxième position, etc. Les élèves peuvent aussi mettre en place cette stratégie en partant de la dernière carte vers la carte qui se trouve à la première position. Il est aussi possible que l'élève, à cette étape, mette en place une stratégie qui nécessite moins de comparaisons de cartes (cf. stratégie 6 ci-dessous).

Dans le jeu 3, les élèves n'ont plus la possibilité de contrôler les échanges des cartes car ils donnent les instructions à plusieurs exécuteurs en même temps dont les suites initiales de cartes sur les grilles ne sont pas forcément les mêmes. Ainsi, ceux qui ont élaboré les stratégies qui sont basées sur l'information de la présence ou de l'absence

d'échanges de cartes lors d'une comparaison sont amenés à les faire évoluer ou à chercher d'autres stratégies à employer pour gagner dans ce jeu. Par exemple, la stratégie 1 disparaîtra alors que les stratégies 2, 3 et 4 pourront évoluer en prenant en compte les nouvelles contraintes (à la condition que l'élève analyse les relations qui sont impliquées dans les stratégies).

Le défi auquel l'élève est confronté dans le jeu 3 diffère selon la stratégie élaborée. Notamment, les élèves qui ont élaboré, à l'issue du jeu 2, les stratégies 2 et 3 seront confrontés dans le jeu 3 à la question du moment de l'arrêt du processus de comparaison des cartes. Pour répondre à cette question, l'élève doit comprendre que le processus de comparaison selon la stratégie 2 conduira à placer d'abord la carte avec la plus grande valeur à la dernière position, ensuite la seconde plus grande carte sur l'avant-dernière position et ainsi de suite jusqu'à ce que la plus petite carte soit placée sur la première position.[3]. D'après la stratégie 3, les cartes sont progressivement rangées des extrémités de la suite vers la ou les cartes placées au milieu de la grille. De ce fait, la partie non-triée de la suite diminue à chaque passage jusqu'à ce qu'il ne reste qu'une paire des cartes à comparer ce qui indiquera le moment pour dire « stop »[4].

L'élève qui a mis en place dans le jeu 2 la stratégie 4, afin de gagner dans le jeu 3 doit envisager le pire cas de placement des cartes (quand la carte avec la plus petite valeur est placée à la dernière position). Cela conduira à l'évolution de la stratégie, ce qui implique que la comparaison des cartes dans le sens inverse à chaque fois doit être faite jusqu'à la première position[5].

Le jeu 4 pose la question du nombre minimal de coups nécessaires pour trier les cartes à l'aide de la stratégie trouvée. Cela exigera un travail d'analyse plus profond de la stratégie élaborée. A la fin de cette phase, il est attendu que les stratégies non optimales issues des jeux précédents évoluent vers des stratégies qui sont moins coûteuses du point de vue du nombre de coups à effectuer pour trier la suite de cartes. Par exemple, la stratégie 5 peut être modifiée en demandant de comparer chaque carte (en partant de la carte placée à la première position) avec les autres cartes dont le numéro de place est supérieur à celle concernée (stratégie 6)[6]. Cela permet de placer toutes les cartes progressivement à leurs places définitives en utilisant moins de coups que dans la stratégie 5. Il est aussi envisageable que les élèves essaient de remplacer les stratégies trouvées par d'autres stratégies, plus efficaces en termes de nombre de coups.

4 Expérimentations et la récolte des données

Les expérimentations ont eu lieu pendant les années 2022 et 2023 en spécialité NSI au lycée Saint John Perse (Pau) en classe de Première (16 élèves) et au lycée de Nay en classe de Première (12 élèves) et de Terminale (10 élèves). Le nombre de séances et le temps consacrés à chaque phase a été laissé au libre choix de l'enseignant. Les deux

[3] Il s'agit de l'algorithme de tri à bulles.
[4] Il s'agit de l'algorithme de tri cocktail.
[5] Il s'agit de l'algorithme de tri par insertion.
[6] Il s'agit de l'algorithme de tri par sélection.

enseignants impliqués dans les expérimentations sont des professeurs de mathématiques agrégés qui interviennent également dans la spécialité NSI dans leurs lycées. Une telle prise en charge de la spécialité NSI par les professeurs de mathématiques est assez courante en France. Même si un CAPES[7] informatique a été créé, le nombre de postes reste encore limité.

Pour la classe de Première, il s'agissait d'une première « rencontre » avec les algorithmes de tri alors que pour la Terminale une redécouverte (les algorithmes de tri ayant été introduits pour ces élèves en classe de Première). La séquence proposée commençait par les quatre jeux évoqués (ce qui a pris une ou deux séances selon la classe), puis les élèves ont travaillé sur le problème général formulé de la façon suivante : « Trouver une/des méthode/s pour trier n cartes ($n \in \mathbb{N}$) pour p élèves ($p \in \mathbb{N}$) en même temps dans le moins de coups possibles (un coup, c'est en enchaînant les trois opérations de base) ». Par la suite, trois questions ont été proposées aux élèves. Les deux premières portaient sur la preuve des méthodes trouvées (correction et terminaison) alors que la troisième concernait le nombre de coups minimal nécessaires pour garantir un rangement dans l'ordre croissant en fonction du nombre n de cartes qu'on possède au départ. Une séance a été consacrée à la présentation par les élèves devant la classe des méthodes trouvées ainsi que les résultats de leur analyse. Dans les classes de Première, les élèves ont également eu une séance consacrée à la programmation des algorithmes trouvés et une qui était dédiée au bilan.

Le corpus de ce travail est constitué par les données audiovisuelles des échanges collectifs (prises par la vue de la classe entière) et du travail des élèves au sein des groupes ainsi que par les productions des élèves. Ces dernières sont représentées par les fiches d'élèves que les élèves ont dû remplir après chaque jeu ainsi qu'à l'issue du travail sur le problème général et les trois questions évoquées précédemment. Les élèves étaient libres dans le choix de la représentation des méthodes élaborées (ce qui a été annoncé explicitement par les professeurs). Le matériel a été à disposition des élèves pendant toutes les séances débranchées y compris lors du travail sur le problème général.

5 Les éléments d'analyse et premiers résultats

Dans ce papier nous ne présentons qu'une partie des résultats qui concerne plutôt la production des algorithmes par les élèves et l'analyse de leurs complexités.

L'analyse des données nous a permis de faire plusieurs constats. Premièrement, les élèves rentrent assez facilement dans les jeux proposés. Ces derniers fonctionnent comme des situations adidactiques au sens de la TSD conduisant les élèves à la recherche et la rédaction des méthodes générales sans interventions directes de l'enseignant. Pour les stratégies apparues lors des situations d'actions, le déroulement effectif confirme l'analyse faite a priori. En effet, pour gagner au jeu 1 les élèves se sont basés principalement sur leurs mémoires alors que dans le jeu 2 la saturation de la mémoire

[7] Le certificat d'aptitude au professorat de l'enseignement du second degré est un concours professionnel du ministère français de l'Education national, de l'Enseignement supérieur et de la Recherche.

les a amenés à la recherche des stratégies plus efficaces. Ces dernières se sont modifiées ou ont été remplacées par les autres stratégies dans le jeu 3. A la fin du jeu 4, dans la majorité de groupes de chaque classe, nous avons pu constater l'émergence des algorithmes de tri tels que le tri à bulle, les tris par insertion et par sélection, le tri pair/impair ainsi que le tri cocktail.

Un constat surprenant pour la classe de Terminale est que les élèves qui sont censés avoir un répertoire didactique [6] concernant les algorithmes de tri depuis de la classe de Première, ne se sont pas rendu compte que la situation proposée est celle où ces connaissances peuvent être utilisées. Ils ont redécouvert les algorithmes à nouveau. Cela renforce l'hypothèse que l'enseignement de nature ostensive des algorithmes de tri (ce qui a été fait dans ce classe) ne permet pas l'acquisition suffisante de ces derniers.

Nous pouvons aussi constater quelques difficultés des élèves, notamment celle de formuler les méthodes trouvées. Dans les productions des élèves, nous retrouvons une variété de représentations, notamment celles où les élèves font recours au langage naturel ou de programmation ou encore utilisent des schémas pour illustrer leurs stratégies. Un exemple d'utilisation d'un langage qui peut s'apparenter à du pseudo-code (combinaison de langage courant et de langage de programmation) utilisé par un élève de la classe de Première est montré sur la Fig. 2. Dans cet exemple, nous constatons que dès le jeu 3 et même dans un contexte débranché, l'élève se projette dans la programmation de l'algorithme trouvé. Cela peut être expliqué d'une part par l'existence d'un contrat implicite selon lequel le travail dans une classe de NSI a pour finalité la programmation, et d'autre part la facilité que cet élève a trouvé dans ce mode de représentation pour décrire de manière générale la méthode trouvée par rapport aux autres registres sémiotiques.

Nous avons pu aussi constater que la démarche des élèves dans la (re)découverte des algorithmes de tri varie (en fonction des connaissances mobilisées ou pas, l'interprétation des rétroactions du milieu, la manière de traiter le singulier, la méthode en cours du développement, etc.). Par exemple, dès la première partie du jeu 2, la plupart des élèves dans leurs décisions concernant le choix des cartes à comparer utilisaient les connaissances sur les propriétés de la relation d'ordre alors que les autres ont eu besoin de rejouer plusieurs fois (en répétant le cycle « action-rétroaction-décision » lors de la situation d'action) pour arriver au constat que certains coups sont nécessaires ou, au contraire, inutiles. Un autre exemple concerne la recherche du nombre de coups pour trier les cartes quel que soit le nombre de cartes au départ. Certains élèves ont procédé par les jeux avec les cartes ouvertes afin ensuite généraliser la réponse alors que les autres étaient capables de raisonner dans le cas général sans recours aux exemples particuliers. En outre, les raisonnements mis en place par les élèves dans cette question semblent être de nature différente. Ci-dessous nous présentons les trois cas pour illustrer ce propos.

Cas 1. L'élève 1 à découvert le tri par sélection qu'il décrit sous forme d'un programme avec deux boucles imbriquées (Fig. 2). Il calcule le nombre de coups pour trier n cartes en calculant le nombre d'itérations dans chaque boucle. Ainsi, il écrit la formule avec une double sommation présentée sur la Fig. 3, et en simplifiant celle-ci, il obtient la somme de n-k pour k de 1 à n. Comme nous le voyons sur la fiche d'élève et dans l'extrait de vidéo, l'élève ne possède pas les connaissances mathématiques pour aller

plus loin dans ses réflexions et donner le résultat dans une forme n'utilisant pas le symbole de somme. En même temps, le changement entre deux cadres (informatique et mathématiques) ne semble pas être difficile pour cet élève. En particulier, comme on peut le remarquer dans la production écrite, l'élève utilise la numération commençant par 0 dans le cas du programme et par 1 dans la somme.

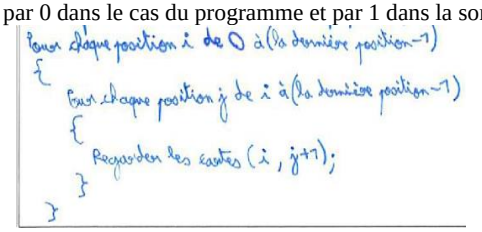


Fig. 2.

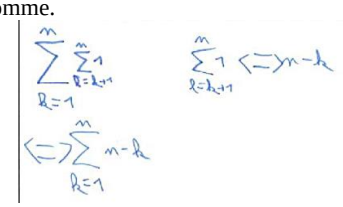


Fig. 3.

Cas 2. L'élève 2 au fil des jeux a découvert le tri à bulle. Il se rend compte que le nombre de « tours » (comparaisons des cartes deux par deux en partant de la première jusqu'à la dernière carte) nécessaires pour trier les cartes est égal au nombre de cartes moins un et qu'à chaque tour, il y a un coup de moins à effectuer. Dans la discussion au sein du groupe, il exprime l'idée que « le nombre de coups minimal nécessaires pour trier *n* cartes est égal à *n*-1 fois le nombre de cartes moins 1 moins 1 mois 1 etc. ». Même si l'élève se rend compte d'avoir une suite arithmétique, il ne trouve pas une autre façon pour formaliser son idée autrement qu'à l'aide d'une procédure en langage de programmation (Fig. 4) qui permet de calculer le nombre de coups pour une valeur donnée de *n*.

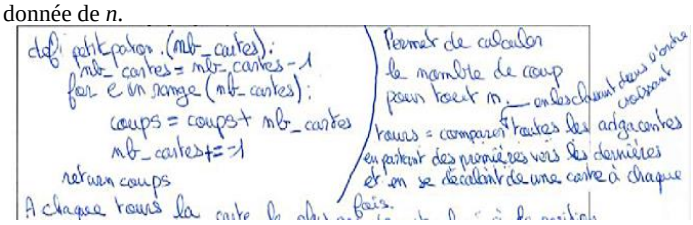


Fig. 4.

Fig. 5.

Nous constatons un lien faux dans la conception de l'élève entre le nombre d'itérations dans la boucle et l'opération de multiplication en mathématiques. On peut aussi retrouver la difficulté liée au changement de cadre dans une autre production de cet élève (Fig. 5). L'utilisation du symbole « sigma » (écrit après l'intervention du professeur qui a invité l'élève à utiliser des outils mathématiques pour décrire le nombre de coups nécessaires pour trier *n* cartes) montre que l'élève reste toujours dans le contexte de programmation avec l'idée de décrémenter *n* à chaque itération.

Cas 3. L'élève 3, quant à lui, a élaboré le tri par insertion. En calculant le nombre de coups effectués pour trier les jeux avec différents nombres de cartes, l'élève s'est aperçu qu'il pouvait modéliser le nombre minimal de coups nécessaires pour trier un nombre quelconque de cartes comme la somme d'une suite arithmétique de raison 1. Ensuite, il

a utilisé ses connaissances mathématiques pour exprimer cette somme en fonction du nombre *n* de cartes (Fig. 6).

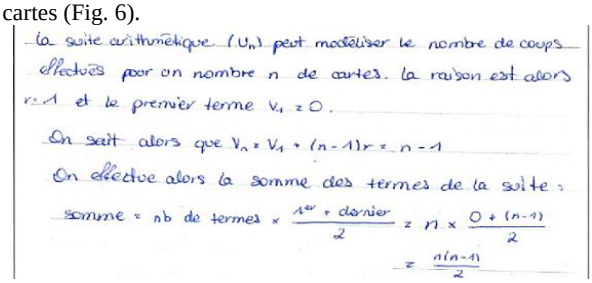


Fig. 6.

En comparant les trois cas, nous pouvons remarquer que les raisonnements des élèves 1 et 2 sont de nature informatique alors que celui de l'élève 3 est plutôt mathématiques. Même si l'élève 1 ne rencontre pas de difficultés pour changer entre les deux cadres contrairement à l'élève 2, dans deux cas le manque des connaissances mathématiques semble constituer un frein dans le travail sur la complexité des algorithmes.

6 Conclusions et perspectives

Dans ce papier nous avons présenté des premiers résultats issus des expérimentations dans des classes de la spécialité NSI de la séquence comportant l'activité « Tri de cartes ». Ces résultats confirment les potentialités didactiques de ces jeux algorithmiques débranchés pour faire découvrir par les élèves des algorithmes de tri évoqués dans les programmes de NSI. En outre, la séquence fournit des conditions favorables pour faire rentrer les élèves dans l'analyse des algorithmes retrouvés du point de vue de nombre de coups minimal nécessaires pour effectuer le rangement en fonction de nombre *n* d'objets. Nous pouvons admettre l'importance du matériel proposé dans l'activité lors de cette phase car les élèves ont très souvent le besoin de revenir dans les situations d'actions en faisant des tris de suites particulières avant passer à la généralisation. Le contexte de travail dans une classe de NSI oriente parfois le raisonnement des élèves vers le raisonnement informatique. Par contre, les connaissances mathématiques semblent être essentielles pour accéder au concept de la complexité des algorithmes de tri. Cela reste un point délicat car tous les élèves suivant la spécialité NSI ne suivent pas forcément aussi la spécialité Mathématiques. En même temps, une analyse plus fine est nécessaire pour identifier les conditions qui peuvent influer sur l'activité des élèves dans les situations construites, notamment le répertoire de connaissances des élèves, les modalités de travail (individuelles, collectives), etc. Une autre perspective dans ce travail concerne l'analyse des preuves, produites par les élèves lors de la séquence, présentes dans notre corpus.

Références

1. Artigue, M. : Ingénierie didactique. Recherches En Didactique Des Mathématiques, 9(3), 281–308 (1988). https://revue-rdm.com/1988/ingenierie-didactique-2/
2. Bell, T., Witten, I., Fellows, M. : Computer Science Unplugged: Off-line Activities and Games for All Ages. Citeseer. (1998)
3. Bloch, I., Gibel, P. : Situations de recherche pour l'accès aux concepts mathématiques à l'entrée à l'université. Revue EpiDEMES, Épijournal de Didactique et Epistémologie des Mathématiques pour l'Enseignement Supérieur. Numéro spécial (2021). ⟨hal-03711877⟩
4. Brousseau, G. : Théorie des Situations Didactiques. La pensée sauvage. Grenoble. (1998).
5. Gibel, P. : Mise en œuvre d'un modèle d'analyse des raisonnements en classe de mathématiques à l'école primaire. Éducation & didactique, 9, 51-72 (2015). https://doi.org/10.4000/educationdidactique.2278
6. Gibel, P. : Elaboration et usages d'un modèle multidimensionnel d'analyse des raisonnements. Note de synthèse de l'Habilitation à diriger les recherches soutenue à l'Université de Pau et des Pays de l'Adour. (2018)
7. Haspekian, M., Nijimbéré, C. : Favoriser l'enseignement de l'algorithmique en mathématiques : une question de distance aux mathématiques ? Éducation & didactique, 10, 121-135 (2016) https://doi.org/10.4000/educationdidactique.2609
8. Modeste, S. : Enseigner l'algorithme pour quoi ? Quelles nouvelles questions pour les mathématiques ? Quels apports pour l'apprentissage de la preuve ? Thèse de doctorat, Université de Grenoble. (2012) ⟨tel-00783294⟩
9. Nishida, T., Kanemune, S., Idosaka, Y., Namiki, M., Bell, T., Kuno, Y. : A CS unplugged design pattern. ACM Sigcse Bulletin, 41(1), 231-235 (2009).
10. Rafalska, M. : Task design for promoting pupils' algorithmic thinking in problem-solving context without using computers. In : J. Hodgen, E. Geraniou, G. Bolondi, & F. Ferretti (Eds.), Proceedings of the Twelfth Congress of the European Society for Research in Mathematics Education (CERME12). (pp. 1981-1989). Free University of Bozen-Bolzano, Italy and ERME. (2022) ⟨hal-03748490⟩

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Vers l'analyse de ressources d'apprentissage de la programmation à l'aide d'un référentiel de types de tâches

Sébastien Jolivet[1][0000−0003−3915−8465], Eva Dechaux[2][0000−0001−8579−1398], Anne-Claire Gobard[3][0000−0002−2866−084X], and Patrick Wang[4][0000−0003−3117−8189]

[1] IUFE & TECFA, Université de Genève, Suisse
sebastien.jolivet@unige.ch
[2] IUFE, Université de Genève, Suisse
eva.dechaux@unige.ch
[3] Lycée Kastler, Académie de Versailles, France
anne-clair.gobard@ac-versailles.fr
[4] Haute école pédagogique du canton de Vaud, Lausanne, Suisse
patrick.wang@hepl.ch

Résumé Dans cette contribution nous abordons la problématique de la description didactique de ressources d'apprentissage de la programmation. Nous nous concentrons sur une compréhension fine des savoirs en jeu dans ces ressources. À cette fin, nous définissons un référentiel de types de tâches, puis l'exploitons pour décrire des problèmes de programmation en illustrant le processus sur deux exemples. Dans la conclusion, nous proposons quelques perspectives pour obtenir, sur cette première base, une description plus complète des ressources.

Keywords: Apprentissage de la programmation · Description de ressource · Référentiel de types de tâches.

1 Introduction

L'enseignement de l'informatique, comme discipline scolaire, s'est généralisé ces dernières années. Ceci entraîne la production de nombreuses ressources d'enseignements. Elles peuvent être en format papier (brochures, manuels, cahiers d'exercices) ou numérique (plateformes en ligne, EIAH, etc.). Avec le partage et la diffusion des ressources, la question n'est pas de savoir s'il existe une ressource qui aborde telle ou telle notion, de telle ou telle manière, mais bien de pouvoir identifier les ressources adéquates à un projet didactique donné. La description des ressources selon des critères de nature bibliothécaire est standardisée – voir par exemple le standard international IEEE LOM (Learning Object Metadata) dont la place centrale est mise en évidence dans [13] (p. 94). Dans [10] les auteurs montrent les limites de différentes approches pour proposer une description didactique des ressources d'apprentissage. Or une telle description est un enjeu important pour aider un agent, humain ou logiciel, à choisir les bonnes ressources

Volume 1, pages 87–98.

pour un projet didactique (enseignant qui prépare sa classe, EIAH avec de l'apprentissage adaptatif, etc.). Dans cet article nous apportons une contribution partielle à la description didactique de ressources : 1) en présentant un référentiel permettant de décrire les savoir-faire, et indirectement les savoirs, mobilisés dans une ressource ; 2) en illustrant, à l'aide de deux exemples, son exploitation pour décrire les types de tâches à mobiliser lors de la construction d'une solution correcte pour des exercices de programmation.

L'utilisation d'un référentiel permet, au-delà de contribuer à la description didactique de la ressource, de répondre à un problème important lié à la description de ressources, qui est qu'elle ne doit pas dépendre de l'annotateur.

Dans [11] et [12] nous avons présenté un premier référentiel de types de tâches mobilisés dans les premiers apprentissages de la programmation et l'avons utilisé pour étudier trois EIAH d'apprentissage de Python. Ce premier référentiel présente certaines limites, en particulier en termes de couverture des types de tâches liés au travail sur la conception d'algorithmes et l'implémentation de programmes. Nous ne reprenons pas le détail de la méthode mise en œuvre pour le construire, signalons simplement qu'elle est en particulier basée sur une étude de divers moyens d'enseignement pour le secondaire.

Dans cette contribution (Sect. 2), nous enrichissons le référentiel et précisons les fondements de sa construction. nous commençons par présenter le référentiel enrichi utilisés, ainsi que les fondements de sa construction. La Sect. 3 explique son utilisation pour décrire des ressources de type exercice en nous appuyant sur deux problèmes fréquemment proposés lors de l'apprentissage de la programmation au secondaire : le problème du *juste prix* et le jeu *pierre, feuille, ciseaux*. Dans chaque section nous pointons les apports et les limites des éléments présentés.

2 Référentiel de type de tâches de la programmation

Dans cette section nous commençons par identifier différents travaux relatifs à la description de l'activité de programmation, puis nous expliquons les spécificités et les fondements de notre référentiel. Enfin, nous insistons sur les principales nouveautés par rapport à la version présentée dans [11]

2.1 Travaux existants

Comprendre et décrire ce que signifie *programmer* est interrogé depuis plus de 50 ans, dans [7] les auteurs proposent une perspective historique. Ils identifient deux grands courants qui ont orienté les motivations et finalités de ce questionnement. Le premier est une approche psychologique de l'activité de programmation, en particulier pour identifier un "bon développeur" pour l'industrie, le second est centré sur l'apprentissage et l'enseignement de la programmation.

Nous nous intéressons plutôt à l'approche psychologique. Dans [14], les auteures situent l'activité de programmation dans le domaine plus large de la

conception (*design*). La programmation y est décrite au travers de grands domaines (compréhension du problème, conception, codage, maintenance) qui organisent les tâches à réaliser, mais ces tâches ne sont pas explicitement identifiées.

Plusieurs travaux visent également à répertorier les éléments constitutifs d'un langage de programmation particulier (e.g., [3,18]) ou partagés par plusieurs langages (e.g., [6,15,17]) afin d'en construire une représentation sous forme d'ontologies. Ces travaux mettent en évidence les objets de savoirs que les apprenants peuvent rencontrer. Le référentiel présenté dans la suite de cet article se distingue de ces travaux en explicitant les manipulations possibles de ces objets de savoir dans le cadre de la résolution d'exercices de programmation.

2.2 Délimitations et fondements du référentiel

Dans ce travail, nous nous situons dans le paradigme de la programmation impérative. Nous considérons qu'un algorithme (ou un programme) est une séquence d'instructions permettant la réalisation d'une tâche t ; t pouvant être de nature et d'ampleur très diverses (e.g., calculer la somme de trois entiers, gérer le décollage d'une fusée, trier une liste, etc.). Nous considérons également qu'un algorithme s'écrit dans un registre autre que celui du langage de programmation (e.g., pseudo code, logigramme, langage naturel) tandis qu'un programme est nécessairement écrit avec un langage de programmation. Cette distinction est motivée par les différents moyens existants pour concevoir un algorithme ou implémenter un programme et l'effet de ces moyens sur l'activité induite. En effet, l'utilisation d'un environnement de développement fournit des moyens spécifiques pour réaliser certaines tâches (par exemple l'exécution, éventuellement pas à pas, d'un programme ou l'affichage de messages en console pour comprendre son fonctionnement). Enfin, le référentiel ne privilégie aucun langage de programmation, ce qui implique que nous ne définissons pas de types de tâches basés sur des caractéristiques syntaxiques (e.g., l'opérateur ternaire `?` en Javascript) ou sémantiques (e.g., la compréhension des listes en Python) différentes d'un langage à l'autre.

Notre travail se place dans une approche anthropologique visant à décrire l'activité humaine mise en œuvre lorsque l'apprenant doit réaliser une tâche de programmation ainsi que ses fondements théoriques. Nous nous appuyons sur le modèle praxéologique développé dans le cadre de la théorie anthropologique du didactique [2], que nous ne détaillons pas dans cet article car non nécessaire pour sa compréhension. Dans ce modèle, l'activité humaine est décrite à l'aide d'un quadruplet constitué de deux blocs, le bloc praxique avec un type de tâches et le moyen de réaliser ce type de tâches (appelé technique dans le modèle) et le bloc du logos qui est constitué des éléments du savoir qui permettent de justifier, organiser, guider la technique, et sur lequel nous ne reviendrons pas dans la suite de cet article. Un des intérêts de choisir ce cadre est qu'il permet de décrire toutes les activités humaines liées à la programmation (modélisation, travail algorithmique, codage, etc.) à l'aide du même cadre et identifie les liens entre savoir-faire et savoirs.

2.3 Structuration et présentation du référentiel

Pour construire le référentiel, nous nous sommes tout d'abord appuyés sur un état de l'art relatif à la question "qu'est ce que programmer ?" (voir par exemple [8,9,16,14]). La synthèse que nous en retenons est qu'il y a des activités qui portent sur d'une part les algorithmes et d'autre part les programmes (conception et maintenance). C'est cette partie du référentiel que nous développons dans cette contribution.

D'autre part, ces activités mobilisent notamment des éléments fondamentaux que sont les variables et les structures de contrôle (boucles bornées et non bornées, instructions conditionnelles). Une seconde partie du référentiel, détaillée dans [11], précise les types de tâches spécifiques à ces éléments.

Pour des raisons de place nous ne présentons que des extraits [5] du référentiel en nous concentrant sur les algorithmes et les programmes. La seconde partie du référentiel est toutefois aussi exploitée dans la section 3.

Pour organiser les principaux types de tâches relatifs à l'activité sur les algorithmes (*types de tâches relatifs aux algorithmes*, Fig. 1) et les programmes (*types de tâches relatifs aux programmes*, Fig. 2) nous nous inspirons des travaux de Pennington et Grabowski [14]. Les auteures distinguent deux activités fondamentales, les tâches de *création d'un programme (composition of a program)* et les tâches de *compréhension d'un programme (comprehension of a program)*, que nous utilisons comme premier niveau de structuration avec les verbes *Produire* et *Analyser*. Dans les types de tâches relatifs aux programme nous ajoutons la catégorie *Mettre au point un programme* qui correspond à l'activité *Maintenance* identifiée dans [14].

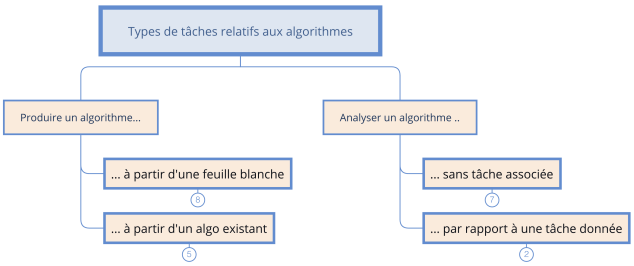


Figure 1. Classification des types de tâches relatifs aux algorithmes.

La catégorie *Produire* est relative au cas où le type de tâches vise à la production d'un algorithme ou d'un programme. Nous distinguons deux situations : 1) on ne dispose d'aucune base de travail, 2) un algorithme ou un programme est déjà existant et il s'agit de le faire évoluer.

5. L'intégralité des éléments disponibles est accessible à ce lien `https://link.infini.fr/ref_prog`

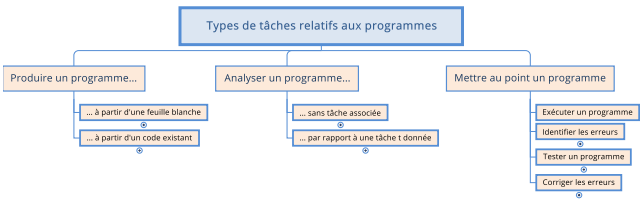


Figure 2. Classification des types de tâches relatifs aux programmes.

La catégorie *Analyser* est aussi scindée en deux cas selon ce qui guide l'analyse. Soit elle porte sur l'objet considéré de manière indépendante de toute tâche à réaliser, par exemple déterminer la complexité algorithmique d'un algorithme ne nécessite pas de lier cet algorithme à la tâche qu'il réalise. Soit la tâche t que doit réaliser l'algorithme ou le programme sert de critère pour mener l'analyse, comme c'est le cas par exemple quand il s'agit de prouver la correction d'un algorithme.

La catégorie *Mettre au point un programme* regroupe différents types de tâches relatifs à la situation où l'on dispose d'un programme non fonctionnel et qu'il s'agit de le faire fonctionner (tester, déboguer, etc.).

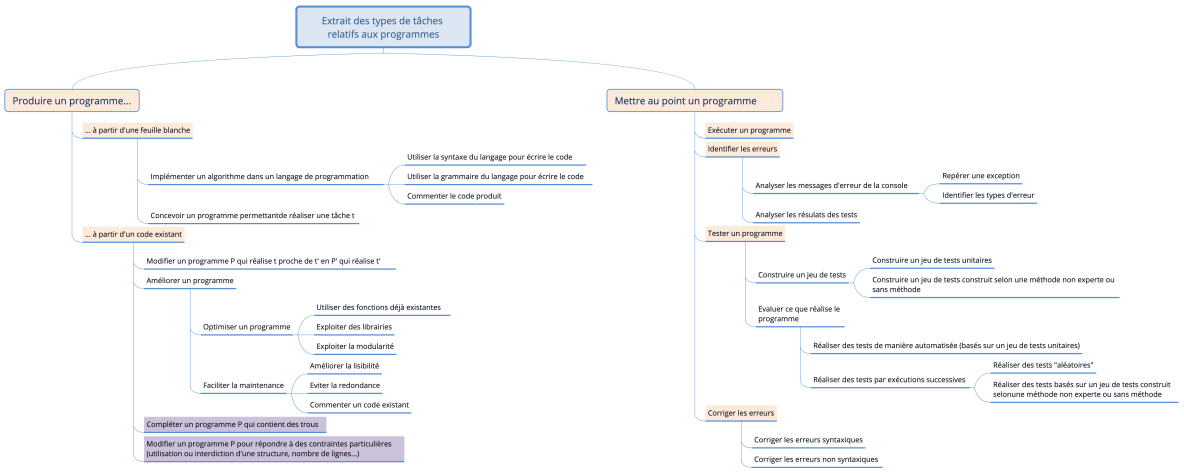


Figure 3. Principaux types de tâches relatifs aux programmes.

Pour être réalisés, chacun de ces types de tâches nécessite de mettre en œuvre d'autres types de tâches. Nous en présentons quelques-uns en Fig. 3, relatifs aux catégories *Produire un programme* et *Mettre au point un programme*. Ils sont organisés à l'aide de nouveaux types de tâches *Identifier les erreurs ; Tester un programme ; Corriger les erreurs*, pour lesquels on précise à nouveau différents types de tâches permettant de les réaliser.

2.4 Synthèse sur le référentiel : apports et limites

Les éléments présentés dans la Sect. 2.3, ainsi que ceux décrits dans [11] , permettent d'avoir un référentiel qui couvre à la fois les types de tâches relatifs aux objets algorithmes et programmes et ceux relatifs aux variables et structures de contrôle.

Il est structuré selon la nature de l'activité à réaliser (produire à partir de rien ou avec une base ; analyser de manière autonome ou relativement à une tâche à réaliser ; mettre au point). Les niveaux inférieurs permettent d'identifier des types de tâches nécessaires à la réalisation de ces types de tâches de plus haut niveau.

Le nombre de types de tâches identifiés est déjà conséquent. Il est aussi nettement plus précis que ce que l'on peut rencontrer dans les curriculums ou dans les représentations ontologiques signalées dans la Sect. 2.1. Il reste cependant possible de raffiner encore ce travail sans remettre en cause la structuration en place. Le niveau de finesse visé dépend essentiellement de l'exploitation souhaitée du référentiel. Par exemple :

- ajouter des types de tâches permettant de mettre en œuvre dans un langage spécifique un type de tâches déjà identifié. Ainsi on pourrait définir le type de tâches *Indenter un ensemble d'instructions* comme ingrédient permettant de réaliser le type de tâches *Déterminer les instructions à exécuter à chaque itération* dans le cas de Python.
- préciser certains ingrédients des techniques permettant de réaliser un type de tâches. Ainsi, on peut ajouter *Dresser un tableau de valeurs des variables* comme moyen, non unique, de réaliser *Déterminer la valeur d'une ou plusieurs variables à l'issue d'une séquence d'instructions*. Ce type de tâches, lui-même, pourra être réalisé de diverses manières en fonction des outils à disposition (e.g., papier-crayon ou IDE) et ainsi donner lieu à la définition de nouveaux types de tâches.

Enfin, le travail d'analyse des moyens d'enseignement nous a amené à identifier certains types de tâches qui n'ont pas de raison d'être si on se positionne du point de vue de l'activité du développeur, mais qui ont une existence comme types de tâches didactiques. Ils sont identifiés en violet dans notre référentiel, par exemple le type de tâches *Compléter un programme à trous* (Fig. 3).

Comme évoqué en Sect. 2.2 nous appuyons notre travail sur l'approche praxéologique. Cela signifie que pour chaque type de tâches il y a une praxéologie qui permet de lier le type de tâches et la technique permettant de le réaliser aux éléments du logos les justifiant. Ainsi ce référentiel, construit et présenté autour des

savoir-faire, permet aussi d'identifier les savoirs. La conclusion est l'occasion de dessiner d'autres perspectives sur les utilisations et évolutions de ce référentiel.

3 Description de la résolution de deux problèmes

Dans cette section nous revenons sur notre problématique de description didactique de ressources d'enseignement de la programmation. Un premier élément, nécessaire mais non suffisant comme expliqué dans [10] , est l'identification des savoir-faire et savoirs mobilisables pour réaliser les tâches prescrites.

Nous nous concentrons sur cet aspect et montrons en quoi le référentiel contribue à répondre à cet objectif. La démarche se déroule en deux étapes. Tout d'abord il s'agit d'identifier les types de tâches situés aux niveaux *algorithme* et *programme* qui sont mobilisés pour produire un programme permettant de réaliser la tâche t visée. Dans un deuxième temps, à partir d'une ou plusieurs solutions du problème, nous identifions les types de tâches relatifs aux variables et structures de contrôle. Nous illustrons maintenant cette démarche sur deux problèmes qui peuvent être rencontrés lors de l'apprentissage de la programmation au secondaire : les jeux *Pierre - Feuille - Ciseaux* et *Devine le juste prix*.

3.1 Types de tâches du niveau algorithmes et programmes

Nous considérons que, quel que soit le problème à résoudre (i.e. la tâche t à faire réaliser par le programme visé), l'ensemble [6] de types de tâches suivant est potentiellement mobilisé : *Concevoir un algorithme réalisant une tache t* ; *Concevoir un programme réalisant une tache t* ; *Exécuter un programme* ; *Tester un programme*. La réalisation de chacun de ces types de tâches va mobiliser au moins une partie des types de tâches identifiées dans le référentiel (par exemple *Modéliser* ; *Choisir une structure de contrôle* ; etc.). Le fait que ces types de tâches soient effectivement mobilisés ou non peut être déterminé à partir d'une analyse de l'énoncé permettant la dévolution du problème à l'apprenant. Nous n'abordons pas cette étape supplémentaire dans ce document. Nous illustrons maintenant la deuxième étape en nous basant sur une solution "classique".

3.2 Pierre - Feuille - Ciseaux

L'énoncé considéré pour mener l'analyse de ce problème est le suivant : *écrire un programme qui permet de jouer une manche de* Pierre - Feuille - Ciseaux *contre un ordinateur qui joue de manière aléatoire. Le joueur doit savoir si le résultat est gagné, nul ou perdu.* Pas de gestion des saisies erronées attendue. Un extrait du code, en Python, d'une solution possible à ce problème est :

6. Nous utilisons le terme *ensemble* à dessein compte tenu qu'il n'y a pas de chronologie systématique entre ces types de tâches et qu'ils peuvent être mobilisés plusieurs fois dans une démarche, partiellement, itérative.

```
import random
choix = int(input( "PIERRE (1) ou FEUILLE (2) ou CISEAUX (3)? "))
hasard = random.randint(1,3)
if choix == hasard :
    print("Match nul !")
elif choix == 1 and hasard == 3:
    print("Victoire !")
# Autres branches absentes pour des raisons de longueur d'article
else :
    print("Perdu :-(")
```

Cette solution permet d'identifier la mobilisation des types de tâches suivants : *Déclarer une variable* ; *Affecter une valeur à une variable* ; *Utiliser une variable dans une expression* ; *Concevoir une instruction conditionnelle.*

3.3 Devine le juste prix

L'énoncé considéré pour mener l'analyse de ce problème est le suivant : *écrire un programme qui permet à l'utilisateur de deviner un nombre entier entre 1 et 100, choisi aléatoirement par l'ordinateur. Il fait des propositions et indique si le nombre est plus grand, plus petit ou affiche gagné si le nombre est trouvé. La partie s'arrête dès que c'est gagné.* Il n'est pas demandé que le programme gère des saisies erronées ou compte le nombre d'essais. Un extrait du code, en Python, d'une solution possible à ce problème est :

```
import random
devine = 0
prix = random.randint(1,100)
while prix != devine:
    devine = int(input("Devinez le prix : "))
    if devine > prix :
        print("C'est moins !")
    elif devine < prix :
        print("C'est plus !")
    else :
        print("Bravo, le prix est bien", devine)
```

Cette solution permet d'identifier la mobilisation des types de tâches suivants : *Déclarer une variable* ; *Affecter une valeur à une variable* ; *Utiliser une variable dans une expression* ; *Concevoir une instruction conditionnelle* ; *Concevoir une boucle non bornée.*

3.4 Synthèse sur la description de ressources : apports et limites

Nous venons d'illustrer l'utilisation du référentiel pour identifier les savoir-faire mobilisables pour résoudre un problème donné. Tout d'abord en établissant un premier ensemble de types de tâches permettant d'aller de l'analyse du problème à résoudre à la production d'un programme puis à sa validation, ou sa correction

s'il ne réalise pas la tâche visée. Certains de ces types de tâches sont nécessairement précédés par d'autres (on ne produit pas un programme avant d'avoir réalisé un travail sur l'algorithme), d'autres vont nécessiter de remobiliser des types de tâches déjà utilisés au moins une fois précédemment. Par exemple, le type de tâches *Corriger les erreurs non syntaxiques* va exploiter des types de tâches pouvant relever des catégories *Analyser un programme* et/ou *Produire un programme à partir d'un code existant*, *Analyser un programme* ayant possiblement déjà été utilisé lors de la réalisation du type de tâches *Chercher un programme réalisant une tâche t' proche de la tâche t.*

Puis, sur la base d'une solution au problème, nous utilisons à nouveau le référentiel pour identifier un second ensemble de types de tâches à mobiliser, qui sont, cette fois, relatifs aux structures de contrôle et variables qui sont mobilisées.

Ceci permet de proposer pour chaque problème une étude des savoirs-faire en jeu et, en remontant au niveau des praxéologies, d'identifier les savoirs mobilisés. Par exemple, par rapport au type de tâches *Concevoir une boucle non bornée*, on va avoir des savoirs tels que *la définition de la valeur de vérité d'une expression* ou *le fait qu'une boucle non bornée est constituée d'un test d'arrêt et d'un corps.*

En travaillant sur un corpus de ressources, cette analyse des ressources peut servir à évaluer la couverture d'un ensemble de savoirs ou types de tâches visés, comme cela a été fait dans [12] où il est montré que des catégories entières de types de tâches sont absentes des EIAH analysés.

La description de ressources, telle que présentée ici, présente trois limites significatives :

- La première est intrinsèque au niveau de précision du référentiel exploité. Par exemple, dans les types de tâches relatifs aux instructions conditionnelles, nous identifions le type de tâches *Déterminer les différentes branches* mais notre référentiel ne nous permet pas de distinguer s'il y a 2 ou 10 branches dans la solution analysée. Ou encore, nous identifions en Sect. 3.3 les types de tâches *Concevoir une IC* ; *Concevoir une boucle non bornée* mais notre description ne rend pas compte du fait qu'elles sont imbriquées dans cette solution. Si on estime que ce type de précision présente un intérêt pour la qualité de la description des exercices, au regard de l'usage souhaité, il est possible de préciser le référentiel en reprenant par exemple la démarche présentée dans [10] qui exploite les générateurs de types de tâches définis dans T4-TEL [1].
- La deuxième est liée au fait que nous n'avons pris en compte qu'une seule solution du problème pour la deuxième phase de la description. Face à cette limite on peut choisir de prendre en compte plusieurs solutions et d'avoir des ensembles de types de tâches (et de praxéologies) qui seront spécifiques à chaque solution. L'étude des différences entre ces ensembles peut, en fixant un contexte d'utilisation de la ressource donnée, déterminer leur viabilité.
- La troisième concerne l'absence de prise en compte de l'énoncé utilisé pour communiquer le problème à l'apprenant sous forme d'exercice. Cet énoncé peut contenir des indications particulières (par exemple sur les

variables à déclarer, ce qui reviendrait à la prise en charge partielle du type de tâches *Modéliser*) ou des questions intermédiaires (par exemple donner une fonction à définir, ce qui reviendrait à une prise en charge partielle du type de tâches *Modulariser*). Cette prise en compte peut se réaliser lors d'une deuxième étude permettant de raffiner les types de tâches identifiés, par exemple en les caractérisant avec des indications du type *identifié dans l'énoncé*, *(partiellement) pris en charge par l'énoncé*, etc.

In fine, les choix à réaliser sur les limites identifiées ci-dessus sont essentiellement liés aux besoins de description des ressources. Par exemple, s'il s'agit de savoir si un code donné est pertinent comme exemple pour illustrer une notion, il n'est pas nécessaire d'envisager tous les codes solutions du problème. S'il s'agit de comparer des exercices entre eux il pourra être nécessaire de prendre en considération les éléments apportés par les énoncés et pas uniquement le problème à résoudre. S'il s'agit, pour un agent humain ou logiciel, de recommander des exercices à un apprenant, le niveau de description des types de tâches et les informations associées seront à adapter à la connaissance de l'apprenant (l'institution où il apprend ; son rapport aux types de tâches étudiés ; etc.).

4 Conclusion

Dans cette contribution nous avons présenté un référentiel de types de tâches d'apprentissage de la programmation en nous centrant sur les aspects liés aux algorithmes et aux programmes. Il enrichit significativement celui présenté dans [11] . La diversité des types de tâches identifiés, sans viser l'exhaustivité, montre l'ampleur de ce que veut dire *apprendre à programmer* et de ce que l'apprenant doit travailler. Et, par contraste, cela peut aussi mettre en évidence certains types de tâches qui ne sont pas traités lors de l'enseignement et laissés *de facto* à la charge complète de l'élève [7].

Le deuxième apport est une esquisse de méthode de description de problèmes de programmation. Les premières limites et perspectives associées à ce travail sont précisées en Sect. 3.4.

Du point de vue du référentiel, divers travaux sont engagés, ou en voie de l'être, pour répondre à certaines limites identifiées en Sect. 2. En particulier, améliorer la couverture avec l'intégration des types de tâches relatifs aux fonctions et aux listes et raffiner en spécifiant certains sous-types de tâches. Un autre axe est la construction de praxéologies complètes en liant types de tâches et techniques avec le logos. Une perspective liée à cette axe est la construction d'une ontologie comme évoqué dans [5] dans un autre domaine.

Pour ce qui est de la description des ressources, une suite envisageable concerne bien évidemment l'exploitation du référentiel étendu pour décrire une plus grande

7. Nous n'affirmons pas ici qu'il faut impérativement travailler tous ces types de tâches, simplement que la plupart d'entre eux vont sans doute être rencontrés à un moment ou un autre par les apprenants de la programmation.

diversité de ressources. Un autre axe consisterait à mettre en relation cette description avec d'autres caractéristiques comme la complexité d'une ressource [4], le niveau d'étayage proposé, ou son adéquation à une intention didactique. Cet axe offre d'autres perspectives d'utilisation de ce référentiel et de cette méthode de description qui pourraient aider un enseignant ou un EIAH dans leurs choix de ressources. Ce sujet a été partiellement exploré dans [10] autour de ressources en mathématiques, mais il s'agit maintenant de prendre en compte les spécificités des ressources pour l'enseignement de la programmation.

Références

1. Chaachoua, H., Bessot, A., Romo, A., Castela, C. : Developments and functionalities in the praxeological model. In : Bosch, M., Chevallard, Y., Javier Garcia, F., Monaghan, J. (eds.) Working with the anthropological theory of the didactic : A comprehensive casebook, pp. 41–60. New perspectives on research in mathematics education - ERME Series, Routledge, routledge edn. (2019)
2. Chevallard, Y. : Concepts fondamentaux de la didactique : perspectives apportées par une approche anthropologique. Recherche en didactique des mathématiques **12**(1), 83–121 (1992)
3. Diatta, B., Basse, A., Ouya, S. : PasOnto : Ontology for learning Pascal programming language. In : 2019 IEEE Global Engineering Education Conference (EDUCON). pp. 749–754. IEEE (2019)
4. Feitelson, D.G. : From Code Complexity Metrics to Program Comprehension. Communications of the ACM **66**(5), 52–61 (May 2023). https://doi.org/10.1145/3546576
5. Grugeon-Allys, B., Jolivet, S., Lesnes, E., Luengo, V., Yessad, A. : Mindmath : didactique des mathématiques et intelligence artificielle dans un EIAH. In : Vandebrouck, F., Emprin, F., Ouvrier-Buffet, C., Vivier, L. (eds.) Nouvelles perspectives en didactique des mathématiques : preuve modélisation et technologies numériques, vol. Volume des ateliers, actes de EE21, pp. 133–152. IREM de Paris - Université de Paris (2023)
6. Grévisse, C., Botev, J., Rothkugel, S. : An Extensible and Lightweight Modular Ontology for Programming Education. In : Solano, A., Ordoñez, H. (eds.) Advances in Computing, vol. 735, pp. 358–371. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-66562-7_26
7. Guzdial, M., Du Boulay, B. : The History of Computing Education Research. In : Fincher, S.A., Robins, A.V. (eds.) The Cambridge Handbook of Computing Education Research, pp. 11–39. Cambridge University Press, 1 edn. (Feb 2019). https://doi.org/10.1017/9781108654555.002
8. Hermans, F., Aldewereld, M. : Programming is Writing is Programming. In : Companion Proceedings of the 1st International Conference on the Art, Science, and Engineering of Programming. pp. 1–8. Programming '17, Association for Computing Machinery, New York, NY, USA (Apr 2017). https://doi.org/10.1145/3079368.3079413
9. Hoc, J.M. : Quelques remarques sur l'analyse du travail et la formation de l'analyste-programmeur. Bulletin de Psychologie **28**, 357–359 (1974)

10. Jolivet, S., Chaachoua, H., Desmoulins, C. : Modèle de description didactique d'exercices de mathématiques. Recherche en Didactique des Mathématiques **42**(1), 53–102 (2022)
11. Jolivet, S., Dechaux, E., Gobard, A.C., Wang, P. : Construction et exploitation d'un référentiel de types de tâches d'apprentissage de la programmation. In : Actes du colloque EIAH2023 : 11ème Conférence sur les Environnements Informatiques pour l'Apprentissage Humain. Brest (2023)
12. Jolivet, S., Dechaux, E., Gobard, A.C., Wang, P. : Description et analyse de trois EIAH d'apprentissage de Python. In : Actes de l'atelier APIMU : 11ème Conférence sur les Environnements Informatiques pour l'Apprentissage Humain. Brest (2023)
13. Loiseau, M. : Élaboration d'un modèle pour une base de textes indexée pédagogiquement pour l'enseignement des langues. Thèse de doctorat, Université Stendhal - Grenoble III, Grenoble (Dec 2009)
14. Pennington, N., Grabowski, B. : The tasks of programming. In : Psychology of programming, pp. 45–62. Elsevier (1990)
15. Pierrakeas, C., Solomou, G., Kameas, A. : An Ontology-Based Approach in Learning Programming Languages. In : 2012 16th Panhellenic Conference on Informatics. pp. 393–398. IEEE, Piraeus, Greece (Oct 2012). https://doi.org/10.1109/PCi.2012.78
16. Rogalski, J., Samurçay, R. : Acquisition of Programming Knowledge and Skills. In : Psychology of Programming, pp. 157–174. Elsevier (1990). https://doi.org/10.1016/B978-0-12-350772-3.50015-X
17. Shishehchi, S., Mat Zin, N.A., Abu Seman, E.A. : Ontology-Based Recommender System for a Learning Sequence in Programming Languages. International Journal of Emerging Technologies in Learning (iJET) **16**(12), 123 (Jun 2021). https://doi.org/10.3991/ijet.v16i12.21451
18. Sosnovsky, S., Gavrilova, T. : Development of educational ontology for C-programming. International Journal "Information Theories & Applications" **13**, 303–308 (2006), publisher : Institute of Information Theories and Applications FOI ITHEA

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Vers une cartographie des Situations d'Informatique Débranchée ⋆

Lecocq Mage Julian[1,3], Modeste Simon[1], Beffara Emmanuel[2], Duchêne Éric[3], Parreau Aline[3], and Rafalska Maryna[4]

[1] IMAG, Université de Montpellier, CNRS, Montpellier
[2] Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, 38000 Grenoble, France
[3] Univ. Lyon, CNRS, UCBL, INSA Lyon, Centrale Lyon, Univ. Lyon 2, LIRIS, UMR5205, F-69622 Villeurbanne
[4] LINE, Université Côte d'Azur, France

Résumé Nous nous intéressons aux situations d'informatique débranchée, qui sont conçues et proposées dans divers lieux, en France, et dans d'autres pays (Royaume-Uni, Nouvelle-Zélande). Notre objectif est d'essayer de "cartographier" ces situations, afin de mieux comprendre le mouvement de l'informatique débranchée et les ressources qu'il propose, en tenant compte de ses développements récents. Nous présentons le périmètre de notre étude et la collecte des situations de notre corpus. Nous explicitons les critères d'analyse et le codage des situations, suivant les contenus abordés, les positionnements didactiques et les modalités matérielles de mise en œuvre, que nous avons appliqués à notre corpus. Les analyses nous permettent de faire de premières observations sur les spécificités des situations proposées et du mouvement de l'informatique débranchée, et d'esquisser une "cartographie" des situations de notre corpus.

Keywords: Informatique débranchée · didactique · classification

1 Introduction et problématique

Le développement de l'informatique dans la société entraîne un besoin de personnes qualifiées mais aussi celui d'une culture informatique pour tous. Ainsi, progressivement, l'informatique s'est développée dans l'enseignement mais aussi dans les actions de médiation scientifique. Les premiers enseignements scolaires d'informatique, en France comme dans plusieurs autres pays, datent des années 1980, et impliquaient essentiellement une initiation à la programmation [1,2]. Si des oscillations ont eu lieu dans l'enseignement entre informatique comme outil et informatique comme objet [3], un consensus sur la nécessité d'enseigner l'informatique pour elle-même et de développer une pensée *computationnelle* [15] semble se dessiner aujourd'hui [14, par exemple]. La programmation et l'utilisation de dispositifs technologiques (ordinateurs, robots...) ont toujours occupé une

⋆. Cette recherche a été soutenue par le projet ASMODÉE, ANR-21-SSMS-0001.

place importante dans l'enseignement de l'informatique, sous différentes formes [3], mais la nécessité de développer chez un large public la compréhension de concepts propres à ce domaine soulève des questions d'ordre didactique.

C'est à cela que tente de répondre le mouvement de l'*informatique débranchée* ou *informatique sans ordinateur*, apparu au tournant des années 1990, en proposant des activités qui se détachent de la seule programmation. Le projet *Computer Science Unplugged* (CSU) a été pionnier avec la rédaction d'un manuel et l'ouverture d'un site web [5]. Les intentions étaient clairement de "transmettre des idées fondamentales qui ne dépendent pas d'un logiciel ou d'un système particulier, des idées qui seront encore d'actualité dans 10 ans" [5]. Aujourd'hui, ce mouvement s'est étendu dans de nombreux pays, et de nombreuses autres activités d'informatique débranchée ont été développées en dehors du projet CSU [13, par exemple]. En France, l'informatique débranchée s'est beaucoup développée en parallèle du système d'enseignement, notamment dans le contexte de la médiation scientifique.

La question des savoirs en jeu et des connaissances développées dans les activités d'informatique débranchée a été peu soulevée dans la littérature, comme le note Drot-Delange [9]. On peut supposer que tous les contenus de l'informatique ne sont pas identiquement concernés par l'informatique débranchée, et se pose la question des champs et des notions couvertes par les activités débranchées.

Si l'on peut clairement identifier les grands principes qui guident le mouvement, on note aussi une grande diversité dans les situations existantes et dans les façons dont sont pensées leurs mises en œuvre et l'activité dans laquelle sont plongés les participants. Ainsi, se posent des questions de nature didactique [10], relatives aux objectifs des activités (apprentissage de notions, découverte de concepts, de grands principes ou de la pensée informatique), mais aussi sur les modalités de fonctionnement de ces situations d'informatique débranchée.

Notre question principale est : sur quels critères peut-on analyser et comparer les *Situations d'Informatique Débranchée* entre elles afin d'étudier leurs spécificités et leurs caractéristiques du point de vue des contenus et des approches didactiques ? Et quelles observations en ressortent ?

2 Méthodologie et critères d'analyses

Pour répondre à cette question, nous avons constitué un corpus de situations d'informatique débranchée, et construit des critères de classification. Chaque situation du corpus a alors été analysée selon ces critères. Nous présentons d'abord nos critères d'analyse, puis la constitution du corpus et l'application des critères. Dans la dernière section nous présentons nos premiers résultats.

Nos critères d'analyse concernent d'une part les contenus mis en jeu, et d'autre part les questions didactiques liées à la mise en œuvre de la situation.

5. Nous distinguons le projet CSU et le mouvement de l'*informatique débranchée*.

2.1 Contenus informatiques : critères d'ordre épistémologique

Nous allons positionner les activités recensées selon la nature du contenu informatique abordé. Dowek [8] défend le constat d'une incomplétude des définitions communes de l'informatique. Il y oppose un recouvrement de ce dont l'informatique est l'objet. D'autres manières de présenter et structurer la discipline informatique ont été proposées : Denning [7], qui a notamment participé au projet *Computer Science Unplugged*, Wing [15, cité par [6]], défendant "l'intégration de la pensée computationnelle (*computational thinking*) au socle intellectuel fondamental" [6], ou encore Hartmann *et al.* [11, cité par [6]]. Nous avons retenu la catégorisation de Dowek, prégnante dans la structuration de l'enseignement de l'informatique en France, qui s'appuie sur une réflexion épistémologique et philosophique, et la catégorisation de Denning, que l'on retrouve chez CSU, mais aussi dans l'analyse de Drot-Delange [9].

Classification de Denning – Conçue dans l'optique de démontrer la légitimité de l'informatique comme domaine à part entière, elle identifie 7 "principes" [7] :

- **Calcul** : algorithmique, vitesse de calcul, complexité, programmation,
- **Communication** : représentation, compression et transmission fiable de données, cryptographie, encodage,
- **Coordination** : interaction, coopération d'agents en réseau,
- **Enregistrement** : enregistrement, stockage et récupération de données,
- **Automatisation** : automatisation d'un système, intelligence artificielle, intelligence collective,
- **Évaluation** : efficacité (performance) d'un système, vérification formelle,
- **Conception** : conception d'un système informatique fiable et efficace.

Ces principes peuvent sembler marqués par une forte prédominance technique ou pratique. Nous faisons donc le choix, en parallèle de cette première catégorisation, de mobiliser celle développée par Dowek [8].

Piliers de Dowek – Cette typologie est structurée en 4 "piliers" ou "concepts" constitutifs de l'informatique, sans être des champs distincts ou des sous-disciplines :

- **Algorithme :** concept d'algorithme, égalité entre deux algorithmes, déterminisme et non déterminisme, objet des algorithmes,
- **Machine :** diversité des machines, puissance de calcul et limites, espace et réseau, ordinateurs, machines parallèles et spécialisées,
- **Langage :** langages de programmation, langue naturelle et langage formel,
- **Information :** représentation symbolique des données, compression, cryptographie, bases de données.

Ces deux classifications sont complémentaires, l'une décrit plutôt de grands types d'activité, la seconde des grands concepts transversaux de la discipline.

Pour ajouter une dimension liée à la structuration des différents champs de l'informatique, nous avons utilisé la nomenclature thématique de la section 27 (informatique) du Conseil National des Universités (CNU 27), notamment utilisée pour la description des thématiques de recherche [4]. Nous ne détaillerons pas cette dimension ici, afin de nous concentrer sur les premiers résultats obtenus, liés aux deux autres classifications.

2.2 Aspects didactiques

Nous complétons ces catégorisations relevant de l'épistémologie de l'informatique par des critères se rapportant davantage à la dimension didactique. Ainsi, nous proposons les critères suivants, construits pour cette étude, et inspirés d'éléments classiques des études en didactiques des sciences [12] :

Approche didactique par rapport au contenu visé – Nous proposons de distinguer deux type de situations :

- celles qui abordent un contenu par une approche de **résolution de problème**, c'est-à-dire qui proposent un problème aux participants, dont la solution construite dans l'activité est l'objet visé par la situation ;
- celles qui relèvent de la **mise en œuvre ou de l'application** par les participants du contenu visé.

Position des participants vis-à-vis du dispositif débranché – Nous en distinguons deux :

- **incarnation du dispositif** : les participants incarnent des éléments d'un dispositif et "simulent" leur comportement ;
- **extérieur au dispositif** : les participants manipulent ou observent un dispositif dont ils ne font pas partie.

Échelle matérielle du dispositif débranché – Nous distinguons :

- l'échelle **macro** : le dispositif débranché est de l'échelle de la "salle de classe" ou plus grand ;
- l'échelle **micro** : le dispositif débranché est de l'échelle du participant ou plus petit ;
- et des situations **multi-échelles** ou les deux cas interviennent.

Nature de la visée didactique – Nous entendons par là distinguer :

- une visée d'**apprentissage** : il est attendu que les participants aient construit une connaissance identifiée à l'issu de la situation ;
- une visée de **découverte** : il est attendu de faire explorer un champ ou une notion, ou de démystifier un aspect de l'informatique.

Format de travail en groupe – Dans l'analyse, nous identifierons aussi cette modalité de travail des participants : travail individuel, en petits sous-groupes, en groupe complet (grand groupe).

Intervalles d'âges concernés – L'âge ou le niveau visé est une information qui nous semble importante à prendre en compte.

3 Recueil des données et "codage"

Recueil des activités – Une des difficultés provient de la large variété des acteurs, de leur dispersion, et de la diversité linguistique. La lecture de rapports issus du groupe InfoSansOrdi, rassemblant les personnes intéressées par l'informatique débranchée en France [6] nous a permis de lister un certain nombre d'acteurs. Nous avons recensé dans un premier temps les activités CSU (version la plus récente du manuel). Nous avons ensuite recensé les activités produites par certains groupes IREM [7], des laboratoires d'informatique comme le LORIA (Nancy), l'IRISA (Rennes) ou le LIRIS (Lyon), ou des projets ou lieux de médiation et diffusion de l'informatique (Maison des Mathématiques et de l'Informatique de Lyon, TerraNumerica). Enfin, nous ajoutons des acteurs dont certaines activités sont mentionnées dans des ressources analysées, notamment le projet Computer Science 4 Fun (cs4fn) à la *Queen Mary University of London*. L'ensemble est détaillé dans le Tableau 1.

Acteurs	**Nombre**	**Méthode de recensement des activités**
CSU	12	Lecture du manuel
IREM Clermont-Fd	13	Inspection du site web, téléchargement des fiches d'activité
IREM Grenoble	7	
LORIA	20	
IRISA	4	
cs4fn	22	
LIRIS / MMI	6	Récupération des déroulés d'activités
TerraNumerica	40	Inspection du catalogue d'activités en ligne

Table 1. Nombre de références par acteur (≥ 3 références)

À l'issue de ce processus de recensement – non-exhaustif mais couvrant une bonne partie des acteurs visibles en France – nous avons listé 141 ressources correspondant à des activités. Parmi ces ressources, un certain nombre correspondait en fait à la même activité, avec un déroulement très proche. Nous avons au final un corpus de 116 situations.

Méthodologie de classification – L'affectation des valeurs pour chacun des critères s'est faite de deux manières. Certaines données étaient fournies (âge du public, nombre de participants) et nous avons seulement inféré les informations quand elles n'étaient pas explicites. Pour les autres, nous avons analysé les ressources pour affecter une valeur (ou plusieurs parfois, par exemple pour les piliers de Dowek, les principes de Denning, ou les visées...). La méthode d'affectation pour chacun des critères est exposée dans le tableau 2.

6. https ://github.com/InfoSansOrdi/InfoSansOrdi/blob/master/README.md
7. Instituts de Recherche sur l'Enseignement des Mathématiques.

4 Analyses, et premiers résultats

Cette analyse nous permet de faire de premiers constats sur les situations d'informatique débranchée étudiées, d'abord sur un plan descriptif, puis en croisant certains critères. Nous ne pouvons pas détailler ici tous les aspects.

Critère	**Méthode**	
Intervalle d'âge, cycle Format groupe	Donnée	Donnée fiche pédagogique quand disponible ; conversion cycle-âge en fonction de la valeur donnée.
Principe(s) de Denning Pilier(s) de Dowek Mots-clés CNU27	Evaluation par une analyse	Analyse fiche pédagogique : prise en compte de la/des thématique(s), pré-requis, objectif(s) d'apprentissage En l'absence de fiche : analyse de l'éventuelle description ou des autres ressources éventuellement disponibles.
Approche didactique du contenu Position des participants Échelle		Analyse du déroulé et/ou de la fiche pédagogique : analyse du point de vue du matériel (espace nécessaire), des éventuels déplacements physiques, de l'éventuelle distribution de "rôles" aux participant·e·s. Analyse de la présence, ou non, de solutions à appliquer, ou de phases de recherches.
Nature de la visée		Analyse des objectifs d'apprentissages éventuellement explicités, de la thématique indiquée ; prise en compte du type d'acteur proposant la ressource.

Table 2. Mode d'affectation des valeurs par critère

Description des situations du corpus – Une grande majorité des situations du corpus (88%) sont proposées pour les cycles 2 (6-8 ans) ou 3 (9-11 ans). En fait, on trouve le plus souvent un niveau recommandé minimum en terme d'âge ou de niveau scolaire, les situations sont donc considérées comme adaptées à tout niveau supérieur. Ainsi, peu de situations considèrent que des notions de collège ou plus sont nécessaires. Ces niveaux pourraient être confrontés avec une analyse fine des prérequis de chaque situation, et de la complexité du travail attendu, mais nous nous basons ici sur les déclarations des auteurs dans les ressources (afin de pouvoir croiser les choix des auteurs avec les publics qu'ils visent).

Au niveau des contenus (figure 1), les principes de Denning les plus présents sont *Calcul*, très majoritairement, et *Enregistrement*. En ce qui concerne les piliers de Dowek, ce sont les piliers *Algorithme*, majoritairement, et *Information* qui dominent. Ces deux constats sont cohérents entre eux, et semblent cohérents avec les principes fondateurs de l'informatique débranchée d'être complémentaire des apprentissages développés dans une activité de programmation.

Relativement aux aspects didactiques, environ 2/3 des situations sont dans une approche de résolution de problème pour environ 1/3 de mise en application. En ce qui concerne la position des participants dans le dispositif, environ 2/3 des situations proposent une position extérieure au dispositif débranché et 1/3 une incarnation du dispositif. Au niveau de l'échelle, on observe une répartition

plus équilibrée (50 macro, 60 micro, 3 multi). Enfin, concernant la visée, les situations du corpus concernent principalement la découverte (66 découverte, 15 apprentissage, et 31 qui sont dans les deux catégories). Cette dernière observation découle certainement du positionnement de l'informatique débranchée (en France en tout cas) plutôt du côté de la médiation que de l'enseignement.

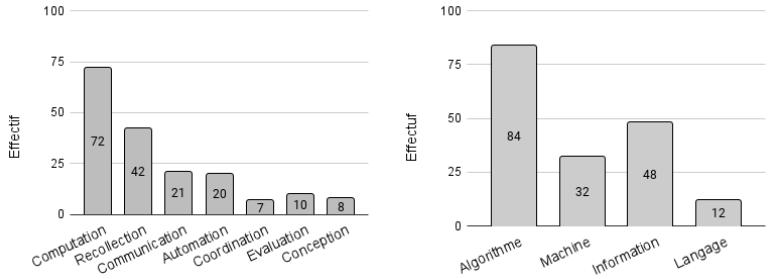


Figure 1. Effectifs des situations par principe de Denning et pilier de Dowek.

Exploration des données, premières observations – La figure 2 croise les critères d'approche didactique, de position des participants, et de visée. Nous avons identifié un lien qui semble étroit entre approche "résolution de problème" et position extérieure au dispositif d'une part, et approche "mise en œuvre" et incarnation du dispositif d'autre part (figure 2, gauche). Cette relation a été confirmée par test du Khi-deux d'indépendance, avec seuil de signification à 5 %. On y reconnaît deux approches bien distinctes de l'informatique débranchée : l'une qui vise à la construction de connaissance par la résolution de problèmes impliquant des manipulations, tandis que l'autre vise à développer la compréhension du fonctionnement d'un système, par sa mise en œuvre sous forme d'une expérience incarnée. Concernant les visées (figure 2, milieu et droite), nous constatons un lien entre l'approche par résolution de problème et la visée d'apprentissage (confirmé par par test du Khi-deux d'indépendance, avec seuil de signification fixé à 5 %). De façon cohérente, nous observons aussi un lien entre la position extérieure au dispositif et la visée "apprentissage", bien que plus ténu (confirmé par test du Khi-deux d'indépendance, avec seuil de signification à 10 %). Nous n'avons pas identifié de lien entre la visée de découverte et d'autres critères (contrairement à la visée apprentissage mentionnée précédemment, ce qui peut questionner la dualité découverte-apprentissage de notre catégorisation). En ce qui concerne les liens entre critères didactiques et contenus, nous n'avons pas identifié, pour l'instant, de lien particulier, mais nous allons continuer d'explorer ces données avec des outils statistiques plus avancés.

Discussions et perspectives – Notre étude propose une première cartographie de 116 situations d'informatique débranchée identifiées à partir du réseau des acteurs français. Nous avons proposé des critères qui nous ont permis de classifier

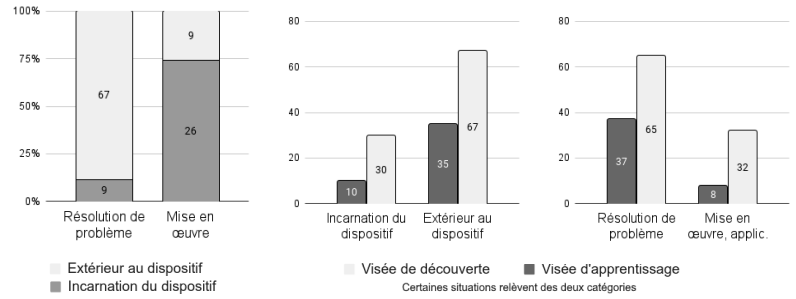


Figure 2. Statistiques croisées deux à deux des critères d'approche didactique, de position vis-à-vis du dispositif, et de visée.

les contenus et d'identifier les contenus privilégiés en informatique débranchée, ainsi que les aspects didactiques caractéristiques des situations. Cela rend notamment visible deux modalités didactiques privilégiées, rassemblant certains choix d'approche et de visée didactiques, et de position vis-à-vis du dispositif débranché. Ce travail de construction de critères a aussi nourri la réflexion au sein du collectif InfoSansOrdi, contribuant à la structuration d'une future base de données en ligne des ressources en informatique débranchée.

Une perspective du travail est d'approfondir l'exploration de nos données, notamment pour rechercher des liens entre types de contenus et modalités didactiques. Nous souhaitons aussi affiner la cartographie des contenus. Approfondir la compréhension des spécificités des situations développées en informatique débranchée peut aussi s'avérer nécessaire pour accompagner le passage du monde de la médiation à celui, plus formel, de l'enseignement scolaire.

Références

1. Aigle, M. : "Vous avez dit binaire ?". In : Baron, G.L., Baudé, J., Cornu, P. (eds.) Colloque francophone sur la didactique de l'informatique. pp. 177–192. Association EPI, Paris, France (1988), https ://edutice.hal.science/edutice-00362447
2. Baron, G.L., Bruillard, É. : L'informatique et son enseignement dans l'enseignement scolaire général français : enjeux de pouvoirs et de savoirs. Recherches et expertises pour l'enseignement scientifique. Bruxelles : De Boeck pp. 79–90 (2011)
3. Baron, G.L., Drot-Delange, B. : L'éducation à l'informatique à l'école primaire. 1024 : Bulletin de la Société Informatique de France **8**, 73–79 (Nov 2016), https ://hal.science/hal-01403598
4. CNU 27 : Nomenclature thématique (2013), http ://cnu27.iut2.upmf-grenoble.fr/Qualifications/Nomenclature-2013.html
5. CS Unplugged : À propos - principes, https ://www.csunplugged.org/fr/principles/
6. Delmas-Rigoutsos, Y. : Proposition de structuration historique des concepts de la pensée informatique fondamentale. In : Parriaux, G., Pellet, J.P., Baron, G.L., Bruillard, É., Komis, V. (eds.) Didapro 7 – DidaSTIC. De 0 à 1 ou l'heure de l'informatique à l'école. Lausanne, Switzerland (2018), https ://hal.science/hal-01752797

7. Denning, P.J. : The great principles of computing. In : Pitici, M. (ed.) The Best Writing on Mathematics 2011, pp. 82–92. Princeton University Press (2011). https://doi.org/10.1515/9781400839544.82
8. Dowek, G. : Les quatre concepts de l'informatique. In : Baron, G.L., Bruillard, É., Komis., V. (eds.) Sciences et technologies de l'information et de la communication en milieu éducatif : Analyse de pratiques et enjeux didactiques. pp. 21–29. Athènes : New Technologies Editions (2011), https ://edutice.hal.science/edutice-00676169
9. Drot-Delange, B. : Enseigner l'informatique débranchée : analyse didactique d'activités. In : AREF. pp. 1–13. France (2013), https ://archive-sic.ccsd.cnrs.fr/sic_00955208
10. Fluckiger, Cédric : Une approche didactique de l'informatique scolaire. Paideia, Presses Universitaires de Rennes (2019)
11. Hartmann, W., Näf, M., Reichert, R. : Enseigner l'informatique. Springer (2011). https://doi.org/10.1007/978-2-8178-0262-6
12. Johsua, S., Dupin, J.J. : Introduction à la didactique des sciences et des mathématiques, vol. 327. Presses Universitaires de France, Paris (1993)
13. Nishida, T., Kanemune, S., Idosaka, Y., Namiki, M., Bell, T., Kuno, Y. : A CS unplugged design pattern. In : Proceedings of the 40th ACM technical symposium on Computer science education. pp. 231–235. ACM, Chattanooga TN USA (Mar 2009). https://doi.org/10.1145/1508865.1508951
14. OECD : PISA 2022 Assessment and Analytical Framework. PISA, OECD (2023). https://doi.org/10.1787/dfe0bf9c-en
15. Wing, J.M. : Computational thinking. Commun. ACM **49**(3), 33–35 (2006). https://doi.org/10.1145/1118178.1118215

Deuxième partie

Journée jeunes chercheurs et chercheuses

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Adapting the Programming Language to a Teacher's Didactic Approach

Jesse Hoobergs, KU Leuven

jesse.hoobergs@kuleuven.be

Abstract

When a teacher designs a programming course, they devise the didactic approach that seems most appropriate for the goals and target audience of their course. To implement their course, they will need to choose a programming language. I argue that regardless of the chosen programming language, the implementation of the didactic approach is hampered by the chosen language. In my PhD, I am creating a Programming Education Runtime System that allows teachers to easily create educational languages specifically tailored to their didactic approach. A teacher can create such educational languages by selecting the needed programming language constructs. A construct can be obtained from a library of reusable implementations, or it can be implemented by the teacher themself when a needed construct is not yet available.

1 Short Introduction

I started my PhD in February 2023 at the PL group of Tom Schrijvers at the KU Leuven. Felienne Hermans from the VU Amsterdam is my co-supervisor. The topic of my PhD originated from my experiences in teaching programming to students in both secondary education and higher education. The main goal of my PhD is to allow teachers to change the language to fit their didactic approach instead of them needing to adapt their approach to fit the language.

2 Theoretical Model

A *notional machine* refers to the machine that you are learning to control when learning to program. The term was first used by Du Boulay [3]. It is important to note that each programming language has different features and semantics and therefore a different notional machine. Previous research has implied that it is important to show the appropriate notional machine to novices early on [10]. Appropriate means that the presented notional machine contains the part of the language that can be understood by the novice at that point in time. The presented notional machine will change throughout a course when new programming constructs are explained or known ones are refined. The specific syntax of the language is not part of the notional machine of the programming language. The notional machine of a language and its block-based counterpart, are identical.

In my research, I am currently trying to define two concepts related to the notional machine concept: *notional machine discrepancy* and *programming language inadequacy*. Before explaining these concepts, it is important to stress the distinction between (1) the notional machine of a language and (2) the presented notional machine. The first concept describes a machine corresponding to the entire programming language, while the second concept describes a machine that is explained to students at a *particular point in time*. To make this distinction clear, I will use the following terms: *language machine* and *presented machine*. The presented machine will always be smaller than the language machine and might even be inconsistent with it.

The *notional machine discrepancy* is the difference between the *presented machine* and the *language machine*. It can be seen as the difference between the language that you want to explain at a particular point in time and what your language actually is. To put it more strongly, it is the difference between the machine that you want at that point in time and the machine that you have got. This discrepancy

Volume 1, pages 109–113.

can lead to issues in understanding error messages as these are written in the context of the *language machine*. Error messages are known to be unreadable by novice programmers [10, 11]. Other issues might arise when a novice–maybe accidentally–uses a feature in a way that is not contained within the presented notional machine. It is impossible for the novice to understand the execution of that program within the presented notional machine. Put differently, the *notional machine discrepancy* is about what a novice can understand at a particular moment and about the fact that the language allows things that the novice cannot yet understand at that moment. A novice cannot make the distinction between (1) not understanding code because they did not fully grasp the presented machine or (2) not understanding code because it cannot be understood within the presented machine.

A *programming language inadequacy* occurs when the language makes it hard to explain or use a concept. Making it hard means that the language imposes you to explain additional concepts that you do not want to explain (yet). One example is using a media computation approach [8] in Python. Python does not have a built-in image class, so you need to explain how to import your image library. Another example in Python is using a `for x in range(N)` construct where the concept of default parameter values is actually used. It might didactically be better to only allow the explicit `for x in range(0, N)` when novices first learn to program.

Each *programming language inadequacy* hampers the didactic approach of a teacher because a teacher has to remedy the inadequacy. Currently, I see the following three options from which a teacher can choose to achieve this:

1. Tell the novice to write certain *magic characters* in the source code and tell them that they cannot understand them but should use them.
2. Explain the concept together with the additional concepts.
3. Change the order of concept introduction.

Option 1 will introduce *notional machine discrepancies* as the novices will use constructs that are not in the *presented machine*. Option 2 will probably lead to a very high cognitive load for the novice, which will hamper learning. The last option seems a reasonable approach, but it is not. When a teacher chooses that option, they are changing their didactic approach to match the language. The language makes them adapt their didactic approach to an approach that they would otherwise never use.

A fourth option would be to look for another programming language that does not suffer from this specific *programming language inadequacy*. However, this programming language will in turn have other *programming language inadequacies* that have to be remedied.

3 Related Work

Prior work on mini languages, sub-languages, educational programming languages and teaching languages can be interpreted as reducing the problems of *notional machine discrepancies* and *programming language inadequacies*. Mini languages like Scratch [13] and Logo reduce the *programming language inadequacy* for a specific didactic approach. However, using Scratch to explain concepts that are not part of Scratch, is impossible. Scratch might also suffer a lot from *notional machine discrepancies* as all blocks are always available to novices which makes it easy to use a block that is not yet explained or cannot yet be understood.

Sub-languages like MiniJava [14] and ProfessorJ [7] contain only a subset of the features of another language (in this case Java). Reducing the amount of features might reduce *notional machine discrepancies* for certain didactic approaches. However, they might also increase the *programming language inadequacy* depending on which subset is chosen. These languages will fit a specific didactic approach but are not adaptable to other approaches.

Educational programming languages like Grace [1], Pyret [2], MuLE [4], Quorum [16] and Hedy [9] are full-fledged standalone programming languages. Most of them will have a rather small *programming language inadequacy* for some didactic approaches. Many of these languages contain, for example, data

types to work with images. They might, however, still suffer from quite some *notional machine discrepancies*. Some of these languages (like Hedy [9] and Grace [1]) remedy this by having a gradual approach with different language levels or dialects. These languages are created with a particular didactic approach in mind and adapting them to your own didactic approach is hard. The most well known example of languages tailored to a particular didactic approach are the teaching languages used in *How to design programs* [5].

Prior work on misconceptions can also be useful, as misconceptions could arise from *notional machine discrepancies*. I have looked at the list of misconceptions from Sorva's dissertation [15] and found a couple of misconceptions that could stem from a *notional machine discrepancy*. Unfortunately, it is hard to be sure that the mentioned misconceptions stem from a *notional machine discrepancy* as such a discrepancy inherently contains a time aspect about which machine has been presented and this information is not available in most papers about misconceptions.

4 My solution

All existing languages for education have one common property: they are created with a particular didactic approach in mind. Adapting these languages to a different approach–or even a slightly different approach–is infeasible for most teachers and computing science education researchers. In my research I am creating a Programming Education Runtime System (PERS) that allows the creation of educational programming languages by combining features and programming language constructs.

After devising their didactic approach, teachers can create the language (or languages) for their course by selecting the features from the library of features in the PERS. When a feature / programming language construct is not yet available, they can add it as a reusable component to the PERS. The modular approach of the system allows teachers to easily create gradually changing languages. The second language can be created as an extension of the first language with some new features and the removal of some old features. A reasonable approach would be to create a different language for each lesson or for each chapter or section of a book. These languages will have no *notional machine discrepancy* and no *programming language inadequacy* as they are tailored to the didactic approach and lesson plan.

The modular approach of the system yields the following advantages:

- Code can be reused between different teaching languages.
- Teachers can easily make small tweaks to languages of other teachers: e.g. changing the order of concept introduction.
- Teachers can easily see which features and constructs are added and removed throughout the different levels. This gives the teacher an overview of which notional machine should be presented to students throughout the learning process.

The current version of the PERS has been used to reimplement Hedy [9]. This implementation has many benefits over the existing implementation:

1. executing the language in the browser works out-of-the-box,
2. no notional machine discrepancy,
3. changing the order of concept introduction is trivial,
4. improved debugging support due to step-by-step execution with source location,
5. explicit interaction with the web platform due to the use of algebraic effects [12] in the PERS,
6. creating a Blockly [6] block-based version works out-of-the-box.

The current version of the PERS does, however, also have some shortcomings:

1. the parsing of the modular languages still has to be done manually
2. features cannot yet be removed when extending a language.

5 Next steps

The definitions and naming of the *notional machine discrepancy* and *programming language inadequacy* concepts are still subject to change. I will undertake the following actions to improve and validate these concepts:

- Look at more misconceptions that are reported in literature and determine whether a *notional machine discrepancy* might be the underlying cause.
- Take an in depth look at mini languages, sub-languages and educational programming languages to see how their language design decisions can be explained by the two concepts.
- Observe students in a CS1 course to see whether the *notional machine discrepancy* does lead to issues.
- Interview course developers to get a clear view of the impact of *programming language inadequacy* on their didactic approach.

The PERS system needs to be improved in a number of ways:

- The parsing of the modular languages needs to be implemented.
- The library of implemented features should be extended.
- Allow removing features from the language when extending a language.
- Create a UI system for teachers to create the languages instead of needing to write Rust code.

This second item will be done by implementing some example teaching languages in the system for different didactic approaches to teach programming in Python. One of these approaches will be used in a course text for computer science in Flemish secondary education.

After improving the PERS, it will be evaluated by letting course developers and teachers use the system.

References

[1] Andrew P Black, Kim B Bruce, Michael Homer, James Noble, Amy Ruskin, and Richard Yannow. Seeking grace: a new object-oriented language for novices. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 129–134, 2013.

[2] The Pyret Crew. The pyret programming language.

[3] Benedict Du Boulay. Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1):57–73, 1986.

[4] Nikita Dümmel, Bernhard Westfechtel, and Matthias Ehmann. Mule–a multiparadigm language for education-the procedural sublanguage. In *EDUCON 2020*, pages 392–401. IEEE, 2020.

[5] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. *How to design programs: an introduction to programming and computing*. MIT Press, 2018.

[6] Neil Fraser. Ten things we've learned from blockly. In *2015 IEEE blocks and beyond workshop (blocks and beyond)*, pages 49–50. IEEE, 2015.

[7] Kathryn E. Gray and Matthew Flatt. Professorj: A gradual introduction to java through language levels. In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2003.

[8] Mark Guzdial. A media computation course for non-majors. In *Proceedings of the 8th annual conference on Innovation and technology in computer science education*, pages 104–108, 2003.

[9] Felienne Hermans. Hedy: a gradual language for programming education. In *Proceedings of the 2020 ACM conference on international computing education research*, pages 259–270, 2020.

[10] Shriram Krishnamurthi and Kathi Fisler. 13 programming paradigms and beyond. *The Cambridge handbook of computing education research*, 2019.

[11] Samim Mirhosseini, Austin Z Henley, and Chris Parnin. What is your biggest pain point? an investigation of cs instructor obstacles, workarounds, and desires. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 291–297, 2023.

[12] Gordon D. Plotkin and Matija Pretnar. Handlers of algebraic effects. In Giuseppe Castagna, editor, *Programming Languages and Systems, 18th European Symposium on Programming, ESOP 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5502 of *Lecture Notes in Computer Science*, pages 80–94. Springer, 2009.

[13] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[14] Eric Roberts. An overview of minijava. *SIGCSE Bull.*, feb 2001.

[15] Juha Sorva et al. *Visual program simulation in introductory programming education*. Aalto University, 2012.

[16] Andreas Stefik and Richard Ladner. The quorum programming language. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pages 641–641, 2017.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Analysing Programming Misconceptions

Julien Liénard[0009−0009−2966−7179]

Second year PhD Student, UCLouvain, ICTEAM, Louvain-la-Neuve, Belgium

Keywords: CS Education · Pattern Mining · Automatic Feedback on Autograders · Misconceptions · Programming Flaws

1 Goals of the research

For programming education to become truly accessible to any student, whether they be primary, mid school or university students, students from less privileged socio-economic or cultural backgrounds, female students (an underrepresented audience for computer science studies), adults returning to study, or future informatics teachers themselves, any tool that can help students and their teachers to lower the bar to access and complete these studies is worth exploring. Of particular interest are automated teaching tools adapted to the particular needs, skills and pace of each individual student. Ensuring a good informatics background to students also depends a lot on the motivation, quality and background of the teachers; having tools that can help less skilled teachers provide accurate automated feedback to their students can be very helpful [13]. Such automated tools are also beneficial to allow scaling up to larger student audiences, as is increasingly the case for first year computer science courses, while still being able to guarantee an individualised quality feedback.

Teaching something new to students is at least partly a matter of informing them that some of what they think they know just isn't so [34]. Identifying and addressing students' misconceptions should be a key part of computer science teachers competences [12][30]. Students experience many difficulties for various reasons including unfamiliarity of syntax, incomplete prior knowledge, misconceptions, and lack of problem-solving strategies. Knowing what misconceptions students hold in programming helps educators offer appropriate teaching methods to correct those erroneous conceptions. In this project we will study not only how to discover and encode these typical errors and what misconceptions underly them, but also how to extend existing autograders to provide automated feedback to students on their false conceptions and how to reduce, overcome and correct them.

Existing classifications of common misconceptions by novice programmers (cf. §2) serve as a starting point to build supportive tools that use this information to react to student's (wrong) behaviours and provide feedback tailored to address suspected misconceptions. Exploring the technology, building prototypes of and evaluating such advanced automated teaching tools will be the main goal of this research project. We will first complete and improve upon existing classifications, by applying pattern mining and program analysis to large code

Volume 1, pages 114–120.

repositories of student submissions we have at our disposition [23], as well as by collecting new data from new generations of students. This classification has been used in a first tool that we developed [20]. Our tools creates unit test to detect those coding flaws classified before. It help create more advanced feedback for the student that make common mistakes in the code submitted on the autograder.

2 State of the art

Many authors have studied students' misconceptions and other difficulties in introductory programming [30]. Chiodini proposes a curated inventory of programming language misconceptions [5]; Caceffo proposes the creation of a concept inventory based on such misconceptions [2]; Sorva presents a catalogue of misconceptions, drawn through exploratory research from actual novice programmers' misconceptions [32]; and Grover also categorises common mistakes and misconceptions made by novice programmers [12]. Several other authors have studied possible misconceptions and types of errors made by novice students in introductory programming education as well [1][31][17][6][18][35][19][33]. Such taxonomies will serve as our basis to build supportive tools that use this information to react to student's (wrong) behaviours and provide feedback tailored to address suspected misconceptions.

Research on source code mining [24][29] has been explored to discover interesting structural regularities [21], coding idioms [28], API usage patterns [36], code clones [8], bugs [4], crosscutting concerns [3], systematic changes [11][26]. refactoring opportunities, etc.

In a recent article [23] which formed the seed for the current research project, we conducted an initial experiment where we used a frequent subtree pattern mining algorithm to mine for interesting good, bad or ugly coding idioms made by over 500 undergraduate students from their exam submissions to an introductory programming course. We did so by looking for patterns that distinguished positive examples, corresponding to the more correct answers to each question, from negative examples, corresponding to solutions that failed the question. That paper was a first promising exploratory step to prove the potential of using this technique to find possible misconceptions (corresponding to the bad coding idioms that were discovered).

Many automatic graders [14][16], autograders in short, have been proposed to grade programs submitted by students, either to scale up to larger audiences through automation [25][7][10] or to provide alternative forms of feedback [27][22][9][15]. In this project we will explore how extend the INGInious autograder [10] with information on mined misconceptions to tell students not only where their code is wrong and what tests it doesn't pass, but also why it is wrong and how it could be improved. Even students who solved a question correctly could receive recommendations on how to improve their coding skills.

3 Research project

In this research project we will investigate the following research questions:

1. What are the typical programming mistakes made by novice programmers?
2. What misconceptions do these typical mistakes correspond to?
3. What recommendations can help students overcome such misconceptions?
4. How to automate the detection of such misconceptions and their remediation?
5. How do such automated tools help students and their teachers?

To answer the first research questions, during my first year as a PhD student,, we used a pattern miner to detect common mistakes in first year computer science exam.

The repository we used is the set of programs submitted by students following the first-year introductory programming course for computer scientists and engineers at UCLouvain. This course is followed by over 700 students yearly and makes use of an autograder which collects and corrects students' programs throughout the year and during exam sessions. The course has been running for 4 years using Python as programming language and before that using the Java language. All this data is available for analysis. For all the submissions we even have a trace of all subsequent submissions each student made per question. As outcome of this analysis, we found recurring errors and misconceptions.

For the second research question, research literature contains a vast body of knowledge on programming misconceptions, including several classifications and taxonomies of typical programming errors and their corresponding misconceptions. However, many of those are still incomplete or cover only some of the programming concepts for some programming languages. We used these classifications as a starting point on which to build our own.

Research questions 3 and 4 are dedicated to the development of our automated tool support. For this we will build upon and extend the existing INGInious autograder that was developed at our university. Since it was developed in house, the necessary expertise is available to extend it. As the autograder is already used by our students, the developed tool support will be integrated in an environment that they are already familiar with.

I have integrated a new feature into this autograders to identify a wider range of errors beyond just syntax and input-output issues. This includes detecting conceptual errors, semantic errors, and poor coding practices [20].

Following this initial investigation, during this second year, we have recognized the potential benefit of introducing a user-friendly language for defining code patterns that can be identified within students' work. This development aims to assist educators who may not possess an extensive background in computer science. Our ongoing research on this topic will be the subject of a forthcoming paper.

A quantitative evaluations has been conducted by comparing the performance of students using our tools support with those that are not. We also want to evaluate their performance when using our tools as compared to using similar

tools if they exist. We will also assess whether our automated tool support is helpful for students and their teachers is by evaluating it through them. Qualitative evaluations will be conducted through interviews with volunteering tutors and students involved in our first-year programming course. Their feedback will help us understand whether the tool meets its objectives and steer its further development and improvement if not. Such evaluations will happen at regular intervals throughout the project rather than only at the end.

We also want to try to use more variety of techniques (such as formal concept analysis, association rule mining, frequent subtree mining and clustering) to mine for good or bad coding idioms in student code, in order to reveal potential misconceptions.

Another outcome will be a comparative study of what techniques or combinations of techniques prove most useful to mine for such errors and idioms, thus contributing to the growing research domain of mining software repositories.

This entails conducting iterative research in which we aim to uncover increasing numbers of misconceptions through techniques like pattern mining and others. Subsequently, we will employ these methods to develop tools and evaluate their effectiveness with our students on an annual basis.

References

1. Bayman, P., Mayer, R.E.: A diagnosis of beginning programmers' misconceptions of BASIC programming statements. Communications of the ACM **26**(9), 677–679 (Sep 1983). https://doi.org/10.1145/358172.358408, https://doi.org/10.1145/358172.358408
2. Caceffo, R., Frank-Bolton, P., Souza, R., Azevedo, R.: Identifying and Validating Java Misconceptions Toward a CS1 Concept Inventory. In: Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education. pp. 23–29. ITiCSE '19, Association for Computing Machinery, New York, NY, USA (Jul 2019). https://doi.org/10.1145/3304221.3319771, https://doi.org/10.1145/3304221.3319771
3. Ceccato, M., Marin, M., Mens, K., Moonen, L., Tonella, P., Tourwe, T.: A qualitative comparison of three aspect mining techniques. In: 13th International Workshop on Program Comprehension (IWPC'05). pp. 13–22 (May 2005). https://doi.org/10.1109/WPC.2005.2, iSSN: 1092-8138
4. ChandraYadav, D., Pal, S.: Software Bug Detection using Data Mining. International Journal of Computer Applications **115**(15), 21–25 (Apr 2015). https://doi.org/10.5120/20228-2513, http://research.ijcaonline.org/volume115/number15/pxc3902513.pdf
5. Chiodini, L., Moreno Santos, I., Gallidabino, A., Tafliovich, A., Santos, A.L., Hauswirth, M.: A Curated Inventory of Programming Language Misconceptions. In: Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1. pp. 380–386. ITiCSE '21, Association for Computing Machinery, New York, NY, USA (Jun 2021). https://doi.org/10.1145/3430665.3456343, https://doi.org/10.1145/3430665.3456343

6. Danielsiek, H., Paul, W., Vahrenhold, J.: Detecting and understanding students' misconceptions related to algorithms and data structures. In: Proceedings of the 43rd ACM technical symposium on Computer Science Education. pp. 21–26. SIGCSE '12, Association for Computing Machinery, New York, NY, USA (Feb 2012). https://doi.org/10.1145/2157136.2157148, https://doi.org/10.1145/2157136.2157148
7. Danutama, K., Liem, I.: Scalable Autograder and LMS Integration. Procedia Technology **11**, 388–395 (Jan 2013). https://doi.org/10.1016/j.protcy.2013.12.207, https://www.sciencedirect.com/science/article/pii/S2212017313003617
8. Deknop, C., Mens, K., Baars, S., Oprescu, A.: Clone Detection vs. Pattern Mining: The Battle p. 7
9. DeNero, J., Sridhara, S., Pérez-Quiñones, M., Nayak, A., Leong, B.: Beyond Autograding: Advances in Student Feedback Platforms. In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education. pp. 651–652. SIGCSE '17, Association for Computing Machinery, New York, NY, USA (Mar 2017). https://doi.org/10.1145/3017680.3017686, https://doi.org/10.1145/3017680.3017686
10. Derval, G., Gego, A., Reinbold, P., Frantzen, B., Van Roy, P.: Automatic grading of programming exercises in a mooc using the inginious platform. European Stakeholder Summit on experiences and best practices in and around MOOCs (EMOOCS'15) pp. 86–91 (2015)
11. Gerlec, , Krajnc, A., Heričko, M., Božnik, J.: Mining source code changes from software repositories. In: 2011 7th Central and Eastern European Software Engineering Conference (CEE-SECR). pp. 1–5 (Oct 2011). https://doi.org/10.1109/CEE-SECR.2011.6188468
12. Grover, S., Basu, S.: Measuring Student Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables, and Boolean Logic. In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education. pp. 267–272. ACM, Seattle Washington USA (Mar 2017). https://doi.org/10.1145/3017680.3017723, https://dl.acm.org/doi/10.1145/3017680.3017723
13. Henry, J., Joris, N.: Informatics at secondary schools in the french-speaking region of belgium: myth or reality. ISSEP 2016 **13** (2016)
14. Hollingsworth, J.: Automatic graders for programming classes. Communications of the ACM **3**(10), 528–529 (Oct 1960). https://doi.org/10.1145/367415.367422, https://doi.org/10.1145/367415.367422
15. Iosup, A., Epema, D.: An experience report on using gamification in technical higher education. In: Proceedings of the 45th ACM technical symposium on Computer science education. pp. 27–32. SIGCSE '14, Association for Computing Machinery, New York, NY, USA (Mar 2014). https://doi.org/10.1145/2538862.2538899, https://doi.org/10.1145/2538862.2538899
16. Jackson, D., Usher, M.: Grading student programs using ASSYST. In: Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education. pp. 335–339. SIGCSE '97, Association for Computing Machinery, New York, NY, USA (Mar 1997). https://doi.org/10.1145/268084.268210, https://doi.org/10.1145/268084.268210
17. Kaczmarczyk, L.C., Petrick, E.R., East, J.P., Herman, G.L.: Identifying student misconceptions of programming. In: Proceedings of the 41st ACM technical symposium on Computer science education. pp.

107–111. SIGCSE '10, Association for Computing Machinery, New York, NY, USA (Mar 2010). https://doi.org/10.1145/1734263.1734299, https://doi.org/10.1145/1734263.1734299
18. Karpierz, K., Wolfman, S.A.: Misconceptions and concept inventory questions for binary search trees and hash tables. In: Proceedings of the 45th ACM technical symposium on Computer science education. pp. 109–114. SIGCSE '14, Association for Computing Machinery, New York, NY, USA (Mar 2014). https://doi.org/10.1145/2538862.2538902, https://doi.org/10.1145/2538862.2538902
19. Kohn, T.: Variable Evaluation: an Exploration of Novice Programmers' Understanding and Common Misconceptions. In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education. pp. 345–350. SIGCSE '17, Association for Computing Machinery, New York, NY, USA (Mar 2017). https://doi.org/10.1145/3017680.3017724, https://doi.org/10.1145/3017680.3017724
20. Lienard, J., Mens, K., Nijssen, S.: Extracting unit tests from patterns mined in student code to provide improved feedback in autograders. In: Seminar Series on Advanced Techniques & Tools for Software Evolution (SATToSE) (2023)
21. Lozano, A., Kellens, A., Mens, K., Arevalo, G.: Mining Source Code for Structural Regularities. In: 2010 17th Working Conference on Reverse Engineering. pp. 22–31 (Oct 2010). https://doi.org/10.1109/WCRE.2010.12, iSSN: 2375-5369
22. MacWilliam, T., Malan, D.J.: Streamlining grading toward better feedback. In: Proceedings of the 18th ACM conference on Innovation and technology in computer science education. pp. 147–152. ITiCSE '13, Association for Computing Machinery, New York, NY, USA (Jul 2013). https://doi.org/10.1145/2462476.2462506, https://doi.org/10.1145/2462476.2462506
23. Mens, K., Nijssen, S., Pham, H.S.: The good, the bad, and the ugly: mining for patterns in student source code. In: Proceedings of the 3rd International Workshop on Education through Advanced Software Engineering and Artificial Intelligence. pp. 1–8. EASEAI 2021, Association for Computing Machinery, New York, NY, USA (Aug 2021). https://doi.org/10.1145/3472673.3473958, https://doi.org/10.1145/3472673.3473958
24. Mens, K., Tourwé, T.: Delving source code with formal concept analysis. Computer Languages, Systems & Structures **31**(3), 183–197 (Oct 2005). https://doi.org/10.1016/j.cl.2004.11.004, https://www.sciencedirect.com/science/article/pii/S1477842405000059
25. Milojicic, D.: Autograding in the Cloud: Interview with David O'Hallaron. IEEE Internet Computing **15**(1), 9–12 (Jan 2011). https://doi.org/10.1109/MIC.2011.2, conference Name: IEEE Internet Computing
26. Molderez, T., Stevens, R., De Roover, C.: Mining change histories for unknown systematic edits. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). pp. 248–256. IEEE (2017)
27. Nicol, D.: From monologue to dialogue: improving written feedback processes in mass higher education. Assessment & Evaluation in Higher Education **35**(5), 501–517 (Aug 2010). https://doi.org/10.1080/02602931003786559, https://doi.org/10.1080/02602931003786559, publisher: Routledge _eprint: https://doi.org/10.1080/02602931003786559
28. Nucci, D.D., Pham, H.S., Fabry, J., Mens, K., Molderez, T., Nijssen, S., Roover, C.D., Zaytsev, V.: Language-Parametric Modular Framework for Mining Idiomatic Code Patterns p. 7

29. Pham, H.S., Nijssen, S., Mens, K., Di Nucci, D., Molderez, T., De Roover, C., Fabry, J., Zaytsev, V.: Mining Patterns in Source Code Using Tree Mining Algorithms. In: Kralj Novak, P., Šmuc, T., Džeroski, S. (eds.) Discovery Science. pp. 471–480. Lecture Notes in Computer Science, Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-33778-0$_3$5
30. Qian, Y., Lehman, J.: Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. ACM Transactions on Computing Education **18**(1), 1:1–1:24 (Oct 2017). https://doi.org/10.1145/3077618, https://doi.org/10.1145/3077618
31. Seppälä, O., Korhonen, A., Malmi, L.: Observations on student errors in algorithm simulation exercises. Koli Calling, 17-20.11.2005, Koli, Suomi pp. 81–86 (2005), https://research.aalto.fi/en/publications/observations-on-student-errors-in-algorithm-simulation-exercises, publisher: TURKU CENTRE FOR COMPUTER SCIENCE
32. Sorva, J.: Visual program simulation in introductory programming education. Aalto University (2012), https://aaltodoc.aalto.fi:443/handle/123456789/3534, accepted: 2012-05-31T07:29:53Z ISSN: 1799-4942 (electronic)
33. Swidan, A., Hermans, F., Smit, M.: Programming Misconceptions for School Students. In: Proceedings of the 2018 ACM Conference on International Computing Education Research. pp. 151–159. ICER '18, Association for Computing Machinery, New York, NY, USA (Aug 2018). https://doi.org/10.1145/3230977.3230995, https://doi.org/10.1145/3230977.3230995
34. Taylor, A.K., Kowalski, P.: Student misconceptions: Where do they come from and what can we do? In: Applying science of learning in education: Infusing psychological science into the curriculum, pp. 259–273. Society for the Teaching of Psychology, Washington, DC, US (2014)
35. Veerasamy, A.K., D'Souza, D., Laakso, M.J.: Identifying Novice Student Programming Misconceptions and Errors From Summative Assessments. Journal of Educational Technology Systems **45**(1), 50–73 (Sep 2016). https://doi.org/10.1177/0047239515627263, https://doi.org/10.1177/0047239515627263, publisher: SAGE Publications Inc
36. Zhong, H., Xie, T., Zhang, L., Pei, J., Mei, H.: MAPO: Mining and Recommending API Usage Patterns. In: Drossopoulou, S. (ed.) ECOOP 2009 – Object-Oriented Programming. pp. 318–343. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03013-0$_1$5

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Computational Thinking Integrated within a Social Robot Project

Natacha Gesquière[1][0000−0002−5577−5124] and Seppe Hermans[2][0000−0001−8375−7831]

[1] Ghent University – imec, IDLab-AIRO, Belgium
natacha.gesquiere@ugent.be
[2] Department of Training and Educational Science, University of Antwerp, Antwerp, Belgium
seppe.hermans@uantwerpen.be

Abstract. This case study explores the integration of computational thinking into the mandatory STEM curriculum in Flanders. It investigates the execution of the 'Social Robot' project in three separate classes within the Flemish education system, each representing a distinct study track. These tracks include general education with a strong emphasis on STEM, vocational education with a STEM focus, and vocational education without a specific STEM focus. Initial findings reveal variations in student interest, autonomous motivation, and self-efficacy among these three classes. In contrast to students in vocational education classes, those in the general education class display a heightened interest, perceiving the project as both more challenging and engaging. This case study provides valuable insights into the intricacies of integrating computational thinking within STEM education, emphasizing the necessity for customized computational thinking initiatives, and underlining the importance of infusing creativity and real-world relevance to enhance student motivation and learning outcomes.

Keywords: computational thinking · STEM · physical computing · Flanders · Belgium · K-12 · education.

1 Introduction

1.1 Computational Thinking and Integrated STEM

Since September 2019 minimum goals[3] on computational thinking (CT) and STEM (Science, Technology, Engineering, and Mathematics) have been part of the compulsory curriculum in Flanders. One way to address these objectives is by integrating CT into STEM projects. The integration of computational thinking within STEM subjects is driven by the recognition of the fundamental importance of STEM competencies for economic growth and the preparation of the future workforce to meet the challenges of global competitiveness [1].

[3] https://onderwijsdoelen.be, in Dutch

Volume 1, pages 121–125.

Equally important, CT is a way of understanding and acting upon the digital world; a basic skill in CT enhances one's ability to understand and interact with technological developments, which can counteract the fear of technology [2].

Integrated STEM education (iSTEM) has emerged as a way to remove the perceived barriers between the four disciplines, enabling pupils to understand the relevance of STEM to address real-world technical and societal issues. Introducing CT into compulsory education aims that pupils acquire the skills of CT and to explore what they mean for the various disciplines [3, 4].

However, the successful implementation of iSTEM education is a multifaceted endeavor that extends beyond mere parallel teaching of these subjects [5]. This challenging task is an obstacle many teachers experience in adopting an integrated STEM methodology [6, 7]. Furthermore, teaching CT and programming presents its own set of challenges for educators [8, 9]. Some teachers tend to choose ready-made assignments that often lack opportunities for open-ended experimentation that are characteristic of iSTEM pedagogy but far more challenging to coach [10]. As integrating CT into STEM projects is new for many teachers, the 'Social Robot' project was developed.

1.2 Social Robot Project

The 'Social Robot' project aims to address the new learning objectives on CT and STEM through an exciting theme that also incorporates learning goals on privacy and societal issues. The implementation of the project was carried out by high school teachers instructing pupils in grades 9 or 10; the pupils were in different fields of study within both general and vocational education. The overarching goal was to empower teachers to navigate open-ended problems and enhance their confidence in the domain of physical computing. Simultaneously, pupils had the opportunity to engage in a programming task that stimulated their creativity and imagination, and prompted them to contemplate the applicability of their work in real-world contexts. To support the teachers, a comprehensive set of resources[4] was made available, such as online materials designed to provide insights into the cutting-edge realms of social robotics and human-robot interaction (HRI) [11]. The teachers harnessed a customized robotics kit, allowing their pupils to collaboratively build personalised social robots.

1.3 Our Study

Given the relatively recent introduction of CT within the Flemish education context, there is a limited understanding of how pupils and teachers perceive and experience CT when integrated into STEM projects like the 'Social Robot' project. This case study seeks to fill this knowledge gap. Following research questions serve as guiding pillars for our endeavor, shaping the direction and focus of our study:

[4] https://www.dwengo.org/en/socialrobot

1. How do high school pupils perceive and experience the integration of CT in the context of the 'Social Robot' project?
2. To what extent do high school teachers feel prepared to implement CT as part of iSTEM projects, specifically within the 'Social Robot' project, and what challenges do they encounter in the process?
3. What are the key lessons learned from the implementation of CT within iSTEM education, and how do they inform the future development of CT initiatives?

2 Methodology

In line with our research questions, we have adopted a mixed-methods approach. This approach allows us to triangulate findings and gain insights from both qualitative and quantitative data sources. In this study, we have two key participant groups. Five high school teachers with diverse subject expertise that actively took part in the 'Social Robot' project. A total of seventy-two high school students all of whom participated in the 'Social Robot' project.

2.1 Instruments and Analysis

Surveys: We administered surveys to high school pupils to gauge their perceptions, experiences, and motivation in relation to CT within the project. The survey included questions regarding their CT competencies, levels of interest, and perceived impact on their understanding of CT and STEM principles. Semi-Structured Interviews: High school teachers who have implemented the 'Social Robot' project were interviewed to provide qualitative insights into their experiences. These interviews explored the challenges faced, their level of preparedness for CT integration, and their observations regarding pupils' engagement. The survey instrument employed in this study has been previously validated in a separate research effort [12]. Given the relatively small sample size in this study, our analysis primarily relies on descriptive statistics, correlation values, and qualitative analysis. These analytical methods enable a comprehensive exploration of the collected data. The semi-structured interviews conducted with the participants were meticulously transcribed and subjected to qualitative content analysis. This approach allows for in-depth exploration of the interview data, unveiling nuanced insights and perspectives. Both the survey data and interview data are integrated in our analysis.

3 Results

Table 1 presents partial results from our study: data from pupils in three classes: one in general education and two in vocational education and training (VET). The class in general education is situated in a science field, thus STEM. The first VET class is also in a STEM domain, while the second VET class belongs to a non-STEM field.

Table 1. Pupils' Self-Reporting on the 'Social Robot' Project.

	Classes		
Constructs	**(VET non-STEM)**	**(General STEM)**	**(VET STEM)**
Interest in STEM (1-4)	2.25	3.38	2.92
Confidence in STEM (1-4)	2.67	2.90	3.16
Abstraction (1-5)	3.22	3.08	3.04
Decomposition (1-5)	3.67	2.75	3.22
Pattern recognition (1-5)	3.00	2.88	3.39
Generalisation (1-5)	3.67	3.56	3.36
Algorithmic thinking (1-5)	3.00	3.38	3.39
Evaluation (1-5)	3.08	3.94	3.61
Autonomous motivation (1-4)	2.63	3.09	3.08
Controlled motivation (1-4)	2.13	2.91	2.63
Amotivation (1-4)	2.42	1.50	2.25
Fun project (0-10)	3.00	8.00	6.78
Interesting project (0-10)	4.00	8.25	6.44
Difficulty project (0-10)	4.67	5.50	3.22

4 Discussion and Conclusion

Our preliminary findings reveal notable variations in the perceptions and experiences of high school pupils in the Flemish education system as they engage with the integration of CT within the 'Social Robot' project. Importantly, the study indicates that pupils' interest STEM varies: specifically, pupils in the general education class exhibit a higher level of interest, while the non-STEM pupils express the least interest. The pupils in the non-STEM class also show the least self-efficacy in STEM.

On average, pupils from all three classes perceive their CT competencies similarly. If we look closer at the concepts of CT, we see that STEM pupils find decomposition more difficult than algorithmic thinking, while for pupils in the non-STEM class it is just the other way around. However, a noteworthy observation is that pupils in the general education class may underestimate their abstraction and pattern recognition skills, potentially linked to their greater exposure to abstract thinking. This underscores the need to ensure that pupils are aware of their own CT competencies and highlights the significance of accurate self-assessment. Autonomous motivation seems to differ between pupils in the STEM classes and the ones in non-STEM: the pupils in the non-STEM class show lesser autonomous motivation than their peers in STEM. The pupils' interest in the project corresponds to their overall interest in STEM. The pupils in non-STEM seemed to not like the project, which stands out considering the creativity and social aspects in the context of the project. Notably, pupils in the general education class rate the project as more challenging than their VET peers. This observation may be associated with the potential gap in technical skill training. However, these pupils find the project more fun and interesting, suggesting a nuanced relationship between challenge and engagement. This find-

ing emphasizes the importance of tailoring CT and STEM initiatives to align with the diverse interests and backgrounds of students across study tracks. In conclusion, these findings highlight the multifaceted nature of CT integration in STEM education, emphasizing the need for a holistic approach. It is important to note that the qualitative nature of this study means that the results cannot be generalised to broader populations. However, the insights gained have generated valuable knowledge that warrants further in-depth research.

References

1. Kennedy, T., Odell, M. R. L.: Engaging pupils In STEM Education. Science education international, **25**, 246-258 (2014)
2. Bocconi, S., Chioccariello, A., Dettori, G. et al.: Developing Computational Thinking in Compulsory Education - Implications for policy and practice. EUR 28295 EN. Luxembourg: Publications Office of the European Union (2016)
3. Barr, V., Stephenson, C.: 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 (2011)
4. Hemmendinger, D.: A Plea for Modesty. ACM Inroads, **1**(2), 4–7 (2010)
5. Knipprath, H., Thibaut, L., Buyse, M. P., Ceuppens, S., Loof, H. D., De Meester, J., Goovaerts, L., Struyf, A., De Pauw, J. B., Depaepe, F., Deprez, J., De Cock, M., Hellinckx, L., Langie, G., Struyven, K., Van de Velde, D., Van Petegem, P., Dehaene, W.: STEM Education in Flanders: How STEM@school Aims to Foster STEM Literacy and a Positive Attitude towards STEM. IEEE Instrumentation & Measurement Magazine, **21**(3), 36-40 (2018)
6. Margot, K. C., Kettler, T.: Teachers' perception of STEM integration and education: a systematic literature review. International Journal of STEM education,**6**(1), 1-16 (2019)
7. Thibaut, L., Ceuppens, S., De Loof, H., De Meester, J., Goovaerts, L., Struyf, A., Boeve-de Pauw, J., Dehaene, W., Deprez, J., De Cock, M.: Integrated STEM education: A systematic review of instructional practices in secondary education. European Journal of STEM Education, **3**(1), 2 (2018)
8. Benitti, F. B. V. : Exploring the educational potential of robotics in schools: A systematic review. Computers & Education, **58**(3): 978-988, 2012.
9. wyffels, F., Martens, B., Lemmens, S.: Starting from scratch: experimenting with computer science in emish secondary education. In Proceedings of the 9th Workshop in Primary and Secondary Computing Education, WiPSCE '14, pages 12-15. Association for Computing Machinery. New York, NY, USA (2014)
10. Neutens, T., wyffels, F.: Unsupervised functional analysis of graphical programs for physical computing. In: 2020 International Conference on Computational Science and Computational Intelligence (CSCI), IEEE, pages 1-6 (2021)
11. Van de Staey, Z., Gesquière, N., wyffels, F.: A social robot activity for novice programmers. In M. Merdan, W. Lepuschitz, G. Koppensteiner, R. Balogh, D. Obdržálek (Eds.), RIE 2021, 12th International Conference on Robotics in Education, Proceedings, **1359**: pages 72–77 (2022)
12. Hermans, S., wyffels, F., Van Petegem, P.: Assessing Computational Thinking: A Validation of Computational Thinking Measures. Manuscript submitted for publication in the Journal of Science Education and Technology (2023).

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Computational thinking competencies of Flemish college students: vision on data collection

Willem Lapage*
willem.lapage@ugent.be

Tom Neutens
tom.neutens@ugent.be

Francis wyffels
francis.wyffels@ugent.be

IDLab, Department of Information Technology, Ghent University - imec, Ghent, Oost-Vlaanderen, 9000, Belgium

October 20, 2023

Abstract

Computational thinking has become an increasingly vital competence in our technologically driven world. As a problem-solving methodology, it can be considered a competence that transcends disciplines and plays an important part in multiple diverse fields. It has also gained a more prominent role in the Flemish education system. Therefore, assessing computational thinking and collecting the necessary data to do so has become increasingly important during students' education. This paper describes how the computational thinking competencies of college students can be monitored in a controlled environment. By combining a literature study as well as knowledge of the context wherein the data will be collected, a subset of data sources has been selected that show potential for a multimodal assessment of computational thinking. This paper outlines an envisioned data collection method to gauge computational thinking competencies among second-year computer science engineering students at Ghent University. The desired end result is a collection of data that can be managed and processed as an input source to assess computational thinking and affect educational practices. This paper describes a way of collecting data that shows potential for a multimodal assessment of computational thinking. It also opens the door for future research exploring the potential of AI-driven methods for automatic assessment and the development of interactive visualisation of said assessments.

1 Introduction

Computational thinking (CT) is a prerequisite for understanding and managing the possibilities, opportunities, consequences and risks of the digitisation of our world, including in the realm of information and communication [1]. This competence transcends disciplinary boundaries, becoming a fundamental competence across diverse fields [3]. In the context of Flemish secondary education, CT can be characterized as a process that involves six subconcepts; pattern recognition, generalisation, problem decomposition and abstraction, modelling of solutions and algorithm usage.[1, 4] It offers a better approach to tackling complex problems and empowers students to leverage computers in solving these challenges [1]. For each of these subconcepts, it's also important to properly apply them. Students may for example want to apply decomposition and split a problem into parts. While different approaches might initially lead to similar results, improper decoupling of responsibilities and consequently less effective decomposition might lead to future problems [8]. Nurturing this competence is thus of paramount importance, as individuals possessing it may be better positioned to leverage the ubiquitous computing in this world [5]. Current adaptive educational technologies aim to personalize learning experiences by leveraging the measurement, collection, analysis, and reporting of multimodal cognitive, metacognitive, affective, and motivational data [2]. However, the use of multimodal online

*First year PhD. student

Volume 1, pages 126–130.

trace data like log files or contextual data such as voice recordings can enhance our understanding of the learning process and provide a more comprehensive view of both the learner and the learning environment [2, 6]. A better understanding of this process and, consequently, a more context-sensitive approach can be very beneficial for educational technologies if they want to achieve educational goals. These technologies must evolve to become context-sensitive and thus more responsive to learners' characteristics and situations. Remaining stagnant and only working in terms of narrow cognitive attributes could lead to their ineffectiveness in achieving these goals [9].

Assessing CT with the aid of artificial intelligence holds significant promise, as it can assist educators to engage and support their students better. It equips educators with the ability to adapt to students' learning processes, receive suggested resources, and interact with computational resources [9]. Moreover, it could alleviate the burden on teachers when it comes to identifying learning challenges during in-lecture exercises, especially when dealing with large class sizes or time constraints [10]. The current lack of tools to assess the development of CT can partly be attributed to the lack of a unified definition and consensus on its incorporation into educational curricula [7].

This paper focuses on describing how we envision gauging and mapping computation thinking competencies among second-year engineering technology students at Ghent University during a computer science course. It outlines data collection methods as well as assessment instruments to map out student performance. This will result in a comprehensive dataset that is both manageable and processable while showing potential as an input source for a multi-modal assessment of CT. The envisioned end result encompasses diverse data types, including plain text, diagrams, and audio files. By combining these varied data sources, the dataset is intended to serve as input for artificial intelligence models, showcasing its desired potential for an automatic multi-modal evaluation of CT.

2 Methodology

The method we present revolves mainly around the selection of data inputs. This to accomplish our goal of constructing a diverse and comprehensive dataset with potential as an input source for a multi-modal assessment of CT. The selection process is driven by multiple factors such as the context of the data and the feasibility of using these data types as input for artificial intelligence models.

2.1 Educational context

The data will be collected in a Flemish educational context and take into account that the collected data and found results must be applicable to said context. Given the Flemish government's six-fold definition of CT; we considered data collection and assessment for the given concepts in this context to be a good representation of doing it for CT as a whole. This definition, however, is aimed at year 7 and year 8 in K-12 while we chose a computer science course at Ghent University. We made this decision because we can exert a high level of influence as supervisors and ensure a very controlled environment. This provides us with a method of mapping out and collecting different sorts of possible input data that could also be applied in K-12.

2.2 Course context

We will, as previously mentioned, collect data from students at Ghent University. To be more specific, all data will be collected during a second-year course called "Engineering Project" in the program "Bachelor of Science in Engineering (Computer Science Engineering)" in the academic school year 2023-2024. The lessons revolve around basic microcontroller programming. This course is estimated to have a workload of 75 to 90 hours spread over a combination of 11 lessons as well as a project students have to present by the end of the course. The lessons of this course are planned over 15 weeks and are primarily focused on giving students practical experience with microcontrollers. All theoretical knowledge is shared online at the start of the course through written text and video-based explanations. Students are expected to independently study this material by the third lesson, but questions can be asked throughout the course.

The 11 lessons consist of the following:

- Lesson 1: Receiving and testing of microcontroller platform.

- Lesson 2-3: Supervised practical exercises with microcontrollers.
- Lesson 4-11: Supervised time to work on the project.

Data collection will revolve exclusively around the project, as it exhibits the complexity required to assess all the components that define the Flemish government's CT description.

For the project, students are organized into pairs, with each pair programming their project using the Dwenguino platform, a microcontroller platform developed by Dwengo vzw. The Dwenguino platform integrates an AT90USB646 microcontroller and supports hardware components such as an LCD, LEDs, buttons, and a motor driver. To program the Dwenguino, students are mandated to use Visual Studio Code and PlatformIO, utilising AVR libraries and the provided datasheet. During the project, students are free to ask questions to supervisors and have to perform a stand-up to describe current progress, plans and challenges they encounter. The microcontroller platform is accessible to students only during the supervised lessons and cannot be taken home. Regarding the subject of the project, there are three choices;

1. Drawing table: In this project, students have access to a planar "robot arm" with two degrees of freedom, controlled by a servo motor. The goal is to use it to visualize complex figures on paper.
2. Accelerometer: The objective is to create a game where interaction is facilitated by the movement of the Dwenguino platform, with the board's movement measured using an accelerometer.
3. Infrared remote controller: In this project, students will establish wireless communication between two Dwenguino platforms using infrared light. For this, they will need to create their own communication protocol.

3 Data modalities to enable automatic assessment of computational thinking

As previously mentioned, our research aims to assemble a diverse dataset, spanning from plain text to diagrams and audio files. This selection process is guided by the Flemish educational context and the practicality of employing these data types as inputs for artificial intelligence models. As a result, we have chosen to collect the following data:

Diagrams to show the structure of the projects: Diagrams, created using draw.io, offer insights into how students apply CT concepts like problem decomposition, abstraction, and modelling. They showcase the students' ability to break down complex problems into manageable parts, conceptualize these components, and demonstrate how they interconnect. It also reveals the granularity students employ when applying these concepts.

Audio recordings of the weekly stand-up sessions: These depict how students plan their projects, as well as the challenges they face and the solutions they find. Decomposition, abstraction and modelling can be found in the way they solve the project as a whole by dividing it into parts and planning how they will solve these. Students may also notice and describe how some parts of the problem follow a pattern or that solutions or algorithms for a more generic, but relevant problem are applicable. Depending on the stand-up they may contain information about every subconcept used to describe CT.

Reports detailing the end result: These reports offer a written summary of implemented functionality, optimisation efforts, challenges faced, and their respective solutions. They serve as an extension of the audio recordings and a consolidated overview of the entire project, providing insights into students' problem-solving and CT processes.

Implementation of the project solutions: The written code represents the practical realisation of a group's solution. It encapsulates all the CT concepts, serving as a tangible manifestation of the drawn diagrams and planned solutions. The use of GitHub allows us to track how the code evolves, providing valuable insights into students' problem-solving progress.

Kanban boards: A kanban board, provided by GitHub, can be used to visually depict each stage of a project. During the project students are advised to use three stages; "To Do", "In progress" and "Done". Students use this tool to organize their projects, creating issues that detail project components that have to be implemented and problems to be solved. The Kanban board showcases students' step-wise planning and how it translates into practical solutions.

Our choice of these diverse data sources not only encompasses each subcomponent of CT but also leverages that the same underlying concepts can be discerned across different information and data inputs. This multifaceted approach also demonstrates how aspects of the project are addressed across various data inputs, emphasising the importance of gathering data from different sources to provide a more comprehensive context for understanding CT. Additionally, these data inputs serve as feasible inputs for artificial intelligence models, making them applicable for automatic CT assessments.

This data will only be collected from students who received an information letter and signed a consent form.

4 Conclusion

In conclusion, this research has unveiled our comprehensive vision for collecting and processing data to assess CT while underlining the pivotal role of incorporating context into the assessment process. Nevertheless, the practical implementation of this vision stands as a crucial next phase. Data collection and processing represent only the initial steps in the assessment process. Once the data has been cleaned and prepared, the focus must transition to automatically assessing the various subcomponents of CT. Identifying appropriate models and fine-tuning them constitutes an indispensable element of this process. Moreover, validating the automatic assessment by comparing it to human-made assessments will be an ongoing endeavour, fostering improvements in assessment methodologies.

Another crucial facet of our future work is the visualization of assessment data on a dashboard. To ascertain the most effective visualization methods for educators and their teaching practices, insights gathered through interviews with educators and a comprehensive literature study will serve as valuable guidance. These inputs will steer the development of visualizations that can empower teachers in their teaching activities.

As the educational landscape continues to evolve, the importance of assessing CT within a multifaceted context becomes increasingly evident. With the right data sources, methods, and visualization techniques, we have the opportunity to not only advance the understanding of CT but also to provide educators with practical tools to enhance their teaching practices. This work, in its essence, represents a stepping stone towards a more nuanced and context-sensitive approach to assessing and fostering CT, contributing to the broader effort of equipping students with essential competencies for the digital age.

References

[1] Agentschap voor Hoger Onderwijs, Volwassenenonderwijs, Kwalificaties en Studietoelagen. Onderwijsdoelen - uitgangspunten.

[2] M. Bannert, I. Molenaar, R. Azevedo, S. Järvelä, and D. Gašević. Relevance of learning analytics to measure and support students' learning in adaptive educational technologies. In *Proceedings of the Seventh International Learning Analytics & Knowledge Conference*, LAK '17, pages 568–569. Association for Computing Machinery.

[3] V. Barr and C. Stephenson. Bringing computational thinking to k-12: what is involved and what is the role of the computer science education community? 2(1):48–54.

[4] N. Gesquière and F. Wyffels. Computational thinking in flanders' compulsory education. In *Proceedings of Sixth APSCE International Conference on Computational Thinking and STEM Education 2022 (CTE-STEM)*, pages 58–63. TU Delft Open.

[5] S. Grover and R. Pea. Computational thinking in k–12: A review of the state of the field. 42(1):38–43.

[6] I. Molenaar. Towards human-AI learning technologies. 57(4):632–645.

[7] J. Moreno-León, G. Robles, and M. Román-González. Dr. scratch: Automatic analysis of scratch projects to assess and foster computational thinking.

[8] T. Neutens. Physical computing in primary and secondary education : techniques and tools for teaching and assessment. ISBN: 9789463556224.

[9] U.S. Department of Education, Office of Educational Technology. Artificial intelligence and future of teaching and learning: Insights and recommendations.

[10] A. G. Zhang, Y. Chen, and S. Oney. VizProg: Identifying misunderstandings by visualizing students' coding progress. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–16. ACM.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Subgoal Learning Integration in a CS1 Course

Olivier Goletti[1] [0000−0002−1610−4985]

ICTEAM/INGI, UCLouvain, Louvain-la-Neuve, Belgium
olivier.goletti@uclouvain.be

Abstract. Introductory programming courses are sometimes too ambitious for novice programming students. Since a lot of university level programming courses make use of undergraduate teaching assistants, we decided to train them with evidence-based instructional strategies that will benefit students' learning. Over the course of five years, we iterated on the integration of instructional strategies through the training of the undergraduate teaching assistants of a CS1 course. Based on an analysis of the course, subgoal learning was selected as an evidence-based instructional strategy to promote learning transfer and reduce cognitive load for students. This paper details the methodology used to integrate subgoal learning in our third iteration. We discuss how insights from previous iterations instructed our design choices, challenges to change an established CS1 course and how we selected the resources and where changes were made in the course to integrate subgoal learning.

Keywords: Subgoal Learning · CS1 · Educational Change.

Context

I am in the fifth year of my doctoral studies at UCLouvain where I am a teaching and research assistant. This means that I dedicate nearly half of my time to teaching and the other half to research. My supervisor at UCLouvain is Prof. Kim Mens and Prof. Felienne Hermans is my co-supervisor at Vrije Universiteit Amsterdam.

I am in the process of completing the third and last iteration step of my research on the integration of evidence-based instructional strategies to undergraduate teaching assistants (UTAs). I hope that the Didapro 10 Doctoral Consortium will allow me to take a step back on this last step in order to help me plan the organisation of my dissertation text.

Motivation

A lot of computer science pedagogic approaches seem to be inspired by constructionism and learner-centered methodologies [6]. The goal of my research is to explore how I can foster learning transfer in this context through the training of UTAs to use explicit evidence-based programming strategies for difficult CS concepts.

Volume 1, pages 131–135.

Background

Learning CS and programming in particular has been described as hard [12] while others insist that the ambitions of CS1 courses are to blame instead [13]. In the course of multiple iterations guided by the principles of *Design-Based Research* [1][4], I selected and integrated instructional strategies to UTAs. Multiple evidence-based strategies were selected based on *learning transfer* and *cognitive load theory*. I then explore(d) how undergraduate teaching assistants could grasp and use these instructional strategies to teach their students. The third iteration of my research focuses on *Subgoal Learning*; other studied strategies were discussed in prior work [9, 10]. We will briefly define all these key notions below.

Design-Based Research is a research approach focusing on the iterative development of a pedagogical innovation while observing and researching it during the intervention. Research is done in an ecological setup such as classrooms, guided by design principles that will be refined, adjusted or dropped in subsequent iterations [1]. Each iteration cycle is starts by a preparation and design phase, then the intervention itself takes place and finally, a redesign is done based on the analysis of the previous iteration.

Learning Transfer is the ability to reuse previously acquired knowledge in a new target task. Learning transfer has been studied and modeled as multiple interacting processes, including knowledge encoding in memory and knowledge retrieval conditioned by its organisation and accessibility in long-term memory [18, 3].

Cognitive Load Theory (CLT) [19] is an instructional theory based on human cognitive architecture. CLT assumes a limited working memory. The impact of CLT on instructional design is that one should try to reduce load on the working memory when teaching new material. A lot of effects and instructional impact have been proposed by CLT proponents [17].

Subgoal Learning Among the effects predicted by cognitive load theory, the worked example effect shows that exposing students to detailed solutions to problems improves their learning [20] in programming courses among other disciplines [16, 5]. Labelling the steps used by experts to solve these problems draws the learner's attention to the generic aspect of these solution steps [14, 16]. Subgoal learning is the idea of explicitly teaching the steps of recurring patterns in the resolution of similar problems to favour learning transfer. Margulieux et al. [15] combined the idea of worked examples with labeling the steps behind common code constructs, proposing subgoal labels worked examples (SLWEs). In the end, the subgoals are taught to the students by interleaving SLWEs and practice exercises [15] progressively diminishing the guidance as prescribed by CLT [11] . The SLWEs studied by Margulieux et al. suggest that *"subgoal materials helped learners to solve problems using the procedure more effectively during the early stages of learning even though no performance difference between groups was found on the exams"*.

Research Goals and Progress

After our analysis of a CS1 course [7], we conducted a first iteration where we trained four undergraduate teaching assistants with four strategies that led us to introduce four criteria to select instructional strategies: the strategy should be easy to understand, straightforward to apply, useful on the long term and supported by literature [9].

Based on those criteria, our analysis of the first iteration led us to select only two strategies to not overload our UTAs and also to introduce both strategies earlier in the semester. We also created dedicated training material to help the UTAs integrate the different aspects of the strategies [10]. For the second iterations, we also measured the *Fidelity of Implementation* [2] of the two strategies by the UTAs by observing, recording and coding recordings of their teaching. While in the first iteration, we used subgoals from the literature, in the second iteration, we developed our own subgoals for the concepts and language of our own CS1 course [8].

Finally, in order to prepare our last iteration focusing on subgoal learning (SL), the main design change was to integrate the strategy globally in the course. The most mentioned issue by UTAs in the focus groups and follow up discussions for them to use SL in the course was that it took them too much time to properly present a specific worked example and the corresponding subgoals and labels. UTAs argued that they would benefit from an introduction of the labels done in the course material. Moreover, we observed that for the other strategy used in the second iteration, an important trigger for strategy use was to have prompts in students' exercises statements.

We identified the resources of the course that could be impacted by the integration of SL and designed a fading approach of the integration of subgoals and labels in the course, in accordance with CLT inspired instructional principles. We identify two different type of Subgoal Labels utilization. First, Step-By-Step SLWEs (SBS) are worked examples that use each label one by one to illustrate at each step how the generic corresponding subgoal for that concept is used in a concrete example. The integration of those SBSs is mainly done by the teachers during courses, but a UTA using the labels and solving procedure to provide students with an exercise solution could also be using SBSs. In order to integrate SBSs in the course, we had to modify teachers slide decks and this took some work and back and forth with teachers. Second, Subgoal Labeled Final Solutions (SLFS) that would be used after the labels have already been introduced once. The idea is to show final solution or solved examples without necessarily going through the whole solving process but by annotating what code pertains to what label. Typically, those could be shown in the teachers slides to remind students of the labels or the result of an UTA annotating code on the blackboard during a lab session.

The integration of SL in the course consisted in the modification of the slides with SBSs and SLFSs. This allowed for the introduction and repetition of the labels for the students in a passive way. Since we knew that prompts in students' exercises statements were also triggers for strategy use, we also added the labels

in the first exercises making use of the targeted concepts. The labels linked each time to a complete SLFS catalogue where the labels and annotated examples were gathered per concept. The last integration step was, next to training of the UTAs like in previous iterations, to add reminders in the correction guide for tutors so that they knew where and when new concepts were used and which labels to use. This information is also discussed each week during coordination meetings with the tutors.

Expected Contribution

We hope that this multi-level fading integration of subgoal learning in the course will push tutors to use the strategy more often than in our previous iteration. For this third iteration, since SL is completely integrated in the course as an instructional strategy, all UTAs and not just only volunteers like in previous iterations will need to apply the strategy. Our hypothesis is that observations will yield more strategy use and with better fidelity.

References

1. Bakker, A.: Design research in education: A practical guide for early career researchers. Routledge (2018)
2. Borrego, M., Cutler, S., Prince, M., Henderson, C., Froyd, J.E.: Fidelity of Implementation of Research-Based Instructional Strategies (RBIS) in Engineering Science Courses. Journal of Engineering Education **102**(3), 394–425 (2013). https://doi.org/10.1002/jee.20020, https://onlinelibrary.wiley.com/doi/abs/10.1002/jee.20020, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jee.20020
3. Bracke, D.: Un Modèle Fonctionnel Du Tranfert Pour l'éducation, pp. 77–106. Presses Université Laval (2004)
4. Collective, D.B.R.: Design-based research: An emerging paradigm for educational inquiry. Educational researcher **32**(1), 5–8 (2003)
5. Ericson, B.J., Margulieux, L.E., Rick, J.: Solving Parsons Problems Versus Fixing and Writing Code. In: Proceedings of the 17th Koli Calling International Conference on Computing Education Research. pp. 20–29. Koli Calling '17, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3141880.3141895
6. Falkner, K., Sheard, J.: 15 pedagogic approaches. The Cambridge handbook of computing education research p. 445 (2019)
7. Goletti, O.: En quoi le dispositif mis en œuvre dans le cours d'introduction à l'informatique en BAC1 ingénieur civil basé sur l'apprentissage par problèmes soutient les processus du transfert des apprentissages : l'encodage et l'accessibilité aux connaissances ? Tech. rep., UCLouvain (2019), http://hdl.handle.net/2078.1/245579
8. Goletti, O., De Pierpont, F., Mens, K.: Création d'exemples résolus avec objectifs étiquetés pour l'apprentissage de la programmation avec python. In: Didapro 9–DidaSTIC (2022)

9. Goletti, O., Mens, K., Hermans, F.: Tutors' Experiences in Using Explicit Strategies in a Problem-Based Learning Introductory Programming Course. In: ITiCSE '21. p. 7. ACM Press, Virtual Event, Germany (Jun 2021). https://doi.org/10.1145/3430665.3456348
10. Goletti, O., Mens, K., Hermans, F.: An analysis of tutors' adoption of explicit instructional strategies in an introductory programming course. In: Proceedings of the 22nd Koli Calling International Conference on Computing Education Research. pp. 1–12 (2022)
11. Kirschner, P.A., Sweller, J., Clark, R.E.: Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching. Educational Psychologist **41**(2), 75–86 (Jun 2006). https://doi.org/10.1207/s15326985ep4102$_1$
12. Lister, R., Adams, E.S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J.E., Sanders, K., Seppälä, O.: A multi-national study of reading and tracing skills in novice programmers. In: ACM SIGCSE Bulletin. vol. 36, pp. 119–150. ACM (2004)
13. Luxton-Reilly, A.: Learning to program is easy. In: Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education. p. 284–289. ITiCSE '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2899415.2899432, https://doi.org/10.1145/2899415.2899432
14. Margulieux, L.E., Guzdial, M., Catrambone, R.: Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. In: Proceedings of the ninth annual international conference on International computing education research. pp. 71–78 (2012)
15. Margulieux, L.E., Morrison, B.B., Decker, A.: Design and Pilot Testing of Subgoal Labeled Worked Examples for Five Core Concepts in CS1. In: Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '19. pp. 548–554. ACM Press, Aberdeen, Scotland Uk (2019). https://doi.org/10.1145/3304221.3319756
16. Morrison, B.B., Margulieux, L.E., Guzdial, M.: Subgoals, Context, and Worked Examples in Learning Computing Problem Solving. In: Proceedings of the Eleventh Annual International Conference on International Computing Education Research. pp. 21–29. ACM Press (2015). https://doi.org/10.1145/2787622.2787733
17. Sweller, J., van Merriënboer, J.J., Paas, F.: Cognitive architecture and instructional design: 20 years later. Educational Psychology Review pp. 1–32 (2019), publisher: Springer
18. Tardif, J.: Le Transfert Des Apprentissages. Editions logiques (1999)
19. van Merriënboer, J.J.G., Sweller, J.: Cognitive Load Theory and Complex Learning: Recent Developments and Future Directions. Educational Psychology Review **17**(2), 147–177 (Jun 2005). https://doi.org/10.1007/s10648-005-3951-0
20. Ward, M., Sweller, J.: Structuring effective worked examples. Cognition and instruction **7**(1), 1–39 (1990)

Troisième partie

Posters

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

A plugin for the INGInious autograder to annotate programs with subgoal labels

Joseph Vankelegom

Université catholique de Louvain

Abstract. Subgoal learning is a pedagogical strategy that can reduce the difficulty faced by novice students when learning how to program. This poster explains how we intend to extend an existing autograder with support for annotating programs with labels representing relevant subgoal, in order to better support this teaching strategy. We discuss the approach, design and objectives of such a plugin for the INGInious autograder.

Keywords: Subgoal labelling · INGInious · Computer science education

1 Introduction

Computer science education plays a pivotal role in shaping the next generation of programmers. Teaching programming skills to inexperienced first-year bachelor students (CS1) is a critical phase. One pedagogical approach currently being explored to support such students is subgoal learning, a strategy that could reduce the difficulty for novices and lead to greater motivation and persistence [2]. The objective of this work, conducted in the context of a master thesis, is to build, test, and validate a plugin for the INGInious [3] autograder that would enable professors to create such exercises for their students.

Subgoal learning The idea of subgoal learning is to highlight the steps of a generic problem solving procedure. Those steps can be reused when reading or writing similar code and thus enable students to comprehend the common structure among programs. Its positive impact has already been studied in Java [1] and other languages. The objective of this approach is to reduce some of the cognitive load on novice students when they learn how to code in their introductory programming course, allowing them to focus on the more important conceptual elements instead and create a mental framework for problem solving.

INGInious INGInious[1] is an open-source platform developed for creating and managing programming assignments, including an autograder for correcting such assignments. One notable feature is its ability to provide students with a more precise and extensive feedback. Another relevant feature is that INGInious is a modular platform so that new problem types (such as exercises with subgoal labels) can easily be added as plugins.

[1] https://github.com/UCL-INGI/INGInious

Volume 1, pages 137–139.

Approach To achieve the goals set out above, we need to decompose the problem into smaller subproblems.

Requirements The first step is to specify our requirements, including the type of exercises the plugin would enable a teacher to create and the correction method. We also would like to include automated assistance for teachers in creating parts of the exercises. Since this automated assistance will require us to work on abstract syntax trees, this decision led us to focus on a specific programming language, and we chose Python since it is one of the most widely-used programming languages.

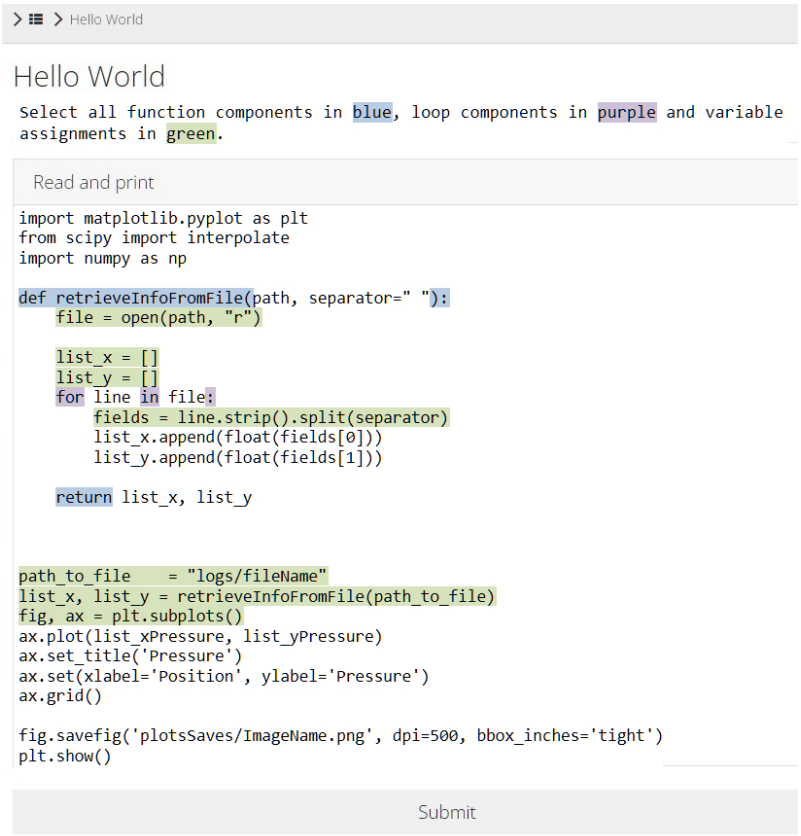


Fig. 1. Mock-up of the student interface

Implementation The implementation process is divided into two principal parts, there is the front-end and the back-end. The front-end it's all the elements that are visible from the user perspective; the teacher interface is where he can create new exercises, and the student's it's where he can answer them. It will be written in JavaScript. The back-end is hidden from the users; there we implement corrections scripts, the assistant tool. It will be written in Python.

Testing and validation We need to ensure that the implementation satisfy the users' requirements in terms of features and experience. To achieve this, active collaboration with our CS1 students and teachers is essential. The teachers will design new exercises, the student solves them; each iteration will provide us with feedback to improve the plugin. Both their validation is crucial since the plugin has two distinct interfaces.

Once it is validated by both parties, we will be able to accomplish our objective, contribute to Inginious with the plugin.

Acknowledgements Thanks to Prof. Kim Mens, Olivier Goletti and Anthony Gego for their help in guiding this master thesis.

References

1. Lauren E. Margulieux, Briana B. Morrison, Adrienne Decker.: Design and Pilot Testing of Subgoal Labeled Worked Examples for Five Core Concepts in CS1. Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education, p 548–554 (2019)
2. (Brunstein, 1993, Locke and Latham, 1990, Soman and Shi, 2003)
3. DERVAL, Guillaume, GEGO, Anthony, REINBOLD, Pierre, et al. Automatic grading of programming exercises in a MOOC using the INGInious platform. European Stakeholder Summit on experiences and best practices in and around MOOCs (EMOOCS'15), 2015, p. 86-91.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Identification des lacunes dans le savoir conceptuel et stratégique dans l'apprentissage de la programmation — *Extended abstract*

Jean-Philippe Pellet[0000−0001−7559−397X] et Patrick Wang[0000−0003−3117−8189]

Haute école pédagogique du canton du canton de Vaud, Lausanne, Suisse
{jean-philippe.pellet, patrick.wang}@hepl.ch

L'apprentissage de la programmation est un sujet largement étudié dans la littérature et de nombreux articles (e.g., [1,2]) cherchent à mettre en avant les principales difficultés que les élèves peuvent rencontrer au cours de cet apprentissage. Ces difficultés se manifestent souvent lorsqu'il s'agit de concevoir une solution algorithmique à un problème donné et/ou lorsqu'il s'agit de traduire cette solution algorithmique en un programme. Dans [2], les auteurs expliquent que ces erreurs peuvent provenir de lacunes dans le savoir *syntaxique*, *conceptuel*, ou *stratégique*. Pour un enseignant, il est *a priori* facile de repérer et d'évaluer les lacunes dans le savoir syntaxique, les messages d'erreurs en console offrant une aide précieuse ici. Mais l'identification précise des points de blocage dans le savoir conceptuel et stratégique est plus complexe.

Dans cette proposition de poster, nous nous intéressons à cette question de l'évaluation du savoir conceptuel et stratégique. La question de recherche qui motive ce travail est la suivante : « Comment identifier et évaluer, dans les productions des élèves, des lacunes dans le savoir conceptuel et stratégique ? ».

Nous avons à cet effet construit un dispositif d'évaluation notamment basé sur des « études de cas » ayant pour objectif de mettre en évidence le savoir conceptuel et stratégique mobilisé durant ces activités. L'idée de base est l'explicitation de l'identification et de la juxtaposition des éléments clés du langage nécessaires pour résoudre une tâche donnée, avant de passer à l'écriture de code.

La question de la stratégie de conception d'un algorithme ou de construction d'un programme n'est pas nouvelle. En effet, Wirth [4] proposait déjà en 1971 la méthode du *stepwise refinement*, qui consiste à formuler une solution dont les instructions sont tout d'abord assez larges, puis à les raffiner jusqu'à obtenir des instructions dont le niveau de précision équivaudrait à celui pouvant être obtenu par un langage de programmation. Cependant, Rist [3] explique qu'un expert, contrairement au novice, sera capable d'alterner (ponctuellement et lorsque nécessaire) entre des approches *top-down* (similaires au *stepwise refinement*) et *bottom-up* lors de la construction d'un programme.

Notre dispositif cherche donc à rendre compte de la capacité des élèves à concevoir une solution algorithmique tout en permettant la mise en évidence des différences de niveau de maîtrise dans les savoirs conceptuels et stratégiques. Pour cela, nous avons conçu une activité en deux temps autour d'études de cas. Tout d'abord il est demandé aux élèves de formuler en langage naturel un algorithme de résolution possible. La présence de constructions algorithmiques et

Volume 1, pages 140–141.

le niveau de précision des instructions sont des indicateurs que nous souhaitons trouver dans les réponses des élèves pour répondre à notre question de recherche. Durant le second temps, les élèves implémentent leur algorithme. Ce travail vise à expliciter un éventuel écart pouvant exister entre une solution algorithmique formulée en langage naturel et son implémentation directe, écart pouvant suggérer des difficultés à aller « jusqu'au bout » du *stepwise refinement*. En guise d'illustration, voici ci-après un exemple d'étude de cas.

Exemple. L'élève, après 4 semaines d'apprentissage de la programmation avec Python, reçoit la consigne : « Pour la tâche suivante, indiquer les éléments du langage ou les structures dont nous aurons besoin. Pour chaque élément, indiquer (a) combien de fois on y fera référence dans le programme et pourquoi, et (b) combien de fois il sera "utilisé" pendant l'exécution du programme. *Tâche :* proposer à l'utilisateur d'entrer une série de notes entre 1 et 6, puis à la fin, afficher la moyenne. » La réponse attendue, dont la forme a été travaillée en classe, ressemble à ceci (étant entendu que les listes Python sont encore inconnues à ce stade). *Il nous faut :*

— *Une* ***boucle****, car il s'agit d'introduire des nombres de manière répétée. Il y aura une seule boucle, dont le corps pourra être exécuté plusieurs fois ;*

— *Un* ***input() avec conversion en int*** *pour entrer les nombres, qui sera visible une fois dans le programme mais qui sera situé dans la boucle et donc exécuté potentiellement plusieurs fois aussi ;*

— *Une* ***condition*** *pour vérifier que soit entre 1 et 6, après chaque input() – aussi une fois dans le programme, mais exécuté dans la boucle ;*

— *Des* ***variables auxiliaires****, une pour faire la somme des valeurs entrées, l'autre pour compter le nombre de valeurs entrées. Elles sont initialisées avant la boucle, mises à jour dans la boucle, et utilisées après la boucle pour calculer la moyenne.*

L'analyse des données n'est actuellement pas terminée mais serait, en cas d'acceptation, complétée pour le poster selon les pistes principales suivantes : (a) capacité à identifier les structures algorithmiques nécessaires et (b) écart entre formulation textuelle et écriture du programme.

Références

1. Du Boulay, B. : Some Difficulties of Learning to Program. Journal of Educational Computing Research **2**(1), 57–73 (Feb 1986). https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9
2. Qian, Y., Lehman, J. : Students' Misconceptions and Other Difficulties in Introductory Programming : A Literature Review. ACM Transactions on Computing Education **18**(1), 1 :1–1 :24 (Oct 2017). https://doi.org/10.1145/3077618
3. Rist, R.S. : Schema creation in programming. Cognitive Science **13**(3), 389–414 (1989)
4. Wirth, N. : Program development by stepwise refinement. Communications of the ACM **14**(4), 221–227 (1971)

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Integrating Parsons problems in a CS1 programming course autograder

Corentin Lengelé

UCLouvain University
Place Sainte corentin.lengele@student.uclouvain.be

Abstract. Parsons problems are a type of exercises in computer science education where students have to assemble a correct program from a set of scrambled or disordered program elements, by arranging the relevant program elements into the correct order. This work discusses how to integrate such Parsons problems into an existing autograder platform, and to observe the effectiveness of first-year bachelor students receiving such problems as compared to how they solve traditional coding tasks.

Introduction This poster will present our ongoing master thesis on integrating interactive learning methods for first-year bachelor programming courses through the integration of Parsons problems into an educational programming platform. A Parsons problem is an interactive exercise that requires rearranging code blocks rather than writing code. [1] The main purpose of integrating such exercises into an exsting autograder is to give students access to a different learning method and to observe the effectiveness of it.

Approach We aim to implement a Parsons problems plugin for the INGInious [1] autograder platform [2], based on some existing interactive Parsons problem examples[2]. The plugin will make this new problem type available to students, tutors, and teaching staff on the platform, allowing them to use or define such problems in the context of an existing course. An alternative programming course utilising Parsons problems will be developed for students to explore. This will provide valuable insights into the efficacy of this learning approach for first-year bachelor students.

Parson problems A Parsons problem is a coding challenge where learners must rearrange predefined blocks in the correct order. It is an intermediate exercise between a worked example where a solution is given and a classic problem solving exercise where the student needs to write code from scratch. This approach has shown comparable learning outcomes to traditional coding tasks while requiring less time. [3] This type of exercise assesses the understanding of multiple

[1] https://inginious.org/
[2] https://js-parsons.github.io
https://runestone.academy/ns/books/published/overview/Assessments/parsons.html

Volume 1, pages 142–144.

skills such as knowledge of syntax, problem decomposition, algorithm development, and debugging [4]. It can be structured as a drag-and-drop activity, where learners select code blocks from a provided list and assemble a solution within a designated area, as shown in Fig. 1.

Numerous additional options can be incorporated into this type of problem. [4] For instance, the code's indentation could either be pre-set or left for the student to determine, adding an extra layer of complexity. Introducing distractor blocks is another possibility, which are blocks that shouldn't be included in the solution. Alternatively, a pair can be presented with either a valid code block or a distractor that closely resembles it, further enhancing the problem-solving aspect.

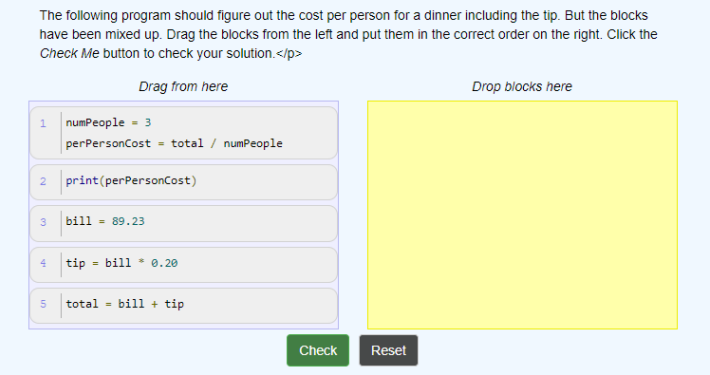


Fig. 1. Example of a Parsons problem structured as a drag-and-drop activity (from https://runestone.academy/ns/books/published/overview/Assessments/parsons.html).

INGInious INGInious [2, 5] is an intelligent autograder platform that allows secured and automated testing of code made by students. It is written in Python and uses Docker to run students' code inside a secured environment. It provides a backend which manages interaction with Docker to grade code, and a frontend which allows students to submit their code in an ease-to-use interface. The frontend also includes an administration interface for teachers to check the progress of their students and to modify exercices in a simple way. Various types of problems such as coding exercises, multiple-choice questions or mathemical problems are already integrated into the platform. Integrationg new problem types is q quite easy process that does not need modifications to the codebase.

References

1. P. Ihantola and V. Karavirta, "Two-dimensional parson's puzzles: The concept, tools, and first observations," *Journal of Information Technology Education. Inno-*

vations in Practice, vol. 10, p. 119, 2011.
2. G. Derval, A. Gego, P. Reinbold, B. Frantzen, and P. Van Roy, "Automatic grading of programming exercises in a mooc using the inginious platform," *European Stakeholder Summit on experiences and best practices in and around MOOCs (EMOOCS'15)*, pp. 86–91, 2015.
3. B. J. Ericson, L. E. Margulieux, and J. Rick, "Solving parsons problems versus fixing and writing code," in *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 20–29. [Online]. Available: https://doi.org/10.1145/3141880.3141895
4. B. J. Ericson, P. Denny, J. Prather, R. Duran, A. Hellas, J. Leinonen, C. S. Miller, B. B. Morrison, J. L. Pearce, and S. H. Rodger, "Parsons problems and beyond: Systematic literature review and empirical study designs," in *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 191–234. [Online]. Available: https://doi.org/10.1145/3571785.3574127
5. "Inginious grading platform for students," https://github.com/UCL-INGI/INGInious.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Intégration du Subgoal Learning dans l'enseignement de la programmation

Antoine Demblon

UCLouvain, Louvain-la-Neuve, Belgique

Abstract. Les exemples résolus avec objectifs étiquetés est une méthode d'enseignement facilitant le transfert d'apprentissage et réduisant la charge cognitive des apprenants. Ce travail présente et analyse l'intégration de cette méthode dans un cours de programmation existant.

1 Introduction

Un des axes de recherche sur l'enseignement de l'informatique est l'utilisation d'exemples résolus avec objectifs étiquetés ("subgoal-labeled worked examples" en anglais), qui a fait l'objet de plusieurs études par Margulieux et al. [4] [3] , ainsi que Goletti et al [2] [1]. L'intégration de cette nouvelle méthode d'apprentissage a été démontré comme efficace dans l'enseignement de la programmation[4] [3].

Le premier objectif de ce poster est de présenter (cf. section 3) comment nous avons intégré cette méthode dans un premier cours d'informatique à l'université. Le second objectif est d'analyser le processus de cette intégration (cf. section 4).

2 Concept Théoriques

Avant d'aborder notre approche, présentons d'abord quelques concepts liées à la notion des exemples résolus avec objectifs étiquetés.

Apprentissage par objectifs étiquetés L'apprentissage par objectifs étiquetés ("subgoal-learning" en anglais), a pour but de mettre en évidence les étapes génériques de la résolution d'un problème type. Les apprenants sont guidés à identifier et utiliser ces étapes pour des problèmes ou sous-problèmes similaires.

Exemples résolus L'utilisation d'exemples résolus permettent d'illustrer efficacement les différentes étapes abstraites d'une procédure de résolution. Les novices peuvent cependant confondre le contexte spécifique de l'exemple avec la procédure elle-même.

Exemples résolus avec objectifs étiquetés Les exemples résolus avec objectifs étiquetés (ou "Subgoal Labelled Worked Examples (SLWE)" en anglais) sont la fusion de 2 méthodes d'apprentissages précédentes. Ils ont été proposées par Margulieux et al [3]. Étiqueter les étapes des exemples résolus permet de distinguer le contexte spécifique de la résolution générique ainsi que diminuer la charge cognitive des apprenants.

Volume 1, pages 145–147.

3 Intégration

Nous avons intégré les exemples résolus avec objectifs étiquetés (SLWEs) dans un cours d'informatique qui possède 3 axes principaux :

- un cours magistral donné par plusieurs professeurs;
- des exercices que les étudiants doivent résoudre;
- des séances de tutorat, dans lequel les solutions des exercices précédent sont présentées et discutées.

L'intégration des SLWEs dans un cours existant posent plusieurs soucis.

- Le support du cours doit être modifié. Cela inclut les diapositives utilisées lors des cours magistraux, les exercices donnés aux étudiants ainsi que les notes données aux tuteurs.
- Les professeurs doivent changer leur méthode d'apprentissage.
- Les SLWEs, ainsi que leur utilisation correcte, doivent être enseignés aux tuteurs.
- Les professeurs doivent approuver chacun des changements précédents.

4 Analyse

Dans cette section, nous allons présenter le processus d'analyse des axes principaux.

Analyse du cours Un questionnaire anonyme sera mis à disposition des étudiants. Ce questionnaire comportera des questions portant sur leur perception de l'utilisation et de l'impact des SLWEs dans le cours.

Analyse des exercices Les étudiants réalisent leurs exercices sur la plateforme en ligne inginious, nous permettant ainsi de comparer les données entre cette année et les précédentes. Par exemple, le nombre moyen de soumissions nécessaire pour réussir un exercice sera évaluée.

Analyse des séance de tutorat Pour analyser l'utilisation des SLWEs lors des séances de tutorat, nous allons les observer avec l'aide de plusieurs étudiants volontaires. Pendant 3 semaines, ils filmeront les séances de tutorat et jugeront l'usage des SLWEs sur base de critère définis. Chaque séance sera évaluée par 2 observateurs différent afin de comparer et valider les données.

References

1. Goletti, O., Mens, K., Hermans, F.: Tutors' experiences in using explicit strategies in a problem-based learning introductory programming course. In: Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1. p. 157–163. ITiCSE '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3430665.3456348, https://doi.org/10.1145/3430665.3456348

2. Goletti, O., Mens, K., Hermans, F.: An analysis of tutors' adoption of explicit instructional strategies in an introductory programming course. In: Proceedings of the 22nd Koli Calling International Conference on Computing Education Research. Koli Calling '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3564721.3565951, https://doi.org/10.1145/3564721.3565951
3. Margulieux, L.E., Morrison, B.B., Decker, A.: Design and pilot testing of subgoal labeled worked examples for five core concepts in cs1. In: Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education. p. 548–554. ITiCSE '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3304221.3319756, https://doi.org/10.1145/3304221.3319756
4. Morrison, B.B., Margulieux, L.E., Guzdial, M.: Subgoals, context, and worked examples in learning computing problem solving. In: Proceedings of the Eleventh Annual International Conference on International Computing Education Research. p. 21–29. ICER '15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2787622.2787733, https://doi.org/10.1145/2787622.2787733

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

La méta-programmation logique sur du code source Python pour rechercher des bonnes ou mauvaises pratiques dans le code des étudiants

Nathan Corbisier

UCLouvain

Résumé. La capacité à interroger et extraire des informations à partir de code source de programmes rédigés par les étudiants revêt une importance significative, permettant ainsi d'automatiser la réponse aux questions suivantes : "Ce code contient-il des erreurs?", "Est-il conforme aux conventions d'écriture en Python?", "Présente-t-il de mauvaises pratiques?", "L'étudiant a-t-il substitué un 'return' par un 'print'?", etc. Nous proposons d'explorer la technique de la méta-programmation logique pour écrire des requêtes dédiées à l'analyse de pratiques spécifiques dans le code des étudiants. L'idée sous-jacente est de traduire du code Python en une base de données logique, et ensuite de pouvoir interroger cette base de données au moyen de différentes requêtes écrites dans un langage de programmation logique.

Introduction. Les cours d'introduction à la programmation utilisent régulièrement des correcteurs automatiques pour aider les étudiants en leur fournissant des retours automatisés sur leurs exercices de programmation. Ces retours, conçus pour aider les étudiants à tirer des enseignements de leurs erreurs et à les rectifier, devraient être de haute qualité. Toutefois, ces correcteurs ont généralement comme objectif principal de vérifier la validité du code par rapport à la question posée. Cependant, tenir compte et corriger les pratiques moins souhaitables, telles que l'utilisation de réponses hardcodées, est tout aussi important.

Notre motivation est d'utiliser la programmation logique comme moyen expressif et déclaratif de décrire et identifier certains motifs dans le code source des étudiants. De cette manière, notre objectif est de mettre à disposition des enseignants un langage déclaratif de haut niveau permettant la détection des mauvaises ou bonnes pratiques dans le code source des étudiants. À terme, nous aimerions pouvoir intégrer cette méthode dans les systèmes de correction automatisée afin d'améliorer la qualité de retours fournis aux étudiants.

Méthodologie. Pour pouvoir effectuer des requêtes logiques sur du code source Python, nous avons opté pour l'utilisation du langage Prolog. Cependant, avant de pouvoir exécuter des requêtes logiques, il est impératif de convertir le code source en faits logiques. Cette transformation s'opère via les étapes suivantes. Tout d'abord, un parser intégré à Python permet la création d'un Abstract Syntax Tree (AST). Ensuite, cet AST est converti en format JSON. SWI-Prolog propose une librairie officielle pour convertir un JSON en données manipulables

Volume 1, pages 148–149.

par un programme logique, ce qui justifie notre choix de cette implémentation du langage Prolog. Une fois l'AST Python transmis au programme SWI-Prolog sous forme de JSON, nous traduisons de manière automatique chaque noeud de l'arbre en faits logique. C'est à partir de ces faits logiques que nous pouvons analyser et écrire des premières requêtes pour trouver certaines pratiques au sein du code source d'étudiants.

Premiers résultats. Après avoir converti un premier programme simple en faits logiques, nous avons constaté la nécessité de garder un lien entre différents faits logiques séparés pour pouvoir traverser notre AST aussi bien dans le sens descendant qu'ascendant. En suivant cette idée nous avons décidé de simplifier nos faits en y ajoutant un identifiant unique. À partir de là et en exploitant les propriétés de la programmation logique nous avons créé des premières fonctions nous facilitant la navigation dans l'arbre ainsi que la collecte d'informations.

Prédicat	Signification
id_type(I, T)	identifiant I lié au type T
child_parent(C, P, L)	C est un enfant de P à un niveau de profondeur L
child_parent(C, C_T, P, P_T, L)	Idem avec le type de l'enfant C_T et du parent P_T
take_info(I, N, X)	X est l'information N du noeud avec identifiant I
count_parentid_childrentype(S, P, C_T)	Parent P a au moins S enfant de type C_T

En utilisant ces premiers prédicats on peut créer des prédicats plus complexes.

```
% trouve les fonctions sans return
function_no_return(Name) :-
    id_type(Id, 'FunctionDef'),
    not(child_parent(_, 'Return', Id, _, _)
    take_info(Id, 'name', Name).
```

Ce prédicat nous permet d'identifier dans n'importe quel programme Python, le nom des fonctions n'ayant pas de 'return', ce qui peut dire que l'étudiant n'a pas vraiment compris le concept d'une fonction qui retourne une valeur. Un des avantages d'utiliser un langage logique est qu'il est plus déclaratif et que les prédicats de base son multi-directionnel, permettant d'explorer l'AST dans différentes directions.

Conclusion. Nous avons fait part dans ce document de nos premiers résultats dans la transformation de code source Python en une base de données de faits logiques, ainsi que des premières possibilités d'exécution de requêtes sur ces données. Combinées avec une détection automatique des patterns positifs ou négatifs au sein des codes sources étudiants, nous pouvons envisager à terme de proposer un outil supplémentaire qui se veut déclaratif afin d'identifier automatiquement les erreurs récurrentes des étudiants, améliorant ainsi les feedback proposés aux étudiants lors de l'utilisation d'un correcteur automatique.

Nous tenons également à souligner que, si les exemples ont été réalisés sur des codes sources Python, notre capacité à traduire automatiquement ces codes en faits logiques signifie que notre démarche est facilement adaptable à d'autres langages de programmation présentant des structures d'AST similaires ou légèrement différentes.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Robots humanoïdes et stratégies cognitives mises en œuvre dans les activités robotiques au primaire

Julien Bugmann[1] [0000-0002-1205-9005] et Nathalie Cloux Renard[2]

[1] HEP Vaud, Lausanne
[2] HEP Vaud, Lausanne

Abstract. Les robots sont de plus en plus présents dans les salles de classe et notamment à des fins d'apprentissage de la programmation et de développement d'une culture et citoyenneté numérique. Cependant, les défis sont multiples dans ce type d'activités et en particulier en ce qui concerne la représentation de l'espace et la décentration des élèves. Nous présentons ici une démarche de recherche visant à comparer les stratégies cognitives mobilisées par les élèves dans deux situations différentes : l'une dans un contexte de programmation avec un robot de sol qui se déplace en roulant, et l'autre avec un robot dit humanoïde qui se déplace, comme un être humain, en marchant. Nous avons donc comparé les stratégies cognitives mises en œuvre par les élèves dans ces deux situations et en présenterons les principales conclusions.

Keywords: Robots humanoïdes, programmation, décentration, école primaire.

1 Problématique

Les robots sont de plus en plus présents dans notre quotidien et sont désormais régulièrement intégrés dans les programmes scolaires à des fins d'apprentissages, notamment en informatique. Aussi, leurs formes sont multiples dans le contexte pédagogique mais ce sont principalement des robots de sol qui sont utilisés dans les établissements scolaires. En Suisse romande, par exemple, les enseignant·e·s du primaire utilisent essentiellement les robots Thymio ou Bluebot, en fonction des degrés concernés, suite aux propositions effectuées dans le cadre du déploiement de l'éducation numérique. Les défis sont multiples dans ces activités de robotique et les plus jeunes élèves ont notamment encore du mal à s'orienter d'un point de vue spatial pour effectuer les tâches, et donc, développer des apprentissages en termes de construction ou d'optimisation des programmes.

En effet, puisque ces différents outils fonctionnent différemment, ils amènent les élèves à adopter un axe d'orientation spatial différent [1]. Symbolisé par la capacité de décentration spatiale des élèves, le défi est d'autant plus grand qu'il faut parfois attendre l'âge de 11 ans pour que cela soit acquis par les élèves [2]. Mais que faire alors pour les élèves plus jeunes ? Comment leur permettre d'apprendre au contact de tels outils ? Certaines recherches ont mis en évidence le fait que les activités de programmation avec des robots de sol de type Beebot n'amenaient pas nécessairement d'amélioration significative de l'activité décentrée chez les élèves de 6 à 7 ans. Pourtant, en allant vers une décentration facilitée et maîtrisée, les élèves pourraient d'avantage se représenter

Volume 1, pages 150–151.

le parcours de programmation et potentiellement aller plus loin dans les apprentissages en informatique.

2 Méthodologie de recherche appliquée

Aussi, pour stimuler cette capacité de décentration et éventuellement la rendre plus aisée, nous proposons de mettre en place une activité de programmation de robot humanoïde à destination d'élèves de 1-2P (4 à 6 ans) et d'en observer les effets sur le développement de leur culture numérique mais aussi sur l'impact que cela pouvait avoir sur leur décentration spatiale. En effet, certaines recherches mettent en évidence le fait que, plus le médium est éloigné de l'élève, plus il peut devenir virtuel [3].

Et à contrario, nous pouvons donc penser que plus il en serait proche, plus il en serait réel et donc davantage en lien avec l'élève. Dans le cadre de cette recherche, 50 élèves de 1-2P ont participé à l'ensemble de ce projet qui comportait 4 étapes majeures, à savoir : (1) des défis Bluebot avec des activités de programmation (pré-test) ; (2) une séance de culture numérique avec le robot humanoïde NAO ; (3) des défis NAO avec des activités de programmation débranchée du robot et (4) des défis Bluebot avec des activités de programmation en guise de post-test.

3 Premiers résultats

Nous avons relevé lors du pré-test (1) des difficultés lors de l'orientation et des erreurs d'instruction de pivot droite ou gauche. Certaines stratégies sont particulièrement présentes, à savoir le « pas à pas » lorsque l'élève dit « avance, avance, tourne à .. », la planification partielle lorsque l'élève procède par étapes successives ou encore la programmation physique en amont lorsque l'élève déplace le robot d'un point à un autre pour en étudier les instructions à mettre en œuvre. Lorsqu'ils programment le robot humanoïde NAO (3), les élèves vont utiliser différentes stratégies cognitives, que cela soit dans le traitement de l'information, dans l'exécution, dans l'anticipation ou dans l'auto-régulation. Certains plaçaient les cartes différemment selon l'orientation du robot, d'autres décrivaient oralement les stratégies, d'autres enfin ont simulé le déplacement du robot dans son ensemble, comme s'ils jouaient au jeu du robot.

References

[1] Rigal, R. (1996). Right-left orientation, mental rotation, and perspective-taking: When can children imagine what people see from their own viewpoint? *Perceptual and Motor Skills*, *83*(3), 831-842. https://doi.org/10.2466/pms.1996.83.3.831
[2] Romero, M. Duflot M., Viéville T. (2019). Le jeu du robot: analyse d'une activité d'informatique débranchée sous la perspective de la cognition incarnée.. Review of science, math- ematics and ICT education, 13 (1), 10.26220/rev.3089 . hal-02144467
[3] Greff JP. (2004). « Le corps d'abord! », Éducation enfantine, 1056, pp. 62-63.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

UMLChecker : un outil vérifiant la conformité entre une spécification et du code dans un enseignement de programmation orientée objet

Arnaud Lanoix[1,2] and Emmanuel Desmontils[1,3[0000−0002−6150−278X]]

[1] LS2N, Nantes Université, France
[2] IUT de Nantes, France
[3] INSPÉ de Nantes, France
arnaud.lanoix-brauer@univ-nantes.fr ; emmanuel.desmontils@univ-nantes.fr

Résumé Nous présentons UMLChecker, un outil permettant de vérifier la conformité entre des spécifications UML et du code objet. Il est important que les étudiants en informatique soient capables de produire le code correspondant à un diagramme de classes. L'outil automatise le processus de vérification de la correction d'un programme dans un langage à objets selon un diagramme UML. Il a été utilisé en première année de BUT d'informatique. Basé sur les capacités d'introspection du langage cible et sur la disponibilité d'outil de tests, il permet d'intégrer la conception et la programmation UML dans le processus d'apprentissage. Il favorise l'auto-évaluation et l'évaluation continue des étudiants.

Keywords: Programmation orientée objet · UML · Test de conformité

Dans le cadre du programme national du BUT [4] informatique [3], plusieurs modules abordent les concepts, méthodes et langages de la Programmation Orientée Objet (POO), largement reconnue. En première année, 2e semestre, une ressource se concentre sur deux aspects complémentaires : i) la conceptualisation d'un problème à travers une conception orientée objet avec l'utilisation de diagrammes de classes UML, ii) le développement de programmes conformes à la POO. L'un des apprentissages critiques du programme national est l'AC11.01, qui consiste à implémenter des conceptions simples. Ceci nécessite un lien fort entre la conception UML et la programmation en POO. En d'autres termes, à la fin de la première année, un étudiant doit être capable de lire et comprendre la spécification d'un problème présentée sous forme de diagramme de classes UML, puis produire le code correspondant sans recourir à des outils de génie logiciel dédiés [3,7]. Outre la nécessité d'une traduction rigoureuse, cela exige une compréhension approfondie des subtilités d'un diagramme de classes pour les refléter fidèlement dans le code. Par exemple, l'étudiant doit pouvoir rendre avec précision la visibilité des attributs, les cardinalités, la navigabilité, les noms de rôles, etc. Ces compétence sont complexes à enseigner [5,9,11]. Elles faisaient jusqu'ici l'objet de situations didactiques demandant une forte implication de

4. Bachelor Universitaire de Technologie

Volume 1, pages 152–154.

l'enseignant. En effet, vérifier les travaux réalisés par 25 à 30 étudiants d'un groupe TD pendant la séance est extrêment lourd à gérer, car s'assurer qu'un étudiant a correctement traduit un diagramme UML est compliqué et prend du temps [12,13]. Nous avons donc cherché à améliorer les retours étudiants en automatisant une partie du processus de vérification. Sans en arriver à une situation adidactique (au sens de Broussau [2]), nous cherchons à permettre à l'enseignant de se concentrer sur l'accompagnement des étudiants ayant des difficultés plus importantes.

Pour s'assurer qu'un étudiant a bien compris une spécification UML et a correctement développé le code correspondant, notre approche consiste à évaluer la conformité de ce code par rapport à la spécification en nous appuyant sur le développement guidé par les tests. Cette évaluation, également connue sous le nom de test de conformité, a déjà fait l'objet de recherches approfondies [1,8,10]. Cependant, sa mise en œuvre implique généralement des processus complexes qui peuvent dépasser les capacités d'un étudiant en informatique en première année de BUT.

Afin de faciliter l'évaluation de la conformité structurelle du code des étudiants par rapport à une spécification UML, nous avons développé UMLChecker. Cet outil permet à un enseignant de générer automatiquement un ensemble de cas de tests JUnit [4,6] spécifiques à partir de la spécification UML. Une caractéristique clé de notre outil est qu'il n'ajoute pas de complexité aux étapes de développement, étant donné que les étudiants sont déjà habitués à valider leurs développements par des tests unitaires. Les retours fournis par UMLChecker permettent aux étudiants de corriger facilement de nombreuses erreurs basiques. L'enseignant est donc moins sollicité pour des questions élémentaires. Cela renforce l'autonomie des étudiants, les encourageant à progresser plus rapidement. La démarche TDD, en général, et l'utilisation d'UMLChecker, en particulier, accélèrent également le processus d'évaluation, facilitant ainsi un contrôle continu tout au long de la ressource.

La mise en œuvre a montré que cet outil est efficace. ll a été utilisé en BUT1 informatique à Nantes en 2021-2022 et en 2022-2023, de manière ad hoc, sur deux promotions d'environ 100 étudiants découpées en 4 groupes TD et 8 groupes TP. Actuellement, aucune réelle évaluation systématique des apports n'a été réalisées, par exemple, en réalisant des séances "avec/sans" et en comparant le travail réalisé par les étudiants. Notre objectif actuel est d'effectuer une évaluation plus formelle d'UMLChecker afin de perfectionner l'outil. Nous envisageons de soumettre des groupes d'étudiants à des séances de travaux pratiques, avec ou sans cas de tests de conformité, et d'évaluer leurs réalisations. Nous envisageaons aussi d'utiliser ce dispositif afin d'analyser les erreurs faites par les étudiants en fonction du diagramme proposé en utilisant des dispositifs de trace (avec xAPI [5]). Cela nous permettra aussi de voir comment les étudiants utilisent les retours et en quoi les messages proposés leur permettent de progresser.

5. https ://xapi.com/

Références

1. K. Beck. *Test Driven Development : By Example.* Addison-Wesley Professional, 2002.
2. Guy Brousseau. Théorie des situation didactiques. *Recherches en Didactiques des Mathématiques. Paris : La pensée Sauvage*, 1998.
3. Licence professionnelle "bachelor universitaire de technologie" - informatique - programme national 2022, 2022. MAJ 2023 - https ://www.enseignementsup-recherche.gouv.fr/fr.
4. S. Gulati and R. Sharma. *Java Unit Testing with JUnit 5 : Test Driven Development with JUnit 5.* Apress, 2017.
5. R. B. Jackson and J. W. Satzinger. Teaching the Complete Object-oriented Development Cycle, Including OOA and OOD, with UML and the UP. *Inf. Syst. Educ. J*, 1(28) :1–20, 2003.
6. *JUnit 5*, 2023. https ://junit.org/junit5/.
7. P. Leiding and H. Salmela. *IS2020 : A Competency Model for Undergraduate Programs in Information Systems.* Association for Computing Machinery (ACM), Association for Information Systems (AIS), Education SIG of the Association for Information Systems and Computing Academic Professionals (ED-SIG), December 2020.
8. L. Madeyski. *Test-Driven Development.* Springer, 2009.
9. S. Moisan and J.-P. Rigault. Teaching Object-Oriented Modeling and UML to Various Audiences. In *Models in Software Engineering*, volume 6002, pages 40–54. Springer Berlin Heidelberg, 2010.
10. J. Printz and J.-F. Pradat-Peyre. *Pratique des tests logiciels.* Dunod, 2021.
11. R. Rivera-Lopez, E. Rivera-Lopez, and A. Rodriguez-Leon. Another approach for the teaching of the foundations of programming using UML and Java. In *WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering.* World Scientific and Engineering Academy and Society, 2009.
12. D. Silva, I. Nunes, and R. Terra. Investigating code quality tools in the context of software engineering education. *Comp Applic In Engineering*, 25(2) :230–241, March 2017.
13. J. Van Eyck, N. Boucké, A. Helleboogh, and T. Holvoet. Using code analysis tools for architectural conformance checking. In *Proceedings of the 6th International Workshop on SHAring and Reusing Architectural Knowledge*, pages 53–54. ACM, May 2011.

Quatrième partie

Résumés

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Améliorer le retour aux étudiants par des tests unitaires générés à partir de motifs trouvés dans le code de leurs programmes

Julien Liénard[1 [0009−0009−2966−7179]], Kim Mens[1 [0000−0003−0303−1630]], and Siegfried Nijssen[1 [0000−0003−2678−1266]]

UCLouvain, ICTEAM, Louvain-la-Neuve, Belgium

Abstract. Les premiers cours de programmation a l'université comprenant de nombreux étudiants utilisent couramment des correcteurs automatiques pour fournir aux étudiants des corrections et des retours automatisés sur les exercices de programmation de base. Implémenter un tel retour pour l'intégrer dans un correcteur automatique est une tâche chronophage pour les enseignants. De plus, ce retour est souvent biaisé par la perception de l'enseignant des erreurs que les étudiants pourraient commettre, plutôt que sur les erreurs qu'ils commettent réellement. Nous présentons un prototype d'outil et d'une méthodologie de soutien pour résoudre ce problème. La méthodologie commence par l'analyse du code des programmes soumis par les étudiants des années précédentes, afin d'identifier des motifs récurrents dans leurs solutions. Nous classifions ces motifs en fonction des scores des étudiants: est-ce qu'ils ont réussi ou raté l'exercice? Ensuite, notre outil génère automatiquement des tests unitaires pour ces motifs qui correspondent typiquement aux mauvaises pratiques, aux failles ou aux anomalies observés dans le code des étudiants. Ces tests générés peuvent facilement être intégrés dans un correcteur automatique afin de donner un retour sur ces mauvaises pratiques observés.

Keywords: Enseignement de l'informatique · Minage de motif · Test unitaire · Génération de code · Correcteur et retour automatique

Volume 1, pages 156–156.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Des métaphores pour la programmation de robots

Oregan Segalen[1], Natacha Caouren[1] et Vincent Ribaud[1]

ENSTA Bretagne, Brest, France
prenom.nom@ensta-bretagne.fr

Résumé Cet article présente, au travers de métaphores, cinq exercices de programmation Scratch de robots, évalués lors d'une compétition.
En premier, le robot se déplace le long d'un carré. La métaphore est celle d'un dessin sur une feuille. Les déplacements du robot se programment en paramétrant le temps de déplacement ; la métaphore du dessin fait appel à une métaphore sous-jacente "un temps qui passe est un mouvement", cette seconde métaphore reposant sur le concept de proportionnalité. Bien que cet exercice plaise aux enfants, seulement 1/3 arrivent à le résoudre en autonomie. Les enfants, procédant par essai-erreur, ont du mal à établir le temps nécessaire à une rotation de 90°.
En second, le robot clignote en bleu en jouant la note A4, en blanc en jouant D4. Les enfants ont appris à programmer un feu tricolore et sonore, sans qu'on leur explique que lumière et son sont gérés concurremment. Cet exercice est difficile, il leur manque la métaphore de l'orchestre. Le chef d'orchestre - un ordonnanceur - distribue les ordres aux différents instrumentistes - son ou lumière. Cette métaphore révèlerait une machine notionnelle simple et observable, qui permettrait une meilleure réussite.
En troisième, le robot évite un obstacle devant lui en tournant. La métaphore est celle de la chauve-souris qui permet d'expliquer le fonctionnement du capteur ultrason. L'observation du comportement d'évitement réifie son algorithme qui complète la métaphore et la rend opérationnelle.
En quatrième, le robot joue le rôle d'un chronomètre qui affiche les minutes et les secondes. La métaphore est celle d'un compteur kilométrique : les chiffres défilent et il y a propagation des retenues. Tous les enfants détectent les cas d'erreur (i.e. les secondes dépassent 60). Peu d'enfants peuvent gérer la situation sans aide, probablement car aucun algorithme de comportement ne peut être réifié de l'observation d'un compteur. Des capacités de programmation avec des variables sont nécessaires.
En cinquième, le robot suit une ligne noire sur fond blanc. Ce suivi fait appel à deux capteurs noir/blanc qui permette de détecter si le robot est sur la ligne noire, partiellement sorti à gauche ou à droite, complètement sorti. En CM2, cet exercice est résolu par induction, en faisant observer un programme de suivi et en complétant un programme "à trous".
En conclusion, une compétition de robotique est un dispositif qui engage et motive les élèves, qui sont des facteurs de réussite. Les métaphores « aident à comprendre » le fonctionnement des éléments du robot et le comportement des programmes. Elles sont utiles aussi pour les séances débranchées. Lorsque l'exercice nécessite une maitrise de la programmation, la métaphore doit être complétée par des algorithmes.

Keywords: Scratch · robotique · métaphore

Volume 1, pages 157–157.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Éduquer au numérique en multipliant les points de vue

Halaert Élise[1a], Henry Julie[1a,b][0000-0003-4354-9848] et Payn Mathieu[2][0009-0005-6493-2223]

[1a] Université de Namur, NAmur Digital Institute (NADI), Faculté d'Informatique, Belgique
[1b] Université de Namur, Institut de Recherches en Didactiques et Éducation de l'UNamur (IRDENa), Belgique
prenom.nom@unamur.be
[2] Haute École Pédagogique Fribourg | Pädagogische Hochschule Freiburg & Université de Fribourg
mathieu.payn@edufr.ch
Les auteur·e·s ont contribué·e·s de façon égale à cet article.

Résumé. L'éducation au numérique fait son retour dans les programmes scolaires des écoles en Belgique francophone. Si cette réintroduction est nécessaire au regard de ce qui se passe ailleurs en Europe, elle oblige les futur·e·s enseignant·e·s à devenir, au minimum, des spécialistes du cadre de références DigComp. Étant donné le volume de savoirs à acquérir et le temps imparti dans la formation initiale des enseignants à cet apprentissage, il s'agit avant tout de se concentrer sur des savoirs développant l'autonomie numérique des citoyen·ne·s de demain qui restent accessibles sans trop d'effort à la majorité des enseignant·e·s. Dans leur intervention, les auteur·e·s s'essaient à identifier certains de ces savoirs « accessibles » à travers une approche pluridisciplinaire (sciences de l'éducation, informatique, psychologie, sociologie) et réfléchissent ainsi à l'intérêt de la « décentration » et de la multiréférentialité dans un contexte d'éducation au numérique.

Mots-clé: didactique du numérique, éducation à la citoyenneté numérique,approche multiréférentielle, approche pluridisciplinaire

Volume 1, pages 158–158.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Modélisation des connaissances mobilisées dans une situation de généralisation de motif

Gaëlle Walgenwitz[1], Marie-Caroline Croset[1] [0000-0003-2399-5883] et Emmanuel Beffara[1] [0000-0003-1993-3401]

[1] Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, 38000 Grenoble, France
Gaelle.Walgenwitz@univ-grenoble-alpes.fr
Marie-Caroline.Croset@univ-grenoble-alpes.fr
Emmanuel.Beffara@univ-grenoble-alpes.fr

Résumé. Cette communication présente une recherche visant à concevoir un modèle praxéologique des connaissances mobilisées lors de la généralisation de motif en contexte de programmation. Ce modèle se fonde, d'une part, sur une typologie de raisonnements algébriques et, d'autre part, sur des outils issus de la théorie anthropologique du didactique. Nous présentons un générateur de types de tâches qui permet la création d'une situation nécessitant la mobilisation du concept de boucle. La mise en œuvre de cette situation expérimentale chez des élèves de 11 ans et l'analyse des traces d'activité ont permis de valider le choix du générateur et d'enrichir ce modèle de connaissances. Ce travail souligne l'importance de l'explicitation des concepts et pratiques informatiques, et met en lumière le rôle essentiel de l'identification du motif dans le processus de généralisation, tant en algèbre qu'en algorithmique. Il ouvre ainsi des perspectives pour l'enseignement de l'informatique, comme l'exploration d'autres objets-frontière afin d'enrichir l'enseignement de cette discipline.

Mots clés : pensée algorithmique, motif, praxéologie

Volume 1, pages 159–159.

Cinquième partie

Ateliers

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Concevoir un robot avec des élèves de 10-12 ans

Felipe Martinez <felipe.martinez@epfl.ch> (1)

1 - Ecole Polytechnique Fédérale de Lausanne (Suisse)

Résumé · Cet atelier présente une activité débranchée visant la conception d'un robot en réponse à un besoin spécifique. Reposant sur une approche méthodique, elle est principalement destinée à des élèves de 10-12 ans. Avant d'entamer le processus de conception, plusieurs étapes préliminaires sont entreprises. Elles incluent notamment l'observation et la classification de robots selon différents critères, ainsi que la définition et le partage des avantages et des inconvénients associés à leur utilisation. La variété des composants embarqués dans un robot est également abordée afin de saisir l'articulation fonctionnelle "capteurs-programmes-actionneurs". Ces différentes étapes conduisent les élèves à effectuer des choix réfléchis et à se positionner de façon critique quant à la place des robots dans notre société. Ils · elles sont finalement amené · e · s à concevoir leur robot à l'aide d'un plateau de conception et de cartes à agencer. La présente ressource a été développée dans le cadre du projet EduNum du Canton de Vaud, Suisse, et est diffusée sous licence Creative Commons (CC BY NC SA 4.0).

Mots Clés · Education Numérique, robotique, activité débranchée, conception, démarche par projet

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Découverte de l'électronique, de la Micro :bit à la carte Arduino

Céline Colas <contact@kodowallonie.be> (1)

1 - Kodo Wallonie (Belgique)

Résumé · Afin de répondre aux demandes actuelles reprises dans les nouveaux référentiels de la Fédération Wallonie-Bruxelles, l'asbl Kodo Wallonie a fait un bref état des lieux des ressources disponibles pour les enseignants pour qu'ils puissent s'approprier les domaines travaillés et encadrer leur classe. Dans le but de combler un manque dans le domaine "objets technologiques", un carnet pour les élèves, accompagné du guide enseignant, a été développé. À cette fin, l'équipe de Kodo Wallonie s'est reposée tant sur les écrits autour de la progression de l'élève dans les apprentissages que sur la didactique de l'informatique. Le carnet a été créé itérativement grâce aux animations scolaires et est proposé aux enseignants pour expérimentation tout au long de l'année scolaire 2023-2024 afin d'être amélioré.

Mots Clés · Informatique, Électronique, Référentiel, FMTTN, Progression

Volume 1, pages 162–162.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Exercices autocorrigés avec la plateforme INGInious

Anthony Gego <anthony.gego@uclouvain.be> (1)

1 - UCLouvain (Belgique)

Résumé · INGInious est une plateforme open-source et libre d'accès permettant la conception d'exercices corrigés automatiquement. Elle supporte les exercices de programmation de type code, par bloc, interactifs en ligne de commande, l'analyse de trace réseaux ou encore les formules mathématiques. Les réponses de l'étudiant sont évaluées à l'aide d'une logique de correction intégrée ou spécifiée manuellement au moyen de scripts et d'une interface de programmation. L'enseignant dispose d'une interface web pour gérer son cours et consulter les soumission des étudiants afin d'améliorer le feedback des exercices. L'atelier vise à prendre en main de la plateforme, la création de séquences de cours sur base d'un catalogue d'exercices existants ainsi que la conception de nouveaux exercices.

Mots Clés · exerciseur, juge en ligne, lms, correction automatique, oer, lti, programmation, ligne de commande réseaux

Volume 1, pages 163–163.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Jeu de rôle sur les enjeux de l'automatisation de la conduite : Des navettes autonomes dans ma commune ?

Sonia Agrebi <sonia.agrebi@epfl.ch> (1)

1 - Ecole Polytechnique Fédérale de Lausanne (Suisse)

Résumé · Cet atelier présente une ressource pédagogique débranchée pour sensibiliser les élèves aux enjeux liés à la conduite autonome, tout en ajoutant une touche ludique à leur apprentissage. Pour cette activité, la classe se transforme en un conseil communal fictif, créant ainsi un jeu de rôle autour de l'introduction des navettes autonomes. Les élèves se voient attribuer divers rôles, ce qui contribue à leur engagement. Ils · elles débattent de l'adoption de ces navettes dans leur commune et explorent les différentes perspectives des acteurs impliqués de manière interactive. Les ressources nécessaires à cette activité ont été conçues dans le cadre de la formation CAS pour l'Enseignement de la Science Informatique en secondaire 1 faisant partie du projet EduNum du Canton de Vaud, Suisse. Elles sont destinées aux élèves âgé · e · s de 12 à 15 ans. Cet atelier offre une opportunité aux participant · e · s de découvrir cette ressource pédagogique ainsi que sa mise en pratique en classe.

Mots Clés · Activité débranchée, Jeu de rôle, Navettes autonomes, Enjeux de l'automatisation et de l'intelligence artificielle

Volume 1, pages 164–164.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Jeux de cartes sérieux et notions de culture informatique

Felipe Martinez <felipe.martinez@epfl.ch> (1), Yann Secq <yann.secq@epfl.ch> (1)

1 - Ecole Polytechnique Fédérale de Lausanne (Suisse)

Résumé · Cet atelier présente une modalité pédagogique débranchée à la croisée d'un escape game et d'un livre dont vous êtes le héros pour initier des élèves de 13 à 16 ans à des notions de culture informatique liées à la cybersécurité. L'objectif est de plonger les élèves dans un scénario les invitant à comprendre une situation en découvrant des notions de culture informatique lors de leur parcours. à travers deux jeux de cartes, les élèves plongent dans un scénario les invitant à réaliser des choix et à se confronter à une variété de problématiques : le premier jeu se centre principalement sur des situations pouvant être rencontrées quotidiennement par les élèves et le second sur d'autres plus complexes et propres à un contexte professionnel. L'atelier permettra aux participant · e · s d'expérimenter cette modalité avant de découvrir la conception scénaristique et technique de ces jeux sérieux. Cette modalité a été développée dans le cadre de la formation continue CAS pour l'Enseignement de la Science Informatique en Secondaire 1 au sein du projet pilote Edunum du canton de Vaud en Suisse.

Mots Clés · Jeu sérieux, activité débranchée, culture générale sur la cybersécurité

Volume 1, pages 165–165.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

La machine R2E2 pour l'enseignement de l'architecture des ordinateurs en première NSI

Christophe Declercq <christophe.declercq@univ-reunion.fr> (1)

1 - Laboratoire d'Informatique et de Mathématiques (France)

Résumé · Nous proposons une architecture de machine permettant de faire manipuler les élèves, pour exécuter pas à pas un programme simple de type langage machine. La « machine en papier » permet tout d'abord une activité débranchée où les élèves partagent les rôles de la partie contrôle et de la partie opérative. Une activité en ligne, utilisant le simulateur logique pour l'enseignement (Pellet & Parriaux, 2022), permet ensuite de vérifier les résultats obtenus et de visualiser le comportement de la machine.

Mots Clés · Architecture des ordinateurs, machine de Von Neumann, informatique débranchée, NSI

Volume 1, pages 166–166.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Modéliser les flexagones grâce à l'utilisation des graphes

Maryline Althuser <maryline.bruel@ac-grenoble.fr> (1), Anne Rasse <anne.rasse@univ-grenoble-alpes.fr> (2), Benjamin Wack <benjamin.wack@univ-grenoble-alpes.fr> (2), Gaëlle Walgenwitz <Gaelle.Le-Corre@ac-grenoble.fr> (1)

1 - Institut de Recherche sur l'Enseignement des Mathématiques [Grenoble] (France)
2 - Institut de Recherche sur l'Enseignement des Mathématiques [Grenoble] (France)

Résumé · Dans cet atelier, nous explorons les flexagones, une curiosité mathématique rendue célèbre par Martin Gardner en 1956. Nous nous concentrons spécifiquement sur la modélisation des transformations subies par un flexagone lors de sa manipulation. Ces transformations sont représentées au moyen d'un automate, un formalisme largement utilisé dans divers domaines informatiques comme la théorie des langages ou les systèmes discrets. Les premières manipulations de flexagones et un questionnement adéquat permettent de faire émerger les notions d'état et de transition. Les participants sont encouragés à créer une trace écrite ou schématique de leurs manipulations. Une discussion approfondie de ces traces met en lumière les processus de modélisation utilisés et les avantages attendus de cette approche. Nous présentons aussi un retour des expérimentations menées dans des classes de collège et des évolutions des choix didactiques autour cette activité.

Mots Clés · modélisation, flexagone, cycle, automate, état, transition

Volume 1, pages 167–167.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Numérique et environnement : Cartographie du cycle de vie de nos équipements numériques

Sonia Agrebi <sonia.agrebi@epfl.ch> (1)

1 - Ecole Polytechnique Fédérale de Lausanne (Suisse)

Résumé · Cet atelier présente une activité pédagogique débranchée pour sensibiliser les élèves aux enjeux environnementaux et sociaux liés à l'utilisation des technologies du numérique sur la base d'une cartographie de leur cycle de vie : fabrication, utilisation et fin de vie. L'activité vise une prise de conscience de la matérialité du numérique et de ses multiples conséquences. Elle encourage également les élèves à réfléchir aux actions possibles, à la fois individuelles et collectives, pouvant être entreprises pour limiter voire réduire les impacts environnementaux du numérique. Les cartes servant à la réalisation de la cartographie ont été réalisées dans le cadre du projet EduNum du Canton de Vaud, Suisse. Elles sont destinées à des élèves de 12 à 15 ans. Des déclinaisons adaptées aux élèves plus jeunes et plus âgés existent également. Cet atelier offre aux participant · e · s l'occasion de découvrir cette activité ainsi que sa mise en œuvre avec des élèves.

Mots Clés · Activité débranchée, Cycle de vie, Sobriété numérique, Impacts environnementaux du numérique

Volume 1, pages 168–168.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Platon une présentation

Dominique Revuz <dominique.revuz@univ-eiffel.fr> (1), Olivier Champalle <olivier.champalle@univ-eiffel.fr> (2)

1 - Laboratoire dÍnformatique Gaspard-Monge (France)
2 - Dispositifs d'Information et de Communication à l'Ère du Numérique - Paris Île-de-France (France)

Résumé · Présentation de la plateforme PLATON. Platon est avec ces Cercles un environnement de gestion et de mise en œuvre de ressources ouvertes. Les ressources sont basées sur des activités programmables adaptables pour de nombreuses disciplines. La présentation en atelier est réalisée pour chercher des partenaires afin de concevoir un module d'analytics permettant la valorisation de stratégies pédagogiques en éliminant en amont les biais expérimentaux et permettre la réutilisation (ou la re-expérimentation) de dispositifs développés dans la cadre de recherches en pédagogique ou en didactique. L'atelier montrera les différentes possibilités de la plateforme en terme d'exercices, d'activité, le look and feel.

Mots Clés · Atelier Exerciseur programmable, activités pédagogiques, stratégies pédagogiques

Volume 1, pages 169–169.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Un jeu de plateau pour comprendre la dualité en logique

Emmanuel Beffara <emmanuel.beffara@univ-grenoble-alpes.fr> (1), Martin Bodin <martin.bodin@inria.fr> (2)

1 - MeTAH; LIG (France)
2 - Spades; Inria; LIG (France)

Résumé · Cet atelier présente une activité sous forme de jeu de plateau à deux joueurs destinée à construire une représentation intuitive des concepts d'opérateurs logiques, de quantificateurs, et des règles de raisonnement associées. La conception met en avant la dualité entre opérateurs et la symétrie entre preuve et réfutation, dans un décor inspiré de l'imaginaire pirate où des cartes au trésor scénarisent des formules logiques. Un jeu de base correspond à la logique propositionnelle et une extension ajoute des quantificateurs et prédicats. On présentera des retours d'expérience issus d'une utilisation du dispositif dans un contexte de médiation auprès d'élèves de lycée.

Mots Clés · logique, dualité, médiation, jeu

Volume 1, pages 170–170.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Une approche de la notion de récursivité à l'aide d'outils de visualisation graphique

Charles Poulmaire <charles.poulmaire@ac-versailles.fr> (1), Pascal Remy <pascal.remy@ac-versailles.fr>

1 - Ministère de l'éducation nationa (France)

Résumé · L'introduction de l'Informatique dans les programmes de l'Enseignement Secondaire en France date de 2012 avec l'apparition de l'option ISN en Terminale. A l'issue de la réforme du lycée général en 2019, celle-ci est devenue une discipline à part entière dont l'enseignement s'étale sur les deux années du cycle terminal (classes de Première et Terminale). L'un des points clés de cet enseignement est la récursivité, introduite en classe de Terminale, après seulement un an de formation initiale. Ce point est souvent difficile à aborder avec les élèves, en raison de l'abstraction sous-jacente à cette notion qui peut entraîner de nombreuses difficultés de compréhension. Dans cet atelier, nous proposons, sous la forme de notebooks Jupyter, deux ressources permettant une meilleure appréhension par les élèves des notions de base de la récursivité (appels récursifs, cas de base et piles d'exécution) en nous appuyant sur une stratégie liée à la manipulation de trois outils de visualisation graphiques : Python Tutor, Code Puzzle et Recursion Visualizer.

Mots Clés · récursivité, programmation, outils graphiques, informatique

Volume 1, pages 171–171.

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Volume 1, pages 172–172.