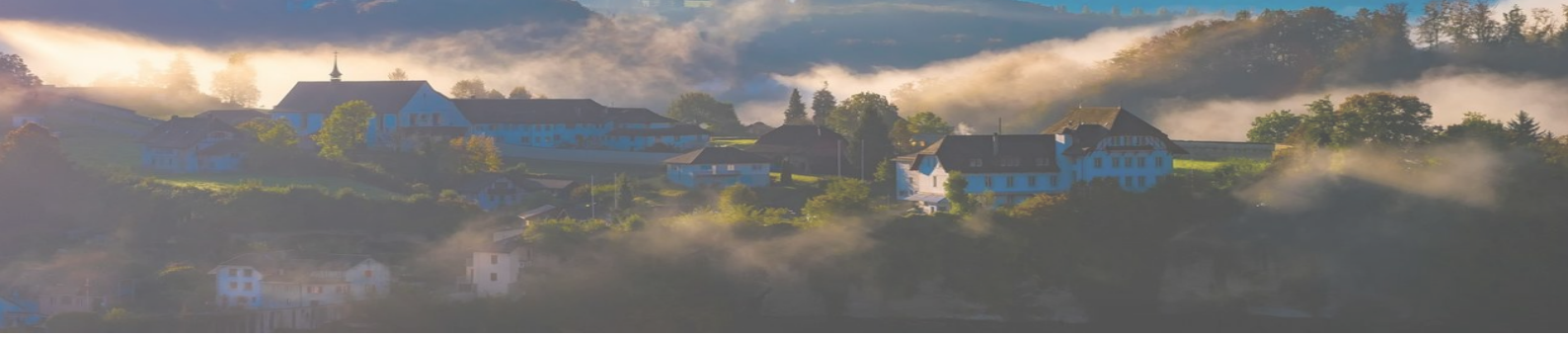
Actes de l'atelier APIMU 2021 @ EIAH : Apprendre la Pensée Informatique de la Maternelle à l'Université

Julien Broisin, Christophe Declercq, Cédric Fluckiger, Yannick Parmentier, Yvan Peter, Yann Secq



APIMU 2021

http://eiah21.apimu.org

Actes de l'atelier
Apprendre la Pensée Informatique
de
la Maternelle à l'Université

Lundi 07 juin 2021 (distanciel)
Dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH)
Fribourg, Suisse

Julien Broisin, Christophe Declercq, Cédric Fluckiger, Yannick Parmentier, Yvan Peter, Yann Secq

(Éds.)

Ces actes sont disponibles sur l'archive ouverte HAL à l'adresse

https://hal.archives-ouvertes.fr/hal-03241714

Préface

Depuis une vingtaine d'années, l'enseignement de l'informatique n'était plus réalisé que dans le supérieur. Un mouvement de réintroduction de l'informatique dans le contexte scolaire en primaire et secondaire s'est produit à différents rythmes dans certains pays anglo-saxons et européens depuis une dizaine d'années. Depuis 5 ans environ, en France et plus récemment en Suisse, ces enseignements ont trouvé une part encore plus importante dans l'ensemble des cycles suite à leur inscription dans les programmes scolaires. Cette introduction massive de l'enseignement de l'informatique dans les programmes scolaires induit des problématiques cruciales en termes de didactique d'une discipline principalement enseignée dans le supérieur ces dernières années, et de formation des enseignant · e · s qui dispensent ces cours dans le primaire et le secondaire.

Cet atelier qui poursuit les sessions organisées depuis 2017 (Journées Orphée RDV 2017, EIAH 2017, RJC'EIAH 2018, EIAH 2019) vise à fédérer une communauté de chercheurs et chercheuses issu · e · s de différents champs disciplinaires autour de la problématique de l'apprentissage de la pensée informatique de la maternelle à l'université afin de bâtir une vision commune du domaine, d'identifier les principales problématiques et de proposer des recherches et ressources pour développer la didactique de l'informatique et sa diffusion à large échelle auprès des enseignant · e · s.

Comme lors des sessions précédentes, les contributions attendues devront principalement porter sur les axes suivants :

— Apprendre et enseigner la pensée informatique,
— Former et accompagner les enseignant · e · s,
— Produire, évaluer et diffuser des outils et des ressources éducatives.

Les contributions qui accordent une attention particulière aux problématiques liées à l'instrumentation de ces apprentissages sont encouragées et une attention particulière est apportée aux contributions s'intéressant au déséquilibre de genre en informatique. D'autre part, des retours d'expérience sur la mise en œuvre des nouveaux programmes scolaires intégrant l'apprentissage de l'informatique sont les bienvenus.

Cet atelier sera co-organisé pour la seconde fois avec le projet ANR IE CARE (enseignement de l'informatique à l'école obligatoire, 3-16 ans) et le projet ERASMUS+ PIAF (Pensée Informatique et Algorithmique dans l'enseignement Fondamental) offrant l'opportunité d'un regard croisé sur les enjeux de cette massification de l'enseignement de l'informatique du plus jeune âge jusqu'aux adultes.

Comités

Comité d'organisation

— Julien BROISIN (Université de Toulouse)
— Christophe DECLERCQ (INSPÉ de Nantes)
— Cédric FLUCKIGER (Université de Lille)
— Yannick PARMENTIER (Université de Lorraine)
— Yvan PETER (Université de Lille)
— Yann SECQ (Université de Lille)

Comité scientifique

— Georges-Louis BARON (Université de Paris)
— Julien BROISIN (Université de Toulouse)
— Isabelle COLLET (Université Genève)
— Christophe DECLERCQ (INSPÉ de Nantes)
— Brigitte DENIS (Université de Liège)
— Liesbeth DE MOL (Université de Lille)
— Marie DUFLOT-KREMER (Université de Lorraine)
— Cédric FLUCKIGER (Université de Lille)
— Colin de la HIGUERA (Université de Nantes)
— Bern MARTENS (Universiteit Leuven)
— Yannick PARMENTIER (Université de Lorraine)
— Gabriel PARRIAUX (HEP Lausanne)
— Yvan PETER (Université de Lille)
— Robert REUTER (Université du Luxembourg)
— Margarida ROMERO (Université Côte d'Azur)
— Yann SECQ (Université de Lille)
— Thierry VIÉVILLE (Inria Sophia Antipolis)

Comité de lecture

— Florent BECKER (Université Orléans)
— Charles BOISVERT (Université Sheffield)
— Laure BOLKA-TABARY (Université de Lille)
— Kris COOLSAET (Universiteit Gent)
— Yannis DELMAS (Université de Poitiers)
— Fahima DJELIL (IMT Atlantique)
— Béatrice DROT-DELANGE (Université de Clermont-Ferrand)
— Olivier GOLETTI (Université de Louvain)
— Monique GRANDBASTIEN (Université de Lorraine)
— Pascal LEROUX (Le Mans Université)
— Kim MENS (Université de Louvain)
— Christophe REFFAY (Université Franche-Comté)
— Eric SANCHEZ (TECFA, Université de Genève)

Table des matières

Apprendre à penser les algorithmes

Christian Blanvillain
HEP Lausanne, Suisse & Université de Patras, Grèce
christian.blanvillain@hepl.ch

RÉSUMÉ

Nous présentons un algorithme de résolution de problèmes algorithmiques. Cet algorithme est utile aux enseignants qui souhaitent faire apprendre aux élèves comment trouver des solutions à des problèmes algorithmiques. Nous proposons un schéma des processus cognitifs mobilisés dans l'acte de conception d'algorithmes. C'est un outil d'analyse pragmatique susceptible d'aider les enseignants à identifier les éventuelles lacunes cognitives des élèves en difficulté. Il est accompagné d'une liste de stratégies utiles aux élèves. Nous concluons avec une réflexion sur les facteurs pouvant influencer l'aptitude à concevoir des algorithmes.

ABSTRACT

How to think algorithms.

We present an algorithm for solving algorithmic problems. This algorithm is useful for teachers who wish to teach students how to find solutions to algorithmic problems. We propose a schema of the cognitive processes mobilized in the act of designing algorithms. It is a pragmatic analysis tool that can help teachers to identify possible cognitive deficiencies of students in difficulty. It is accompanied by a list of useful strategies for students. We conclude with a discussion of factors that can influence the ability to design algorithms.

MOTS-CLÉS : Intelligence algorithmique, Fonctions cognitives, Stratégies cognitives, Noèsiologie.

KEYWORDS: Algorithmic intelligence, Cognitive functions, Cognitive strategies, Noesiology.

1 Introduction

Lorsque l'on souhaite enseigner l'informatique aux jeunes élèves, plusieurs questions se posent : peut-on se contenter de développer leur pensée informatique ou bien faut-il leur apprendre à programmer ? Finalement, l'essentiel n'est-il pas de leur apprendre à penser les algorithmes ? Développer la pensée informatique sans leur apprendre à penser les algorithmes, c'est passer à côté d'une opportunité extraordinaire de leur enseigner des stratégies cognitives pour la résolution de problèmes en cultivant un état d'esprit rigoureux et méthodique, qui leur serviront bien au-delà des exercices proposés et tout particulièrement dans les situations scolaires ou extrascolaires où une approche intuitive ne suffit plus pour venir à bout des obstacles rencontrés. Leur apprendre à développer leur intelligence algorithmique dès leur plus jeune âge, c'est leur apprendre à développer leur intelligence créative, pratique et logico-mathématique (en référence à la conception triarchique de l'intelligence de Robert J. Sternberg (1985)). Les transferts possibles dans les autres disciplines scolaires et dans la vie quotidienne sont nombreux. Nous les préparerons ainsi à mieux appréhender le monde numérique dans lequel ils évoluent quotidiennement.

Atelier « Apprendre la Pensée Informatique de la Maternelle à l'Université », dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH), Fribourg, Suisse, 07 juin 2021, pages 1–12,

2 Algorithme de résolution de problèmes algorithmiques

Dans ce chapitre, nous décrivons un algorithme présentant comment l'élève pourrait s'y prendre pour résoudre des problèmes algorithmiques. Cette théorie est issue d'une analyse introspective de notre propre processus de résolution de problème algorithmique, ainsi que de nombreuses lectures :

- Jean-Yves Fournier, *À l'école de l'intelligence* (1999, 2e partie) ;
- Pierre Vianin, *L'aide stratégique aux élèves en difficulté scolaire* (2009, p. 72 à 116) ;
- Todd Lubart, Christophe Mouchiroud, Sylvie Tordjman, Franck Zenasni, *Psychologie de la créativité* (2015, p. 117 et 122) ;
- Patrick Lemaire et André Didierjean, *Introduction à la psychologie cognitive* (2018, chap. 8).

2.1 Lecture et compréhension de l'énoncé

Pour résoudre un problème algorithmique, on commence par lire son énoncé. En première lecture, on peut facilement saturer notre mémoire de travail et ne pas comprendre le texte, ce n'est pas grave. Il ne faut pas paniquer et garder confiance en soi. Reprendre la lecture en essayant d'éliminer l'inutile et conserver les informations importantes pour identifier ce qui est demandé de faire ou de résoudre. Trier les informations fournies dans les consignes en cherchant les données de départ et les choses à produire **[Récolte]**. Cette étape est parfois nécessaire pour pouvoir être plus sélectif sur les informations pertinentes pour se représenter le travail à réaliser et les distinguer des informations qui le sont moins, qui posent juste le contexte. L'objectif est de ne conserver en mémoire de travail que les informations nécessaires pour que l'on puisse se créer une représentation mentale du problème à résoudre **[Assemblage]** en écartant les informations parasites. Il faut réussir à se représenter le problème à résoudre dans son ensemble : ce que l'on sait, ce que l'on cherche. Il s'agit de faire une synthèse de ce qui est demandé afin de pouvoir émettre une hypothèse heuristique sur la stratégie à utiliser et sur la démarche qui conduira à la solution **[Projection]**. Stratégie qu'il va falloir mentalement tester en une approche exploratoire rapide sur des données simples, pour voir si elle est efficiente ou pas. L'exploitation de la représentation mentale du problème à résoudre que l'on s'est créée en lisant l'énoncé va permettre d'explorer plusieurs possibilités en éliminant rapidement les mauvaises idées, juste en testant les situations triviales.

Parmi les hypothèses faites sur les différentes stratégies de résolution possibles, conserver celle qui nous semble être la plus simple à mettre en œuvre, même si ce n'est pas forcément la plus élégante. Attention cependant à ne pas se précipiter sur la première idée qui vient à l'esprit ni se lancer dans l'élaboration d'une réponse sans avoir pris le temps de vérifier si l'idée semble valable ou pas. Tester cette hypothèse : appliquer le plan d'action prévu sur les données mémorisées, toujours en commençant par les situations les plus simples, car on cherche encore à valider notre hypothèse heuristique de résolution. Si notre hypothèse est correcte, on a un début de solution. Continuer alors d'analyser les autres données pour résoudre la suite du problème. Si notre hypothèse bloque sur certaines données et ne permet plus d'obtenir la réponse à la question posée, revoir l'hypothèse de départ et envisager d'utiliser une stratégie moins simple, mais plus précise. Un changement d'hypothèse peut aussi avoir lieu en cours de résolution : l'hypothèse initiale peut tout à fait fonctionner pour les cas simples, mais pas pour tous les cas. Les résultats obtenus ne sont pas forcément tous à remettre en cause : il ne faut pas obligatoirement jeter les réflexions faites et les conclusions partielles auxquelles on est arrivé. Une manière de chercher de nouvelles hypothèses est de faire des liens avec ce que l'on connait en cherchant des similitudes avec d'autres situations déjà

rencontrées. Observer les ressemblances et les différences avec ce que l'on a déjà vécu par le passé peut nous aider à penser à une nouvelle manière de faire.

Lorsque l'on réussit à produire le résultat escompté dans les situations triviales, vérifier que la solution trouvée est consistante avec l'énoncé du problème. La compréhension partielle du problème va nous aider à mieux cerner ce qui est demandé et mieux comprendre ce qu'il faut faire pour pouvoir gérer les situations non triviales. Les réflexions initiales ne sont donc pas perdues. Elles étaient nécessaires pour arriver à ce stade du raisonnement. Elles nous permettent maintenant d'affiner nos réponses, de pouvoir reprendre l'ensemble de notre approche avec une conscience plus globale de ce qu'il faut faire en prenant en considération toutes les données du problème.

Le processus peut sembler coûteux en temps, mais il n'en est rien. Chaque étape permet de se familiariser davantage avec le problème et de mieux appréhender les difficultés, les choses à faire et à ne pas faire, les informations essentielles et les informations moins cruciales. Cette expertise qui se construit en nous va nous permettre de nous approcher, mentalement et de manière itérative vers ce qui est attendu. Ce chemin intellectuel de compréhension est nécessaire pour se construire une représentation mentale fidèle du problème à résoudre. Parfois, le problème est suffisamment simple pour qu'il n'y ait pas besoin de faire de démarche particulière : on comprend tout de suite ce qui est demandé et ce qu'il faut faire. Mais parfois, le véritable problème n'apparait pas tout de suite et se découvre au fil du processus de lecture et de tentative de résolution. L'établissement d'une stratégie de résolution s'effectue conjointement au processus de compréhension du problème. Si l'on ne réussit pas à trouver une stratégie de résolution satisfaisante, c'est peut-être que l'on n'a pas assez bien exploré les données du problème. Une boucle se crée alors, entre la relecture pour une meilleure compréhension du problème, la recherche d'une nouvelle hypothèse heuristique et la vérification de l'hypothèse retenue. Nous appelons cette boucle le cycle d'élaboration d'une image mentale du problème (figure 1).

2.2 Résolution du problème

La phase de résolution commence lorsque l'on met en œuvre la stratégie de résolution imaginée sur l'ensemble des données du problème en envisageant les cas plus complexes. Concrètement, nous allons manipuler (ou plutôt *mentipuler* ?) les informations disponibles (données du problème) pour déplacer avec les yeux les données en faisant attention à ne pas oublier les nouveaux états du système que nous sommes en train de produire au travers de nos manipulations mentales **[Réflexion]**.

À ce stade de la réflexion, il est très important de rester concentré sur la tâche : écarter tout évènement extérieur perturbateur, pour ne pas altérer notre état mental et la représentation que nous nous faisons du système. Notre mémoire de travail est fragile et éphémère. Si une perturbation que nous ne réussissons pas éviter à lieu (quelqu'un qui nous pose directement une question par exemple), ne pas hésiter à abandonner le raisonnement dans son ensemble et à recommencer depuis le début. Le nouvel état mental du système doit être fiable et dénué de toute erreur pour que nous puissions l'utiliser dans la suite du raisonnement, pour calculer les prochains états du système de manière itérative jusqu'à ce que nous atteignions la fin de notre algorithme de résolution associé à notre stratégie et à nos hypothèses. Ces opérations mentales sont nécessaires et importantes pour que nous puissions nous assurer d'avancer, dans le raisonnement, vers une solution au problème posé. De temps en temps, si l'image mentale devient floue ou imprécise, reprendre rapidement les dernières étapes pour rafraichir notre mémoire de travail et éliminer les risques d'erreur de raisonnement dus aux imprécisions inhérentes à notre mémoire de travail.

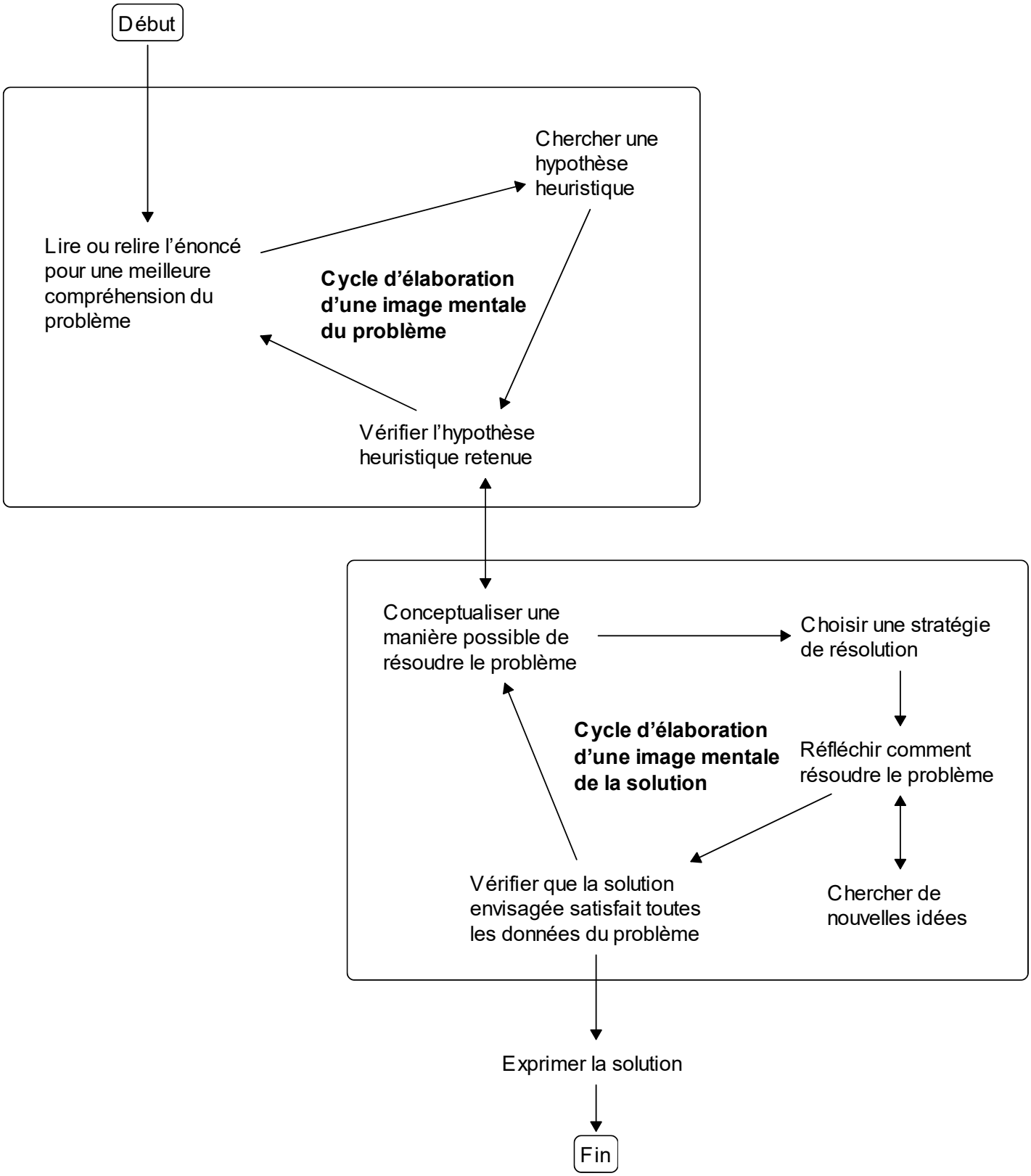


FIGURE 1 – Les cycles d'élaboration des images mentales

Concrètement, réfléchir au code à écrire ou bien aux opérations à décrire pour élaborer un algorithme solution au problème posé, consiste à visualiser dans son esprit les différents états du modèle mental du système en cours de conception, puis de décrire les opérations qui vont faire évoluer cet état pour obtenir un état objectif qui est soit l'objectif final, soit une étape intermédiaire que l'on imagine nous rapprocher de l'objectif final. La représentation des états internes du modèle du système que nous programmons mentalement se fait en utilisant une image de l'algorithme en cours de construction. Cette image permet de visualiser les valeurs et leurs évolutions. Nous pouvons nous représenter les différentes opérations à effectuer comme si c'était nous qui devions physiquement réaliser ces opérations. En nous projetant à la place du système que nous sommes en train de programmer, nous mobilisons notre intelligence pratique : il suffit de faire appel au bon sens et de réfléchir à ce que nous devrions faire et pouvons faire pour obtenir ce que l'on souhaite, puis une fois que ces opérations sont conscientisées, de les décrire en langage naturel.

Souvent, une bonne stratégie de résolution consiste à découper le problème en étapes plus simples à résoudre et se concentrer sur chacune des étapes pour trouver une solution sans se préoccuper pour le moment des autres étapes. Mais lorsque l'on ne sait plus quoi faire et que l'on est bloqué, il faut observer avec un peu de recul l'état global du système dans notre tête, vérifier que nous avons bien exploité toutes les données à disposition, que nous avons bien exploré tout ce qu'il est possible de faire avec ces données, y compris les configurations de données improbables lorsque le problème est corsé. On peut donc faire un inventaire des données déjà utilisées et traitées, des résultats de réflexion intermédiaires obtenus et des données qu'il reste à traiter pour atteindre l'objectif qui nous est demandé. Si l'on ne sait vraiment plus quoi faire après avoir fait cet inventaire, on peut repasser en revue toutes les informations de l'énoncé du problème au lieu de se contenter de reparcourir uniquement les données conservées dans notre mémoire de travail. La recherche d'une éventuelle information oubliée dans l'énoncé risque de nuire à notre représentation avancée de l'état du système et parfois, cet exercice nous obligera à reprendre notre raisonnement depuis le début pour intégrer la nouvelle information glanée dans l'énoncé. Cette gymnastique n'est pas aussi coûteuse en temps que ce que l'on pourrait croire, car refaire mentalement le chemin que nous venons juste de parcourir est assez rapide. Notre mémoire de travail va nous aider à aller bien plus vite la deuxième fois.

Si l'on est toujours bloqué dans l'étape de résolution du problème, avant d'abandonner notre hypothèse de travail, de tout jeter et de repartir sur une autre hypothèse plus complexe pour reprendre nos réflexions à partir de zéro, il convient de faire une pause. Relâcher la tension intellectuelle intérieure, calmer son esprit, laisser le cerveau se reposer quelques secondes en ne pensant à rien : une nouvelle idée pourrait alors jaillir spontanément nous permettant de nous débloquer **[Idée]**. À ce stade du raisonnement, il est hyper important de ne pas se laisser distraire par les évènements extérieurs, ou par nos émotions intérieures : stress, doute, peur de ne pas y arriver, peur de se tromper, peur d'être en échec, autant d'émotions qui viennent perturber notre tranquillité et notre raisonnement. Toutes ces émotions doivent être mises en sourdine, pour laisser au cerveau la possibilité d'activer sa pensée créatrice et nous montrer une alternative que nous n'avions pas jusqu'alors envisagée. Mais pour que la magie opère, il faut accepter de ne pas savoir quoi faire, de ne pas avoir de réponse, d'être perdu et rempli de doute, sans sourciller, sans paniquer, sans arrêter de faire l'effort de chercher à faire des associations d'idées pour en générer de nouvelles ou bien de ne penser à rien pour laisser émerger spontanément de nouvelles idées.

Si toujours rien ne se passe, nous pouvons passer en revue des techniques de résolution connues, vues précédemment en nous demandant comment faire autrement, en tentant résoudre le problème en lui soumettant des principes déjà vus, des approches apprises, jusqu'à ce qu'une piste de solution apparaisse et que nous puissions repartir dans une nouvelle hypothèse de résolution, avec une stratégie

adaptée. Nous pouvons également, à ce stade de la réflexion, nous aider de l'intelligence du groupe et demander aux autres élèves de partager leurs idées avec nous. Une idée fait émerger d'autres idées. Parfois, le chemin qu'emprunte la solution peut être assez détourné et c'est une idée à priori anodine qui pourrait nous mettre sur la voie d'une solution possible.

2.3 Expression de la solution

Lorsque l'on a terminé le processus de réflexion, que l'on a construit une solution à proposer, il faut vérifier qu'elle est bien juste **[Vérification]**. C'est une vérification plus lente qui s'opère par rapport à la vérification rapide de l'heuristique initiale, car ici on travaille avec une conscience complète de toutes les données du problème. Si la solution envisagée ne satisfait pas toutes les données du problème, chercher une nouvelle manière possible de résoudre le problème, adapter la stratégie de résolution et reprendre nos réflexions. Nous appelons cette boucle le cycle d'élaboration d'une image mentale de la solution (figure 1).

On peut être amené à élaborer des explications plus ou moins complexes pour présenter nos idées et notre résultat **[Expression]** à d'autres personnes. Lorsque l'on exprime notre solution, bien s'assurer que nos idées soient structurées, concises et compréhensibles par nos interlocuteurs. Enfin, lorsque tout est fini, repenser à ce qui a été réalisé. Se remémorer la stratégie qui nous a conduits à la résolution du problème posé et essayer alors d'identifier des situations similaires où cette stratégie serait à même de fonctionner pour favoriser les transferts, ou bien des situations déjà vécues dans lesquelles nous aurions déjà exploité cette stratégie de résolution. Ce faisant, nous créons des liens entre des expériences distinctes. Ces prises de conscience viennent alors enrichir notre propre expérience et nous préparent à mieux affronter des situations inédites.

3 Fonctions cognitives

Les fonctions cognitives listées dans cette section sont issues des travaux de plusieurs chercheurs :

- Christine Mayer qui reprend les travaux de Reuven Feurenstein
  dans son *Manuel de métapédagogie* (2005) ;
- Pierre Vianin, tableau récapitulatif
  dans *Neurosciences cognitives et pédagogie spécialisée* (2010) ;
- Todd Lubart, Franck Zenasni, Baptiste Barbot,
  dans *Créativité en éducation et formation* (2016) ;
- Todd Lubart, Christophe Mouchiroud, Sylvie Tordjman, Franck Zenasni,
  dans la *Psychologie de la créativité* (2015).

Le lien entre l'algorithme de résolution de problèmes algorithmique et les fonctions cognitives listées ci-dessous est réalisé par un mot clé en gras et entre crochets dans le texte des chapitres précédents. Il identifie l'étape du processus durant lequel nous considérons que les fonctions cognitives associées sont plus particulièrement susceptibles d'être mobilisées par l'élève. Nous avons regroupé les différentes étapes identifiées dans trois phases de l'algorithme : (a) lecture et compréhension de l'énoncé, (b) résolution du problème et (c) expression de la solution. Elles sont représentées dans le diagramme ci-dessous (figure 2) :

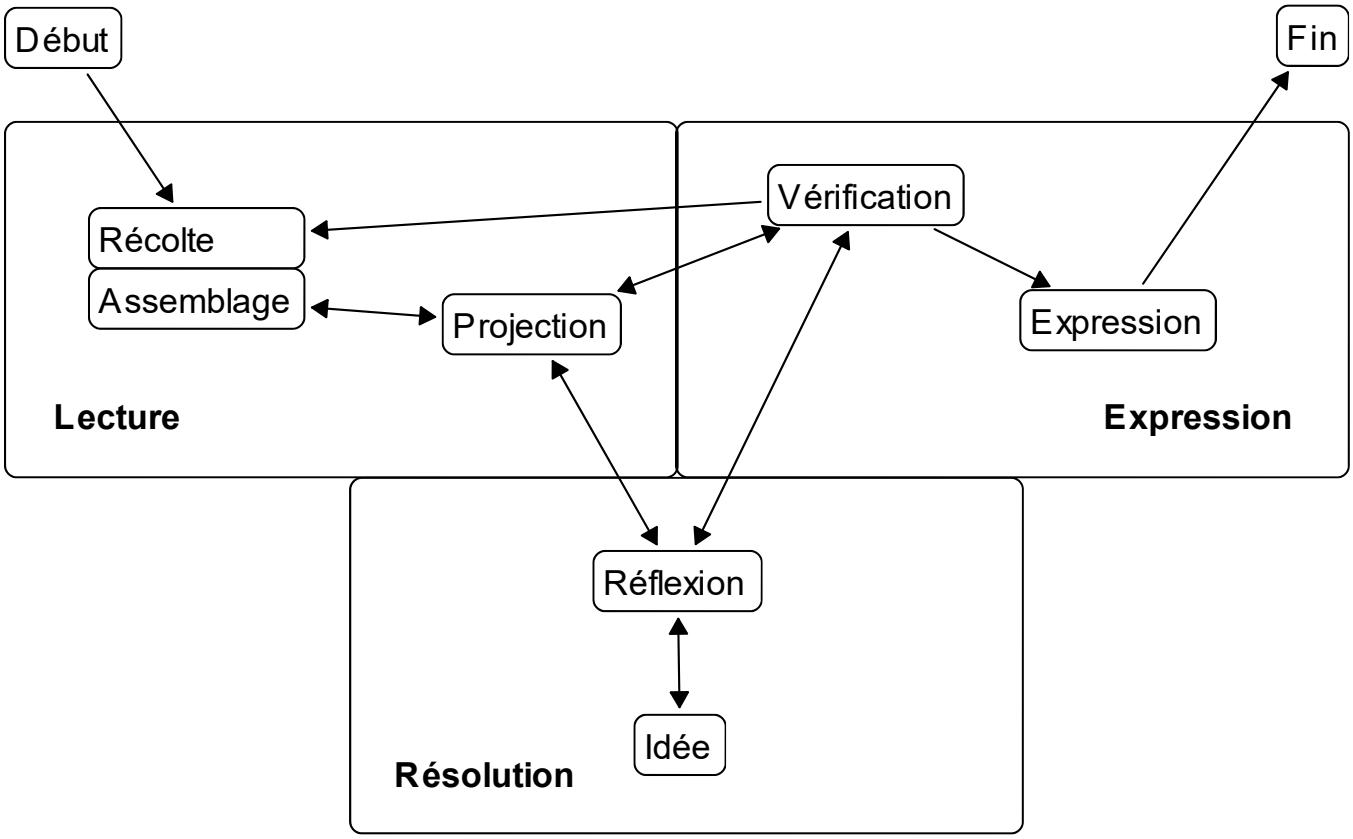


FIGURE 2 – Les étapes de l'algorithme de résolution de problèmes algorithmiques

Par souci de concision, nous listons ici uniquement les intitulés des fonctions cognitives sans en donner une description précise, que le lecteur retrouvera dans les travaux cités en référence. Ce travail de synthèse est un outil utile pour l'enseignant qui chercherait à faire une analyse clinique des éventuelles causes de blocage chez l'élève en difficulté.

- **[Récolte]** *La récolte des données* : attention, exploration, méthode, rigueur, identification, gestion.
- **[Assemblage]** *L'assemblage des données* : évocation, connexion, catégorisation, organisation, comparaison, sériation, sélection.
- **[Projection]** *La projection d'une stratégie de résolution* : reformulation, délimitation, fluidité, problématisation, anticipation, structuration, planification, modélisation.
- **[Réflexion]** *L'élaboration d'une réflexion* : inférence, abduction, rétention, déduction, induction, intériorisation.
- **[Idée]** *La recherche d'une idée* : désinhibition, flexibilité, pensée convergente, pensée divergente, pensée analogique, pensée intuitive.
- **[Vérification]** *La vérification d'une réponse* : récapitulation, justification, précision, circonspection, monitoring.
- **[Expression]** *L'expression d'une réponse ou d'une solution* : régulation, décentration, dénomination, autocontrôle.

Cette liste offre aux enseignants de nouvelles lunettes de compréhension sur les principaux mécanismes cognitifs potentiellement mis en œuvre par l'élève dans les étapes de conception d'une solution à un problème algorithmique. Elle ouvre de nouvelles pistes de travail de remédiation ciblé, lorsqu'une éventuelle défaillance cognitive est suspectée ou identifiée. Loin d'être exhaustive, cette liste pragmatique se concentre sur les 42 fonctions cognitives que nous considérons être les plus pertinentes dans la tâche de résolution de problèmes algorithmiques. Il se peut que les raisons du blocage de l'élève soient ailleurs, mais nous espérons que dans la majorité des situations l'enseignant trouvera dans ce répertoire une des origines possibles à ses difficultés d'apprentissage.

4 Stratégies cognitives

Les stratégies cognitives listées dans cette section reprennent une bonne partie des idées présentées dans l'algorithme de résolution de problèmes algorithmiques et les reformulent pour les rendre accessibles aux élèves. Certaines de ces stratégies sont directement inspirées de Mayer (2005). D'autres ont été élaborées à partir des discussions et réflexions de groupe faites en classe. C'est le fruit d'un travail empirique de médiation (Chappaz, 1995) de la part de l'enseignant réalisant, avec les élèves, une analyse réflexive et métacognitive des stratégies concrètes et opérationnelles utilisées. Elles sont utiles pour aider les élèves en difficulté à trouver des solutions aux problèmes algorithmiques proposés. Elles sont présentées aux élèves sous la forme d'une phrase mnémotechnique (fournie ici en italique), suivi d'un texte simple et compréhensible, leur permettant de se souvenir de la stratégie qui aura été au préalable discutée en classe. Ces 16 stratégies portent sur 4 moments distincts du processus de résolution d'un problème algorithmique :

1. Avant de se mettre au travail, pour apprendre à se concentrer et se mettre en condition de travail :
   - *Je suis capable de le faire.* Qu'est-ce que je dois faire ? Est-ce que j'ai bien toutes les données ? C'est OK de ne pas y arriver du premier coup ou de faire des erreurs : garde confiance en toi. Si c'était trop facile, tu n'apprendrais rien.

— *Je me concentre pour réussir.* Est-ce que tu as besoin qu'on te résume ce qu'il faut faire ? Focalise ton attention sur la tâche que tu es en train d'accomplir. Tu penseras au reste plus tard. Il y a un temps pour apprendre et un temps pour s'amuser.
— *Je résiste à donner la première réponse sans vérifier.* Est-ce que tu as pris le temps avant de donner ta réponse ? Vérifie bien l'exactitude de ton idée. Parfois, la vérité se cache ailleurs. Une approche critique et réfléchie permet d'éviter les pièges.
— *Je pense avant d'agir.* Comment ne pas répondre au hasard ? Demande-toi, si tu fais ça, alors qu'est-ce qui se passe ? Tu exploreras très rapidement beaucoup plus de possibilités en raisonnant dans ta tête qu'en essayant pour de vrai.

2. Pendant la recherche d'une solution, pour apprendre à résoudre des problèmes :
— *Je fais des liens avec ce que je connais.* Qu'est-ce qui est pareil ? Qu'est-ce qui est différent ? Cherche des similitudes avec ce que tu sais déjà et fais une hypothèse "c'est comme si…". Observe les différences et essaye de voir si ton intuition est juste.
— *Je me construis une image mentale du problème.* Qu'est-ce qui est utile ? Qu'est-ce qui ne l'est pas ? Qu'est-ce que je garde dans ma tête ? Représente-toi visuellement ce qu'il faut faire avant de commencer, ça va t'aider à mieux réfléchir.
— *Je manipule mon image mentale pour résoudre le problème.* Utilise l'image mentale que tu as construite pour chercher une solution. Déplace les choses avec tes yeux pour vérifier tes idées.
— *Je cherche à simplifier le problème.* Découpe le problème en plus petits problèmes ou bien résous un problème semblable, mais plus simple. Qu'est-ce qui est constant ? Qu'est-ce qui est variable ? Essaye de généraliser ta solution.

3. Au moment où l'on est bloqué, pour apprendre à être créatif :
— *J'explore une autre piste.* Lorsqu'une piste de réflexion ne mène à rien, fais marche arrière et envisage d'autres alternatives partout où tu as dû faire des choix.
— *Je change de chemin.* Tu es bloqué et tu as listé toutes les possibilités et leurs alternatives ? Cherche alors à changer d'approche, de stratégie ou à faire totalement autrement.
— *Je trouve de nouvelles idées.* Ferme les yeux et évacue toutes les pensées parasites en utilisant ton attention pour faire taire la petite voix dans ta tête. Des idées vont émerger d'elles-mêmes.
— *J'utilise l'intelligence du groupe.* Vraiment bloqué ? Cherche de l'aide et lorsque tu as trouvé, regarde si tu peux à ton tour aider les autres à trouver par eux-mêmes. Demande-toi si les autres vont comprendre ce que tu as fait ou ce que tu dis.

4. Lorsque le travail est terminé, dans une étape d'analyse réflexive pour enrichir les apprentissages :
— *Quand la tempête est passée, j'apprends à baisser les tours.* Lorsque je ne trouve pas, que je suis bloqué, je ne dois pas stresser ni paniquer. Je dois juste m'habituer à accepter le doute et l'incertitude. C'est la condition pour laisser émerger de nouvelles idées.
— *Je réfléchis aux chemins pris pour apprendre.* Quelles sont les étapes que tu as suivies ? Juste après avoir résolu un problème, pose-toi la question de ce qui t'a permis de résoudre le problème pour t'en souvenir plus tard.
— *Je réfléchis plus pour travailler moins et apprendre mieux.* Qu'est-ce que tu as vu ou entendu dans ta tête ? Comment as-tu su que c'était correct ? Dirige ton attention sur ta manière d'apprendre, pour apprendre mieux et avec moins d'efforts.
— *Je réfléchis aux applications futures.* Quel principe général peux-tu dégager de cette expérience ? Comment l'appliquer ailleurs ou autrement ? Imagine comment réutiliser ce que tu apprends pour pouvoir faire des liens avec des situations futures.

5 Conclusion

Finalement, le meilleur moyen de développer l'intelligence algorithmique de l'élève, c'est de la mobiliser sur des problèmes simples au début pour alléger sa charge cognitive (Chanquoy, Tricot, Sweller, 2007) développer la précision de son modèle mental (Johnson-Laird, 1983) et la puissance de sa mémoire de travail en activant ses neurones de manière répétée et espacée (Masson, 2020), puis sur des problèmes variés de plus en plus complexes, pour qu'il puisse acquérir de nouvelles stratégies de résolution de problème et enrichir ses expériences. Mais dans tous les cas, il faut être capable d'accepter le doute de ne pas savoir faire, pour dépasser cet état et apprendre, petit à petit, à développer sa créativité, son esprit pratique et sa logique mathématique : les trois composantes fondamentales de l'intelligence algorithmique.

L'aptitude à résoudre des problèmes algorithmiques chez l'élève dépendra de 5 paramètres :

- Sa faculté à rester concentrer sur la tâche en cours, malgré les perturbations extérieures, afin de réussir à se construire une représentation mentale fidèle de l'état du système durant chaque étape de son processus de transformation, au fil de l'avancement de la réflexion.
- L'étendue de sa mémoire de travail et le nombre d'objets mentaux qu'il sera à même de prendre en considération simultanément afin de se constituer un modèle mental fidèle du système avec lequel il doit travailler.
- De bonnes hypothèses et stratégies cognitives de résolution de problèmes, avec un riche panel d'expériences et de stratégies ayant eu du succès auparavant. Cette richesse ne s'acquiert qu'avec l'expérience. Il faut donc s'entrainer à concevoir des algorithmes pour développer cette faculté.
- Son aptitude à gérer ses doutes et ses émotions lorsqu'il sera bloqué et qu'il ne saura plus quoi faire. Sa capacité à créer un état de tranquillité intérieure, en situation de stress, afin de laisser au cerveau la possibilité de trouver de nouvelles idées face au problème rencontré. Sa faculté à cultiver sa créativité en situation de blocage en acceptant sereinement de ne pas savoir quoi faire, sans paniquer, sans douter, sans se soucier du fait de ne pas savoir.
- La puissance du plaisir narcissique qu'il ressentira en résolvant les premiers problèmes et qui déterminera sa motivation à explorer davantage de problèmes algorithmiques.

Ainsi, un élève qui manquera de confiance dans sa capacité d'apprendre à résoudre des problèmes algorithmiques, se retrouvera en difficulté non pas parce qu'il ne peut pas apprendre à les résoudre, mais par ce qu'il ne supportera pas l'état de doute et d'incertitude nécessaire pour traverser la phase où l'on ne sait ni quoi faire ni comment faire. Une autre cause possible à son manque de persévérance est son sentiment d'impuissance perçue (Le Bossé, 2016) : "à quoi bon faire l'effort de continuer d'essayer si de toute façon je perds mon temps, car je n'en suis pas capable ?". Il faut accepter avec humilité le fait que l'acte intellectuel qui conduit à la maitrise de l'intelligence algorithmique n'est pas inné, mais qu'il se développe avec du temps, des efforts et en utilisant de bonnes stratégies.

Nous pensons que l'explicitation de fonctions et de stratégies cognitives en classe est une aide précieuse pour l'enseignement de l'intelligence algorithmique au plus grand nombre. Être capable d'identifier les blocages potentiels chez les élèves en difficulté demande une connaissance fine des mécanismes cognitifs à l'œuvre dans la dynamique d'apprentissage de l'algorithmique. L'algorithme de résolution de problèmes algorithmiques est une aide, à la fois pour l'enseignant et pour les élèves, qui permet d'avoir une vision d'ensemble des processus cognitifs en œuvre lors de la conception d'une solution algorithmique à un problème posé. Son apprentissage permet de prendre conscience de la complexité de l'acte de résolution de problèmes algorithmiques et des difficultés qui y sont liées.

L'étude du développement de l'intelligence algorithmique, c'est-à-dire des intelligences créative, pratique et logico-mathématique des élèves, est en soi un vaste domaine. Il est nécessaire de se préoccuper de l'expérience vécue par l'élève dans son ensemble : de ne pas se limiter à la chose à apprendre, mais d'étendre notre enseignement au développement, en classe, de l'affect et de toutes les intelligences de l'élève, quelque soit la discipline enseignée, y compris dans l'acte même d'apprendre. Ainsi, nous pensons qu'il faudrait compléter la formation des enseignants par l'étude approfondie des mécanismes de développement des intelligences de l'élève. Nous nous devons d'étudier l'acte de compréhension de l'élève via la perception, l'imagination et la mémoire : c'est-à-dire étudier le *noos* (dans l'œuvre de Platon « intelligence, esprit en tant qu'il perçoit et qu'il pense ») ou la *noèse* (dans la pensée phénoménologique d'Edmund Husserl (Leclercq et Richard, 2016) « l'activité du sujet connaissant »). Pour ce faire, nous proposons d'envisager la création d'une nouvelle discipline, complémentaire à la didactique et la pédagogie, fondée sur la psychologie de la créativité, la psychologie cognitive et les neurosciences de l'éducation, dont l'objet d'étude serait le développement des intelligences de l'élève à l'école et que nous pourrions nommer : la *noèsiologie*.

6 Références

Chanquoy, L. Tricot, A. Sweller, J. (2007). *La charge cognitivie. Théorie et applications*. Paris : Armand Colin.

Chappaz, G. (1995). *Comprendre et construire la médiation*. Marseille : CRDP, Université de Provence.

Fournier, J-Y. (1999). *À l'école de l'intelligence*. Paris : ESF.

Johnson-Laird, P. N. (1983). *Mental Models*. Cambridge, MA : Harvard University Press.

Le Bossé, Y. (2016). *Sortir de l'impuissance. Tome 2 : aspects pratiques*. Québec : Ardis.

Leclercq, B. Richard, S. (2016). « Husserl (A) ». Dans M. Kristanek, *l'Encyclopédie philosophique*. En ligne : $https://encyclo-philo.fr/husserl-a$

Lemaire, P. Didierjean, A. (2018). *Introduction à la psychologie cognitive*. Bruxelles : De Boeck Supérieur.

Lubart, T. Mouchiroud, C. Tordjman, S. Zenasni, F. (2015) *Psychologie de la créativité*. Paris : Armand Colin.

Lubart, T. Zenasni, F. Barbot, B. (2016). Le potentiel créatif : de la mesure à son développement. Dans I. Capron Puozzo, *La créativité en éducation et formation* (pp. 65-78). Bruxelles : De Boeck Supérieur.

Masson, S. (2020). *Activer ses neurones : pour mieux apprendre et enseigner*. Paris : Odile Jacob.

Mayer, C. (2005). *Manuel de métapédagogie*. Menoncourt : Upbraining. En ligne : $https://www.upbraining.net/produit/manuel-de-metapedagogie/$

Sternberg, R. J. (1985). *Beyond I.Q. : A triarchic theory of human intelligence*. New York : Cambridge University Press.

Vianin, P. (2009). *L'aide stratégique aux élèves en difficulté scolaire – Comment donner à l'élève les clés de sa réussite ?* Bruxelles : De Boeck.

Vianin, P. (2010). *Neurosciences cognitives et pédagogie spécialisée : un exemple d'évaluation diagnostique des processus cognitifs*. Berne : Centre Suisse de Pédagogie Spécialisée.

Changer la représentation de l'informatique chez les jeunes : recommandations

Julie Henry Cécile Lombart Bruno Dumas
Faculté d'Informatique, Institut NADI, Université de Namur - 21 rue Grandgagnage, 5000 Namur (Belgique)
julie.henry@unamur.be

RÉSUMÉ

La Belgique francophone ne développe pas les compétences numériques dans l'enseignement primaire et secondaire. En conséquence, de nombreux jeunes ont une représentation incomplète de ce qu'est l'informatique, ce qui entraîne un manque d'intérêt de leur part pour ce domaine et les métiers associés. Pour pallier à ce problème, des ateliers ont été développés, permettant d'initier aux concepts-clé de l'informatique et de ses sous-domaines (programmation, robotique, réseaux informatiques, l'intelligence artificielle et la cybersécurité. Pendant deux ans, des données ont été collectées permettant de mesurer l'impact de ces ateliers sur les représentations de plus de 200 élèves de 12 à 15 ans, dans six écoles différentes. L'analyse des données met en évidence le rôle important joué tant par les thèmes abordés que par le discours de l'enseignant à leur sujet. En particulier, la comparaison des résultats obtenus lorsque les ateliers étaient animés par un expert ou par des enseignants souligne le problème du manque de formation des enseignants.

ABSTRACT

Changing the Way Youth Think About Computer Science : Recommendations.

French-speaking Belgium does not develop digital skills in primary and secondary education. As a result, many young people have an incomplete representation of what computer science is, which leads to a lack of interest on their part for this field and the associated jobs. To address this problem, workshops have been developed to introduce key concepts of computer science and its subfields (programming, robotics, computer networks, artificial intelligence and cybersecurity). For two years, data was collected to measure the impact of these workshops on the representations of more than 200 students aged 12 to 15, in six different schools. The analysis of the data highlights the important role played by both the topics addressed and the teacher's discourse on them. In particular, the comparison of the results obtained when the workshops were led by an expert or by teachers highlights the problem of the lack of teacher training.

MOTS-CLÉS : enseignement de l'informatique, K12, didactique, atelier, représentation erronée.

KEYWORDS: computer science education, K12, didactic, workshops, misconception.

1 Introduction

Selon le rapport évaluant les politiques éducatives numériques dans 43 pays [1] publié par le réseau Eurydice en 2019, la Belgique francophone est un des rares pays à ne pas développer les compétences

1. Digital Education at school in Europe, rapport consulté en ligne le 20 avril 2021 - https ://eacea.ec.europa.eu/national-policies/eurydice/content/digital-education-school-europe

Atelier « Apprendre la Pensée Informatique de la Maternelle à l'Université », dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH), Fribourg, Suisse, 07 juin 2021, pages 13–24,

numériques dans l'enseignement primaire et secondaire (élèves de 5 à 18 ans). D'autres études avaient déjà souligné ce manque (Henry and Joris, 2016, 2015) qui devait être comblé par la réforme du programme d'étude qu'est le Pacte pour un Enseignement d'Excellence [2]. Le Pacte stipule que "dès l'école primaire, une introduction à la logique numérique peut être réalisée par la programmation de machines simple" et évoque également une "maîtrise minimale de la logique des outils - programmer ou être programmé". Un référentiel de Formation Manuelle, Technique, Technologique et Numérique [3] (FMTTN) a été édité, composé de deux volets liés : le volet "Formation manuelle, technique et technologique" et le volet "Numérique". Malgré la volonté de développer les compétences numériques dès le plus jeune âge, le contenu de cet enseignement reste encore à définir, et une lecture de ce référentiel met en évidence l'absence d'un enseignement spécifique de la pensée informatique et, en ce qui concerne l'informatique, une focalisation sur les algorithmes et la programmation. En outre, la mise en oeuvre du Pacte est prévue d'ici 2030.

Depuis des années, attirer plus de jeunes, et notamment des filles, vers les métiers du numérique est un défi majeur. Mais comment les jeunes pourraient-ils penser s'inscrire dans une filière dont ils ne connaissent finalement pas grand chose ? Comment lutter contre l'impact des stéréotypes ? Une solution pourrait être la création d'ateliers pouvant être directement intégrés dans un cours existant lié à la technologie et qui pourraient, par la suite, être intégrés dans le cours FMTTN. C'est l'objectif du projet School-IT [4]. Depuis 2017, ce projet promeut la culture numérique des élèves de la maternelle à la fin du secondaire avec trois objectifs : enseigner l'informatique comme une discipline fondamentale, modifier la conception qu'en ont les jeunes pour favoriser leur intégration dans les filières de recrutement, et viser une éducation citoyenne pour leur permettre d'être autonomes et de réfléchir à leurs pratiques numériques (Henry et al., 2018).

Les ateliers développés dans ce cadre visent à apporter des compétences techniques, en privilégiant la variété dans les domaines couverts (communication, réseau, interaction humain-machine, intelligence artificielle, etc.), dans les processus mis en évidence (analyse, codage, test, etc.), et dans les équipements utilisés (activités débranchées, micro :bit, Makeblock, thymio, Bee-Bot, etc.) Ces ateliers visent à accroître l'intérêt des jeunes pour le numérique en leur faisant prendre conscience de la richesse et de la diversité qui caractérisent cette discipline. Cependant, changer les représentations des jeunes n'est pas simple et des représentations non désirées peuvent apparaître. Prêter attention à ces représentations erronées doit, dès lors, devenir un exercice de didactique à mettre en place dans la phase de conception d'un atelier.

Cet article tente d'apporter des éléments de réponse à la question **Comment changer les représentations des jeunes en matière d'informatique ?** en mesurant l'impact que peuvent avoir des ateliers d'initiation à l'informatique, mais également le discours de l'enseignant mettant en place ces ateliers. Des recommandations sont fournies en guise de conclusion.

2. Le Pacte pour un Enseignement d'Excellence, consulté en ligne le 20 avril 2021 - https ://www.wbe.be/ressources/ressources-pedagogiques/pacte-pour-un-enseignement-dexcellence/

3. Référentiel FMTTN, version provisoire consultée en ligne le 20 avril 2021 - http ://www.ares-ac.be/images/FIE/Referentiels/Referentiel-FMTTN.pdf

4. https ://school-it.info.unamur.be/

2 Travaux antérieurs

2.1 Les représentations existantes

Il n'est pas facile pour les apprenants d'abandonner leurs représentations existantes, parfois erronées, et d'en adopter de nouvelles (Davis, 2001 cité par (Orey, 2010)).

Selon Ben-Ari Ben-Ari (2001), la théorie du constructivisme "affirme que la connaissance est activement construite par l'apprenant (...). Comme la construction se fait de manière récursive sur des connaissances que l'apprenant possède déjà, chaque apprenant construira une version idiosyncrasique de la connaissance. Dans la mesure où ce savoir n'est pas identique au savoir scientifique standard, on dit que l'apprenant a des représentations erronées".

Selon Maier Maier (2004), la façon de résoudre ou de prévenir les représentations fausses est de confronter directement l'apprenant à une expérience qui provoque un déséquilibre. Remettre en question les représentations existantes de l'apprenant l'encourage à détecter les problèmes de compréhension et le motive à construire des compréhensions appropriées (Scott et al., 1997). En général, une stratégie d'enseignement des conflits cognitifs comporte trois étapes : enquêter sur les connaissances antérieures de l'apprenant et sur ses conceptions existantes ; mettre l'apprenant au défi avec des informations contradictoires ; évaluer le changement conceptuel entre les représentations antérieures de l'apprenant et ses représentations actuelles (Limón, 2001). Parce que les apprenants doivent être confrontés à leurs représentations erronées, celles-ci contribuent à l'élaboration de ressources pédagogiques.

2.2 Les représentations de l'ordinateur

Les jeunes grandissent entourés d'ordinateurs et interagissent rapidement avec eux, se construisant ainsi leur propre représentation du fonctionnement et des capacités d'un ordinateur. Rücker et Pinkwart 2016 identifient cinq représentations distinctes. Premièrement, "l'ordinateur est souvent humanisé et considéré comme une sorte d'entité vivante". Deuxièmement, l'ordinateur est une base de données qui sait tout (stockage illimité des données) et retient tout par cœur (récupération rapide des données). Par conséquent, l'ordinateur ne calcule pas. Troisièmement, l'ordinateur est une horloge complexe en raison de la façon dont il est construit. Quatrièmement, l'ordinateur est un dispositif électronique très complexe "qui reste un mystère complet". Enfin et surtout, l'ordinateur est programmable : "son comportement et ses capacités sont déterminés par l'homme et peuvent être modifiés par l'homme". Selon Rücker et inkwart, toutes ces représentations ne semblent pas être persistantes dans le temps : les capacités intrinsèques sont plus tenaces tandis que les représentations liées au matériel sont logiquement plus influencées par les développements technologiques. En outre, un enfant peut détenir simultanément plusieurs représentations qui peuvent être ou non sélectionnées dans un contexte ou une situation donnés.

2.3 Les représentations de l'informatique

Au cours de la dernière décennie, un certain nombre d'études ont été menées sur les représentations erronées des enfants de 12-18 ans sur l'informatique et les perspectives de carrière qui y sont liées. Ces études suggèrent que les jeunes ne savent pas ce qu'est l'informatique. Ces représentations

erronées ont un impact sur leur intérêt pour l'informatique et leur affinité pour ce domaine. En 1998, Greening Greening (1998) en est convaincu : "les étudiants pourraient ne pas s'inscrire à des cours d'informatique (universitaires) à cause de leurs représentations erronées". Cette conviction que les jeunes choisissent de ne pas se spécialiser en informatique parce qu'ils ont une représentation incorrecte ou inadéquate de la discipline (ou même aucune représentation) est étayée par plusieurs études (Carter, 2006; Biggers et al., 2008; Brinda et al., 2009; Ruslanov and Yolevich, 2011).

La solution se trouve dans une initiation à l'informatique proposée dès le plus jeune âge. Greening Greening (1998) définit "une base pour faire évoluer les cours de telle manière qu'ils augmentent la probabilité que les élèves soient enthousiasmés par leur apprentissage". Yardi et Bruckmann 2007 suggèrent de "combler le fossé entre leurs représentations et les possibilités réelles qui sont offertes dans les disciplines informatiques". Ils proposent, par ailleurs, un programme d'études "pour préparer et motiver les adolescents à des carrières dans l'économie mondiale actuelle, en expansion et basée sur l'Internet". Taub et al. 2009 travaillent sur des activités débranchées permettant d'accroître le changement de représentation des jeunes sur l'informatique. Plus récemment, Grover et al. 2014 présente les résultats d'"une intervention dans le cadre du programme scolaire qui vise à montrer l'informatique aux jeunes [de 12 à 14 ans] sous un jour nouveau - dans des contextes du monde réel et en tant que discipline créative et de résolution de problèmes". Deux ans plus tard (Grover et al., 2016), ils recommandent aux enseignants de "déplacer l'objectif initial des cours d'introduction vers les idées plus profondes de l'informatique, du calcul et de la calculabilité".

Selon Hewner et Guzdial 2008, l'enseignement de l'informatique avec des cours d'introduction conçus pour être attrayants, pertinents par rapport aux intérêts des jeunes et axés sur la pratique de la programmation n'a pas d'effet significatif sur l'attitude de ces jeunes vis-à-vis de l'informatique. Pour lui, ce n'est pas un moyen viable d'accroître de manière significative l'intérêt pour l'informatique en tant que discipline. Taub et al. sont légèrement plus optimistes 2009. Leurs résultats montrent que "les activités débranchée en informatique ont effectivement amorcé un processus de changement d'opinion, mais que ce processus était partiel". Cette vision optimiste est partagée par Henry Henry and Dumas (2018). Elle mesure alors l'influence d'un atelier de programmation sur les représentations qu'ont les jeunes de 12 à 14 ans en matière d'informatique. Parmi les leçons à tirer de cette étude, la signification de l'atelier et ses objectifs doivent être bien réfléchis, régis par la vision de l'informatique à transmettre aux jeunes.

2.4 Les représentations des métiers de l'informatique

La technique du "dessine moi..." est souvent utilisée pour déterminer les attitudes et les croyances concernant un sujet quelconque. Selon Martin Martin (2004), les enfants dessinent le plus souvent les informaticiens comme des "hommes blancs à divers degrés de geekitude" (à savoir, portant des lunettes, ayant de l'acné et des cheveux en désordre, les yeux collés à un écran d'ordinateur, présentant un excès de poids, étant accro à la malbouffe, portant un t-shirt avec un code informatique, etc.). Ces résultats sont similaires à ceux mesurés par Hansen et al. 2016; 2017 auprès de jeunes de 9 à 10 ans, avant et après un programme de sciences informatiques. Hansen souligne qu'après le programme, les filles ont été plus nombreuses qu'auparavant à dessiner des femmes informaticiennes, et les actions présentées étaient plus spécifiques à l'informatique (et non à la technologie en général : dactylographie, impression, etc.)

Comprendre les représentations des enfants semble pouvoir aider à identifier le moment où les stéréotypes apparaissent. La prise en compte des représentations dans la constructions de ressources

pédagogiques, mais également dans le discours de l'enseignant, pourrait, dès lors, aider à contrer ces stéréotypes. Cet article décrit la mesure de cet impact auprès des jeunes et les recommandations qui en découlent.

3 Méthodologie

Afin de mesurer l'effet d'ateliers de courte durée sur les représentations des jeunes en matière d'informatique, une étude comparative a été menée en situation réelle, sur une période de deux ans, entre septembre 2017 et juin 2019.

3.1 Échantillon et ateliers

Cinq établissements scolaires ont accueilli les ateliers du projet School-IT. Pour chaque école, l'enseignant responsable du cours de technologie a pris part à l'étude. Au total, onze classes ont participé, soit 232 élèves âgés de 12 à 15 ans.

En 2017-2018, les ateliers ont été donnés par une membre de l'équipe School-It, informaticienne de formation. En 2018-2019, les ateliers ont été donnés par les enseignants responsables des différentes classes, tous scientifiques de formation initiale. Deux échantillons sont donc considérés : 134 élèves (échantillon 1) ont reçu un enseignement par un expert en informatique et 98 élèves (échantillon 2) ont suivi les ateliers avec leur enseignant. Durant la première année de l'étude, les enseignants ont assisté aux ateliers donnés par l'expert.

Les élèves ont eu entre quatre et huit périodes de cours consacrées aux ateliers. Tous les élèves n'ont pas eu les mêmes ateliers, en fonction du temps disponible dans chaque école et des affinités de leur enseignant pour la matière abordée. Deux ateliers, organisés sur trois périodes, ont toutefois été imposés : un atelier introduisant la numérisation de l'information et un atelier expliquant le fonctionnement d'un ordinateur par la découverte des entrées et sorties d'un appareil tangible (ordinateur portable, micro :bit, thymio, tablette ou smartphone). Il restait alors aux enseignants à choisir parmi les ateliers disponibles sur le site School-IT : concepts de base en programmation et intelligence artificielle (IA), entre autre. L'ensemble des enseignants a opté pour les ateliers abordant la programmation. L'expert a donné, durant la première année de l'étude, les ateliers choisis par les enseignants.

3.2 Collecte et analyse des données

Au cours de l'étude, un questionnaire a été administré aux élèves avant et après l'ensemble des ateliers donnés. Une période pouvant aller jusqu'à quatre mois s'est écoulée entre les deux tests. Trois questions ouvertes étaient posées : *Qu'est-ce qu'un ordinateur ? et Qu'est-ce que l'informatique ? Quels sont, selon toi, les aspects positifs et négatifs des métiers de l'informatique ?*

Les réponses à ces questions ont été codées indépendamment par deux chercheurs de façon à les regrouper dans des catégories homogènes. Pour conserver au mieux le vocabulaire utilisé par les participants, certaines catégories ont été doublées bien qu'elles expriment un même thème. Par exemple, pour la question *Qu'est-ce qu'un ordinateur ?*, les catégories "Machine et "Boîte" sont

restées distinctes, de même que les catégories “Écran/Souris/Clavier” et "Entrées/Sorties”. Pour chaque question, les thèmes qui apparaissent de façon récurrente ont été dénombrés. Une comparaison a ensuite été faite entre les résultats obtenus durant la première année d’étude (ateliers donnés par l’expert) et les résultats de la deuxième année.

4 Résultats

Les résultats de l’étude sont structurés en tois sections, en tenant compte des questions des tests administrés aux élèves : les représentations de l’ordinateur (Table 1), de l’informatique (Table 2) et des métiers de l’informatique (Tables 3 et 4). Seules les observations les plus marquantes sont reprises dans le texte.

4.1 Représentations de l’ordinateur

À la question de savoir ce qu’est un ordinateur (Table 1), avant les ateliers, les élèves de l’échantillon 1 parlent surtout d’une machine, avec un écran, un clavier et une souris, possédant une mémoire et capable d’agir seule. Ils parlent également d’un usage très courant : la recherche d’informations. Après les ateliers, ces élèves parlent davantage d’entrée/sortie, font toujours référence à la mémoire, mais l’associent au processeur.

Lorsqu’un enseignant anime les ateliers, les observations sont différentes. Les élèves de l’échantillon 2 décrivent l’ordinateur d’un point de vue matériel (écran, souris, clavier) avant et après les ateliers. Si peu d’élèves ont mentionné l’aspect composant (processeur, mémoire) avant les ateliers, ce nombre diminue encore après les ateliers. Par ailleurs, le nombre d’élèves décrivant les usages de l’ordinateur (communication, recherche) augmente légèrement. Enfin, près de 12,8 % des élèves de l’échantillon 2 mentionnent l’intelligence artificielle après les ateliers.

	Échantillon 1		Échantillon 2	
	Pré-test(%)	Post-test(%)	Pré-test(%)	Post-test(%)
Machine	27.9	18	21.0	27.7
“Boîte”	12	10.5	2.0	13.8
Écran/Souris/Clavier	25.3	6	16	19.1
Entrées/Sorties	0.0	36.1	3.0	1.1
Processeur	2.7	13.5	3.0	1.1
Mémoire	20.0	24.1	3.0	2.1
Code/Programme	4.0	2.3	3.0	6.4
Internet	9.3	1.5	6.0	6.4
Communication	6.7	0.0	0.0	1.1
Recherche	27.9	5.3	3.0	8.5
Logiciels	6.7	6.0	4.0	6.4
IA	5.3	6.0	3.0	12.8
Suivre les ordres	6.7	3.0	7.0	8.5
Faire les choses par soi-même	23.9	3.0	4.0	8.5

TABLE 1 – Qu’est-ce qu’un ordinateur ?

4.2 Représentations de l'informatique

Avant les ateliers, 50,0 % des élèves de l'échantillon 1 ont mentionné le mot "ordinateur" au moins une fois (Table 2). Après les ateliers, ce chiffre tombe à 41,8 %. Ces résultats, pour l'activité présentée par l'expert, montrent que les élèves pensent moins spontanément à un ordinateur lorsqu'on leur demande ce qu'est l'informatique (9,0 % de moins). Si l'on considère les élèves de l'échantillon 2, le nombre de fois où le mot "ordinateur" est mentionné est presque constant, passant de 39,4 % à 38,8 %.

Alors que les élèves parlent de matériel (souris, clavier) avant les ateliers, ce n'est généralement plus le cas après. Cependant, après l'intervention de l'expert, les élèves de l'échantillon 1 sont plus nombreux à parler d'entrée/sortie (+ 7,5 %).

Les élèves de l'échantillon 2 parlent davantage des robots après les ateliers (+ 11,8 % contre + 3,8 % pour les élèves de l'échantillon 1). Dans les deux échantillons, les élèves mentionnent davantage le mot "code" après les ateliers (+ 16,2 % pour l'échantillon 2 et + 5,9 % pour l'échantillon 1).

L'association entre l'informatique et Internet est plus présente en année 1, mais elle diminue avec l'impact des ateliers (- 5,3 %). Une petite augmentation est observée dans l'échantillon 2 après l'intervention des enseignants (+ 2,2 %).

Pour les logiciels, les tendances sont inversées. Alors qu'en 2017-2018, ils étaient mentionnés par près de 9 % des élèves avant les ateliers, c'était le cas pour seulement 1 % des élèves de l'échantillon 2. En revanche, après les ateliers, seuls 1,5 % des élèves de l'échantillon 1 parlent de logiciels, contre 9,5 % des élèves de l'échantillon 2. Le même schéma est observé pour les jeux vidéo.

Enfin, l'intelligence artificielle est évoquée par davantage d'élèves de l'échantillon 2 après les ateliers (+ 5,4 %).

	Échantillon 1		Échantillon 2	
	Pretest(%)	Post-Test(%)	Pretest(%)	Post-Test(%)
Ordinateur	50.0	41.8	39.4	38.9
Smartphone	14.2	4.5	6.1	3.2
Télévision	2.2	0.7	1.0	3.2
Machine électronique	10.4	8.2	13.1	10.5
Robot	0.7	4.5	7.1	18.9
Écran/Souris/Clavier	6.7	0.0	8.1	2.1
Entrées/Sorties	0.0	7.5	0.0	0.0
Code/programme	7.5	13.4	10.1	26.3
Internet	7.5	2.2	2.0	4.2
Communication	1.5	0.0	3.0	0.0
Logiciels	8.2	1.5	1.0	9.5
Jeux vidéo	5.2	0.7	0.0	4.2
IA	2.2	1.5	3.0	0.0
Réseaux	2.2	1.5	1.0	1.1
Technologie	21.6	9.7	13.1	8.4
Données	10.4	0.0	5.1	3.2
Travail	11.2	6.0	0.0	0.0

TABLE 2 – Qu'est-ce que l'informatique ?

4.3 Représentations des métiers de l'informatique

On a demandé aux élèves ce qu'ils trouvaient de positif et de négatif dans les métiers de l'informatique (Tables 3 et 4).

Concernant les aspects positifs des métiers de l'informatique, une augmentation du nombre d'élèves qui les trouvent utiles pour la société est constatée. Cette augmentation est la même pour les ateliers présentés par l'expert et par les enseignants. Les élèves de l'échantillon 1 pensent que l'informatique aide les gens.

Les élèves de l'échantillon 1 s'éloignent de l'idée qu'un métier de l'informatique nécessite l'utilisation d'un ordinateur, alors que cette idée est renforcée chez les élèves de l'échantillon 2.

L'aspect ludique diminue pour tous les élèves, tandis que la créativité n'augmente que pour les élèves de l'échantillon 1.

En ce qui concerne les aspects négatifs des métiers de l'informatique, les réponses des élèves de l'échantillon 1 diminuent sur presque tous les points, à l'exception de l'attrait. Lorsque les ateliers sont présentés par les enseignants, les aspects négatifs de la santé et le côté statique sont mentionnés par davantage d'élèves après les ateliers.

	Échantillon 1		Échantillon 2	
	Pré-test(%)	Post-test(%)	Pré-test(%)	Post-test(%)
Utile	15.8	24.1	22.0	35.0
Utilisation de l'ordinateur	10.5	6.8	4.0	6.0
Apprentissage continu	11.3	8.3	13.0	11.0
Emploi d'avenir	8.2	8.9	4.0	6.0
Fun	6.0	3.0	9.0	4.0
Créativité	4.5	5.3	2.0	1.0
Permet d'aider les gens	3.8	6	1.0	1.0
Métier statique	0.8	3.0	5.0	3.0

TABLE 3 – Aspects positifs des métiers liés à l'informatique

	Échantillon 1		Échantillon 2	
	Pré-test(%)	Post-test(%)	Pré-test(%)	Post-test(%)
Métier statique	37.2	21.1	16.0	21.0
Mauvais pour la vue	31.9	16.5	10.0	25.0
Difficile	16.0	6.8	11.0	8.0
Mauvais pour la santé	14.6	7.5	6.0	6.0
Solitude	10,6	7,5	2,0	2,0
Pas attractif	5.3	6.8	15.0	5.0
Effrayant	1.3	0.0	3.0	4.0

TABLE 4 – Aspects négatifs des métiers liés à l'informatique

5 Discussion

Outre l'objectif éducatif qui est de faire découvrir un domaine, des concepts, des techniques et des applications, l'enseignant doit se fixer l'objectif de faire évoluer les représentations. C'est sur ce deuxième objectif que porte la discussion des résultats.

Un premier constat à faire est la confusion toujours présente chez les jeunes entre l'informatique et les métiers de l'informatique et l'ordinateur sous sa forme la plus classique (avec écran, clavier et souris). Bien que distinctes, les questions posées souffrent de cette confusion. Cependant, il reste possible de tirer des leçons des réponses collectées.

En général, l'expert a pris davantage soin de s'éloigner, dans son discours, de l'ordinateur sous sa forme classique pour en faire comprendre le fonctionnement interne, incluant le fonctionnement des composants (mémoire, processeur, composant). Cela n'est pas surprenant puisque cette étude fait partie d'un projet visant à changer les représentations des jeunes en matière d'informatique. L'expert a été attentif de transmettre "des idées plus profondes de l'informatique, du calcul et de la calculabilité", comme le soutiennent Grover et al. 2014. Compte tenu des ateliers imposés (l'initiation à la numérisation de l'information et au fonctionnement de l'ordinateur), les enseignants devaient également pouvoir transmettre ces idées. La différence mesurée entre les deux échantillons pourrait s'expliquer par le manque de formation des enseignants, par leur manque de confiance lorsqu'il s'agit de donner des cours d'informatique, ou par le fait que les enseignants ont probablement moins insisté sur ces idées jugées cruciales par l'expert.

Les résultats, au regard des ateliers organisés, laissent penser que, comme les jeunes, les enseignants semblent s'inspirer des médias pour développer leurs connaissances dans le domaine de l'informatique. Cela expliquerait la présence, dans les réponses de leurs élèves, de thèmes tels que robot et IA, notions non abordées par les enseignants mais qui font régulièrement la une des médias. L'évocation des logiciels et des jeux vidéo, plus souvent cités par les élèves de l'échantillon 2, montre également que les enseignants s'appuient sur leurs connaissances et leurs utilisations fréquentes.

L'augmentation des références à la programmation est logique considérant le contenu des ateliers. Cependant, elle est beaucoup plus prononcée dans l'échantillon 2. Une fois de plus, cela peut s'expliquer par l'attention portée par l'expert à montrer la variété qui existe dans les métiers de l'informatique.

Les changements concernant les métiers mettent une fois de plus en évidence la précaution prise par l'expert pour valoriser son travail. L'intervention des enseignants semble avoir un impact relativement faible, voire plutôt négatif.

En conclusion, il semble difficile de changer les représentations des jeunes si certaines conditions ne sont pas remplies. Plusieurs recommandations peuvent être faites, nourries par les résultats obtenus mais également par les discussions informelles menées avec les enseignants durant ces deux années d'étude.

Certaines recommandations portent sur les ateliers.

- **Être clair sur le message à transmettre : ne pas enseigner l'informatique pour n'enseigner que l'informatique.** Si l'expert est conscient de ce qu'il doit apporter, le combat est différent pour les enseignants qui se battent dans un domaine qu'ils maîtrisent mal.
- **Découvrir les différents sous-domaines de l'informatique.** Il est important de montrer les différents sous-domaines de l'informatique (IA, cybersécurité, interface humain-machine, etc.) pour éviter de renforcer les stéréotypes et de changer la façon dont les jeunes perçoivent l'informatique.
- **Prendre suffisamment de temps.** Il est clair qu'il est difficile de changer des représentations profondément ancrées, surtout si le temps qui leur est consacré est court. En cela, une introduction uniquement ponctuelle à l'informatique restera insuffisante.
- **Se détacher de l'ordinateur sous sa forme la plus classique (écran, souris et clavier).**

Comme les étudiants parlent principalement de l'ordinateur lorsqu'ils se réfèrent à l'informatique, l'idéal est de s'en éloigner le plus possible pour développer leur vision et leur montrer d'autres aspects et dispositifs (micro :bit, makeblock, thymio, etc.). Une activité débranchée peut être tout aussi efficace pour comprendre la matière et changer les mentalités.

D'autres concernent directement les enseignants.

- **Renforcer la formation.** S'il est nécessaire de jouer sur les thèmes abordés (IA, cybersécurité, interface humain-machine, etc.), cela oblige les enseignants à maîtriser davantage que les compétences de plus en plus demandées dans l'enseignement que sont l'algorithmique et la programmation. Maîtriser la matière, c'est maîtriser les concepts de base, maîtriser le vocabulaire associé, avoir confiance en soi, pouvoir se détacher de la matière elle-même, mais aussi des médias. Prendre du recul est essentiel pour encourager les étudiants à adopter une attitude critique à l'égard de l'informatique.
- **Faire attention à ce qui est dit.** Les enseignants doivent être informés des stéréotypes existants afin de les déconstruire et non de les alimenter. Cela leur permettrait également de porter un regard critique sur les médias et les métaphores couramment utilisées (Boraita et al., 2020).
- **Devenir autant que possible un modèle ou promouvoir des modèles inspirants.** L'expert devient un modèle pour les étudiants qui ont alors devant eux une personne qui devrait travailler quotidiennement devant un ordinateur (selon leurs représentations) mais qui dit faire beaucoup plus que cela. S'il n'est pas possible pour les enseignants d'être eux-mêmes des modèles, ils peuvent faire découvrir à leurs élèves des modèles inspirants à travers des histoires personnelles. Il faut cependant veiller à montrer des modèles accessibles et non de rares cas de réussite (Spieler et al., 2019).

6 Conclusion

Dans un monde de plus en plus numérique, les emplois informatiques manquent de ressources humaines. Les représentations erronnées que les jeunes ont de ce qu'est l'informatique et de ce qu'elle implique en termes d'emplois en sont une cause possible.

De courts ateliers ont été mis au point pour initier les jeunes à l'informatique. Pendant deux ans, ces ateliers ont été testés dans cinq écoles avec 232 élèves âgés de 12 à 15 ans. La première année, les ateliers ont été donnés par un expert, et la deuxième année, par les enseignants habituels. Pour recueillir des données, une enquête a été menée auprès des élèves avant et après chaque séquence d'ateliers. L'objectif de l'enquête était d'identifier les changements dans les représentations des jeunes en matière d'ordinateur, d'informatique et des métiers de l'informatique.

Plusieurs éléments ont pu être mis en évidence dans cette étude. Le plus évident est qu'il existe une grande différence d'impact entre un atelier donné par un informaticien et un atelier donné par un enseignant. L'informaticien ayant une connaissance approfondie des stéréotypes sait comment orienter les ateliers. Il lui est aussi logiquement plus facile de transmettre les concepts-clé. Il peut donc se concentrer plus particulièrement sur sa mission secondaire qui est de faire évoluer les représentation et de déconstruire les stéréotypes que les jeunes ont sur l'informatique. L'idée est de partir des représentations que les jeunes ont et de proposer un atelier leur demandant de réviser leurs connaissances en les confrontant à la réalité des personnes travaillant dans ce domaine. Cette action n'est pas possible sans une formation, malheureusement souvent absente ou incomplète chez les

enseignants.

Une autre conclusion que nous pouvons tirer de cette étude est que le choix des ateliers proposés et du matériel utilisé est également très important. Il est essentiel d'enrichir la vision des jeunes en évitant de proposer uniquement des ateliers qui présentent des sous-domaines déjà connus de l'informatique (programmation, entre autres). Il faut cibler des sous-domaines auxquels les gens pensent moins lorsqu'ils pensent à l'informatique (souvent à cause des stéréotypes et de la vision diffusée par les médias), tels que les interfaces humain-machine, la modélisation, les bases de données, etc.

Références

Ben-Ari, M. (2001). Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching*, 20(1) :45–73.

Biggers, M., Brauer, A., and Yilmaz, T. (2008). Student perceptions of computer science : a retention study comparing graduating seniors with cs leavers. *Acm sigcse bulletin*, 40(1) :402–406.

Boraita, F., Henry, J., and Collard, A.-S. (2020). Developing a critical robot literacy for young people from conceptual metaphors analysis. In *2020 IEEE Frontiers in Education Conference (FIE)*, pages 1–7. IEEE.

Brinda, T., Puhlmann, H., and Schulte, C. (2009). Bridging ict and cs : educational standards for computer science in lower secondary education. *Acm Sigcse Bulletin*, 41(3) :288–292.

Carter, L. (2006). Why students with an apparent aptitude for computer science don't choose to major in computer science. *ACM SIGCSE Bulletin*, 38(1) :27–31.

Greening, T. (1998). Computer science : through the eyes of potential students. In *Proceedings of the 3rd Australasian conference on Computer science education*, pages 145–154.

Grover, S., Pea, R., and Cooper, S. (2014). Remedying misperceptions of computer science among middle school students. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 343–348.

Grover, S., Rutstein, D., and Snow, E. (2016). " what is a computer" what do secondary school students think ? In *Proceedings of the 47th acm technical symposium on computing science education*, pages 564–569.

Hansen, A. K., Dwyer, H., Harlow, D. B., and Franklin, D. (2016). What is a computer scientist ? developing the draw-a-computer-scientist-test for elementary school students. *AERA Online Paper Repository*.

Hansen, A. K., Dwyer, H. A., Iveland, A., Talesfore, M., Wright, L., Harlow, D. B., and Franklin, D. (2017). Assessing children's understanding of the work of computer scientists : The draw-a-computer-scientist test. In *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education*, pages 279–284.

Henry, J. and Dumas, B. (2018). Perceptions of computer science among children after a hands-on activity : a pilot study. In *2018 IEEE Global Engineering Education Conference (EDUCON)*, pages 1811–1817. IEEE.

Henry, J., Hernalesteen, A., Dumas, B., and Collard, A.-S. (2018). Que signifie éduquer au numérique ? pour une approche interdisciplinaire. *De 0 à 1 ou l'heure de l'informatique à l'école*, page 61.

Henry, J. and Joris, N. (2015). Le bagage tic des étudiants en belgique francophone. état des lieux. *Informatique en éducation : perspectives curriculaires et didactiques*, pages 61–81.

Henry, J. and Joris, N. (2016). Informatics at secondary schools in the french-speaking region of belgium : myth or reality. *ISSEP 2016*, 13.

Hewner, M. and Guzdial, M. (2008). Attitudes about computing in postsecondary graduates. In *Proceedings of the fourth international workshop on computing education research*, pages 71–78.

Limón, M. (2001). On the cognitive conflict as an instructional strategy for conceptual change : A critical appraisal. *Learning and instruction*, 11(4-5) :357–380.

Maier, S. (2004). Misconception research and piagetian models of intelligence. In *Proc. 2004 Oklahoma Higher Education Teaching and Learning Conf.*

Martin, C. D. (2004). Draw a computer scientist. *ACM SIGCSE Bulletin*, 36(4) :11–12.

Orey, M. (2010). *Emerging perspectives on learning, teaching and technology.* CreateSpace North Charleston.

Rücker, M. T. and Pinkwart, N. (2016). Review and discussion of children's conceptions of computers. *Journal of Science Education and Technology*, 25(2) :274–283.

Ruslanov, A. D. and Yolevich, A. P. (2011). College student notions of computer science. In *AIP Conference Proceedings*, volume 1373, pages 137–148. American Institute of Physics.

Scott, P., Asoko, H., and Driver, R. (1997). Teaching for conceptual change : A review of strategies. *Connecting Research in Physics Education with Teacher Education*.

Spieler, B., Oates-Indruchova, L., and Slany, W. (2019). Female teenagers in computer science education : understanding stereotypes, negative impacts, and positive motivation. *arXiv preprint arXiv :1903.01190*.

Taub, R., Ben-Ari, M., and Armoni, M. (2009). The effect of cs unplugged on middle-school students' views of cs. *ACM SIGCSE Bulletin*, 41(3) :99–103.

Yardi, S. and Bruckman, A. (2007). What is computing ? bridging the gap between teenagers' perceptions and graduate students' experiences. In *Proceedings of the third international workshop on Computing education research*, pages 39–50.

Construction d'un programme combinant exécution partielle et manipulation directe

Michel Adam[1] Moncef Daoud[1]
Patrice Frison[2]
(1) Université Bretagne Sud, 56100 Lorient, France
(2) IRISA, Université Bretagne Sud, 56000 Vannes, France
michel.adam@univ-ubs.fr, moncef.daoud@univ-ubs.fr,
patrice.frison@univ-ubs.fr

RÉSUMÉ

La programmation est un art difficile à maîtriser. Dans le cas de la programmation impérative, la boucle est l'un des concepts majeurs à connaître. Pour les programmeurs novices, c'est aussi l'un des concepts les plus difficiles à comprendre et à mettre en oeuvre. Cette difficulté est d'autant plus grande quand un algorithme nécessite la mise en place d'une boucle à l'intérieur d'une autre boucle. On parle alors de boucles imbriquées. Dans cet article, nous proposons une méthode associée à un outil facilitant la conception et la programmation d'algorithmes avec boucles imbriquées. Cette méthode originale combine alternativement l'exécution partielle de la boucle externe du programme avec l'exécution manuelle par le programmeur de la boucle interne.

ABSTRACT

Construction of a program combining partial execution and direct manipulation

Programming is a difficult art to master. In the case of imperative programming, loop is one of the major concepts to know. For novice programmers, it is also one of the most difficult concepts to understand and implement. This difficulty is all the greater when an algorithm requires the implementation of a loop within another loop. We then speak of nested loops. In this article, we propose a method associated with a tool facilitating the design and programming of algorithms with nested loops. This original method alternately combines the partial execution of the outer loop of the program with the manual execution by the programmer of the inner loop.

MOTS-CLÉS : concept de boucles, boucles imbriquées, programmation par manipulation directe des données, visualisation d'algorithmes, environnements d'apprentissage interactifs.

KEYWORDS: loops concept, nested loops, programming by data direct manipulation, visualization of algorithms, interactive learning environnements.

1 Introduction

L'apprentissage de la programmation fait l'objet de nombreux travaux de recherche de par le monde. L'informatique étant désormais enseignée dès le primaire (Baron and Drot-Delange, 2016) dans la plupart des pays, de nouvelles approches ont été proposées pour adapter cet enseignement à un large public. La notion de pensée informatique (*computational thinking*) a été introduite par Wing (Wing, 2011). L'idée étant de montrer que de nombreux problèmes de la vie courante peuvent être traités

Atelier « Apprendre la Pensée Informatique de la Maternelle à l'Université », dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH), Fribourg, Suisse, 07 juin 2021, pages 25–33,

de façon systématique et ainsi être programmés sur des ordinateurs. Dans ce contexte, le concept d'algorithme est défini ainsi que la notion fondamentale de boucle. Cependant, le concept de boucle reste difficile à mettre en pratique par les novices lorsqu'ils doivent programmer eux-mêmes un exercice. Dans cet article, nous nous intéressons plus particulièrement à l'apprentissage des boucles.

Le papier est organisé de la façon suivante. Dans la deuxième section, nous présentons le contexte de l'étude. La troisième section aborde la programmation par manipulation directe des données, illustrée par le système AlgoTouch. Puis, dans une quatrième section, nous montrerons comment construire des boucles imbriquées par une approche originale alternant exécution du programme et manipulation directe des données. Le principe est le suivant : la boucle externe s'exécute mais s'interrompt pour laisser à l'utilisateur le contrôle pour réaliser l'objectif de la boucle interne par manipulation directe des données. Ensuite, le programme de la boucle externe reprend et ainsi de suite. Nous conclurons par une analyse des avantages de l'approche proposée et ses perspectives.

2 Contexte de l'étude

Différentes approches ont été proposées pour aider à l'apprentissage de la programmation. L'approche débranchée (Drot-Delange, 2013) propose de décrire les actions à effectuer pour traiter un problème. La description doit être suffisamment précise pour qu'un opérateur humain puisse exécuter les actions décrites et ainsi atteindre l'objectif. L'informatique tangible (Papavlasopoulou et al., 2017) permet aux débutants d'interagir avec des objets physiques et transformer la logique du monde physique en logique du programme. La manipulation directe des données (Adam et al., 2019) permet de construire des programmes en réalisant directement sur un écran les opérations que doit effectuer une machine. Le logiciel traduit les actions de l'utilisateur en langage informatique. Ce programme peut ensuite être exécuté automatiquement.

Pour faciliter la programmation, des environnements éducatifs ont été proposés (Maloney et al., 2010) (Daly, 2013). Le programme est construit par assemblage de blocs qui s'emboitent comme des éléments de puzzle. Ceci simplifie l'apprentissage d'un langage de programmation. L'environnement d'exécution est très visuel, mettant en scène des avatars se déplaçant dans un univers virtuel. Pour les apprenants plus expérimentés, l'apprentissage s'effectue avec des outils plus standard du programmeur au sein d'un EDI (Environnement de Développement Intégré). L'outil au centre de l'EDI étant l'éditeur de texte permettant la saisie assistée du programme dans un langage particulier.

Pour aider à la compréhension et à la réalisation des boucles, diverses méthodes ont été proposées (Cetin et al., 2020). La lecture de code écrit par l'enseignant permet d'illustrer concrètement à travers d'exemples bien choisis comment programmer les boucles. Cet exercice peut être complété en demandant aux étudiants d'exécuter manuellement le programme. Il permet notamment à l'enseignant de vérifier que l'étudiant déroule correctement chaque pas de la boucle. Un autre exercice consiste à demander à un apprenant de compléter un programme constitué d'une boucle. Il manque, par exemple, la condition de sortie de boucle ou l'instruction d'évolution de la boucle (incrément d'un indice par exemple).

Des outils ont été mis au point pour suivre le comportement d'un programme. Le débogueur permet d'exécuter le programme instruction par instruction et de consulter le contenu des variables. Les logiciels de visualisation (SV pour Software Visualisation) ont été développés pour aider les apprenants à surmonter leurs difficultés cognitives dans l'apprentissage des concepts de programmation

(Sorva et al., 2013). Cependant ces systèmes de visualisation n'ont d'intérêt que si l'apprenant a un moyen d'intervenir dans le processus de construction (Hundhausen et al., 2002) : la visualisation du fonctionnement d'un algorithme ne suffit pas.

Si la construction de boucle est difficile pour les étudiants (Kordaki et al., 2008) (Teague and Lister, 2014) (Izu et al., 2016) (Mladenović et al., 2018), c'est d'autant plus vrai pour le cas des boucles imbriquées. Une approche possible consiste à remplacer une boucle interne par un appel de méthode (Caspersen and Kolling, 2009) : "*If you have a nested loop, move the inner loop into a separate method*". Le but de cette approche est d'isoler les parties de code des deux boucles, en s'intéressant à la mise en oeuvre de chacune d'elles séparément. Ce principe hérite d'un principe plus large que les auteurs ont nommé "The Mañana Principle" : "*When – during implementation of a method – you wish you had a certain support method, write your code as if you had it. Implement it later*". En fait, ce principe est souvent utilisé par les développeurs. Il consiste à implémenter une méthode non écrite avec un corps qui rend toujours le même résultat, compatible avec les spécifications. Dans cet article nous proposons également de remplacer une boucle interne par un appel de méthode. Mais contrairement à l'approche décrite précédemment dans laquelle la méthode nouvelle rend toujours le même résultat, nous proposons que l'utilisateur prenne le contrôle du programme pour réaliser lui-même les opérations de la méthode.

Ce dispositif a été mis en oeuvre au sein d'un logiciel appelé AlgoTouch que nous avons développé. Il fait partie des logiciels de manipulation directe et de visualisation. Il permet de manipuler visuellement et manuellement les données d'un programme (variables, tableaux, indices), de construire un programme par enregistrement de suites d'actions. Mais il permet aussi de visualiser l'exécution d'un programme déjà écrit. Un outil particulièrement utile d'une part à un enseignant pour montrer le principe de fonctionnement d'un algorithme, et d'autre part à un élève pour visualiser le déroulement de l'exécution de son programme (Adam et al., 2018). Une première expérience a montré que des étudiants avaient de meilleurs résultats sur la construction de boucles avec AlgoTouch, qu'en écrivant un programme en Python (Adam et al., 2019).

3 Programmation par manipulation des données : exemple du système AlgoTouch

Les systèmes de programation par manipulation directe des données appartiennent à la catégorie des systèmes de programmation par démonstration (Programming By Demonstration : PBD) introduits par Cypher et al. (Cypher and Halbert, 1993). L'idée principale est que le système observe ce que fait l'utilisateur et qu'un programme soit produit de façon automatique ou contrôlée. De nombreuses variantes ont été proposées dans la littérature. Nous nous intéressons ici aux systèmes PBD concernant la manipulation directe des données du programme. Ces systèmes, dont le précurseur est Pygmalion (Cypher and Halbert, 1993), utilisent la métaphore du tableau noir, c'est-à-dire la manipulation directe sur l'écran des objets de la programmation : variables, tableaux, indices (figure 1). Une mise en oeuvre de cette méthode est réalisée avec l'outil AlgoTouch (Frison, 2015).

Les opérations de base sur les variables sont concrètes. Ainsi, l'utilisateur d'AlgoTouch agit directement sur les données et visualise immédiatement le résultat obtenu. A titre d'exemple, l'affectation est effectuée en glissant le contenu d'une variable et en le déposant sur une autre variable. Toutes les actions réalisées directement sur les données correspondent à des instructions d'un langage de

programmation spécifique à AlgoTouch et inspiré du langage Eiffel (Meyer, 2009). Par exemple, faire glisser le contenu d'une variable *y* dans une variable *x* provoquera la génération de l'instruction `x = y`. Par des gestes simples, l'utilisateur peut aussi augmenter les valeurs d'indices, lancer des comparaisons, effectuer les opérations de base (addition, soustraction, multiplication, division). Une séquence d'actions peut être enregistrée et ensuite rejouée. Une partie du programme est créée et produite au fur et à mesure que le programmeur « *joue* » avec les données de l'algorithme.

Le code genéré peut être nommé dans une structure de base d'AlgoTouch appelée macro-instruction ou simplement macro. Une macro est une manière d'abstraire une suite d'instructions. Aussi, une macro peut contenir un appel à une autre macro. Ainsi, il est possible de décomposer un programme en plusieurs macros de manière à resoudre des problèmes complexes. Une macro est différente d'une fonction ou d'une procédure. Dans un souci de simplification, les notions de paramètres et de variables locales n'existent pas dans AlgoTouch. En effet, toutes les variables, les données du programme, sont globales et donc utilisables dans toutes les macros.

AlgoTouch autorise uniquement deux types de macros : la macro simple, une séquence d'instructions et la macro boucle. Cette dernière est composée de 4 parties distinctes :

- la partie *From* permet de définir clairement les instructions nécessaires avant la boucle ;
- la partie *Until* définit les conditions de sortie de la boucle. Ces conditions sont placées en séquence sans opérateur. La première condition est évaluée, si elle est vraie, la boucle s'arrête. Sinon, la seconde condition est évaluée et ainsi de suite ;
- la partie *Loop* constitue le cœur de la boucle. Elle est exécutée quand toutes les conditions de sortie sont fausses ;
- enfin la partie *Terminate* permet de définir les instructions éventuelles à effectuer après la boucle.

La figure 1 illustre la construction d'un programme de recherche dichotomique.

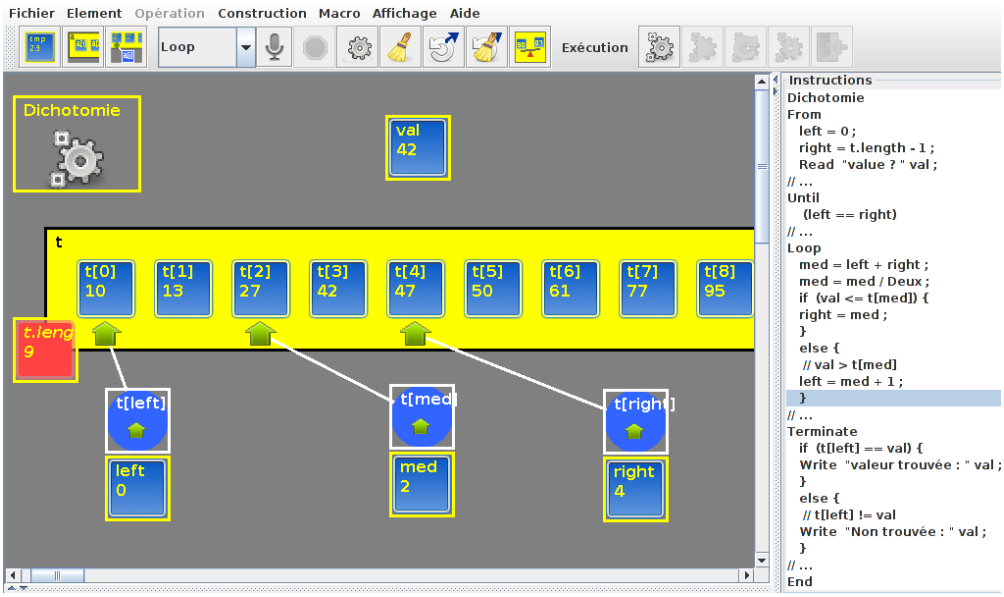


FIGURE 1 – Écran AlgoTouch, à gauche les données manipulées, à droite le code

4 Nouvelle approche de la construction de boucles imbriquées

La programmation de boucles imbriquées dans AlgoTouch est uniquement possible par appel de macro selon le "Mañana Principle" (Caspersen and Kolling, 2009). Une première macro permet de programmer une première boucle. Elle appelle une autre macro boucle dans son corps. L'appel d'une macro dans le corps de la boucle est équivalent à de l'imbrication. L'avantage est que le programmeur ne visualise à un instant donné qu'une seule boucle, qu'une seule macro. Il doit réaliser chacune des macros l'une après l'autre.

Dans cette partie, nous allons montrer qu'avec AlgoTouch, en utilisant la manipulation de données, il est possible de commencer par la boucle la plus externe. Le programmeur génère le code de la boucle externe. La macro représentant la boucle interne est appelée mais son code est vide. Le programmeur peut exécuter la macro externe. Quand la macro interne vide est appelée, l'exécution s'arrête et le programmeur reprend la main. Il peut alors manipuler directement les données de manière à produire le résultat attendu par l'exécution de la macro. Quand le résultat est produit, le programme peut reprendre son exécution. La boucle externe est ainsi testée.

Cette méthode est détaillée avec AlgoTouch à travers l'exemple du tri par insertion. Dans un premier temps, nous présentons le principe du tri, puis la programmation de la boucle externe et enfin celle de la boucle interne.

4.1 Exemple du tri par insertion

Le principe du tri par insertion étudié dans cette partie est de placer les valeurs au fur et à mesure de leur saisie à la bonne position dans le tableau. Ce tri est visuel : les premières valeurs du tableau sont toujours triées et il est facile de déterminer visuellement la place à attribuer à la dernière valeur saisie. La mise en oeuvre avec Algotouch est illustrée sur la figure 2.

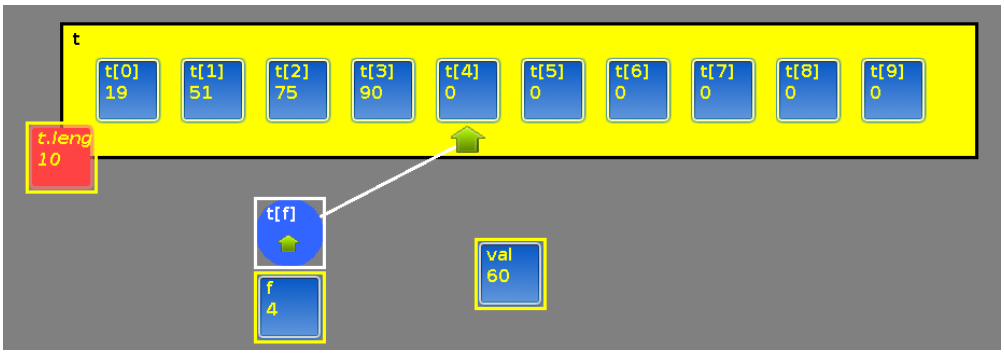


FIGURE 2 – Données du tri par insertion : *t* un tableau d'entiers, *val* valeur à insérer et *f* indice de la première case vide

L'algorithme répète la séquence suivante : saisir une valeur puis la placer dans le tableau déjà trié. Chaque étape de la séquence correspond à une boucle. La saisie s'effectue dans une boucle externe (macro *TriVolee*) et l'insertion dans une boucle interne (macro *Placer*). L'approche proposée étant descendante, nous commençons par concevoir et programmer la macro *TriVolee*, puis la macro *Placer*.

4.2 Construction de la macro *TriVolee*

Le rôle de la macro *Trivolee* est de répéter les actions suivantes : saisir une valeur, l'insérer dans le tableau en appelant la macro *Placer* et d'incrémenter le nombre de valeurs insérées jusqu'à ce que le tableau soit entièrement rempli. Le code généré pour la macro *TriVolee* est visible sur la partie droite de la figure 3.

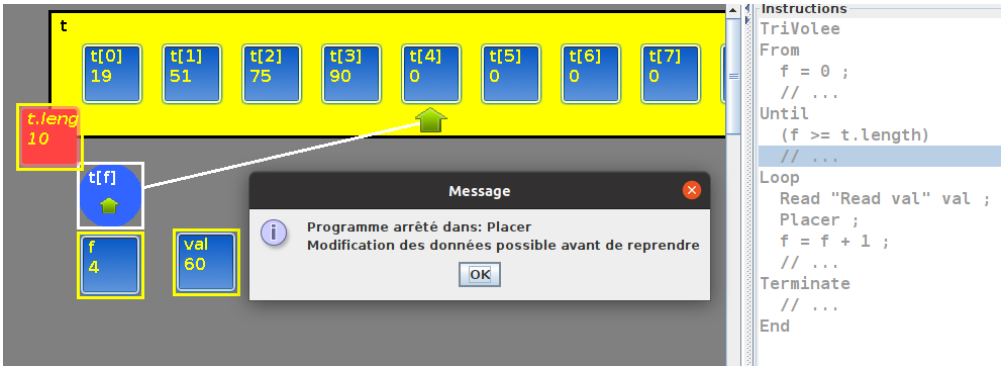


FIGURE 3 – Arrêt sur la macro *Placer* pendant l'exécution de la macro *TriVolee*

À ce niveau le code de la macro *Placer* n'est pas encore écrit. Pourtant, avec AlgoTouch, il est possible de tester la macro *TriVolee*. En effet, l'exécution d'une macro vide provoque l'arrêt de l'exécution. Sur la figure 3, on note l'arrêt sur la macro *Placer* pendant l'exécution de *TriVolee*. Le programmeur prend le contrôle car il a accès à toutes les variables du programme. Par manipulation directe des données, il insère la valeur au bon endroit dans le tableau et relance l'exécution au point d'arrêt. Sur l'exemple de la figure 3, l'utilisateur doit libérer la case t[2] et donc déplacer vers la droite les deux valeurs 75 et 90. En pratique, il doit d'abord déplacer la valeur 90 de t[3] dans t[4], puis la valeur 75 de t[2] dans t[3] et finalement placer la valeur 60 de val dans t[2].

Le programmeur s'est substitué à la macro *Placer*, et il a produit le résultat attendu à la fin de son exécution. À ce stade, il n'a pas eu besoin ni de la coder, ni même de connaitre l'algorithme à utiliser. Il a simplement constaté visuellement où placer la valeur.

Ainsi, la macro *TriVolee* peut être testée de manière unitaire. Le résultat produit ne dépend que de son code et des manipulations du programmeur. Évidemment, un test d'intégration ou fonctionnel sera à effectuer une fois le code de la macro *Placer* généré et testé.

Enfin, par les manipulations qu'il effectue, le programmeur peut déjà imaginer le comportement algorithmique de la macro *Placer*. Naturellement, il effectue des manipulations qui peuvent être répétitives et lui permettre de trouver plus facilement l'algorithme de la macro *Placer*. La génération du code de la macro doit en être facilitée comme illustré dans des travaux antérieurs (Adam et al., 2019).

4.3 Construction de la macro *Placer*

Le rôle de la macro *Placer* est d'insérer la valeur saisie dans le tableau. Évidemment, il existe plusieurs algorithmes pour réaliser la macro *Placer*. Nous avons donc effectué un choix parmi les différentes méthodes de placement existantes. L'algorithme décale les valeurs d'une position vers la

droite et s'arrête s'il trouve une valeur plus petite que celle à insérer ou s'il atteint le début du tableau. Le code généré pour la macro *Placer* est présenté sur la figure 4.

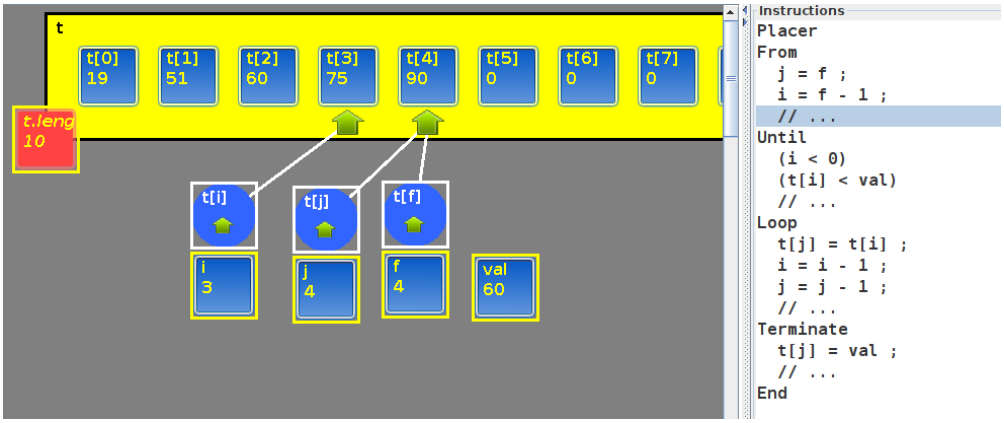


FIGURE 4 – Code la macro *Placer*

Une fois le code généré, il est possible de tester la macro *Placer* de manière unitaire. Ensuite, le test d'intégration peut être effectué en appelant la méthode *TriVolee*. Ce test est dit d'intégration car il teste le bon fonctionnement des deux macros. Dans la partie précédente, le test était unitaire, seul le code de *TriVolee* était complet, seule la macro était vérifiée.

4.4 Analyse

Dans cette partie, nous analysons *a priori* les apports de l'approche descendante avec AlgoTouch présentée dans la partie précédente. Toutes ces hypothèses devront être validées par des expérimentations.

Conception et "écriture" du code de manière concomitante
Avec des langages de programmation classique, la conception est effectuée de manière descendante et l'écriture du code ascendante. Les macros ou méthodes les plus internes sont écrites et testées en premier. Avec l'approche proposée, la conception, la programmation et les tests sont effectués de manière descendante.

Conception et programmation
Le novice ne se concentre que sur un seul niveau de boucle. Quand l'imbrication est sur plusieurs niveaux, le programmeur n'a pas besoin de déterminer à l'avance le nombre de macros qui seront nécessaires, ni toutes les variables. Il les crée en fonction de ses besoins. En générant d'abord la macro la plus externe, le programmeur définit le rôle de la macro interne.

Différentiation forte du comportement et du code
Lors de la construction de la macro externe *TriVolee*, pour notre exemple, seul le comportement de la macro interne *Placer* est utile. Il n'est pas nécessaire de connaître son code. Ainsi, le débutant peut aborder, par l'exemple, la différence entre le rôle de la macro, son algorithme et son code.

Aide à la conception de la macro interne
Quand le novice en programmation génère et exécute la macro externe *TriVolee*, il doit manipuler directement les données pour produire le résultat attendu par l'exécution de la macro interne *Placer*. Par ces manipulations, il pourrait progressivement identifier l'algorithme de la macro imbriquée. Son

codage devrait en être facilité.

Identification des variables nécessaires à chaque étape
Comme les variables ne sont créées par le programmeur qu'au fur et à mesure, le débutant en programmation découvre une première approche de la portée des variables. Une variable utilisée uniquement dans une macro interne peut être vue comme une variable propre à cette macro. Ceci permettrait d'introduire la notion de variables locales présente dans la plupart des langages de programmation.

5 Conclusion

Pour les novices, la programmation des boucles reste difficile même après plusieurs semaines de formation (Teague and Lister, 2014). Dans cet article, nous avons proposé une méthode permettant de faciliter la programmation de boucles imbriquées. La boucle interne est remplacée par un appel d'une macro vide. L'originalité de la méthode réside dans le fait qu'à l'exécution de la boucle, lors de l'appel de la macro vide, le contrôle du programme est rendu au programmeur. Par manipulation directe des données, il peut effectuer les actions sur les données du programme pour satisfaire l'objectif de la macro. Puis il rend le contrôle au programme. L'exécution reprend et s'arrête à nouveau dans la boucle. Le programmeur manipule à nouveau les données et peut progressivement identifier l'algorithme de la macro qu'il simule. Il peut donc tester la boucle principale puis s'intéresser à la construction de la macro interne. Dans l'approche par "*stubs*" proposée dans (Caspersen and Kolling, 2009), la macro interne quand elle est vide produit toujours un résultat identique. Avec notre approche, le programmeur manipule les données pour produire le résultat attendu, qui peut être différent à chaque tour de boucle.

Ce dispositif a été mis en place dans le logiciel de manipulation directe AlgoTouch. A l'heure actuelle, pour des raisons liées à la pandémie, il a été seulement utilisé par les auteurs sur plusieurs exemples. Nous envisageons des expérimentations avec des enseignants et des apprenants. Pour les enseignants, ce système permet de montrer concrètement comment construire progressivement un programme contenant une boucle imbriquée. Pour un apprenant, le système doit permettre de simplifier la construction et le test de tels programmes.

Références

Adam, M., Daoud, M., and Frison, P. (2018). Teaching and learning how to program without writing code. In *International Conference Europe Middle East & North Africa Information Systems and Technologies to Support Learning*, pages 106–117. Springer.

Adam, M., Daoud, M., and Frison, P. (2019). Direct manipulation versus text-based programming : An experiment report. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pages 353–359.

Baron, G.-L. and Drot-Delange, B. (2016). L'éducation à l'informatique à l'école primaire. *1024 : Bulletin de la Société Informatique de France*, 8 :73–79.

Caspersen, M. E. and Kolling, M. (2009). Stream : A first programming process. *ACM Trans. Comput. Educ.*, 9(1).

Cetin, I. et al. (2020). Teaching loops concept through visualization construction. *Informatics in Education-An International Journal*, 19(4) :589–609.

Cypher, A. and Halbert, D. C. (1993). *Watch what I do : programming by demonstration*. MIT press.

Daly, T. (2013). *Influence of alice 3 : reducing the hurdles to success in a cs1 programming course*. University of North Texas.

Drot-Delange, B. (2013). Enseigner l'informatique débranchée : analyse didactique d'activités. In *AREF*, pages 1–13.

Frison, P. (2015). A teaching assistant for algorithm construction. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '15, pages 9–14, New York, NY, USA. ACM.

Hundhausen, C. D., Douglas, S. A., and Stasko, J. T. (2002). A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 13(3) :259–290.

Izu, C., Weerasinghe, A., and Pope, C. (2016). A study of code design skills in novice programmers using the solo taxonomy. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*, pages 251–259.

Kordaki, M., Miatidis, M., and Kapsampelis, G. (2008). A computer environment for beginners' learning of sorting algorithms : Design and pilot evaluation. *Computers & Education*, 51(2) :708–723.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., and Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4) :1–15.

Meyer, B. (2009). *Touch of Class : learning to program well with objects and contracts*. Springer.

Mladenović, M., Boljat, I., and Žanko, Ž. (2018). Comparing loops misconceptions in block-based and text-based programming languages at the k-12 level. *Education and Information Technologies*, 23(4) :1483–1500.

Papavlasopoulou, S., Giannakos, M. N., and Jaccheri, L. (2017). Reviewing the affordances of tangible programming languages : Implications for design and practice. In *2017 IEEE Global Engineering Education Conference (EDUCON)*, pages 1811–1816.

Sorva, J., Karavirta, V., and Malmi, L. (2013). A review of generic program visualization systems for introductory programming education. *ACM Trans. Comput. Educ.*, 13(4).

Teague, D. and Lister, R. (2014). Programming : reading, writing and reversing. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, pages 285–290.

Wing, J. (2011). Research notebook : Computational thinking—what and why ? the link magazine, spring. *Carnegie Mellon University, Pittsburgh. Retrieved*, 1 :2019.

Donner du sens à l'*objet* numérique dans la formation des futur·e·s professeur·e·s des écoles

Yannick Parmentier[1, 2, 3] Sylvie Kirchmeyer[1]
(1) INSPE / Université de Lorraine, 5 Rue Paul Richard, 54320 Maxéville, France
(2) LORIA / Université de Lorraine, Campus Scientifique, 54506 Vandœuvre-Lès-Nancy, France
(3) LIFO / Université d'Orléans, Rue Léonard de Vinci, 45067 Orléans, France
yannick.parmentier@univ-lorraine.fr sylvie.kirchmeyer@univ-lorraine.fr

RÉSUMÉ

Dans la formation des futur·e·s enseignant·e·s du premier degré, les compétences numériques sont considérées au travers du prisme des Technologies de l'Information et de la Communication pour l'Éducation (TICE). Autrement dit, le *numérique* réfère à un outil au service des apprentissages, dont la maîtrise s'acquiert principalement par la pratique, et nécessite des conditions particulières (par exemple avoir du matériel à disposition). Cette vision restrictive du numérique est un frein à l'acquisition de compétences sur l'*objet* numérique lui-même, compétences dont bénéficieraient les pratiques éducatives utilisant les TICE. Nous décrivons ici une introduction transdisciplinaire à l'objet numérique en lien avec les mathématiques, co-conçue par une formatrice en mathématiques et un formateur au numérique. [1] Celle-ci se propose d'expliciter et de faire manipuler deux concepts fondamentaux communs aux deux domaines : la *numération* et l'*algorithmique*, en adoptant des points de vues complémentaires, avec entre autres pour ambition d'ainsi donner plus de sens à l'objet numérique dans la formation des futur·e·s professeur·e·s des écoles.

ABSTRACT

Giving a meaning to digital objects in the training of future school teachers.

In primary school teacher training, digital competencies are seen as related to Information and Communication Technologies for Education (ICTE). In other words, *digital* refers to a tool for learning, mastery of which is acquired mainly by practice and requires particular conditions (such as having hardware at hand). This restrictive view of digital devices hinders the acquisition of competencies about digital *objects* themselves, competencies from which ICTE activities would also benefit. We describe a transdisciplinary introduction to digital objects in relation with mathematics, co-designed by a mathematics and a computer science teacher, which focusses on defining and handling two fundamental concepts : *numeration* and *algorithmics*, while taking complementary points of view, with the goal among others to give more sense to digital objects in primary school teacher training.

MOTS-CLÉS : numérique, mathématique, numération, algorithmique, école primaire.

KEYWORDS: digital, mathematics, numeration, algorithmics, fundamental education.

1. Cette introduction a été dispensée durant le semestre de printemps 2021 à des étudiant·e·s inscrit·e·s en 2e année de master Métiers de l'Enseignement, de l'Éducation et de la Formation (MEEF), mention 1er degré, parcours enseignement et pratique accompagnée à l'INSPE de Lorraine, site de Bar-le-Duc.

Atelier « Apprendre la Pensée Informatique de la Maternelle à l'Université », dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH), Fribourg, Suisse, 07 juin 2021, pages 34–45,

1 Introduction

En France, depuis plus de 15 ans maintenant, la formation des futur·e·s enseignant·e·s du premier degré inclut la possibilité de passer une certification (communément appelée C2I2e, ou Certificat Informatique et Internet niveau 2 - enseignant) attestant de compétences numériques (Boissinot, 2004). Celle-ci regroupe 7 blocs de compétences déclinés en 28 compétences, réparties entre :

— compétences générales en lien avec l'exercice du métier (par exemple « maîtriser l'environnement numérique professionnel ») ;
— compétences nécessaires à l'intégration des TICE dans sa pratique d'enseignement (par exemple « utiliser des outils de travail collaboratif »).

L'accent y est mis sur l'outillage et la capacité à travailler dans un monde numérique en tirant le meilleur parti des ressources que ce dernier peut offrir. Force est de constater qu'au fil des ans, cette certification est devenue un (sinon *le*) repère pour les futur·e·s enseignant·e·s, en ce qui concerne les compétences en lien avec le numérique, que doit avoir tout·e professionnel·le de l'éducation.

Baron et Drot-Delange (2016), dans leur mise en perspective historique de la place de l'objet numérique à l'école primaire, nous permettent d'appréhender le contexte de création du C2I2e. Sans revenir en détail sur ce contexte, on peut noter que la création du C2I2e intervient à un moment où les programmes scolaires ne font aucune référence aux compétences *sur* l'objet numérique (autrement dit aux compétences informatiques), l'accent étant mis sur des compétences d'utilisateur·trice. [2] Conformément à cette vision *utilitaire* du numérique, le C2I2e ne couvre aucune compétence en lien avec l'objet numérique lui-même (telle que par exemple des connaissances de base sur l'architecture des ordinateurs, ou des bases en programmation).

À cela s'ajoute le fait que les futur·e·s enseignant·e·s ont pour la plupart une formation en sciences humaines et sociales. En effet, en 2018 près de 46 % des effectifs des Écoles Supérieures du Professorat et de l'Éducation (ESPE, futurs INSPE et ex IUFM) provenaient des sciences humaines et sociales, contre 14 % des sciences exactes (M.E.S.R.I., 2018). Ce public est donc a priori constitué de personnes ayant été peu exposées à l'objet numérique autrement que comme "simple" outil. [3]

Ce double phénomène pourrait expliquer, au moins en partie, une méconnaissance (voire une méfiance [4]) des futur·e·s enseignant·e·s (en particulier dans le premier degré) envers l'objet numérique. Cette méconnaissance appelle à une formation sur l'objet numérique, comme en attestent certaines enquêtes de terrain. L'enquête PROFETIC nous apprend ainsi que 14 % des enseignant·e·s de premier degré recruté·e·s depuis moins de 3 ans interrogé·e·s (387 enseignant·e·s) estiment qu'une meilleure maîtrise du numérique pourrait favoriser l'usage du numérique en classe, ce chiffre montant à 24 % si l'on considère l'ensemble des 2334 enseignant·e·s interrogé·e·s sans distinction d'expérience (M.E.N.J., 2019). Ce besoin de maîtrise de l'objet numérique est corroboré par les travaux de Becker et Ravitz (2001), qui ont montré que lorsque les enseignant·e·s avaient une certaine expertise informatique, il·elle·s étaient plus enclin·e·s à faire usage du numérique en classe.

Par ailleurs, de la même façon que l'absence de rattachement des compétences numériques à une discipline scolaire entraîne une représentation biaisée du numérique chez les élèves (Fluckiger et Reuter, 2014), on peut craindre qu'une certification universitaire détachée de toute discipline

2. Ce qui a duré jusqu'à l'apparition de l'option *Informatique et Sciences du Numérique* en Terminale Scientifique en 2011.
3. Cette situation est en train de changer, puisque le Cadre de Référence des Compétences Numériques (CRCN) paru en 2019 (M.E.N., 2019), et sur lequel est adossée la formation scolaire au numérique, inclut des compétences de base en informatique (par exemple la capacité d'écrire des programmes simples).
4. Comme l'a montré Summers (1990), compétences et attitudes envers l'objet numérique sont liées.

scientifique n'ait le même effet chez de futur·e·s enseignant·e·s.

Enfin (et c'est sans doute là un des points les plus importants), de par sa formation, le·la futur·e enseignant·e est placé·e dans un rôle d'utilisateur·trice de solutions numériques, sans pour autant avoir les clés lui permettant d'appréhender ces solutions, et ainsi de faire face à certains problèmes techniques pourtant courants (tels que l'incompatibilité entre certains logiciels et formats de fichiers, ou encore entre certains logiciels et architectures matérielles). Cette capacité à gérer (sinon appréhender) ces problèmes techniques constitue en quelque sorte la base d'une pyramide de Maslow de l'usage du numérique. Sans connaissances informatiques (i.e., sur l'objet numérique), il est très difficile (voire impossible) de se sentir pleinement en sécurité par rapport à l'usage d'outils numériques. Capelle *et al.* (2018) précisent notamment que le principal risque lié au numérique identifié par les jeunes enseignant·e·s sont les risques techniques (66,2 % des interrogé·e·s).

Dans ce contexte, la formation à l'objet numérique constitue sans doute l'un des enjeux majeurs de la formation des futur·e·s enseignant·e·s. Notons par ailleurs que cet enjeux est encore renforcé par l'introduction du numérique comme objet d'étude dans les programmes scolaires de 2016 (la programmation y étant abordée dès le cycle 2).

Si l'on s'accorde sur la nécessité de former les futur·e·s enseignant·e·s à l'objet numérique [5], se pose alors la question de savoir comment ? Comme l'indiquent Baron et Drot-Delange (2016), l'histoire de l'enseignement de l'objet numérique en contexte scolaire est riche, avec notamment l'usage de micro-mondes (Touloupaki et Baron, 2019) permettant de placer l'apprenant dans un rôle créatif. Mais qu'en est il de son enseignement dans la formation initiale de futur·e·s enseignant·e·s du primaire ? À notre connaissance, peu d'études sur ce sujet sont disponibles.

Dans le travail exploratoire que nous relatons ici, nous avons fait le choix d'une formation transdisciplinaire (Gérard et Roegiers, 2009). Ce choix est motivé par le fait que la transdisciplinarité permet entre autres de donner plus de sens aux apprentissages (Daudigny, 2017). Cette caractéristique est particulièrement bienvenue dans notre contexte de formation à l'objet numérique, aux vues de la vision utilitaire du numérique citée précédemment prônée par le C2I2e (les futur·e·s enseignant·e·s ne voyant pas toujours quel intérêt il y aurait à apprendre les bases de l'objet numérique).

Dans l'optique de lier l'introduction de l'objet numérique [6] à une discipline scolaire, plusieurs choix sont possibles, l'informatique étant par nature la science du traitement de l'information, quel que soit le domaine d'application de celle-ci. Notre choix s'est naturellement porté sur les mathématiques, pour deux raisons principales :

— les deux disciplines scientifiques (sous-jacentes aux disciplines scolaires) ont des liens [7] historiques très forts [8] ;
— les mathématiques constituent un élément important de la formation des futur·e·s enseignant·e·s du premier degré. [9] [10]

5. À l'instar des travaux du projet DALIE (Didactique et Apprentissage de l'Informatique à l'École) qui pointent notamment un besoin de formation « pour permettre aux enseignant·e·s de mieux appréhender les enjeux d'un enseignement de l'informatique à l'école » (Drot-Delange, 2018).

6. Que nous entendons ici comme apparentée à la discipline scolaire intitulée *Numérique et Science Informatique* introduite dans les programmes scolaires du lycée en 2019.

7. Pour s'en convaincre, rappelons si besoin que le dictionnaire Larousse donne pour définition de l'adjectif *numérique* : « qui relève des nombres ; qui se fait avec des nombres, est représenté par un nombre ».

8. On peut notamment citer les travaux du mathématicien Alan Turing, auteur de la *machine de Turing*, un modèle abstrait de mécanisation du calcul datant de 1936, qui est à l'origine de l'architecture des ordinateurs actuels (Girard et Turing, 1995).

9. Elles font partie des disciplines comportant une épreuve écrite au concours de recrutement de professeur·e des écoles.

10. Plus généralement, elles occupent une place particulière dans le système éducatif français, elles sont par exemple

Concrètement, nous proposons une formation commune au numérique et aux mathématiques, qui permet aux futur·e·s enseignant·e·s d'acquérir (ou de renforcer) les connaissances des objets correspondants. Cette formation se concentre autour de deux concepts clés : la numération et l'algorithmique (dont l'enseignement est décrit respectivement en Sections 2 et 3). Elle utilise des activités mises explicitement en lien avec les objets numérique *et* mathématique, afin de dresser des ponts entre les deux disciplines scientifiques. [11] Sa mise en œuvre, pour un volume total de 10 heures, a eu lieu au sein d'une unité d'enseignement dispensée à un groupe de 15 étudiant·e·s inscrits en deuxième année de Master *Métiers de l'Education, de l'Enseignement et de la Formation* (MEEF), mention 1er degré, parcours enseignement et pratique accompagnée (PEPA), en d'autres termes des étudiant·e·s qui, bien qu'ayant validé leur première année de Master, n'ont pas encore obtenu le concours de recrutement de professeur·e des écoles (CRPE). À l'issue de la présentation de cette formation, nous concluons sur ce travail préliminaire et donnons quelques pistes pour la suite (Section 4).

2 La numération : une compétence commune à l'être humain et à l'ordinateur

La numération occupe une place importante dans les programmes de l'école primaire. Les compétences visées ont trait à la *connaissance et compréhension des nombres, de leur écriture chiffrée (numération décimale) et du calcul* (M.E.N.J.S., 2020). L'un des enjeux ici pour les élèves est la maîtrise de la numération décimale, c'est-à-dire d'un système positionnel de représentation des nombres utilisant la base dix (*i.e.*, où l'on raisonne par paquets de dix éléments). Cet enseignement concerne autant la numération orale qu'écrite, qui présentent des particularités qui rendent leur maîtrise non triviale. En particulier, la numération écrite repose sur un système positionnel utilisant dix chiffres (de 0 à 9), qui s'écrivent de droite à gauche (les unités correspondant au chiffre le plus à droite), où le 0 occupe une place particulière (cf notion de chiffre significatif), et dont le lien avec la numération orale est irrégulier, on utilise par exemple le terme *onze* et non l'expression *dix-et-un*, *trente* et non *trois-dix*, etc. (Le Poche, 2010).

Dans ce contexte, nous proposons d'aborder cette problématique d'enseignement de la construction d'une numération en adoptant une vue transdisciplinaire : nous comparons la numération utilisée par l'être humain enseignée en mathématiques (utilisant la base dix) à celle utilisée par l'objet numérique (utilisant la base deux). Le but ici est de permettre une prise de recul par rapport à l'usage d'une base particulière. Cette distanciation nous paraît primordiale afin que les futur·e·s enseignant·e·s puissent appréhender la didactique de la numération et les enjeux attenants (*e.g.* savoir définir les activités à proposer aux élèves pour leur permettre d'acquérir les compétences visées par les programmes).

Par ailleurs, nous adoptons une approche par investigation, visant à placer l'apprenant·e dans un rôle actif (Hintikka, 1992). Concrètement, il s'agit de faire manipuler les deux bases (base dix et base deux) aux futur·e·s enseignant·e·s afin notamment (i) d'expliciter des propriétés et opérations de la base dix qu'il·elle·s utilisent de manière inconsciente (ce qui amène parfois des difficultés dans l'enseignement de celles-ci à des élèves de primaire), et (ii) d'esquisser le fonctionnement des ordinateurs en montrant comment l'information binaire y est manipulée, tant d'un point de vue théorique (base deux) que pratique (portes logiques). Cette introduction au fonctionnement des

distinguées des autres sciences exactes dans les programmes officiels.

11. Nous dépassons ici le cadre des disciplines scolaires stricto sensu, afin de permettre aux futur·e·s enseignant·e·s d'acquérir un plus grand recul sur les enseignements correspondants.

ordinateurs leur permettra notamment d'appréhender le sens du *bit* (***b**inary dig**it***, unité d'information) et son utilisation notamment pour définir (i) la *taille* d'un fichier, ou encore (ii) le nombre d'adresses mémoires utilisables (cf architectures 32 et 64 bits).

Les compétences visées chez les futur·e·s enseignant·e·s sont les suivantes :

- avoir un recul sur les automatismes mis en œuvre en numération ;
- être capable de modéliser ces automatismes sous forme de propriétés mathématiques ;
- être capable de transférer ces automatismes à une autre base ;
- comprendre la représentation binaire des nombres entiers (et appréhender le fait qu'un nombre fini de valeurs soit représentable en fonction du nombre de chiffres autorisés) ;
- être capable d'additionner des nombres en base deux (en posant l'addition) ;
- convertir un nombre binaire en nombre décimal et vice versa ;
- expliquer de manière simplifiée la représentation des nombres entiers dans un ordinateur.

Deux séances (pour un total de cinq heures) sont dédiées à cette partie. Leur déroulé est le suivant. La première séance débute par une phase d'accroche. On demande aux étudiant·e·s : « quel(s) système(s) de représentation des nombres connaissez vous ? À quoi ceux-ci servent-ils ? » Les réponses incluent les chiffres romains (utilisés par exemple en Histoire) et les représentations en bâtons (utilisés pour le dénombrement). On indique alors qu'on va manipuler une autre représentation en utilisant des cartes. Concrètement, il s'agit de faire découvrir le fonctionnement de la base deux au moyen de cartes représentant des chiffres binaires (bits), comme illustré en Figure 1. Chaque carte contient ainsi un nombre de points noirs en lien avec une puissance de deux (la carte la plus à droite représente le chiffre des unités et contient $2^0 = 1$ point, celle qui la précède $2^1 = 2$ points, et ainsi de suite). Nous réutilisons ici une activité de type *informatique débranchée* (Bell *et al.*, 2015), dont l'intérêt potentiel dans la formation des enseignant·e·s du premier degré a été discuté par ailleurs (Parmentier, 2018).

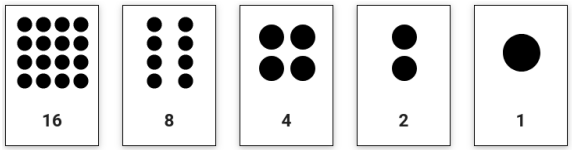


FIGURE 1 – Cartes représentant les puissances de deux par ordre décroissant.

Dans un premier temps, les cartes sont distribuées à 5 étudiant·e·s volontaires, qui viennent se placer en face de leurs collègues. Une discussion s'en suit, où l'on demande aux étudiant·e· : « à votre avis, que représentent ces cartes ? Combien de points au total contiennent ces cartes ? Comment afficher n points ? Quelle est le plus grand nombre de points que l'on peut afficher ? Etc. ». On propose alors d'attribuer un 1 aux cartes visibles et un 0 aux cartes retournées. L'affichage de 11 points correspond par exemple à la représentation fournie en Figure 2, soit, d'après notre convention d'interprétation, au nombre binaire 01011. Autrement dit, le nombre décimal 11 s'écrit 01011 en base deux ou plus simplement 1011 si l'on omet les chiffres non significatifs (ce qui est noté $\overline{1011}^2$ en mathématiques).

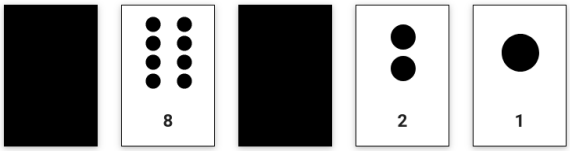


FIGURE 2 – Représentation du nombre décimal 11.

Cette introduction à la base deux nous permet également de revenir sur les démarches calculatoires associées : décomposition additive en somme de puissances de 2, conversion entre nombre décimal et binaire par divisions successives par 2. Cette phase donne lieu à l'élaboration d'une trace écrite qui

constitue une première allusion à la notion d'algorithme, auquel est donné un sens mathématique.

On propose alors aux étudiant·e·s de découvrir une nouvelle représentation des nombres entiers, qui a la propriété d'être manipulable : le boulier chinois [12] (Cumin *et al.*, 1988), illustré en Figure 3 [13]. L'utilisation d'un tel outil pour l'enseignement des mathématiques auprès d'élèves comme d'enseignant·e·s (notamment du primaire), a été étudié par ailleurs (Poisard *et al.*, 2016). Les auteur·e·s de cette étude concluent que ce boulier permet notamment de découvrir un autre moyen de construire les nombres et d'appréhender de nouvelles techniques de calcul. C'est l'occasion pour nous de faire le parallèle entre l'incrémentation (calcul de la fonction qui à un nombre n associe le nombre $n+1$) de nombres binaires et de nombres décimaux (lorsque ces derniers sont représentés par un boulier). Dans les deux cas, un patron sous-jacent (cyclique) est à l'œuvre. Cette confrontation à des calculs itérés permet notamment aux étudiant·e·s de leur faire mesurer les démarches répétitives mises en œuvre pour calculer avec un boulier. Le passage par la verbalisation et le codage permet de mettre en évidence des démarches algorithmiques différentes de celles du calcul posé, étudié par ailleurs avec l'addition.

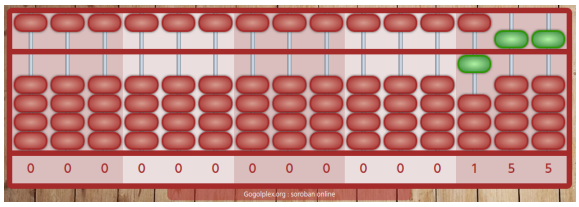
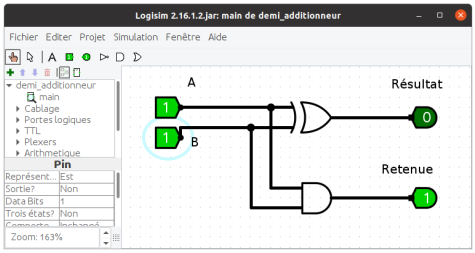


FIGURE 3 – Boulier chinois virtuel de type soroban.

Pour terminer cette première partie, nous présentons aux étudiant·e·s le concept de table de vérité, et montrons comment celui-ci permet de représenter (i) des opérations logiques (utilisant les opérateurs *et*, *ou*, *ou exclusif*, *non*), ou (ii) des opérations sur les nombres binaires (en associant au chiffre binaire 1 la valeur booléenne *Vrai* et à 0 la valeur *Faux*). En particulier, la somme s et la retenue r de deux nombres binaires a et b s'expriment respectivement par les formules logiques $s = a \oplus b$ et $r = a \wedge b$ ($\oplus$ représentant le *ou exclusif* et $\wedge$ le *et logique*). Nous concluons par l'utilisation d'un simulateur [14] pour visualiser le fonctionnement du circuit électronique demi-additionneur de bits (c'est-à-dire sans retenue en entrée), comme illustré Figure 4.



(a) Circuit électronique simulé

a	b	s	r
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(b) Table de vérité

FIGURE 4 – Demi-additionneur de bits.

À l'issue de ces deux séances, nous évaluons oralement les compétences acquises par les étudiant·e·s. Il·elle·s sont généralement capables de représenter des valeurs décimales sous forme de nombres binaires ou au moyen de bouliers chinois, et d'effectuer avec chaque représentation des additions en posant des retenues si nécessaire. Le vocabulaire de l'écriture décimale (par exemple base, rang, retenue) est mobilisé. Enfin, il·elle·s peuvent décrire de manière simplifiée comment les nombres sont représentés dans un ordinateur et comment ce dernier réalise des calculs.

12. Le boulier utilisé ici est de type soroban, et est parfois appelé également boulier japonais.
13. Version accessible en ligne à `http://www.gogolplex.org/app/bouliers/soroban.php`.
14. Logiciel `Logisim` disponible librement à l'adresse `http://logisim.altervista.org/`.

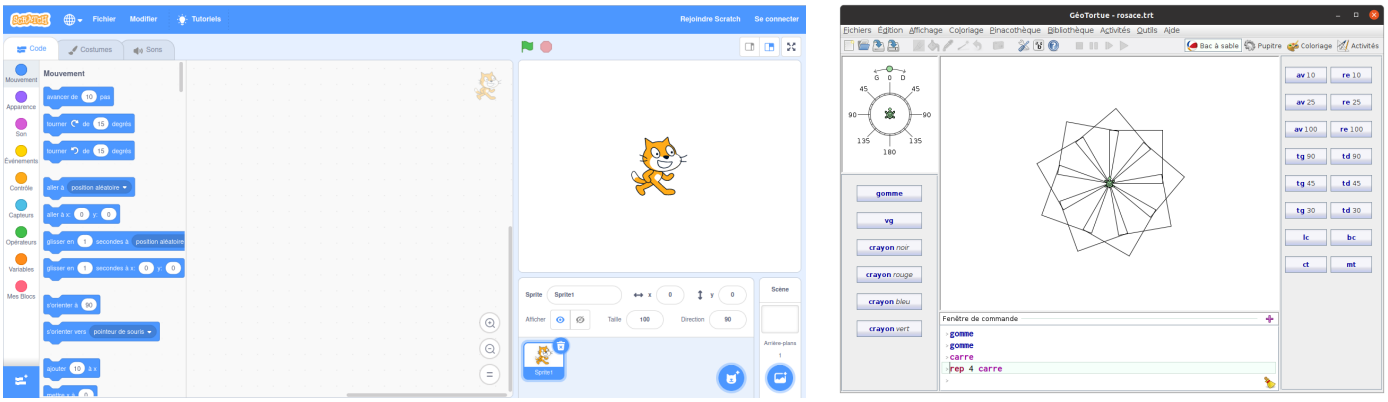
3 L'algorithmique : un outil pour calculer, mais aussi pour tracer des formes géométriques

Le second concept sur lequel s'appuie cette formation est celui d'algorithme. Ce concept est omniprésent en mathématiques. En effet, on l'utilise par exemple dès lors que l'on pose des calculs. Il est relativement aisé de faire concevoir aux étudiant·e·s qu'un algorithme correspond à un procédé systématique permettant de calculer une valeur, ou plus généralement une réponse à un problème. Il est plus délicat de leur faire appréhender celui de programme, car il requiert d'aborder la question du langage (de programmation) utilisé pour transcrire l'algorithme dans une forme aisément exploitable (notamment par un ordinateur).

Nous proposons ici d'utiliser les constructions géométriques comme cadre à la conception d'algorithmes et à l'écriture de programmes. Les motivations à ce choix sont doubles :

- l'analyse de figures et la verbalisation des étapes de construction font partie des programmes scolaires de mathématiques du 1er degré ;
- l'utilisation d'un langage de programmation dédié pour tracer des formes géométriques est un procédé éprouvé depuis les travaux autour du langage LOGO (Solomon et Papert, 1976).

Concrètement, il s'agit de faire concevoir et implanter des algorithmes de construction de formes géométriques du plan (2D), en utilisant deux environnements (et langages) de programmation : [15] `Scratch` [16] et `Géotortue` [17], dont les interfaces graphiques sont présentées en Figure 5.



(a) Logiciel `Scratch` (version 3) (b) Logiciel `Géotortue`

FIGURE 5 – Environnements de programmation utilisés.

Les objectifs visés sont les suivants :

- analyser géométriquement une figure simple ou complexe, tout en mobilisant ses connaissances sur les transformations géométriques (symétrie, translation) ;
- se repérer, décrire ou exécuter des déplacements (*relatifs* ou *absolus*) dans un plan ;
- nommer et répéter une séquence d'instructions ;
- appréhender la représentation d'un plan par un ordinateur (via un repère orthogonal) ;
- réaliser une figure simple, ou composée de figures simples, à l'aide d'un programme (et d'un logiciel interprétant ce programme).

Cette partie a par ailleurs pour but de sensibiliser les étudiant·e·s au fait qu'un ordinateur affiche des informations *numériques* (*i.e.*, une suite de bits, représentant par exemple un cercle ayant un certain

15. Nous utilisons une police différente pour distinguer l'`environnement` du langage (bien qu'ils aient le même nom).
16. `https://scratch.mit.edu`
17. `https://geotortue.free.fr`

rayon) à l'écran en considérant ce dernier comme un plan discret – ou matrice – composé de points coloriables appelés pixels (***pix**[pictures]* ***el**ements*).

Nous dédions à nouveau deux séances (pour un total de cinq heures) à cette partie. Avant de faire programmer des formes géométriques sous `Scratch`, nous choisissons d'introduire le concept de repère orthogonal utilisé par l'ordinateur pour afficher des éléments à l'écran. [18] Pour cela, nous commençons par une activité introductive basée sur une discussion collective à partir d'images projetées (présentées en Figure 6).

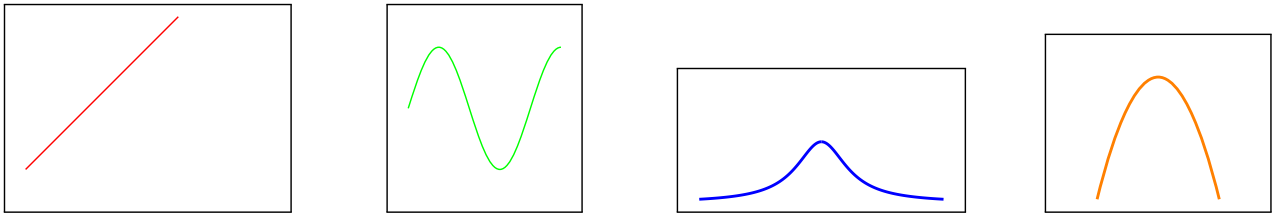
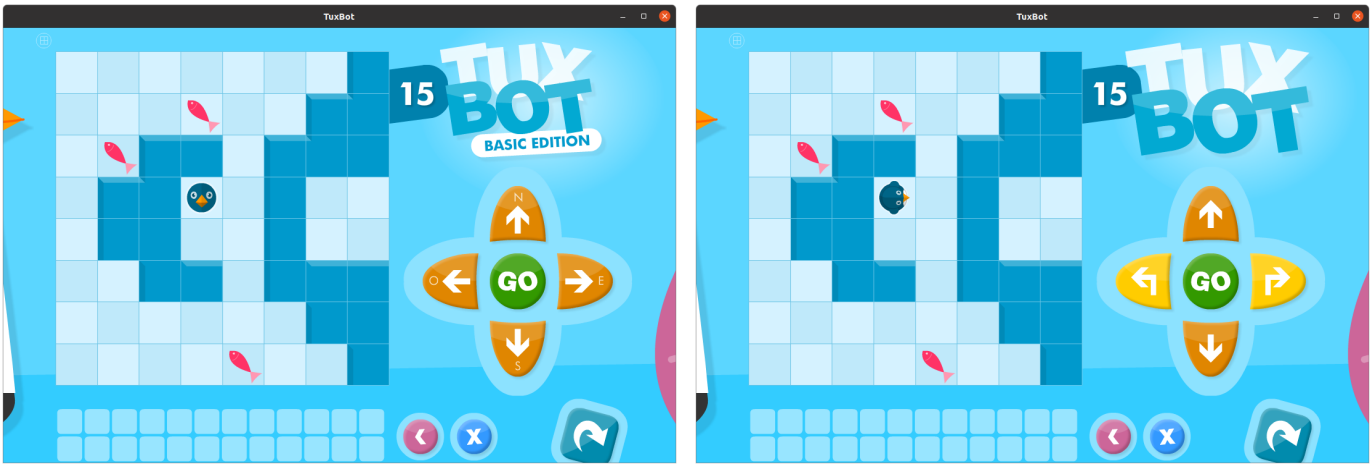


FIGURE 6 – Fonctions d'équation $y = f(x)$.

Le but de cette discussion est de faire émerger le concept de fonction, en tant que relation entre deux nombres : l'image y et son antécédent x, et que nous savons représenter dans un repère orthogonal constitué d'un axe des abscisses (horizontal) et d'un axe des ordonnées (vertical). L'ordinateur procède de même (en notant dans certains contextes des différences, par exemple un axe des ordonnées dirigé vers le bas, cf un rafraîchissement de l'écran opéré de haut en bas impliquant une position de l'origine du repère en haut à gauche).

Une deuxième activité permet d'introduire le concept de déplacement relatif. Celle-ci utilise l'environnement de programmation `Tuxbot` [19], qui permet de programmer les déplacements d'un manchot sur une grille. L'un des intérêts de `Tuxbot` réside dans le fait qu'il est configurable, afin de permettre aux utilisateur·trice·s de définir des déplacements absolus (non liés à une orientation donnée du manchot) au moyen des 4 points cardinaux, ou relatifs au moyen de directions, comme illustré Figure 7.



(a) Mode déplacements absolus (b) Mode déplacements relatifs

FIGURE 7 – Environnement de programmation `Tuxbot`.

À l'issue de ces deux activités, les étudiant·e·s sont capables de programmer des déplacements relatifs (à partir d'un point de départ) sur une grille, et de désigner des positions particulières de cette grille

18. Comme indiqué à juste titre par l'un·e des relecteur·trice·s, il aurait été intéressant de procéder par étapes, en utilisant dans un premier temps uniquement des déplacements relatifs à partir d'une position donnée (ne nécessitant pas de maitrîser les coordonnées dans un repère orthogonal), avant de basculer ensuite sur une utilisation de déplacements absolus.

19. `http://appli-etna.ac-nantes.fr:8080/ia53/tice/ressources/tuxbot/index.php`

au moyen de coordonnées dans un repère orthogonal.

Une troisième activité, inspirée par la fiche eduscol du thème « mathématiques – espace et géométrie » intitulée « Initiation à la programmation – Annexe 5.4 : Scratch – Figures géométriques » [20], consiste à prendre en main le langage Scratch en partant d'un programme à commenter, représentant ici un carré de côté 150 pixels (Figure 8a).

(a) Programme mystère

(b) Figure à coder

FIGURE 8 – Activités Scratch.

Nous utilisons l'environnement `Scratch` en premier lieu car son interface graphique permet d'avoir une vue d'ensemble du langage (blocs disponibles) ainsi que des aides visuelles sur comment ce langage s'utilise (règles *grammaticales* de combinaison des blocs représentées par la forme et la couleur de ces derniers). [21] Avant que les étudiant·e·s ne commencent à utiliser `Scratch`, nous avons pris soin de présenter son fonctionnement et son interface graphique au moyen d'un vidéo-projecteur.

Dans un second temps, nous demandons aux étudiant·e·s de programmer une figure imposée en réinvestissant le travail effectué à l'étape précédente. Par exemple ici nous demandons aux étudiant·e·s de construire une forme contenant entre autres un carré (Figure 8b). Afin de permettre aux étudiant·e·s de contrôler plus facilement la position de leurs figures à l'écran, nous leur montrons le réglage permettant d'afficher le repère orthonormé dans la zone d'affichage de `Scratch`.

En fonction de l'avancée des étudiant·e·s, une dernière activité propose d'utiliser l'environnement de programmation `Géotortue` pour construire les figures vues durant l'activité Scratch dans un nouveau langage. Un intérêt du langage Géotortue est qu'il permet de programmer les déplacements relatifs d'une tortue au moyen d'un jeu d'instructions réduit, tout en offrant une expressivité relativement riche par la possibilité d'utiliser des boucles et procédures nommées (Beaumin, 2016). Pour illustrer ceci, le code Géotortue du programme Scratch de la Figure 8a est donné en Figure 9.

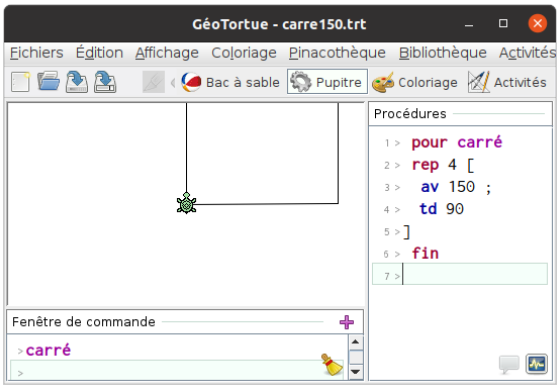


FIGURE 9 – Programme Géotortue.

20. `https://maths.dis.ac-guyane.fr/IMG/pdf/ra16_c2_c3_math_annexe_5_4_scratch_figures_geo_624916.pdf`

21. Sans compter que ce langage est parfois utilisé dans les épreuves de mathématiques du concours de recrutement de professeur des écoles, on peut par exemple demander aux candidat·e·s de donner le résultat d'un programme Scratch fourni.

Cette activité mobilise d'intéressantes compétences en mathématiques, telles qu'utiliser les connaissances relatives aux figures planes et à leurs transformations, être capable de mettre en place une démarche d'investigation (résolution d'un problème ouvert), anticiper une action, un résultat, etc.

À l'issue de ces deux séances, nous vérifions (à nouveau oralement) que les étudiant·e·s sont capables d'expliquer ce qu'est un algorithme et un programme, de lire et écrire des programmes simples au moyen du langage par blocs Scratch (et pour certain·e·s du langage textuel Géotortue). Les étudiant·e·s indiquent notamment qu'il·elle·s ont à présent non seulement une bonne compréhension du fonctionnement de l'environnement `Scratch`, qu'il·elle·s pensent être capables de lire et d'écrire des programmes Scratch décrivant des formes géométriques non triviales, mais reconnaissent également l'intérêt de passer par une représentation sous forme de programme informatique pour expliciter les propriétés mathématiques de formes géométriques. On voit ainsi poindre les effets bénéfiques d'une approche transdisciplinaire.

4 Conclusion et travaux futurs

Nous avons présenté ici une introduction à l'objet numérique à destination de futur·e·s enseignant·e·s du primaire utilisant une approche transdisciplinaire par investigation. Une évaluation superficielle des connaissances et compétences acquises, au moyen d'observations et d'évaluations orales collectives, montre un certain succès de la formation par rapport aux objectifs fixés (notamment une meilleure appréhension du fonctionnement et des fonctionnalités offertes par l'objet numérique).

Outre la sensibilisation au fonctionnement de l'objet numérique, cette formation a permis de renforcer des compétences travaillées par ailleurs en mathématiques et de leur donner du sens dans un autre domaine. Les effets sont bénéfiques pour les deux disciplines scolaires (les futur·e·s enseignant·e·s ont renforcées leurs compétences sur la numération et la géométrie, mais aussi en programmation). On peut noter que ce travail exploratoire a été bien accueilli par les étudiant·e·s (les personnes ne pouvant pas être présentes pour cause d'isolement sanitaire ont demandé expressément à suivre la formation à distance en même temps que leurs camarades sur site). Les interactions ont été riches, aussi bien entre formateur·trice·s, qu'avec les étudiant·e·s, ou encore qu'entre étudiant·e·s eux·elles-mêmes.

Les travaux futurs incluent une étude plus approfondie de la formation proposée (avec recueil de données notamment), et l'évaluation de l'impact de celle-ci sur (1) la représentation et (2) l'usage du numérique chez les futur·e·s enseignant·e·s. L'axe (1) pourrait bénéficier des apports méthodologiques d'Ailincai *et al.* (2018), dont le travail visait entre autres à identifier les représentations sur le numérique chez les enseignant·e·s de premier degré en Polynésie française. Concernant l'axe (2), nous visons le développement d'un pool d'enseignant·e·s composé pour moitié de personnes ayant suivi une formation à l'objet numérique comparable à celle présentée ici, puis d'analyser les usages du numérique en classe, avec pour but d'observer des évolutions de pratiques (et de posture).

Remerciements

Nous sommes reconnaissants envers les deux membres du comité de programme ayant relu cet article pour leurs commentaires constructifs et bienveillants. Nous remercions également les membres du projet Erasmus+ « Pensée Informatique et Algorithmique dans l'enseignement Fondamental » (PIAF, référence 2018-1-BE01-KA201-038611) pour les interactions riches en lien avec ce travail.

Références

AILINCAI, R., GABILLON, Z. et FERRIERE, S. (2018). Des éléments de corpus pour comprendre les représentations sur le numérique en contexte polynésien : préalables à la conception d'un dispositif de formation des enseignants du 1er degré. *Contextes et didactiques [En ligne]*, 11. mis en ligne le 15 juin 2018, consulté le 14 mai 2021.

BARON, G.-L. et DROT-DELANGE, B. (2016). L'informatique comme objet d'enseignement à l'école primaire française ? mise en perspective historique. *Revue française de pédagogie*, 195. mis en ligne le 30 juin 2019, consulté le 12 mai 2021.

BEAUMIN, C. (2016). *Initier ses élèves à la programmation*. Nathan, Paris.

BECKER, H. J. et RAVITZ, J. L. (2001). Computer use by teachers : Are Cuban's predictions correct ? *In Proceedings of the 2001 Annual Meeting of the American Educational Research Association*, Seattle, USA.

BELL, T., WITTEN, I. H. et FELLOWS, M. (2015). *Computer Science Unplugged – An enrichment and extension programme for primary-aged students*. University of Canterbury, Christchurch, NZ.

BOISSINOT, A. (2004). B.O. n°11 du 11 mars 2004 - Ministère de la Jeunesse, de l'Education Nationale et de la Rercherche.

CAPELLE, C., CORDIER, A. et LEHMANS, A. (2018). Usages numériques en éducation : l'influence de la perception des risques par les enseignants. *Revue française des sciences de l'information et de la communication [En ligne]*, 15. mis en ligne le 01 janvier 2019, consulté le 14 mai 2021.

CUMIN, J., DELUCHEY, C. et HOSSENLOPP, J. (1988). *Le boulier*. Edition L'Impensé Radical.

DAUDIGNY, A. (2017). Donner du sens aux apprentissages : les projets transdisciplinaires. Mémoire de Master MEEF, Université Paris-Sorbonne.

DROT-DELANGE, B. (2018). Reconfiguration de l'enseignement de l'informatique à l'école primaire : quelle conscience disciplinaire chez les professeurs des écoles stagiaires ? *Recherches en didactiques*, 1(1):27–40.

FLUCKIGER, C. et REUTER, Y. (2014). Les contenus « informatiques » et leur(s) reconstruction(s) par les élèves de CM2, Étude didactique. *Recherches en Éducation*, 18:64–78.

GIRARD, J.-Y. et TURING, A. (1995). *La machine de Turing*. Éditions du Seuil, Paris.

GÉRARD, F.-M. et ROEGIERS, X. (2009). Fiche 22. interdisciplinarité et transdisciplinarité. *In* GÉRARD, F.-M. et ROEGIERS, X., éditeurs : *Des manuels scolaires pour apprendre : Concevoir, évaluer, utiliser*, pages 293–296. De Boeck Supérieur, Louvain-la-Neuve, Belgique.

HINTIKKA, J. (1992). The interrogative model of inquiry as a general theory of argumentation. *Communication and Cognition*, 25(2-3):221–242.

LE POCHE, G. (2010). Débuter la numération. *In Le nombre au cycle 2*, pages 39–50. SCERÉN, CNDP-CRDP.

M.E.N. (2019). 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. JO, 1er septembre 2019, n° 0203, texte 25.

M.E.N.J. (2019). Enquête PROFETIC 2019 : connaître les pratiques numériques des enseignants du 1er degré.

M.E.N.J.S. (2020). Programmes d'enseignement cycle des apprentissages fondamentaux (cycle 2), cycle de consolidation (cycle 3) et cycle des approfondissements (cycle 4) : modification. Bulletin officiel de l'éducation nationale, n° 31 du 30 juillet.

M.E.S.R.I. (2018). Les effectifs en ESPE en 2017-2018. Note flash du 6 mai.

PARMENTIER, Y. (2018). Enseigner la pensée informatique à l'école primaire : formation initiale et continue des professeurs. *In Atelier "Organisation et suivi des activités d'apprentissage de l'informatique : outils, modèles et expériences" RJC-EIAH 2018*, Besançon, France.

POISARD, C., RIOU-AZOU, G., D 'HONDT, D. et MOUMIN, E. (2016). Le boulier chinois : une ressource pour la classe et pour la formation des professeurs. *MathémaTICE*, 51.

SOLOMON, C. J. et PAPERT, S. (1976). A case study of a young child doing turtle graphics in LOGO. *In American Federation of Information Processing Societies*, pages 1049–1056, New York, NY, USA. AFIPS Press.

SUMMERS, M. (1990). New student teachers and computers : An investigation of experiences and feelings. *Educational Review*, 42(3):261–271.

TOULOUPAKI, S. et BARON, G.-L. (2019). *Apprendre à programmer à l'école primaire ? Une approche exploratoire en cycle 2*. Presses universitaires du Septentrion, Villeneuve d'Ascq.

Hypothèse de la « distance » appliquée à la robot-pédagogie pour les enfants en maternelle

Julian Alvarez[1, 2], Katell Bellegarde[3] , Julie Boyaval[4], Vincent Hurez[5], Jean-Jacques Flahaut[2], Thierry Lafouge[6]
(1) CRIStAL-NOCE, Université de Lille, France
(2) INSPE, Université de Lille, France
(3) CIREL, Université de Lille, France
(4) Académie de Lille, Circonscription de Carvin, France
(5) Académie de Dijon, Circonscription de Dijon-Est, France
(6) ELICO, Université de Lyon 1, France
julian.alvarez@univ-lille.fr, katell.bellegarde@inspe-lille-hdf.fr, julie.boyaval@ac-lille.fr, vincent.hurez@ac-dijon.fr, jean-jacques.flahaut@inspe-lille-hdf.fr, thierry.lafouge@univ-lyon1.fr

RÉSUMÉ

Le projet Blue Bot a impliqué 230 élèves de cinq ans en troisième année de maternelle dans la région Nord-Pas de Calais (France). L'objectif était de les initier au codage, au décodage et à la conception de la programmation. Une séance ludopédagogique a été proposée. Elle comportait trois modalités : corps, robot et tablette. Les résultats statistiques ont confirmé la fiabilité de l'expérience scientifique et mis en avant que la bi-modalité associant robot et tablette a recensé les scores de performance les plus élevés dans une méthode évaluative de type pré et post tests. L'objectif de cette communication est d'exposer l'hypothèse de la « distance » pour tenter d'expliquer ce phénomène.

ABSTRACT

"Distance" Hypothesis applied to robot-pedagogy for children in kindergarten.

The Blue Bot project involved 230 five-year-old pupils of kindergarten in the Nord-Pas de Calais Region (France). The goal was to introduce them to coding, decoding and programming design. A ludopedagogy session was proposed with three main modalities: Body, Robot and Tablet. The statistical results confirmed the reliability of the scientific experiment and highlighted that the two modality combining Robot and Tablet recorded the highest performance scores in a pre- and post-test type evaluation method. The objective of this communication is to expose the "Distance" hypothesis as an attempt to explain this phenomenon.

MOTS-CLÉS : Robot, pédagogie, tablette numérique, étude comparative, maternelle, expérimentation, codage, décodage, conception de programmation, bi-modalité, mono-modalité, hypothèse de la distance.

Atelier « Apprendre la Pensée Informatique de la Maternelle à l'Université », dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH), Fribourg, Suisse, 07 juin 2021, pages 46–56,

KEYWORDS: Robot, pédagogie, tablette numérique, étude comparative, maternelle, expérimentation, codage, décodage, conception de programmation, bi-modalité, mono-modalité, hypothèse de la distance.

1 Introduction

Globalement, le projet de recherche Blue Bot s'est déroulé entre 2016 et 2019 et et s'est attaché à étudier plusieurs aspects en lien avec les apprentissages du codage informatique auprès des enfants de maternelle. Il a impliqué 35 enseignants, 230 élèves de 5 ans répartis dans 28 classes de grande section maternelle du Nord de la France. Parmi les différents axes du projet, il s'agissait d'étudier comment une séance ludopédagogique pouvait contribuer à aider des enfants de 5 ans inscrits en grande section maternelle à s'initier au codage informatique (Bellegarde, Boyaval et Alvarez, 2019). En tant que médiations cognitives, nous avons utilisé trois variantes d'une même séance ludopédagogique convoquant la thématique du robot dans le but d'initier les élèves à la robotique / informatique, et au codage en particulier. En effet, la médiation robotique initiée par Papert dans les années 70 (Papert, 1981) a suscité très tôt bons nombres d'engouements et d'interrogations (Cohen & Milaret, 1987) sur leurs apports en termes de développements cognitifs chez l'enfant. Cette thématique du robot nous a donc fortement inspiré pour bâtir ce projet de recherche. Concrètement, les trois variantes des séances ludopédagogiques se distinguaient au niveau des modalités proposées aux enfants :

- Utilisation du corps : un enfant incarne un robot et doit se déplacer sur un damier tracé au sol. D'autres enfants lui dictent les instructions (Figure 1 - image de gauche).

- Utilisation d'un robot jouet : les enfants programment le robot jouet Blue Bot pour qu'il se déplace sur un damier imprimé sur un tapis en plastique et posé sur une table (Figure 1 - image du centre).

- Utilisation d'une tablette numérique : le jeu est proposé dans un environnement totalement virtuel. Il se joue sur une tablette (Figure 1 - image de droite).

Figure 1: De gauche à droite, les trois modalités utilisées pour la séance ludopédagogique de Blue Bot : Corps, Robot et Tablette

Quelle que soit la modalité mis en présence, Corps, Robot ou Tablette, la séance ludopédagogique propose des activités de programmation qui se déroulent toujours sur un damier de 24 cases (grille de 4 par 6). Le but est chaque fois identique : déplacer le robot d'un point de départ à un point d'arrivée. En parallèle, une progression est proposée au niveau du jeu mis en place. Le parcours devient plus long et plus sinueux tandis que le nombre d'obstacles à éviter augmente de niveau en niveau.

Afin de préparer les enfants à participer aux différentes variantes de la séance ludopédagogique, deux autres séances pédagogiques ont été proposées en amont : D'abord, les enfants sont invités à faire des activités « débranchées » inspirées des œuvres de Margarida Romero et Viviane Vallerand (Romero et Vallerand, 2016) ainsi que de Romero et Loufane (Romero et Loufane, 2016). Il s'agit d'initier les enfants à ce que représente un robot et sa logique de fonctionnement. Puis, des initiations à la prise en main du robot jouet Blue Bot ou de la tablette ainsi que des explications sur la notion de codage et d'algorithmique ont été proposés aux enfants (Bellegarde, Boyaval et Alvarez, 2019).

Dans le cadre de cette communication, nous nous inscrivons dans la lignée de la recherche évaluative (Depover, Karsenti & Komis, 2011) en associant de la robot-pédagogie pour enseigner la pensée informatique même si cette discipline reste à clarifier (Drot-Delange, Pellet, Delmas-Rigoutsos, & Bruillard, 2019). Il s'agit ainsi de nous focaliser sur la partie du projet Blue Bot visant à étudier les performances des élèves pour s'initier à la programmation informatique. La méthode de pré-test et post-test a été mobilisée pour étudier lesdites performances issues des activités ludopédagogiques. Les élèves ont été répartis en différents groupes comme suit : Corps seul (B), Robot seul (R), Tablette seule (T), Corps + Robot (BR) et Robot + Tablette (RT). En parallèle, un groupe témoin appelé P pour « Placebo », n'ayant été exposé à aucune séance ludopédagogique entre les phases de pré et post tests, a également été mis en place. Ceci pour vérifier que les performances associées aux cinq autres groupes (B, R, T, BR et RT) sont bien reliées aux différentes séances ludopédagogiques proposées.

Les activités de pré-test et post-test sont identiques, présentées sur un format papier A4 en mode analogique (utilisation d'un crayon papier) et structurées pour évaluer les performances des élèves sur les 3 aspects suivants :

- Activité n°1 – décodage : Il est demandé à l'enfant de déduire à partir d'une série de flèches symbolisant des instructions la trajectoire effectuée par le robot. Ce tracé doit être dessin par l'enfant sur une grille.

- Activité n°2 – codage : En se basant sur la trajectoire du robot dessiné sur une grille, il est demandé à l'enfant de proposer la liste d'instructions idoine sous forme de flèches.

- Activité n ° 3 - conception de programme : l'enfant doit imaginer et tracer un chemin sur la grille pour que le robot puisse atteindre deux fleurs tout en évitant un oiseau. Puis il doit proposer les instructions idoines sous la forme de flèches.

Une fois les pré et post tests effectués, les enseignants ont remis les éléments aux chercheurs qui les ont évalués. Ceci pour s'assurer de l'homogénéité des évaluations. Ces dernières effectuées, toutes les données ont été vérifiées et traitées par statistiques (Alvarez, Katell, Boyaval, Hurez, Flahaut &

Lafouge, 2021 - publication en cours). Les résultats liés aux traitements statistiques ont révélé que les cinq groupes principaux (B, R, T, BR et RT) présentent une progression significative entre les pré- tests et les post-tests. Elle est bien supérieure à celle du groupe témoin (P). Ce qui signifie que les séances ludopédagogiques proposées ont eu un effet. Cette phase de vérification menée, nous avons éliminé à ce stade de notre analyse le groupe témoin. Nous avons aussi éliminé certains groupes d'élèves dont les tests semblaient incohérents à l'instar des élèves d'une même classe dont les résultats de performances étaient tous identiques et très élevés. Ce traitement opéré nous avons analysé les performances de 177 enfants.

Au niveau des performances liées aux différentes modalités, l'histogramme de la Figure 2 montre que les deux groupes, associant des bi-modalités, collectent le score le plus élevé de notes moyennes et honorables (Honorable-Moyen) : 39% pour BT 26% pour RT. Les mono-modalités B et R présentent des scores de 15% et 14%. La mono-modalité T ferme la marche avec un score de 7%. Dans le cas des notes faibles ou liées à des régressions (Faible-Régression), les bi-modalités RT et BR présentent aussi les scores les plus élevés : 25% pour BR, 29 % pour RT. Les mono-modalités T, R et B ont des scores quasi identiques. Pour le cas des scores faibles les différences entre modalité sont moins significatives. Ce constat permet de classer[1] les modalités en trois groupes du plus performant au moins performant : (RT BR), (R,B), T.

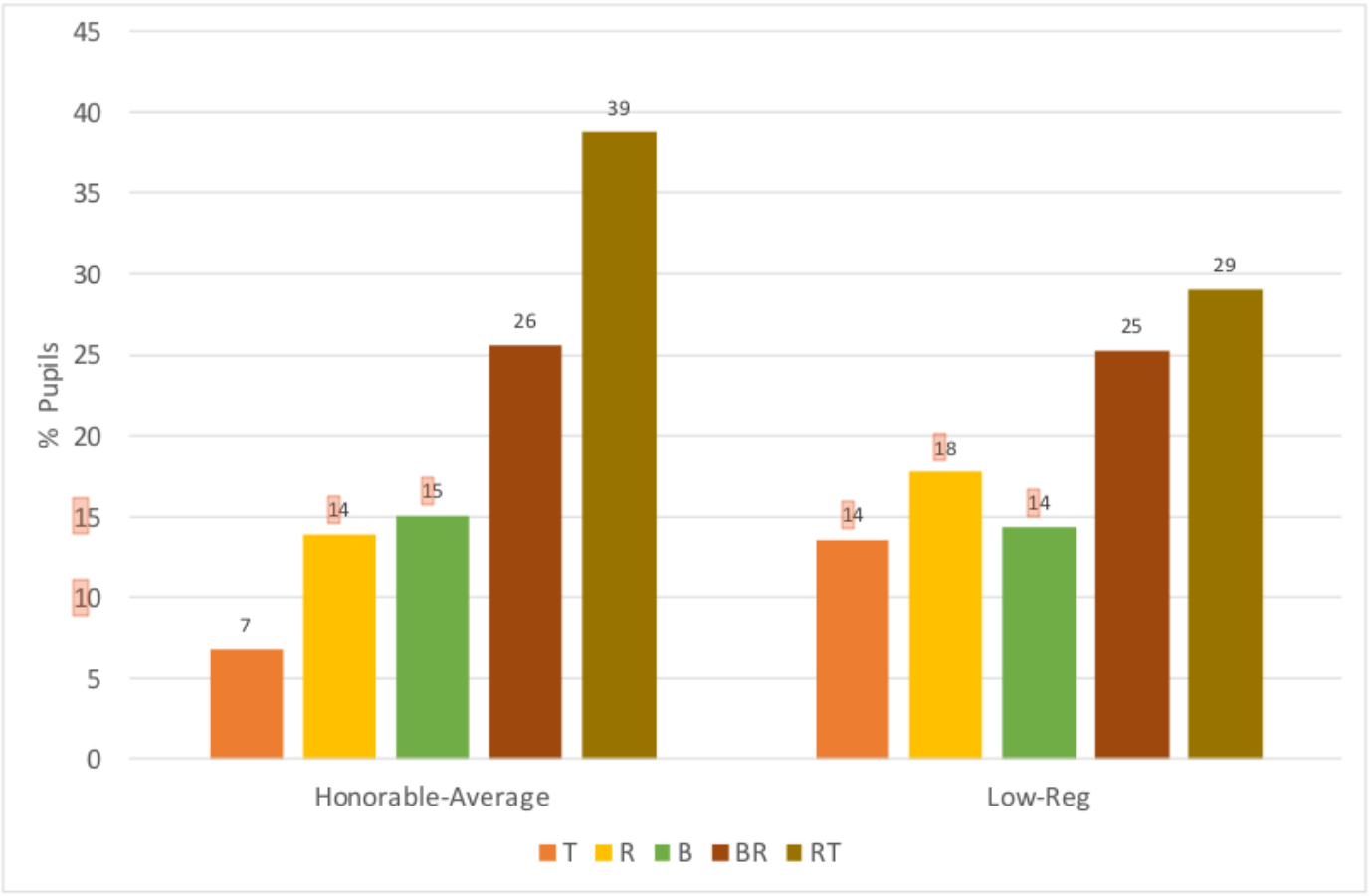


Figure 2: Scores de performances associés aux différentes modalités exprimés en pourcentages

[1] Ces 177 élèves peuvent être considérés comme un échantillon : la théorie des probabilités permet d'indiquer si deux pourcentages sont significativement différents ou non (la valeur la plus élevée de l'estimation de l'erreur est de 7,3% pour un risque de 5%). Les formules exactes peuvent être trouvées : chapitre 12 p 278 dans (Saporta, 2006).

Sur la base des résultats présentés, la modalité RT dépasse en terme de score toutes les modalités au niveau des notes Honorable-Average. Cependant, lorsqu'on se focalise sur les modalités T et R prises séparément, on observe que leurs scores de performances sont les plus faibles, respectivement 7% et 14%. En comparaison, B recense 15%. Il semble surprenant de voir que les deux mono-modalités aux scores les plus faibles composent au final la bi-modalité au score le plus performant. Ce qui induit une autre question : sachant que les mono-modalités B et R présentent réunis un meilleur score Honorable-Average de 29% (15+14) contre 21% (7+14) pour le duo R et T, pourquoi la bi-modalité RT présente-t-elle au final un meilleur score de 39% contre 26% pour BR ? Comment expliquer ce phénomène apparemment paradoxal ?

La présente communication vise à essayer de répondre à ces questions. Pour ce faire, nous allons rendre compte d'une hypothèse dite de la « distance » et la discuter via un raisonnement hypothético -déductif.

2 Deux hypothèses complémentaires

Un symposium en lien avec le projet Blue Bot s'est tenu à l'INSPE en Juin 2019 entre enseignants et chercheurs pour aborder notamment les questions liées aux classements des modalités. Deux hypothèses ont émergé pour tenter d'expliquer le paradoxe lié à la prédominance de la bi-modalité RT au regard de BR, sachant que les modalités T et R présentaient des scores Honorable-Moyen plus faibles que B. Il s'agit de l'hypothèse de la « double modalité » et celle dite de la « distance ». Ces deux hypothèses ne sont ni exclusives, ni antagonistes. Elles semblent au contraire complémentaires.

3 Hypothèse de la « double modalité »

L'hypothèse de la double modalité se base sur la notion qu'il est sans doute plus efficace pour un enfant d'expérimenter deux modalités différentes que d'en utiliser une seule (Alvarez, Katell, Boyaval,Hurez, Flahaut & Lafouge, 2021 - publication en cours). Ainsi, le facteur temps pourrait jouer un rôle prépondérant : les élèves qui ont expérimenté deux modalités ont consacré deux fois plus de temps dans leurs apprentissages que les élèves qui n'en ont exploré qu'une. Par conséquent, le temps supplémentaire peut avoir servi à promouvoir l'apprentissage dans le cas des bi-modalités. Un autre facteur fait référence aux travaux de Gardner (1996) sur les intelligences multiples. Il se pourrait que l'association de deux modalités pédagogiques constitue un moyen de différencier les situations d'apprentissages en termes d'activités cognitives prises en charge et de compétences travaillées par les élèves.

Si les deux bi-modalités BR et RT sont en tête de classement, elles sont également source de division puisqu'elles ont généré un nombre élevé de scores faibles voire de régression. Ainsi, ni la notion de temps, ni la référence aux intelligences multiples n'expliquent ce phénomène. D'autres facteurs, qui doivent encore être identifiés, sont nécessairement impliqués et ont une influence positive ou négative. Il convient de les identifier. C'est là qu'intervient l'hypothèse dite de la « distance » que nous n'avons pas eu l'occasion d'exposer jusqu'à présent.

4 Approche de Greff

L'hypothèse de la « distance » est basée sur les travaux de Greff qui se concentre sur l'écart entre le médium et la réalité physique de l'élève. Plus le médium est éloigné de l'élève, plus il devient virtuel (Greff, 2004). Plus la distance est importante, plus l'élève doit adopter un niveau d'abstraction élevé. En termes pragmatiques, lorsqu'il y a une plus grande distance, l'enfant ne peut plus compter sur ses doigts ou tracer le chemin avec son doigt... L'élève est donc amené à se représenter mentalement les choses, ce qui rend l'activité plus complexe.

Dans le contexte du projet Blue Bot, l'hypothèse de la « distance » se traduit par l'idée que les élèves doivent sans doute mieux réussir dans les activités de post-test lorsque des modalités aux distances réduites sont proposées pour appréhender l'informatique et la robotique. Dans une telle approche, il semble logique de penser que les performances doivent être plus faibles avec la modalité B qu'avec R ou la T. Pourquoi ?

5 Grille anamorphosée

Lors de l'utilisation de la modalité B, les enfants ont été invités à jouer sur un damier de 3 mètres sur 2 (voir figure 1- gauche). Une telle taille a permis aux enfants d'incarner le robot et d'exécuter sa trajectoire. L'implication du corps a permis à l'enfant d'acquérir une bonne conscience sensori-motrice, mais il ne pouvait pas disposer d'une vue panoramique du damier. Cela signifie que les enfants peuvent rencontrer des difficultés à appréhender les distances et à identifier les détails du jeu. Dans le même temps, étant donné que la taille moyenne des enfants de cinq ans est de 1,10 mètre selon l'Organisation mondiale de la santé[2], la représentation qu'ont les enfants du damier est anamorphosée.

Cette représentation a sans doute complexifié l'activité pour les enfants en créant de la distance, au sens où l'entend Greff. A contrario, les modalités du robot et de la tablette ne présentent pas de telles contraintes car les surfaces mobilisées sont plus petites : le tapis Blue Bot Robot mesure 1 x 0,60 m, et l'écran de 10 pouces mesure 0,22 x 0,12 m. Les élèves disposent donc d'une vue en mode plan non anamorphosée et peuvent percevoir le damier dans sa totalité. Dans le cadre de la robotique éducative, Vincent Hurez a proposé une solution pour réduire la distance du damier géant : il a encouragé ses élèves à se tenir en haut d'un escalier et à profiter d'une vue plus panoramique (Figure 3). Mais nous n'avons pas adopté cette solution dans le cadre du projet Blue Bot pour ne pas introduire de biais dans les données de l'étude. En effet, toutes les classes auraient eu besoin de jouer au jeu dans des conditions similaires.

Ajoutons que les évaluations pré-test et post-test (voir figures 3 à 5) ont été réalisées sur des feuilles de papier A4 (0,21 x 0,29 m). Cette modalité offre une distance réduite. Lorsque la feuille de papier A4 est utilisée comme distance de référence, alors la modalité de la tablette numérique (T) correspond à une distance nulle, le Robot (R) à une distance proche et la modalité Corps (B) avec le damier grandeur nature à une distance élevée.

---

[2] Taille filles de 5 ans selon l'OMS : https://www.who.int/childgrowth/standards/cht_lfa_filles_z_2_5.pdf?ua=1 et garçons de 5 ans selon l'OMS : https://www.who.int/childgrowth/standards/cht_lfa_garcs_z_2_5.pdf?ua=1 - pour les deux sexes la norme est de 1,10 mètre en 2021.

Figure 3: Lorsqu'ils se tiennent en haut d'un escalier, les élèves disposent d'une vue panoramique sur l'ensemble d'un damier géant

Selon cette approche, il semble donc logique de penser que les enfants qui ont été exposés à la bi-modalité RT (Robot + Tablette) présentent des performances plus élevées dans les scores de post-tests que les élèves qui ont été exposés à la mono-modalité B. Les analyses statistiques ont confirmé cet aspect (cf. Figure 2). Cependant, cela devrait également être le cas pour les mono-modalités R et T prises séparément. Et pourtant, ce n'est pas ce que nous observons avec la Figure 2. Examinons de plus près la modalité T pour mieux comprendre ce phénomène.

6 Analyse de la modalité T

La modalité T, soit la tablette, soulève plusieurs interrogations : sa distance semble d'abord être une distance nulle puisqu'elle est très similaire au format papier des pré-tests et post-tests. Dans ce cas, on peut parler d'« isomorphisme ». Et pourtant, lorsque les élèves ont été exposés à la modalité tablette, leurs résultats de performance étaient parmi les plus faibles (cf. Figure 2). À première vue, cela sembleassez paradoxal, à moins que la distance entre l'élève et la tablette ne soit considérée comme plus importante que ce que l'on pourrait penser au départ. La taille de l'objet est le seul facteur pris en considération - la tablette et la feuille de papier A4 sont en effet de taille similaire -, mais la nature même de l'objet n'a pas été prise en compte. L'utilisateur se voit présenter un environnement virtuel lorsqu'il manipule la tablette numérique. La dimension haptique - le sens du toucher de l'enfant - change donc et nous savons que les enfants acquièrent un sens plus naturel de cette dimension lorsqu'ils peuvent manipuler des objets tangibles comme le Robot (R). La senso-perception qu'implique le corps (B) devient donc le cadre de référence pour calculer cette nouvelle distance. Dans ce nouveau référentiel, B correspond à la distance zéro, le robot à une distance proche car étant extérieur du corps mais manipulable d'un point de vue haptique et la tablette est vue comme une distance éloignée par son environnement virtuel non saisissable car les éléments

numériques sont immatériels. Cela peut expliquer pourquoi une proximité entre les scores liés aux mono-modalités corps (B) et le robot (R) est mise en évidence par la Figure 2 au niveau du score Honorable-Average (cf. Figure 2). En revanche, la modalité Tablet (T) présente un score bien inférieur.

7 Représentation des distances mises en jeu

Pour représenter ces deux types de distance on construit Figure 4 un repère spatial symbolique avec deux axes : l'axe de la distance du damier (axe « Size ») en abscisse, et l'axe du sens hapitique (axe « Senso-perception ») en ordonnée. Puis les quatre modalités sont projetées.

La modalité R se situe à égale distance des deux axes, contrairement aux modalités C et T qui sont sur les axes. La position centrale de la modalité R pourrait expliquer pourquoi la Figure 2 montre un résultat relativement équilibré en ce qui concerne la répartition des différentes notes obtenues par les élèves.

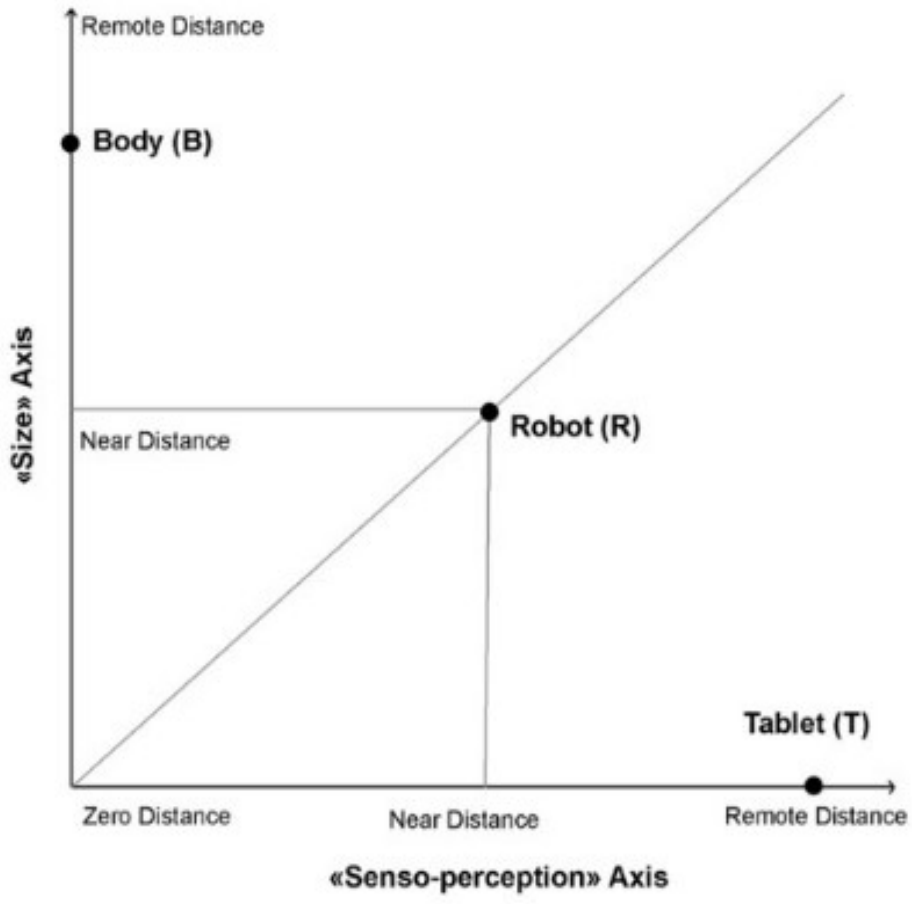


Figure 4: Distribution des modalités exprimées en distance selon les axes "Senso-perception" et "Tailles"

Dans le cas de la modalité B, une forme de distanciation peut être observée sur l'axe « Size », mais il faut rappeler que les enseignants utilisent régulièrement le corps comme modalité lors de l'enseignement aux élèves de maternelle (Bellegarde, Boyaval et Alvarez, 2019). Ces derniers ont donc potentiellement acquis une habitude qui peut les aider à compenser la distance associée à la

taille du damier. Cependant, la modalité de la tablette est probablement trop nouvelle pour que les élèves de maternelle puissent la compenser de quelque manière que ce soit à ce stade. Cela pourrait expliquer en partie pourquoi de si mauvaises performances ont été enregistrées lorsque la modalité T avait été utilisée seule. Cette compensation entre peut-être en œuvre lors des bi-modalités BT et RT ?

8 Perspectives

Si l'hypothèse de la « distance » semble donner des éléments de réponses pour expliquer le score de la bi-modalité RT au regard du faible score lié à la mono-modalité T, nous avons aussi noté des aspects qu'il conviendrait de vérifier. Tout d'abord, étudier si le phénomène de compensation est bien mis en œuvre par les élèves pour appréhender la tablette par exemple pour les bi-modalités BT et RT. Il se pourrait que l'ordre des modalités puisse jouer un rôle : il est peut-être contre-productif d'introduire la tablette avant les modalités B ou R ? Ce qui pourrait expliquer de telles régressions dans les bi- modalités.

Le temps laissé aux élèves pour assimiler et s'approprier une modalité est également un paramètre qu'il conviendrait d'étudier avec plus de rigueur. En effet, dans le cadre des expérimentations liées au projet Blue Bot, il convenait de prêter les robots et les tablettes aux différentes écoles à tour de rôle. Ces temps ont sans doute pu contribuer à créer des mises en tension chez les différents enseignants participant aux expérimentations pour déployer leurs enseignements auprès de leurs élèves. Cela a pu contribuer potentiellement à mettre en situation d'échec certains apprenants qui auraient eu besoin d'un temps d'apprentissage plus long le cas échéant pour appréhender l'usage du robot ou des tablettes.

D'autres hypothèses liées au développement cognitif des enfants de cinq ans peuvent également être étudiées. Peut-être que les enfants de cet âge ne sont pas encore capables d'effectuer des transpositions entre des environnements virtuels comme l'écran et des environnements plus analogiques comme le papier ? Dans ce registre, dans de précédents travaux nous avons ainsi mis en lumière la difficulté pour des enfants à reproduire des flèches en mode analogique (Alvarez, Katell, Boyaval, Hurez, Flahaut & Lafouge, 2021 - publication en cours). Tant que l'enfant ne peut pas établir une telle transposition, il est peut-être prématuré de vouloir l'exposer à des environnements virtuels ? D'autres expériences devraient également être menées avant d'exclure une autre possibilité : que les environnements virtuels puissent, au contraire, contribuer à améliorer le sens de l'abstraction d'un enfant de cinq ans. Mais cette approche doit être soigneusement envisagée d'un point de vue éthique à la lumière des écrits de Serge Tisseron. Ce dernier recommande en effet de limiter l'exposition à l'écran pour les enfants de moins de six ans (Tisseron, 2013).

Avant de poursuivre ces différents projets, nous proposons de refermer le présent article avec des recommandations à destination des enseignants. Employer différentes modalités pour initier des élèves de maternelle au codage, au décodage et à la conception de programmes informatiques semble être une clé pour maximiser les chances d'améliorer leurs performances au regard de la théorie des intelligences multiples avancée par Gardner. Concrètement, avec ces différentes modalités, il s'agit de diversifier les situations d'apprentissage. C'est aussi tenir compte des intelligences propres aux élèves, leur donner la possibilité de développer ou renforcer leurs intelligences.

Ce point étant précisé, il nous semble cependant important de proposer un ordre dans l'introduction de ces trois modalités. A ce stade de notre analyse, même s'il convient encore de le vérifier, nous pensons logique de proposer l'enseignement de séances ludopédagogiques telles que présentées dans le cadre du projet de recherche Blue Bot en débutant par la modalité Corps. En effet, cette dernière est déjà répandue dans les pratiques enseignantes opérées par les enseignants de maternelle et constitue de ce fait une continuité logique et naturelle pour la majorité des élèves. En outre, nous avons proposé des activités débranchées impliquant également le corps de l'élève en amont des séances ludopédagogiques liées avec le projet de Recherche Blue Bot en nous inspirant des travaux de Romero et Vallerand notamment. Au regard des résultats de performances obtenus par les groupes élèves avec l'emploi de la mono-modalité Robot et par sa position en terme de distance intermédiaire si l'on se réfère à la Figure 4, enchaîner avec la modalité Robot semble ensuite appropriée. Enfin, nous proposons de finir par la modalité Tablette, à condition que cela ne vienne pas se heurter à des problèmes éthiques en lien avec une surexposition éventuelle aux écrans chez les jeunes enfants.

Un tel enchaînement, Corps, Robot, Tablette semble donc être la séquence la plus adaptée pour maximiser chez les enfants d'école maternelle la possibilité de s'initier aux activités de codage, décodage et de conception informatique. Bien entendu, il s'agit là d'une proposition normée qui pourrait se retrouver en rupture éventuelle avec le profil de certains apprenants. Ces derniers pouvant présenter un développement de certaines facettes de leurs intelligences qui sera peut-être en déphasage avec l'ordre des modalités proposé. Il conviendra alors pour l'enseignant de le détecter le cas échéant et de voir comment adapter au mieux la situation pour les profils concernés.

Enfin, comme nous l'avons évoqué, la question de la temporalité est à prendre en compte. Si nous devons encore le vérifier et l'évaluer par des études complémentaires, nous pensons qu'il est important de laisser du temps aux élèves pour découvrir sereinement et avec plaisir, donc de manière motivée, les différentes modalités afin de pouvoir se les approprier. Sans cela, l'enseignant réduira sans doute les chances pour les différents élèves d'accéder pleinement aux aspects didactiques visés via l'emploi des modalités Corps, Robot ou Tablette.

Remerciements

Pour leurs conseils, expertises, le partage de leurs expériences, soutiens, traductions et suggestions, nous tenons à remercier : Véronique Alvarez, Anne Losq, Gilles Brougère, Serge Tisseron, Sylvie Leleu-Merviel, Dorothée Hallier-Vanuxemm, Margarida Romero, Jean-François Condette, Alfonsino Cutillo, Romain Deledicq, Yoann Lebrun, Philippe Leclercq, David Detève, Angelino Mascaro, Patrick Pelayo, Gilles Petit, Yvan Peter, Yann Secq et Marielle Léonard ainsi que tous les partenaires associés au projet de Recherche Blue Bot : l'Académie Lille et de Dijon, la DANE de Lille, le laboratoire DeVisu de l'université Polytechnique Hauts-de-France et l'INSPE Nord-de-France ainsi que l'University de Laval.

Références

ALVAREZ, J., BELLEGARDE, K., BOYAVAL, J., HUREZ, V., FLAHAUT, J-J. & LAFOUGE, T. (A paraître). « An educational robotics experiment conducted with five-year old pupils to learn coding / decoding / design », Nature : Science of Learning

BELLEGARDE, K., BOYAVAL, J. & ALVAREZ, J. (2019). « S'INITIER à la robotique/informatique en classe de grande section de maternelle - une expérimentation autour de l'utilisation du robot Blue Bot comme jeux sérieux, ReSMICTE, Vol.13, n°1, https://pasithee.library.upatras.gr/review/article/view/3105/3437

COHEN, R. & MIALARET, G. (1987). « Les Jeunes Enfants, la découverte de l'écrit et de l'ordinateur », Paris : Presses Universitaires de France.

DEPOVER, C., KARSENTI, T. & KOMIS, V. (2011). « La recherche en éducation : étapes et approches » (pp.213-228) Chapter: La recherche évaluative Publisher: Saint-Laurent, QC : ERPI Editors: Thierry Karsenti, Lorraine Savoie-Zajc.

DROT-DELANGE, B., PELLET, J. P., DELMAS-RIGOUTSOS, Y., & BRUILLARD, É. (2019). Pensée informatique: points de vue contrastés. Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation, 26(1), 39-61.

GARDNER, H. (1996). Les Intelligences Multiples Pour Changer l'École: La Prise En Compte des Différentes Formes D'Intelligence, Paris : Retz.

GREFF JP. (2004). « Le corps d'abord! », Education enfantine, 1056, pp. 62-63.

PAPERT, S. (1981). « Jaillissement de l'esprit. Ordinateur et apprentissage ». Paris : Flammarion.

ROMERO, M., VALLERAND, V. (2016). « Guide d'activités technocréatives pour les enfants du 21ème siècle ». Québec, QC: Livres en ligne du CRIRES, ISBN 978-1523809622.

ROMERO, M. & LOUFANE (2016). « Vibot the Robot », Québec : Publications du Québec.

SAPORTA, G. (2006). « Probabilités analyse des données et statistique », Edition Technip, Paris.

TISSERON, S. (2013). « 3-6-9-12 Apprivoiser les écrans et grandir ». Toulouse : Erès.

Machine Notionnelle et Pratiques de la Programmation: Mieux Apprendre avec le Développement Progressif

Charles Boisvert[1]
(1) Department of Computing, Sheffield Hallan University, Sheffield S1 1WB, United Kingdom
c.boisvert@shu.ac.uk

RÉSUMÉ

Cet article définit les problèmes de développement progressif comme des problèmes dont les solutions partielles ou erronées sont utiles au progrès vers une solution complète. Appliquant cette définition à des exemples Scratch, il présente trois exercices (un jeu simple, un en trois dimensions et une simulation mécanique de bielle-manivelle) qui peuvent être traités comme des problèmes de développement progressifs, et montre comment cette présentation permet d'encourager les apprenants à développer des pratiques de programmation efficaces et une machine notionnelle — un modèle mental de l'interprétation du code — plus juste.

ABSTRACT

Notional Machine and Practice of Programming : Learning Better with Progressive Development

This paper defines progressive development problems as problems for which the partial or mistaken solutions help progress to a more complete one. Applying this definition to Scratch examples, it presents three exercises (a simple game, another in three dimensions, and a rod-crank mechanical simulation) that can be treated as progressive development problems, and shows how this presentation helps to encourage learners to develop better programming practices and improve their notional machine — mental model of code interpretation.

MOTS-CLÉS : développement progressif ; Scratch ; ressources éducatives ; machine notionnelle.

KEYWORDS: progressive development ; Scratch ; educational resources ; notional machine.

1 Apprendre à coder

L'enseignement de la programmation dès l'école primaire est grandement facilité aujourd'hui par de nombreux outils qui permettent de faire l'économie des difficultés de syntaxe et qui proposent de réifier certains concepts clés avec des objets virtuels ou même tangibles (Resnick et al. 2009). En particulier, la représentation du code par la métaphore de pièces de puzzle, qui ne permettent que les combinaisons syntaxiquement valides, a donné lieu à de multiples systèmes (Resnick et al. 2009, Harvey & Mönig 2010, Google 2019) utilisés aujourd'hui.

Mais ce progrès a de nombreuses limites. La revue menée par (Lodi et al. 2019) montre le besoin de développer l'enseignement de la programmation dans deux directions : en étendant la perception de la discipline, de la production de code à la résolution de problèmes associés ; et en développant chez l'apprenant la machine notionnelle, le modèle mental qui permet de suivre l'exécution d'un

Atelier « Apprendre la Pensée Informatique de la Maternelle à l'Université », dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH), Fribourg, Suisse, 07 juin 2021, pages 57–67,

programme (Du Boulay 1986). Par exemple, les efforts de soutien menés par l'Université de Sheffield Hallam aux écoles locales (Adshead et al. 2015) se sont réduits après quelques années, les professeurs de primaire et de secondaire se satisfaisant d'un petit nombre d'activités qu'ils maîtrisent.

Il est clair que les enseignants ont relevé le défi de se former et de proposer rapidement cette nouvelle discipline, mais que les premières réponses ne peuvent pas suffire : le soutien doit se poursuivre dans le temps. Cet article propose, pour développer des apprentissages à l'école, de s'appuyer sur quelques notions clés de génie logiciel, et offre une piste pour les traduire dans l'enseignement.

2 Le code s'écrit pas à pas

La pratique de développer les programmes par étapes, vérifiant chacune avant de modifier le code est connue dans le génie logiciel. Comme le relève (Lodi et al. 2019), on retrouve ces pratiques dans les méthodologies agiles (Beck et al. n.d.) aujourd'hui communes chez les professionnels du développement ; cet usage de résultats exécutables incrémentaux est devenu central dans l'approche informatique des problèmes.

2.1 la Refactorisation

Par exemple, Daviaud et Revranche 2019, dans leur cahier de soutien, se servent de cette technique pour mieux faire comprendre le fonctionnement d'une boucle, comme présenté fig. 1. En proposant deux programmes aux résultats identiques, d'abord sans puis avec une boucle, les auteurs ne font pas que montrer l'avantage du second. Ils font usage d'une technique de génie logiciel, la refactorisation, qui consiste à réécrire un programme déjà fonctionnel pour le rendre plus facile à manipuler, sans pour autant en changer le résultat.

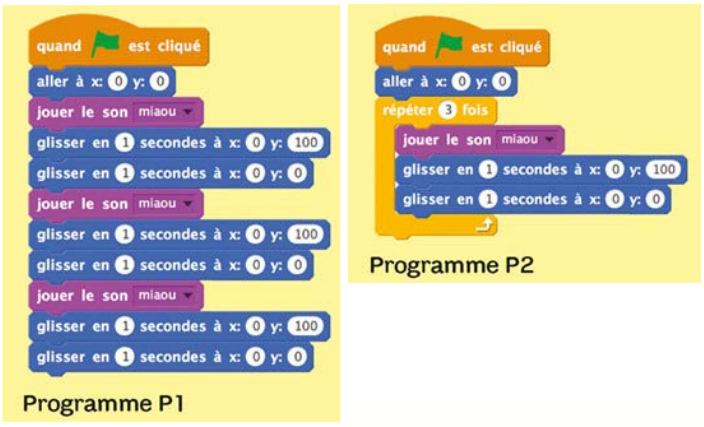


FIGURE 1 – Réécrire un programme : deux scripts aux effets identiques, l'un long, l'autre rendu plus concis par une boucle. Source : (Daviaud & Revranche 2019)

Cette technique permet aux développeurs de s'appuyer pendant la plus grande partie de leur projet sur des résultats intermédiaires exécutables, qui peuvent être évalués par comparaison avec une définition explicite du résultat fini et qui forment part du processus de résolution du problème.

2.2 Développement progressif : définition

On peut envisager que le développement de programmes par essai et erreurs aurait quelque chose de *naturel*, et donc va être *bon* dans nos méthodes d'enseignement. Mais l'exemple de la section 2.1 ci-dessus, n'est pas motivé par une erreur — ou en tout cas pas par une erreur de code ou d'exécution. Cette première idée ne suffit donc pas.

Pour mieux cerner comment cette notion se traduit dans une situation d'apprentissage, une définition plus claire, et restreinte, de ce qui fait qu'un problème de développement est progressif est proposée dans (Boisvert 2009). Il s'agit de problèmes où :

— La solution ne va pas de soi ;
— Les solutions partielles sont productives, c'est à dire qu'elles permettent un progrès vers une solution plus complète ;
— les solutions erronées, ou partiellement erronées, sont productives également.

Les problèmes de ce type proposés dans l'article d'origine sont des exercices bien connus dans l'enseignement de l'informatique au niveau supérieur, comme le tri, l'affichage d'un calendrier, ou le traitement de chaînes de caractères. Les enseignants du primaire et secondaire, ont besoin d'exemples de réalisations interactives et visuelles, plus motivantes pour de jeunes élèves, et adaptées à des environnements comme Scratch.

2.3 Déconstruire nos scripts Scratch

De nombreux manuels destinés aux élèves proposent des exemples Scratch qui sous-utilisent ce bénéfice d'une programmation pas-à-pas. Figure 2 montre, par exemple, un script extrait d'un jeu proposee par (Vorderman 2019). C'est un script assez commun, qui permet au joueur de contrôler les mouvements d'un lutin vers la droite ou la gauche avec le clavier.

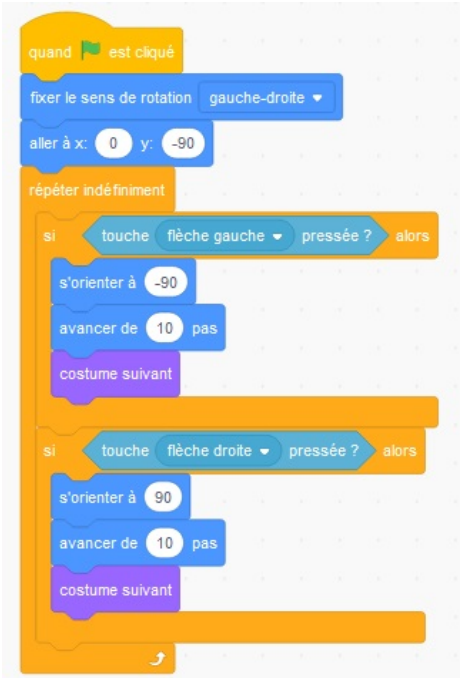


FIGURE 2 – Un script compliqué pour un mouvement simple. D'après (Vorderman 2019)

Le script utilise simultanément des effets visuels simples et un certain nombre de notions difficiles : sélections, événements, boucle, initialisation. Mais surtout il est difficile à utiliser autrement que d'une seule pièce. L'apprenant qui voudrait voir l'effet d'une partie du travail devrait choisir soigneusement quelle section ignorer pour vérifier un résultat. Une erreur de manipulation est aussi plus difficile à repérer pour la corriger. Enfin, si l'on voulait adapter le script, par exemple pour déplacer le personnage de haut en bas, ce point de départ encourage à créer peu de scripts faits de nombreux blocs : un phénomène de 'code spaghetti' bien connu des professionnels. Mêmes plus colorés, les spaghettis de Scratch n'en deviennent pas moins rapidement illisibles.

Une alternative à l'exemple ci-dessus consiste au contraire à refactoriser le code en séparant chaque résultat souhaité en scripts courts à l'objectif clair. Cette version éclatée est présentée figure 3. La version refactorisée permet d'exécuter chaque script sans devoir assembler les autres. On peut aussi plus facilement cibler une erreur de copie ou de conception. Enfin, la série de scripts est plus facile à étendre ou à modifier.

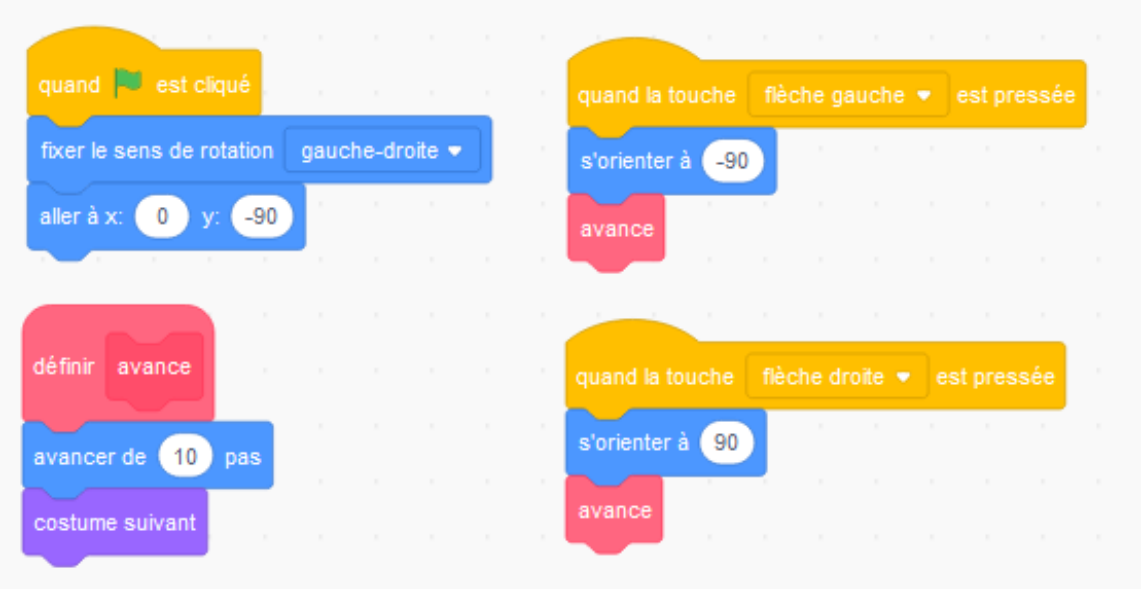


FIGURE 3 – Le même résultat que figure 2 en traitant chaque objectif (initialisation, déplacement dans chaque direction, animation du mouvement) par un script séparé.

On pourrait s'inquiéter de ce que cette seconde version fait appel à des notions complexes de parallélisme et d'événements. Les exemples de cet article utilisent en effet les deux. Cependant, Scratch facilite la compréhension, en permettant de réifier ces concepts. Ces notions sont plus complexes dans un contexte plus large — où le langage de programmation, la diversité des systèmes, la persistance d'anciennes approches, l'exigence de garantir la sécurité et la stabilité, imposent une compréhension détaillée et fine des notions et une pratique plus exigeante qui rend souvent le code confus.

Mitch Resnick raconte qu'avec son équipe, à la conception de Scratch, ils avaient 'peur de tout ce qui pourrait effrayer un enfant de huit ans' [1]. L'équipe Scratch a recherché des moyens de représenter le parallélisme ou les événements de telle façon que leur usage ne fasse pas appel aux notions abstraites nécessaires pour les comprendre en détail. Dès lors, on peut se servir de ces outils en pratique : de la même façon qu'on n'hésite pas à lancer un ballon à des enfants, même s'ils ne sont pas encore en

1. Communication orale : discours d'accueil, Scratch Bordeaux 2017

mesure de saisir tous les détails de la gravitation universelle, il ne faut pas craindre de leur donner deux script qui fonctionnent en même temps, même s'ils ne sont pas encore en mesure de saisir tous les détails de l'exécution parallèle. Peut-être même qu'au contraire, de même que l'enfant qui a joué au ballon pourra un jour se servir de cette expérience pour mieux conceptualiser le calcul balistique, celui ou celle qui aura vu Scratch exécuter deux scripts simultanément pourra s'appuyer sur cette expérience pour intégrer les concepts du parallélisme.

Pour mieux appréhender cette pratique, considérons des exemples plus complets choisis pour associer problème à résoudre et usage de code imparfait comme moyen de résolution.

3 Trois problèmes progressifs en Scratch

De nombreux exemples en Scratch peuvent être adaptés au développement progressif. Les trois cas ci-dessous illustrent l'usage que l'on peut faire des fonctionnalités de Scratch pour présenter un développement dont les étapes sont productives et testables.

3.1 Poser la fusée

L'atterrissage d'une fusée est bien connu comme jeu et comme exercice de programmation. La version présentée ici vient d'une collection de ressources créée par Martin Quinson (2014). Le problème en Scratch, et une solution qui fait usage du parallélisme sont illustrés fig. 4.

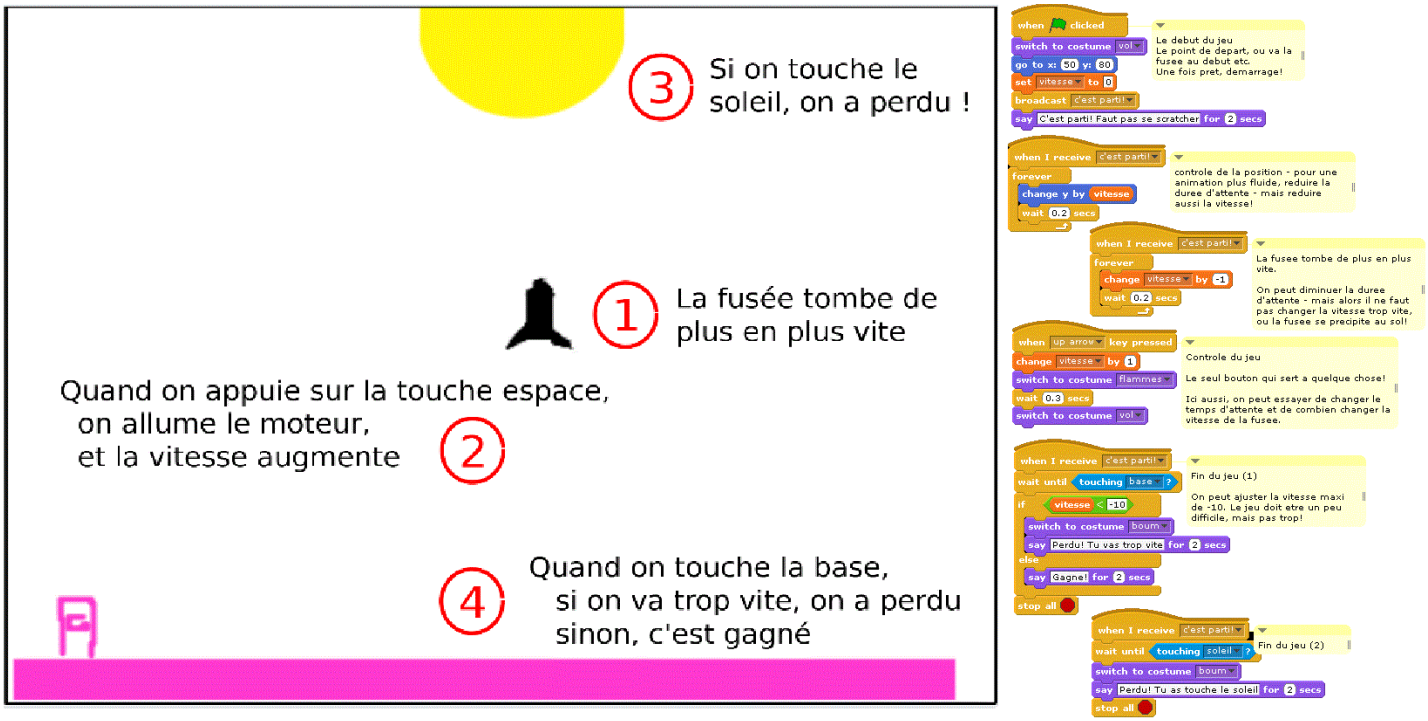


FIGURE 4 – Poser la fusée : le problème (à gauche) et une solution utilisant le parallélisme de Scratch (à droite)

On y retrouve plusieurs techniques qui aident au développement et à la compréhension du programme :

— Plusieurs scripts courts plutôt qu'un long. Chaque script peut-être écrit et testé séparément, permettant de vérifier et de réfléchir sur des résultats partiels : *les solutions partielles sont productives.*
— Décomposition des problèmes. Chaque script accomplit une seule fonction dans le jeu (de haut en bas, fig. 4 : démarrage du jeu, mouvement de la fusée, accélération de sa chute, contrôle par le joueur, et deux conditions de fin).
— Un schéma (*design pattern*) réutilisable. L'initialisation du jeu est confiée à un script unique qui établit les conditions initiales puis appelle le coeur du jeu.

Trop d'exemples proposent, comme dans le cas de la figure 2 (Vorderman 2019), de copier en entier un long script avant de voir un résultat. Cette approche permet d'introduire les concepts un par un et de vérifier leur fonctionnement par étapes. L'exercice complet (Quinson 2014) présente chaque étape du développement et insiste sur la possibilité d'exécuter le programme incomplet et de personnaliser le jeu au fur et à mesure. Elle s'étend facilement à de nombreux jeux et animations sur Scratch.

Le test d'un script en Scratch peut être l'occasion de mettre en pratique une technique de génie logiciel. Même pour de jeunes apprenants, on peut retrouver :

— *Une définition claire de 'fini'* : par exemple, il faut pouvoir exprimer à l'avance que 'si ça marche, la fusée va...'
— *Des tests multiples* : Un essai unique ne suffit généralement pas ; est ce que la fusée peut monter et descendre ? Qu'est-ce qui se passe si elle va plus vite ?
— *Une prise en compte des cas exceptionnels* : que fait la fusée au départ de l'animation, à la fin ?

3.2 Un peu de 3D

Le second exemple est tiré des ressources développées pour l'enseignement secondaire à l'Université de Sheffield Hallam (Adshead et al. 2015). L'une d'elles est une série de jeux du même type : le joueur voit arriver vers lui des objets (ennemis à abattre, obstacles, éléments de décor). Leur mouvement et la perspective créent l'illusion d'un parcours de jeu.

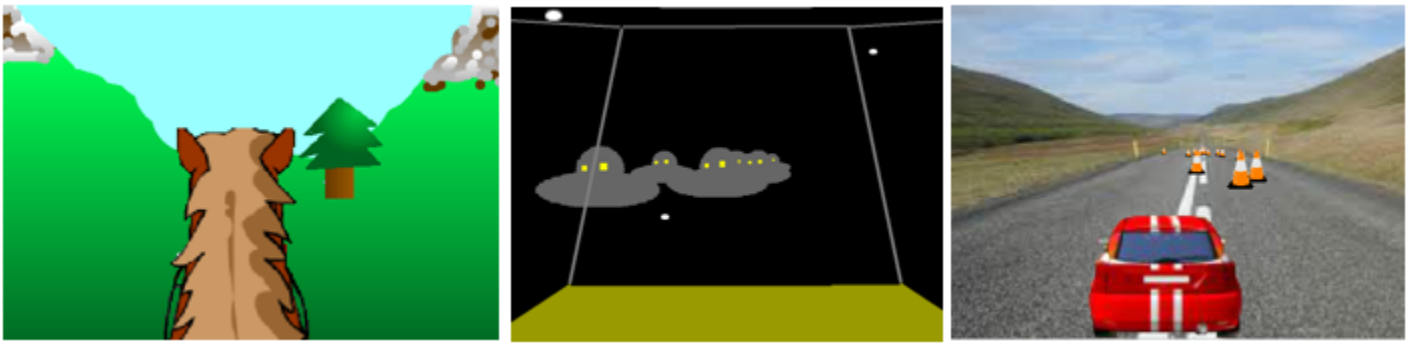


FIGURE 5 – Une série de jeux utilisant la perspective

Une spécificité de cet exemple est que des apprenants jeunes ne maîtrisent pas les calculs nécessaires. Le matériel pédagogique [2] propose donc un bloc déjà prêt, présenté figure 6, pour projeter les trois

2. https ://tinyURL.com/3DScratch

coordonnées spatiales en deux coordonnées apparentes, et explique son usage sans entrer dans les détails mathématiques.

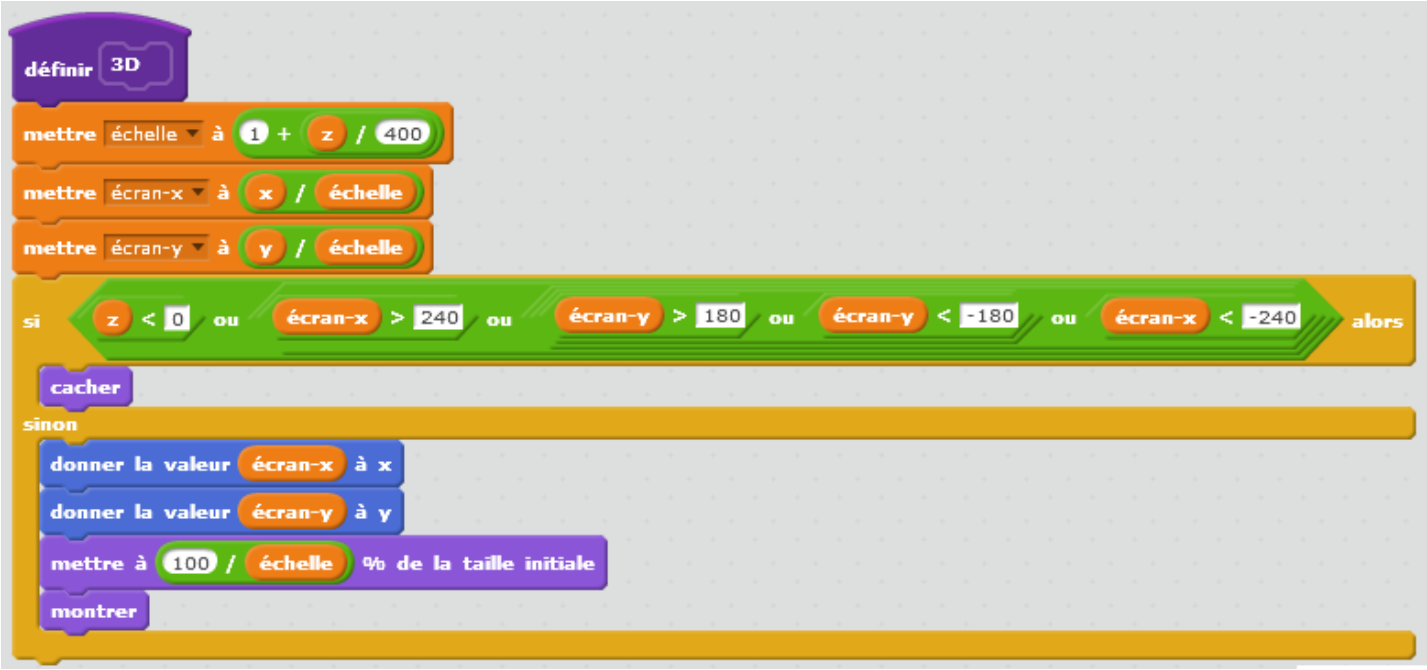


FIGURE 6 – Calcul des coordonnées X, Y et la taille apparentes d'un objet à l'écran à partir de variables de position spatiales x, y, z

Hormis ce bloc de calcul, les techniques précédentes, et en particulier le parallélisme de Scratch, permettent de définir des problèmes distincts et testables séparément : le mouvement des objets vers le joueur, les déplacements du joueur pour les atteindre ou les éviter, la gestion des collisions, forment chacun des fonctions distinctes.

Le matériel propose aussi aux apprenants de prendre le contrôle de leur jeu, de plusieurs façons :

— *Costumes à l'avance* : plusieurs lutins et fonds sont proposés, donnant des thèmes différents (course automobile, espace, balade à cheval). L'objectif est double : éviter que les apprenants ne passent trop longtemps à éditer les graphiques, et donner des options dès le départ. Les costumes sont proposés, avec le bloc de calcul 3D, dans un projet Scratch à modifier.
— *Offrir des options de jeu* : proposer des solutions alternatives au cours du développement. De nombreux exercices proposent une réalisation complète avant d'inviter l'apprenant à la modifier. La réalisation progressive et les tests d'exécution en cours de développement permettent de proposer des variantes alors que le jeu n'est pas terminé.
— *Bien choisir le bloc fourni* : le choix de ce que fait un bloc préparé d'avance demande beaucoup de soin. Il s'agit de faciliter une compréhension du problème en circonscrivant une opération et en la nommant. Ce choix limite et oriente les possibilités des apprenants, autant qu'il facilite certaines créations.

3.3 Un mécanisme bielle-manivelle [3]

Ce troisième exemple n'est pas un jeu mais une animation, montrant le mouvement d'un mécanisme bielle-manivelle. Le résultat est présenté figure 7.

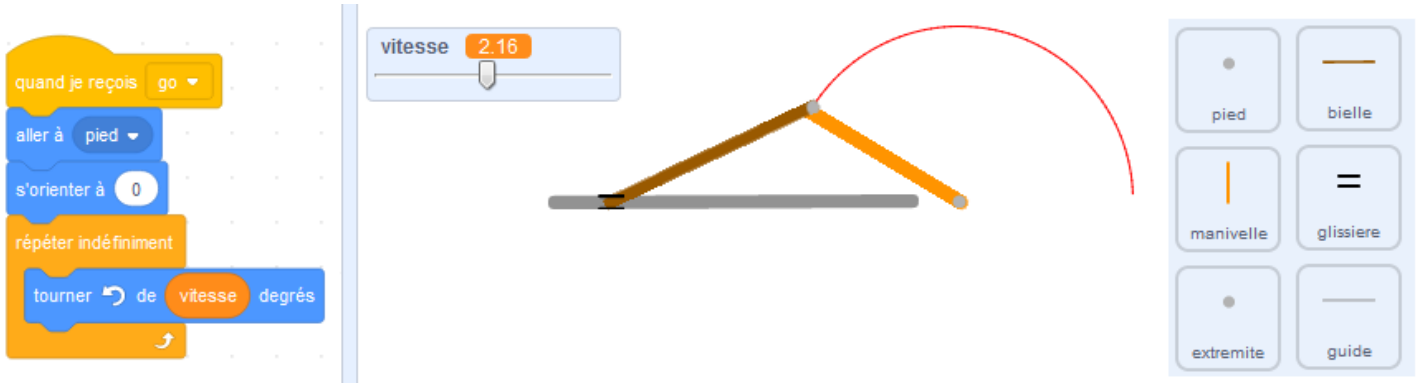


FIGURE 7 – Mécanisme bielle-manivelle. La décomposition permet de traiter le mouvement en coordonnant une série de scripts très simples, comme celui (à gauche) de la manivelle

Comme les deux précédents, le problème s'appuie sur des scripts courts qui font usage du parallélisme, et chaque sous-problème est traité par un script testable indépendamment des autres. Cependant, l'exécution en parallèle n'est pas aussi visible : le mouvement de chaque lutin est contrôlé par un unique script. La particularité de l'exercice se trouve dans l'association très claire entre les parties du mécanisme et la gestion de chaque élément du mouvement. Six lutins (à droite figure 7) modélisent le mécanisme. Deux d'entre eux (le pied et le guide horizontal) sont fixes. Le mouvement des quatre autres est simple : la manivelle et son extrémité tournent autour du pied ; la bielle suit l'extrémité de la manivelle et reste orientée vers la glissière ; la glissière enfin est maintenue à une distance toujours identique par la bielle.

Le positionnement d'un objet qui tourne autour d'un autre demande un calcul trigonométrique, pour lequel un bloc est fourni à l'avance, indiqué figure 8. Ce bloc utilise une autre notion abstraite qui peut inquiéter un enseignant mais qui, dans un contexte pratique d'usage, ne gêne pas les apprenants : le nom du bloc qui sert de point de référence est donné en paramètre [4].

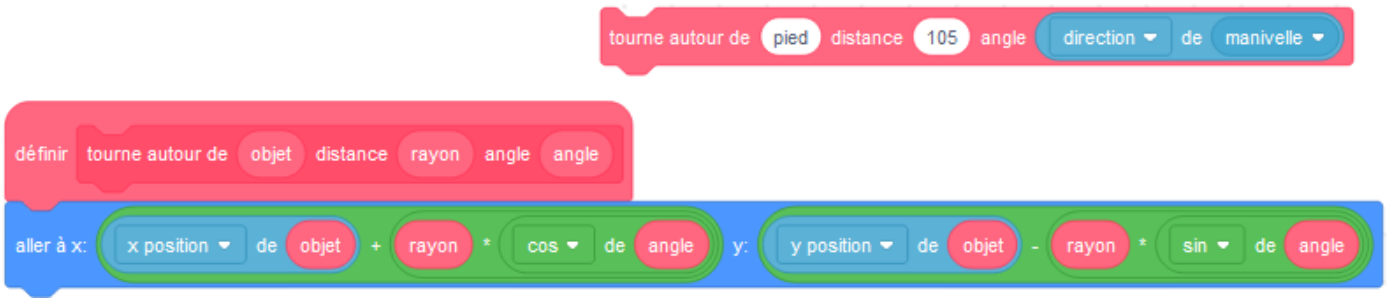


FIGURE 8 – Un bloc pour positionner un lutin autour d'un objet de référence, dont le nom est donné en paramètre.

3. https ://scratch.mit.edu/projects/87248378
4. Scratch interprète ce paramètre correctement mais traduit le bloc incomplètement quand on change la langue de l'interface

Avec des apprenants plus jeunes, il est préférable de fournir à l'avance les lutins décomposant le mécanisme. L'exercice consiste alors à trouver chaque mouvement. Comme dans les exemples précédents, on peut commencer par définir le résultat attendu, afin de tester les scripts, à mesure de leur développement, par comparaison à cette définition.

La recherche d'une solution donne lieu à toutes sortes de résultats incorrects, mais utiles à la compréhension du problème et du code, faisant du code erroné un moyen de réflexion. Seymour Papert illustre ce point magnifiquement dans son travail sur Logo (Papert 1980), reproduit figure 9 : les premières tentatives sont mauvaises, mais en voir le résultat nourrit la réflexion et un enfant ne trouverait peut-être pas le résultat sans ces 'erreurs' qui sont autant d'indications.

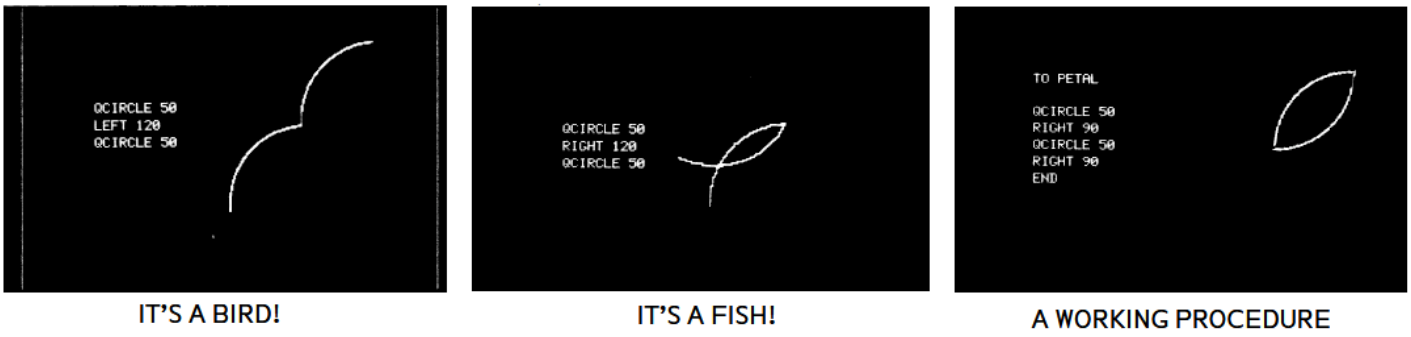


FIGURE 9 – 'C'est un oiseau !', 'C'est un poisson !' : le processus de débogage selon Papert (1980)

Le mouvement de la glissière est le plus difficile. Une solution consiste à comparer la distance de la glissière à l'extrémité de la manivelle pour la positionner : cette option présente l'avantage de s'accorder à un modèle mental du mécanisme — la bielle repousse ou tire la glissière.

Figure 10 présente une seconde option, qui peut être donnée toute prête. La solution part du même principe, mais calcule le défaut de distance pour mieux positionner la glissière.

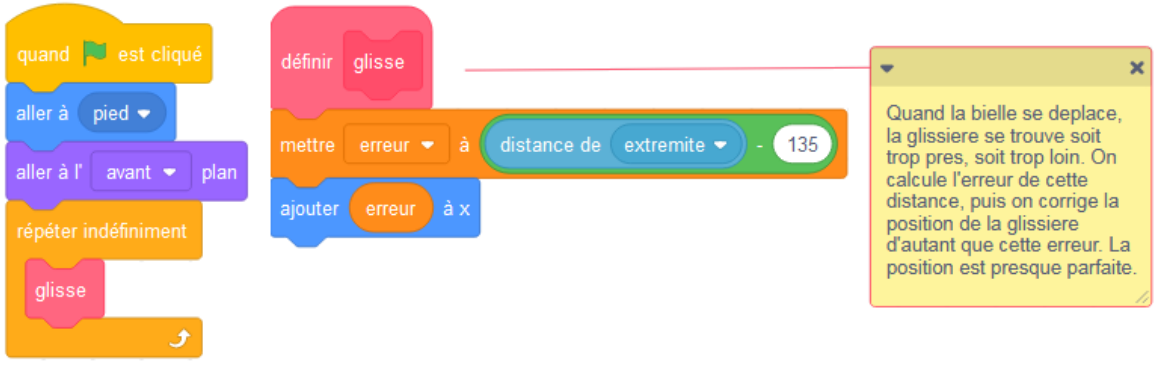


FIGURE 10 – Une manière de calculer la position de l'extrémité de la bielle.

.

Pour des apprenants plus âgés, cet exercice peut être l'occasion de rechercher une décomposition du problème. Le mécanisme bien visible soutient les apprenants dans leur recherche de solution, mais la résolution exige une étape de réflexion préalable, comme le recommande (Chevalier et al. 2020). En particulier, un bon choix du centre de chaque lutin simplifie la programmation de son mouvement, et la solution complète du mouvement de la glissière fait appel à des connaissances mathématiques (théorème de Pythagore).

4 Conclusion : Machine Notionnelle et pratiques de la programmation

La revue de (Lodi et al. 2019) fait valoir que la machine notionnelle — le modèle mental permettant de prévoir le comportement de leur système (Du Boulay 1986) — formée par de nombreux jeunes apprenants est largement simplifiée, s'appuyant souvent sur une personnification de l'interpréteur, qui plus est souvent représenté par un lutin animé. Pour l'auteur, les éducateurs et les apprenants ont besoin de soutien sur ce point, ainsi que de soutien pour appliquer des schémas et des pratiques de développement et de résolution de problèmes. Plutôt que de nouveaux outils technologiques, les exemples proposés ici tentent de répondre par des ressources utilisables dans le cadre technologique existant. En effet, l'usage pédagogique a autant, et quelquefois plus d'importance que l'outil (Hundhausen et al. 2002).

Dans le cas de Scratch, le développement de code spaghetti est une sous-utilisation fréquente du système. Les exemples proposés ici se servent du parallélisme pour à la fois développer une machine notionnelle plus complète, pour introduire une forme simple de décomposition, et pour dissocier les comportements (décrits par les scripts) des agents (décrits par les lutins). Simultanément, ils permettent de mettre en application certaines pratiques établies du développement logiciel : les méthodes de test pour guider le développement, la séparation des problèmes, et l'analyse du problème pour identifier les composants à traiter.

Références

Adshead, D., Boisvert, C., Love, D. & Spencer, P. (2015), Changing culture : Educating the next computer scientists, *in* 'Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education', ACM, pp. 33–38.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunnigham, W. & Fowler, M. (n.d.), 'Manifesto for Agile Software Development'.
URL: *https ://agilemanifesto.org/*

Boisvert, C. R. (2009), A visualisation tool for the programming process, *in* 'Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education', ITiCSE '09, Association for Computing Machinery, New York, NY, USA, pp. 328–332.
URL: *https ://doi.org/10.1145/1562877.1562976*

Chevalier, M., Giang, C., Piatti, A. & Mondada, F. (2020), 'Fostering computational thinking through educational robotics : a model for creative computational problem solving', *International Journal of STEM Education* **7**(1), 39.
URL: *https ://doi.org/10.1186/s40594-020-00238-z*

Daviaud, D. & Revranche, B. (2019), *Mini Chouette Programmer avec Scratch 5e/4e/3e : cahier de soutien en maths*, Hatier.

Du Boulay, B. (1986), 'Some difficulties of learning to program', *Journal of Educational Computing Research* **2**(1), 57–73. Publisher : SAGE Publications Sage CA : Los Angeles, CA.

Google (2019), 'Google blockly'.
URL: *https ://developers.google.com/blockly*

Harvey, B. & Mönig, J. (2010), 'Bringing "no ceiling" to scratch : Can one language serve kids and computer scientists ?', *Proc. Constructionism* pp. 1–10.

Hundhausen, C. D., Douglas, S. A. & Stasko, J. T. (2002), 'A Meta-Study of Algorithm Visualization Effectiveness', *Journal of Visual Languages & Computing* **13**(3), 259–290.
URL: *https ://www.sciencedirect.com/science/article/pii/S1045926X02902375*

Lodi, M., Malchiodi, D., Monga, M., Morpurgo, A. & Spieler, B. (2019), 'Constructionist Attempts at Supporting the Learning of Computer Programming : A Survey', *Olympiads in Informatics* **13**, 99–121.
URL: *https ://ioinformatics.org/page/ioi-journal-index/44volume13*

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

Quinson, M. (2014), 'quatre jeux · coding4kids'.
URL: *https ://github.com/mquinson/coding4kids*

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. & others (2009), 'Scratch : programming for all', *Communications of the ACM* **52**(11), 60–67. Publisher : ACM.

Vorderman, C. (2019), *Computer Coding for Kids : A unique step-by-step visual guide, from binary code to building games*, DK Children.

Pensée informatique et activités de programmation : quels outils pour enseigner et évaluer ?

Kevin Sigayret, Nathalie Blanc, André Tricot
Université Montpellier 3, Laboratoire EPSYLON EA 4556, Site Saint Charles,
1 rue du Professeur Henri Serre, 34080 Montpellier
ksigayret@univ-Montp3.fr, nathalie.blanc@univ-montp.fr,
andre.tricot@univ-montp3.fr

RÉSUMÉ

Au cours de cet atelier, nous présenterons une revue des outils d'évaluation de la pensée informatique et de leurs limites. Nous proposerons également deux nouveaux outils, que nous avons mis au point, et qui permettent de distinguer deux aspects fondamentaux de cette pensée informatique : la compréhension des notions et la capacité à résoudre des problèmes. Nous réaliserons ensuite une analyse comparative des trois principaux dispositifs d'enseignement de la programmation à l'école (activités débranchées, logiciels éducatifs et robotique). Nous verrons qu'il est aujourd'hui impossible de conclure sur l'efficacité relative de ces dispositifs pour développer la pensée informatique, par manque d'études fiables. Nous terminerons par un retour d'expérience relatif à des expérimentations que nous avons menées récemment et qui visaient à recueillir davantage de données empiriques concernant l'impact de ces dispositifs sur la capacité à acquérir la pensée informatique.

ABSTRACT

Computational thinking and programming activities :
which tools should be used to both teach and evaluate ?
In this workshop, we present a review of tools used to assess computational thinking skills and discuss the limitations of these tools. We also present two new tools we developed, to enable the distinction between two fundamental aspects of computational thinking: conceptual understanding and problem solving skills. Then, we provide a comparative analysis of the three main devices used to teach programming at school (unplugged activities, educational software and robotics). Overall, we will highlight that it is impossible today to conclude on the relative effectiveness of these devices to develop computational thinking because of the lack of reliable studies. Finally, we draw some conclusions based on experimental works we recently conducted. In sum, we stress the need to gather more empirical data about the impact of these devices on the ability to acquire computational thinking.

MOTS-CLÉS : Programmation, apprentissage, pensée informatique, évaluation, activités débranchées, Scratch, robotique éducative

KEYWORDS: Programming, learning, computational thinking, assessment, unplugged activities, Scratch, educational robotics

Atelier « Apprendre la Pensée Informatique de la Maternelle à l'Université », dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH), Fribourg, Suisse, 07 juin 2021, pages 68–75,

1 Introduction

Cette contribution s'inscrit dans le cadre d'une thèse réalisée au laboratoire de Psychologie Epsylon, rattaché à l'Université Paul Valéry Montpellier 3, sous la direction des Pr. Nathalie Blanc et André Tricot. La Délégation Académique au Numérique Éducatif (DANE) de l'Académie de Montpellier finance et suit de près ce travail de recherche. Cette thèse porte sur l'apprentissage de la programmation en contexte scolaire et sur l'acquisition de la pensée informatique.

En introduction, nous reviendrons sur la définition de la pensée informatique, telle que formulée à l'origine par Wing (2006). Nous verrons également qu'il n'existe pas véritablement de consensus sur la définition exacte de cette pensée informatique (Selby & Woollard (2013). D'où la difficulté qui en découle pour évaluer l'acquisition de cette pensée informatique et pour se prononcer sur l'efficacité des activités et dispositifs censés la développer.

Quelles sont les différentes approches existantes pour évaluer la maîtrise de la pensée informatique ? Quelles en sont les limites ?

L'apprentissage de la programmation à l'école permettrait de développer cette pensée informatique. Quels outils et dispositifs sont les plus indiqués pour enseigner cette discipline dans un cadre scolaire ?

Nous proposerons une revue de la littérature empirique relative à ces questions tout en apportant également notre contribution à ce débat. Deux nouveaux outils d'évaluation de la pensée informatique, actuellement en cours de validation psychométrique, seront proposés. Par ailleurs, un retour d'expérience sera réalisé concernant une étude expérimentale que nous avons menée au début de l'année 2021.

2 Quels outils pour évaluer la maîtrise de la pensée informatique ?

2.1 Revue des outils déjà existants

Dans cette première partie, nous reviendrons sur les différentes approches qui sont actuellement proposées pour évaluer l'acquisition et la maîtrise de la pensée informatique. Les approches qualitatives fondées sur des entretiens ou des observations peuvent présenter un intérêt certain. Cependant, nous nous focaliserons ici sur les outils permettant la récupération de données quantitatives, objectives et fiables, qui ont l’avantage de permettre des comparaisons.

Nous distinguerons notamment trois catégories d'outils qui correspondent à trois façons d'appréhender cette pensée informatique :

- Korkmaz et al. (2017) proposent le *Computational Thinking Scales* (CTS) qui regroupe des items issus de plusieurs tests distincts visant à évaluer différentes facultés cognitives. Cette approche considère la pensée informatique comme une agrégation de plusieurs compétences déjà bien identifiées (créativité, esprit critique, pensée logico-mathématique, compétences en résolution de problèmes...).

- D'autres chercheurs défendent une approche fondée sur l'analyse objective du code produit par un élève lors de la réalisation d'activités de programmation (Brennan & Resnick, 2012 ; Moreno-Léon & Robles, 2015 ; Alves et al., 2019). Ceux-ci estiment qu'il est possible d'associer l'utilisation de certains blocs issus des logiciels de programmation visuels à la maîtrise de certains concepts de la pensée informatique.

- Enfin, une dernière approche consiste à mettre en œuvre des tâches de résolution de problèmes dont l'accomplissement est censé traduire la maîtrise de certains aspects de la pensée informatique (Grover et al., 2014 ; Atmazidou & Demetriadis, 2016). Le *Computational Thinking Test* de Gonzalez (2015) représente probablement la tentative la plus aboutie en ce sens.

2.2 Deux nouveaux outils pour évaluer la pensée informatique

Globalement, les outils de mesure de la pensée informatique que nous avons détaillés précédemment sont des tentatives louables et pertinentes d'évaluer cette compétence. Cependant, ils ne proposent actuellement aucune distinction claire entre d'une part les connaissances des élèves et d'autre part leur capacité à appliquer ces connaissances pour résoudre des problèmes complexes. Or, nous considérons que cette distinction est capitale.

Tricot et Musial (2020) définissent la compétence comme un ensemble {Tâche ; Connaissances théoriquement nécessaires à la tâche ; Connaissances issues de la tâche}. En suivant ce modèle, la pensée informatique peut être considérée comme un ensemble regroupant une tâche (la résolution de problèmes algorithmiques), des connaissances théoriques nécessaires à la réalisation de cette tâche (les notions fondamentales en programmation) et des connaissances pratiques issues de cette tâche qui permettent au sujet d'utiliser les stratégies adéquates pour y faire face.

Si l'objectif est de développer une méthode d'évaluation permettant de discriminer différents niveaux de maîtrise de la pensée informatique, alors il faut pouvoir en distinguer les aspects les plus basiques, la compréhension des notions et concepts fondamentaux, et les aspects les plus complexes, l'assimilation et l'application pratique de ces concepts et le développement de stratégies permettant la résolution de problèmes.

Nous proposons ainsi deux nouveaux tests que nous avons conçus pour évaluer d'une part la compréhension des notions fondamentales en programmation et d'autre part la capacité à résoudre des problèmes algorithmiques. Le test de compréhension se compose de plusieurs questions à choix multiples. Le test d'algorithmique se présente sous la forme de problèmes à résoudre en produisant un algorithme, c'est-à-dire une série d'instructions qui constituent une solution à ce problème.

2.3 Validation psychométrique

Les deux tests ont suivi une procédure de validation psychométrique. Nous avons fait passer le test de compréhension des notions de programmation à près de 800 participants et le test d'algorithmique a été soumis à plus de 500 participants. Tous les participants étaient des élèves francophones de cycle 3 ou 4.

Chaque élève a dû, au préalable, remplir un questionnaire visant à estimer son niveau de pratique de la programmation. Deux versions de chaque test ont été créés : une version papier et une version numérique.

Les critères retenus pour s'assurer de la validité des tests sont sa sensibilité au niveau de pratique des élèves ainsi que la cohérence interne des items proposés. En d'autres termes, on s'attend à ce que le niveau de pratique de la programmation des participants soit positivement corrélé à leurs performances aux tests. On s'attend également à retrouver une corrélation entre les réponses aux items qui visent à mesurer le même construit (par exemple, la compréhension de la notion de boucle). Enfin, on s'assurera également de l'équivalence entre les versions papier et numérique des tests.

L'analyse des résultats obtenus concernant le test de compréhension des notions de programmation fait état d'une corrélation modérée entre le niveau de pratique des élèves et leurs performances au test ($r = 0.60$). Les indices de cohérence interne des items mesurant la compréhension des notions d'algorithme, de boucle, de condition ou de variable diffèrent suivant la version du test. La cohérence interne oscille entre $\alpha = 0.50$ et $\alpha = 0.54$ pour ces 4 notions dans la version numérique. Elle est plus élevée pour la version papier (entre $\alpha = 0.60$ et $\alpha = 0.66$). Les élèves de Cycle 4 présentent des niveaux de pratique significativement plus élevés que les élèves de Cycle 3, ce qui est cohérent et peut donc être considéré comme un signe de fiabilité du questionnaire visant à mesurer le niveau de pratique des élèves.

Pour le test d'algorithmique, les performances globales au test sont modérément corrélées au niveau de pratique de la programmation des élèves ($r = 0.58$). Malheureusement, peu d'élèves ont répondu à l'ensemble des exercices et seule une faible minorité est allé au-delà du 2ème exercice. Par conséquent, la seule mesure de la cohérence interne qui a pu être effectuée concerne la capacité des élèves à produire des algorithmes comprenant des instructions cohérentes et pertinentes et permettant de réaliser le parcours demandé dans les exercices 1 et 2 ($\alpha = 0.65$).

Les deux tests proposés présentent des résultats encourageants mais ne peuvent pas encore être considérés comme des outils suffisamment fiables et valides pour évaluer l'apprentissage de la programmation et la maîtrise de la pensée informatique. Pour le test de compréhension, la cohérence interne des items est actuellement insuffisante ($\alpha < 0.70$), même si la version papier est proche de cette valeur seuil. Cependant, des améliorations sont possibles pour accroître la fiabilité du test. Par exemples, les différents items présentent des niveaux de difficulté trop disparates et la possibilité de répondre correctement par hasard doit être diminuée, notamment en multipliant le nombre de réponses proposées.

En ce qui concerne le test d'algorithmique, des modifications vont être apportées pour diminuer la complexité des exercices et clarifier certaines formulations de phrases qui semblent n'avoir pas été parfaitement comprises par les élèves. De plus, lors de la passation des tests, les élèves ont souvent dû enchaîner les deux tests à la suite ce qui a rallongé la durée de la tâche et a probablement joué un rôle sur leur motivation. La passation des deux tests doit donc être réalisée séparément.

Une deuxième version de chaque test est en cours de réalisation afin de poursuivre le travail de validation psychométrique de ces outils.

3 Quels dispositifs pour enseigner la pensée informatique ?

3.1 Approche comparative des différents dispositifs utilisés

L'apprentissage de la programmation à l'école a pour finalité l'acquisition de connaissances et de compétences identifiées comme faisant partie de la pensée informatique (résolution de problèmes, aptitudes au raisonnement logique, créativité...). Il existe cependant diverses manières d'enseigner cette discipline. Nous proposerons ici une analyse comparative des trois principaux dispositifs utilisés :

- Les activités débranchées, qui ne font appel à aucun outil numérique, pourraient favoriser l'apprentissage en supprimant la charge cognitive liée à l'utilisation d'une machine tout en s'accordant avec les théories de la cognition incarnée qui soulignent l'importance des actions sensori-motrices concrètes dans le processus d'apprentissage (Romero et al., 2018)

- D'autres chercheurs ont mis en évidence l'intérêt de l'utilisation des logiciels visuels de programmation (Saez-Lopez et al., 2016 ; Xu et al., 2019). La richesse des possibilités offertes et le feedback immédiat renvoyé par l'ordinateur pourraient faciliter l'acquisition des compétences pratiques nécessaires pour résoudre des problèmes algorithmiques. Cependant, le coût cognitif de l'utilisation des logiciels pourrait également constituer une contrainte, diminuant les ressources attentionnelles disponibles et nécessaires à la compréhension.

- L'aspect tangible de la robotique éducative pourrait être un atout dans l'enseignement de la programmation tout en garantissant la grande diversité de possibilités et le feedback immédiat permis par les outils numériques (Sullivan & Heffernan, 2016 ; Scherer et al., 2020). Cependant, la recherche doit encore démontrer la validité empirique de cette supposition.

Nous conclurons cette analyse en précisant qu'à l'heure actuelle, il est impossible de se prononcer avec certitude sur l'efficacité relative de ces dispositifs comme outils permettant l'initiation à la programmation et le développement de la pensée informatique, en raison d'un manque de données empiriques fiables.

3.2 Présentation des expérimentations réalisées

Afin de faire progresser nos connaissances concernant l'efficacité relative des trois dispositifs cités précédemment, nous avons mené une expérience à laquelle ont pu participer plus de 350 élèves de cycle 3 issus de 14 classes de l'Académie de Montpellier. Ces élèves ont été répartis dans trois conditions expérimentales. Un premier groupe participait à des activités de programmation débranchées, sans utiliser aucun outil numérique. Le deuxième groupe était initié à la programmation via le logiciel éducatif Scratch. Enfin, le dernier groupe participait également à des activités sur Scratch mais, au lieu de programmer des personnages virtuels sur l'écran, le logiciel était connecté à un robot pédagogique Thymio que les élèves pouvaient ainsi apprendre à contrôler.

Les expérimentations ont eu lieu au cours des mois de Janvier et Février 2021. Au total, 10 séances de 45 minutes réparties sur 5 semaines ont été mises en place. Ces 10 séances ont été conçues pour être « équivalentes » entre les trois conditions.

Les notions abordées et les compétences travaillées étaient les mêmes dans les trois groupes expérimentaux, la seule différence reposait sur l'utilisation ou non de certains outils (logiciel et robot).

L'expérience s'est déroulée dans le cadre « naturel » de la classe afin de mettre en évidence des effets directement observables en pratique en contexte scolaire. Tous les élèves étaient novices en programmation au moment où les premières séances ont débuté.

3.3 Analyse des résultats obtenus

L'analyse des résultats est en cours de finalisation. L'objectif de cette expérience était de bénéficier de données quantitatives et objectives permettant la comparaison entre trois dispositifs d'enseignement différents afin de mettre en évidence leur efficacité relative dans l'apprentissage de la programmation à l'école. Pour ce faire, plusieurs variables ont été étudiées :

- La compréhension des notions fondamentales en programmation.
- La capacité à résoudre des problèmes algorithmiques.
- La motivation et le sentiment d'auto-efficacité.
- L'intérêt pour les disciplines scientifiques.

Pour les deux premiers facteurs, nous avons utilisé les deux tests que nous avons conçus précédemment.

Les deux autres facteurs concernent le vécu subjectif des élèves. A l'aide d'une comparaison pré-test/post-test, nous sommes en mesure d'estimer si l'un de ces dispositifs est susceptible d'accroître chez les élèves ayant participé à des activités de programmation la motivation, le sentiment d'auto-efficacité ou l'intérêt pour les sciences. Pour ce faire, nous avons sélectionné des items issus de tests existant ayant déjà été validés (Weinburgh & Steele, 2000 ; Marsh et al., 2006 ; Kind et al., 2007).

Pour l'ensemble des résultats à suivre, le coefficient d de Cohen est mentionné entre parenthèses pour indiquer la taille des effets mis en évidence.

Malheureusement, en raison de l'absence de trois des quatre enseignants de la condition « Scratch + Robot Thymio » lors de la dernière séance, les résultats des élèves au test de compréhension des notions et au test d'algorithmique n'ont pas pu être pris en compte dans cette condition.

Cependant, la comparaison entre les élèves en condition « débranchée » et les élèves en condition « Scratch » révèle un niveau de compréhension des notions largement plus élevé chez les élèves en condition « Scratch » ($d = 0.87$). Cet effet n'est pas visible sur la compréhension de la seule notion d'algorithme pour laquelle les différences observées ne sont pas significatives. En revanche, cet effet est assez fort en ce qui concerne les notions d'instruction ($d = 0.73$) et de boucle ($d = 0.67$) et encore plus marqué en ce qui concerne les notions de condition ($d = 0.81$) et de variable ($d = 0.81$).

Les filles présentent des scores de compréhension légèrement plus élevés que les garçons mais cet effet est faible ($d = 0.20$) et non significatif ($p = .08$).

De même, la capacité à résoudre des problèmes algorithmiques est nettement plus élevée dans la condition « Scratch » en comparaison avec la condition « débranchée » (d = 0.74). Cet effet n'est pas significatif sur les deux premiers exercices proposés qui sont plus simples et plus concrets mais très fort sur les deux derniers exercices proposés qui sont plus complexes et plus abstraits (d = 0.95).

Cet effet reste assez faible en ce qui concerne la capacité à produire des instructions conditionnelles (d = 0.35), moyen en ce qui concerne la capacité à produire un algorithme comprenant des instructions pertinentes et cohérentes (d = 0.61) ou la capacité à produire des boucles (d = 0.55), très fort en ce qui concerne la capacité à manipuler des variables (d = 0.95).

La capacité à résoudre des problèmes algorithmiques des élèves de CM2 est largement plus élevée que celle des élèves de CM1 (d = 0.90). Cet effet ne vient cependant pas remettre en cause la supériorité de la condition « Scratch » sur la condition « débranchée ». Les élèves de CM1, largement minoritaires, sont répartis assez équitablement dans les deux conditions.

Le niveau de compréhension des notions et la capacité à résoudre des problèmes algorithmiques sont moyennement corrélés (r = 0.40).

Au niveau du vécu subjectif des élèves, il n'y a pas d'évolution significative pré-test / post-test de l'intérêt pour les sciences dans les trois conditions étudiées.

On peut toutefois constater une légère baisse de la motivation post-test dans la condition « débranchée » par rapport au niveau de motivation évalué pré-test (d = 0.31), alors que motivation et sentiment d'auto-efficacité restent stables dans les deux autres conditions. Pré-test, les élèves dans la condition « Scratch + Robot » présentent un niveau de motivation plus élevé que dans les deux autres conditions (d = 0.72 par rapport à la condition « débranchée », d = 0.50 par rapport à la condition « Scratch »). Enfin, les garçons présentent des scores significativement plus élevés que les filles en ce qui concerne un item relatif à la volonté de faire de la programmation sur son temps libre (d = 0.40), présent dans le post-test seulement.

Références

Alves, N. D. C., Von Wangenheim, C. G., & Hauck, J. C. (2019). Approaches to assess computational thinking competences based on code analysis in K-12 education: A systematic mapping study. Informatics in Education, 18(1), 17.

Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. Robotics and Autonomous Systems, 75, 661-670.

Brennan, K., & Resnick, M. (2012, April). New frameworks for studying and assessing the development of computational thinking. In Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada (Vol. 1, p. 25).

González, M. R. (2015). Computational thinking test: Design guidelines and content validation. In Proceedings of EDULEARN15 conference (pp. 2436-2444).

Grover, S., Cooper, S., & Pea, R. (2014, June). Assessing computational learning in K-12. In Proceedings of the 2014 conference on Innovation & technology in computer science education (pp. 57-62).

Kind, P., Jones, K., & Barmby, P. (2007). Developing attitudes towards science measures. *International journal of science education*, *29*(7), 871-893.

Korkmaz, Ö., Cakir, R., & Özden, M. Y. (2017). A validity and reliability study of the computational thinking scales (CTS). Computers in Human Behavior, 72, 558-569.

Marsh, H. W., Hau, K. T., Artelt, C., Baumert, J., & Peschar, J. L. (2006). OECD's brief self-report measure of educational psychology's most useful affective constructs: Cross-cultural, psychometric comparisons across 25 countries. *International Journal of Testing*, *6*(4), 311-360

Moreno-León, J., & Robles, G. (2015, November). Dr. Scratch : A web tool to automatically evaluate Scratch projects. In Proceedings of the workshop in primary and secondary computing education (pp. 132-133).

Romero, M., Viéville, T., Duflot-Kremer, M., de Smet, C., & Belhassein, D. (2018, August). Analyse comparative d'une activité d'apprentissage de la programmation en mode branché et débranché.

Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. Computers & Education, 97, 129-141.

Selby, C., & Woollard, J. (2013). Computational thinking: the developing definition.

Sullivan, F. R., & Heffernan, J. (2016). Robotic construction kits as computational manipulatives for learning in the STEM disciplines. Journal of Research on Technology in Education, 48(2), 105-128.

Tricot, A., & Musial, M. (2020). Précis d'ingénierie pédagogique. De Boeck Supérieur.

Weinburgh, M. H., & Steele, D. (2000). The modified attitudes toward science inventory: Developing an instrument to be used with fifth grade urban students. *Journal of Women and Minorities in Science and Engineering*, *6*(1).

Wing, J. M. (2006). Computational thinking. Communications of the ACM, 49(3), 33-35.

Xu, Z., Ritzhaupt, A. D., Tian, F., & Umapathy, K. (2019). Block-based versus text-based programming environments on novice student learning outcomes: a meta-analysis study. Computer Science Education, 29(2-3), 177-204.

Permettre l'évaluation par les pairs de projets tuteurés en informatique : une grille critériée adaptée aux jeux sérieux

Ying-Dong Liu [1], Julien Gossa [2], Laurence Schmoll [3]
(1) Université de Strasbourg, LISEC UR 2310, Strasbourg, France
(2) Université de Strasbourg, ICube UMR 7357, Strasbourg, France
(3) Université de Strasbourg, LiLPa UR 1339, Strasbourg, France
yingdong.liu@etu.unistra.fr, julien.gossa@unistra.fr, lschmoll@unistra.fr

RESUME

L'article présente un retour d'expérience à propos de l'utilisation d'une grille critériée pour l'évaluation de jeux sérieux conçus dans le cadre d'un projet tuteuré de DUT informatique. Ce projet se base sur une collaboration entre des étudiants en informatique chargés du développement, et des doctorants en sciences sociales chargés de présenter une notion scientifique que le jeu doit transmettre au joueur. 24 jeux ont été produits, et leur évaluation a été faite par les pairs. Afin d'objectiver l'évaluation des jeux sérieux, une grille critériée est développée puis utilisée, portant sur trois aspects (pragmatique, hédonique et technique) en s'appuyant sur une enquête réalisée auprès des étudiants.

ABSTRACT

Peer-reviewing for Computer Science tutored projects: a criterion-based rubric that fits serious games.

The article presents serious games evaluation criteria and its usage for a serious game student project, part of a "DUT" project. This project is based on a collaboration between students in computer science, who are in charge of development, and doctoral students in social sciences, presenting a scientific notion that the game must transmit to the player. Twenty-four games were produced, then evaluated by peers. To ensure the objectivity of the assessment of serious games, the criteria were developed and used based on a survey of students, focusing on three aspects (pragmatic, hedonic, and technical).

MOTS-CLES : Évaluation entre pairs, jeux sérieux, grille d'évaluation critériée, enseignement supérieur.

KEYWORDS: reviewers, serious games, criterion-based evaluation rubric, higher education.

1 Introduction

Les jeux sérieux (*serious games*, SGs), dont la plupart ont pour ambition de prodiguer un entraînement ou un apprentissage à travers l'activité ludique, combinent les termes « serious » et les « ressorts ludiques » du jeu vidéo (Alvarez, 2007). Ils sont notamment utilisés dans le domaine de la gestion, de

Atelier « Apprendre la Pensée Informatique de la Maternelle à l'Université », dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH), Fribourg, Suisse, 07 juin 2021, pages 76–86,

la santé, de la sensibilisation à diverses thématiques comme celle de l'environnement ou encore, dans notre cas, de l'enseignement-apprentissage universitaire. Dans le cadre d'un projet tuteuré, des étudiants en informatique ont dû concevoir un SG, puis évaluer les SG conçus par leurs pairs. Pour que cette évaluation soit centrée sur l'apprenant et que cela semble plus raisonnable, nous nous posons la question : comment rendre l'évaluation des jeux sérieux plus objective par des jeunes développeurs de manière à ce qu'elle leur soit accessible ?

Dans la littérature, il existe déjà plusieurs approches concernant l'évaluation des jeux sérieux présentant des limites que nous allons identifier. Certaines, par exemple, ne se concentrent que sur une des deux composantes du jeu sérieux. Selon Ávila-Pesántez Rivera et Alban (2017), l'évaluation des jeux sérieux consiste à s'assurer avant tout que les objectifs pédagogiques ont été atteints. Au contraire, le modèle GameFlow (Sweetser et Wyeth, 2005) se concentre surtout sur le plaisir procuré par les jeux tels que la concentration, le défi du jeu, le sentiment de contrôle, l'immersion et l'interaction sociale. Par ailleurs, le game experience questionnaire (GEQ) (IJsselsteijn, de Kort, Y et Poels, K, 2013) a, quant à lui, pour objectif d'évaluer l'expérience de jeu par le moyen concret de scores attribués à des composantes tels que l'immersion (la capacité du jeu à immerger le joueur de façon sensorielle et imaginaire), la compétence (la capacité du jeu à permettre au joueur de se sentir compétent), les ressentis positifs et négatifs éprouvés par les joueurs. Or, le questionnaire AttrakDiff (Hassenzahl, Burmester et Koller, 2003), par exemple, qui est un outil d'évaluation de l'expérience utilisateur pour les logiciels, met en avant qu'au-delà de la qualité hédonique du produit, il est important également de s'intéresser aux qualités pragmatiques/utilitaires de ce dernier.

Ces approches présentent un déséquilibre en ne choisissant d'évaluer que les contenus d'apprentissage ou que les expériences divertissantes. Par ailleurs, les méthodes d'évaluation existantes fixent des critères qui ne sont par conséquent pas toujours adaptés aux besoins réels d'évaluation d'un produit en particulier par des évaluateurs spécifiques. Notamment dans le contexte des environnements informatiques pour l'apprentissage humain (EIAH), l'évaluation des jeux sérieux devrait non seulement prendre en compte l'apprentissage des apprenants, mais aussi leur expérience vécue. C'est la raison pour laquelle nous avons fini par nous intéresser à la méthode d'évaluation : une grille d'évaluation critériée, préconisée par Berthiaume et Rege Colet (2013) en Sciences de l'éducation, ce qui rend l'évaluation plus souple et contextualisée tout en réduisant la subjectivité dans l'évaluation des apprentissages.

Après une rapide introduction, nous présenterons, dans la section 2, le contexte d'étude qui nous a amené à vouloir concevoir et appliquer une grille d'évaluation. La section 3 détaille, quant à elle, la méthode utilisée pour la création de la grille d'évaluation, et les démarches de notre étude. Ensuite, nous présenterons le résultat de la grille dans la section 4 et nous ferons une conclusion dans la section 5.

2 Evaluation des jeux sérieux scientifiques dans l'enseignement supérieur

2.1 Contexte d'étude

L'étude prend place en deuxième année de Diplôme Universitaire de Technologie (DUT), spécialité informatique, sur une promotion de 80 étudiants. Ces derniers ont déjà des connaissances en développement informatique, mais n'ont encore jamais réalisé de projet d'envergure. Le projet représente 100 heures de travail par étudiant, étalées de septembre à décembre, par groupe de 3 ou 4. Les objectifs pédagogiques sont d'obtenir une première expérience de travail collaboratif sur trois

mois, de découvrir les différentes étapes d'un projet informatique, et de mettre en œuvre les connaissances et compétences acquises dans les autres modules. Pour favoriser la motivation des étudiants et s'ancrer dans un univers qui leur est familier, ce projet vise à la production d'un jeu sérieux dit « scientifique » car il a pour vocation de transmettre au joueur une notion scientifique. Cette notion est présentée par des doctorants en sciences humaines et sociales (en l'occurrence : sciences de l'éducation, psychologie sociale et archéologie), sans connaissance particulière de l'informatique, qui présentent leur sujet de recherche au cours d'une rencontre introductive. Après en avoir pris connaissance, les étudiants en informatique sont libres d'élaborer leurs propres concepts de jeu dans un prototype. Leurs travaux se décomposent en trois temps :

1. après la première rencontre avec un(e) doctorant (e) (durant laquelle le ou la doctorant(e) expose l'un des fruits de sa recherche, par exemple, le sujet sur l'archéologie concerne l'ensemble des méthodes qui permettent de mettre en lumière les traces du passé de l'humain), il y a la constitution d'un dossier de conception par petits groupes ;

2. une fois le dossier validé par l'enseignant-coordinateur, les étudiants commencent la réalisation des jeux sérieux scientifiques ;

3. démonstration des jeux auprès des enseignants et des étudiants. Tests des jeux et évaluations effectués par leurs pairs.

Nous avons choisi que l'évaluation soit effectuée par leurs pairs, parce que cela favorise les échanges entre « évaluateurs » et « évalués », ce qui rend les évaluations plus transparentes. De plus, cela permet aux étudiants de mettre en perspective leurs propres travaux avec les travaux des autres, et ainsi de mieux comprendre leurs propres notes. Néanmoins, une évaluation de la part de leurs pairs implique nécessairement une subjectivité importante : les étudiants ont tendance à s'appuyer sur leurs critères personnels et éventuellement à évaluer en fonction des affinités au sein du groupe. Pour une évaluation plus juste et plus objective, nous avons donc opté pour la conception d'une grille critériée. Encore fallait-il que cette grille présente des critères compréhensibles et que l'échelle proposée soit facilement utilisable. Dans cette optique, les attentes et les modes d'évaluation de ce cours ont été communiqués dès le début, la grille d'évaluation a été soumise aux observations et à l'approbation des étudiants.

3 Méthode

3.1 Application de la grille d'évaluation

Une grille d'évaluation critériée permet de réduire des biais de correction comme la fatigue, le stéréotype, la préférence personnelle, et les différents états d'esprit lors de l'évaluation (Berthiaume et Rege Colet, 2013). L'utilisation d'une grille permet par ailleurs, d'une part de clarifier les attentes liées à l'évaluation et, d'autre part, de permettre de donner un feedback détaillé aux étudiants. Berthiaume et Rege Colet (2013) mentionnent la nécessité, pour le développement de la grille, de clarifier dans un premier temps les attentes de l'évaluation, puis d'identifier les critères d'évaluation et d'établir différents niveaux de performance. Ils permettent aussi d'utiliser une échelle à trois niveaux de performance : excellent, acceptable et inacceptable.

3.2 Démarche

Pour construire la grille d'évaluation, nous nous sommes inspirés du modèle d'évaluation AttrakDiff qui a pour objectif d'évaluer l'expérience utilisateur par la qualité pragmatique et la qualité hédonique (Hassenzahl, Burmester et Koller, 2003). Nous considérons que les deux qualités permettent de donner un aperçu de l'expérience vécue des apprenants. Pour que la démarche soit bien centrée sur l'apprenant, nous avons demandé aux étudiants de choisir parmi les 36 critères les 15 qu'ils considèrent comme les plus pertinents pour évaluer leur jeu. La liste des critères est issue d'une revue de littérature sur l'évaluation des serious games (Liu, en cours). A partir de leurs réponses, nous avons pu valider une liste de 12 critères en regroupant les critères similaires et en s'assurant de leur cohérence globale. Nous avons ensuite ajouté un troisième aspect, dit technique, visant à évaluer la bonne maîtrise des outils et technologies dans la production du prototype du jeu sérieux (voir 4.1).

Lors de la phase d'évaluation, après une présentation de 3 minutes pour chacun des 24 prototypes du jeu, l'enseignant-coordinateur a distribué de façon aléatoire les évaluations aux pairs, de sorte que chaque prototype soit évalué au moins 4 fois. Il a été demandé aux évaluateurs d'installer les jeux et de les tester. Limesurvey a été utilisé pour collecter les résultats de l'évaluation, et une place a été réservée aux justifications plein texte des décisions des étudiants-évaluateurs.

4 Résultats

4.1 Grille d'évaluation critériée

Nous avons développé une grille d'évaluation des jeux sérieux qui est issue des 12 critères validés (voir Tableau 1), qui contient six critères pour l'aspect hédonique, six critères pour l'aspect pragmatique et douze critères pour l'aspect technique. Pour encourager les étudiants, nous avons utilisé une échelle à trois niveaux de performance adaptée au contexte, à savoir : excellent, satisfaisant et insatisfaisant dans ce contexte.

Tableau 1: Grille d'évaluation critériée pour les jeux sérieux scientifiques

Grille d'évaluation des jeux sérieux scientifiques					
Aspect d'évaluation	**Module**	**Critère**	**Excellent (1 pt)**	**Satisfaisant (0,5 pt)**	**Insatisfaisant (0 pt)**
Aspect hédonique		**Ludique**	Le jeu emploie divers éléments du jeu (points, badges, temps limite, bonus et classements etc.). Ils sont employés de manière pertinente et reflètent une bonne maîtrise de la ludification.	Quelques éléments du jeu sont employés, le jeu reflète une maîtrise de la ludification.	Le jeu n'emploie aucun élément de jeu.
		Curiosité	Le jeu suscite de la curiosité chez le joueur et l'encourage à	Le jeu encourage par moments le joueur à	Le jeu n'encourage pas le joueur à découvrir de

			explorer continuellement de nouvelles choses.	découvrir de nouvelles choses.	nouvelles choses.
		Interactivité	Le jeu prend constamment en compte les informations fournies par le joueur et implique une participation active de ce dernier.	Le jeu implique la participation du joueur.	Le jeu n'implique pas la participation du joueur.
		Esthétique	Les représentations visuelles correspondent au thème du jeu. Les interfaces graphiques sont de très bonne qualité et agréables à regarder.	Les interfaces graphiques sont agréables à regarder.	Les interfaces graphiques ne sont pas agréables à regarder.
		Jouabilité	Les structures, les règles de jeu, la narration, le jeu et la façon dont le joueur s'approprie les possibilités du jeu sont bien articulés.	La structure du jeu, les règles, la narration, le jeu et la manière dont les joueurs s'approprient les possibilités du jeu sont articulés, mais il manque un peu de cohérence.	La structure du jeu, les règles, la narration, le jeu et la manière dont les joueurs s'approprient les possibilités du jeu ne s'accordent pas.
		Satisfaction	Le joueur a un sentiment relativement plus profond et plus durable que le plaisir éprouvé au moment de jouer.	Le joueur ressent du plaisir au moment de jouer.	Le joueur ne ressent pas le plaisir de jouer.
Aspect pragmatique		**Objectif d'apprentissage clair**	L'objectif d'apprentissage du jeu est clairement annoncé. L'objectif d'apprentissage est parfaitement	L'objectif d'apprentissage du jeu est annoncé. L'objectif d'apprentissage est présent dans le jeu.	L'objectif d'apprentissage n'est pas présent dans le jeu.

			fusionné avec les missions du jeu.		
		Correspondance par rapport aux besoins des apprenants	Le jeu tient compte des besoins d'apprentissage du public visé. Les missions/défis proposés dans le jeu sont alignés avec les objectifs d'apprentissage visés.	Le jeu a pris en compte les besoins d'apprentissage du public visé.	Le jeu n'a pas pris en compte les besoins d'apprentissage du public visé.
		Facilité d'utilisation	On s'approprie le jeu facilement. Le joueur arrive sans difficulté à comprendre et à apprendre comment jouer.	Le joueur arrive à comprendre comment jouer au jeu, mais cela demande une charge de travail élevée.	Le joueur n'arrive pas à comprendre comment jouer.
		Acceptabilité	La mission du jeu est facilement comprise et est naturellement acceptée par le joueur.	La mission du jeu est comprise par le joueur.	La mission du jeu n'est pas comprise par le joueur.
		Réflexivité	Le jeu est capable de faire réfléchir le joueur sur les objectifs d'apprentissage annoncés.	Le jeu fait réfléchir le joueur de façon globale mais sans se fixer sur les objectifs d'apprentissage annoncés.	Le jeu ne déclenche aucune réflexion.
		Efficacité	Le jeu permet de développer des connaissances et/ou des compétences, et permet d'évaluer la maîtrise des objectifs d'apprentissage et informe sur la progression des apprenants.	Le jeu permet d'évaluer la maîtrise des objectifs d'apprentissage.	Le jeu ne permet pas d'évaluer la maîtrise des objectifs d'apprentissage.
Aspect technique	**Documentation**	**Outils**	Un outil de commentaire de code a été	Un outil de commentaire de code a été utilisé	Aucun outil de commentaire de

			pleinement utilisé pour générer un dossier et un site web.	pour générer une documentation.	code n'a été utilisé.
		Exhaustivité documentation code	Tout le code (fichier, fonction, variables critiques) est commenté.	Le code est globalement commenté.	Le code n'est pas suffisamment commenté.
		Qualité documentation code	Les commentaires de code sont très clairs et les règles de nommage sont clairement établies.	Les commentaires de code sont clairs et l'orthographe est soignée.	Les commentaires de code ne sont pas clairs et l'orthographe n'est pas soignée.
	GIT	**Organisation**	Les fichiers sources suivent une organisation et des règles de nommage clairement établies.	Les fichiers sources sont tous partagés et organisés.	Certains fichiers sources ne sont pas partagés ou leur organisation laisse à désirer.
		Versioning	La granularité des commits est pertinente et les commits sont clairement et systématiquement renseignés.	La granularité des commits n'est pas pertinente ou les commits ne sont pas clairement et systématiquement renseignés.	La granularité des commits n'est pas pertinente et les commits ne sont pas clairement et systématiquement renseignés.
		Procédure	La procédure présentée a été scrupuleusement suivie, validation comprise.	La procédure présentée a été scrupuleusement suivie.	La procédure présentée n'a pas été scrupuleusement suivie.
	GITLAB	**README**	Le fichier présente le projet, contient au moins une capture d'écran, les liens vers le téléchargement,la documentation ainsi que les éventuelles instructions d'installation et de lancement.	Le fichier présente le projet.	Le fichier ne présente pas le projet.

		WIKI ou description .md	Le wiki/Description.md développe : - sous-ensemble des objectifs pédagogiques que vous avez choisis ; - description sommaire du jeu (genre son type) ; - actions du joueur ; - informations que le jeu renvoie au joueur.	Le wiki/Description.md présente sommairement : - sous-ensemble des objectifs pédagogiques que vous avez choisis ; - description sommaire du jeu (genre son type) ; - actions du joueur ; - informations que le jeu renvoie au joueur.	Le wiki/Description.md ne présente pas : - sous-ensemble des objectifs pédagogiques que vous avez choisis ; - description sommaire du jeu (genre son type) ; - actions du joueur ; - informations que le jeu renvoie au joueur.
		Tâches	Milestones et issues sont renseignées et utilisées pour définir toutes les fonctionnalités techniques principales de l'application	Milestones et issues sont renseignées.	Milestones et issues ne sont pas renseignées.
	Réutilisation	**Documentation**	La documentation explique tous les détails de l'installation	La documentation donne la procédure d'installation	La documentation ne décrit pas l'installation
		Installation	L'application permet une installation rapide et aisée.	L'application permet une installation.	L'application ne permet pas une installation.
		Exécution	L'application permet une exécution rapide et aisée.	L'application permet une exécution.	L'application ne permet pas d'exécution.
			TOTAL=/24 points	**TOTAL=/12 points**	**TOTAL=/0 points**

	TOTAL général=/24 points	

4.2 Analyse des résultats

Les résultats d'évaluation sont calculés à partir de l'échelle à trois niveaux de performance, à savoir : excellent (1 pt), satisfaisant (0,5 pt) et insatisfaisant (0 pt) concernant les trois aspects : hédonique, pragmatique et technique. Afin de mieux comprendre la divergence des notes de chaque aspect d'évaluation, nous allons analyser l'écart-type. La Figure 1 ci-dessous montre chaque point correspondant à un moyen d'évaluation de prototype du jeu sérieux. La quatrième colonne contient la moyenne des trois aspects et leur moyenne totale. Nous pouvons constater que les évaluations des aspects hédoniques et pragmatiques sont plus dispersées que celles de l'aspect technique. Nous avons fait deux hypothèses : premièrement, l'aspect technique s'appuie sur des connaissances et des compétences acquises par les étudiants pendant leur formation, donc il est plus évident à évaluer pour eux.

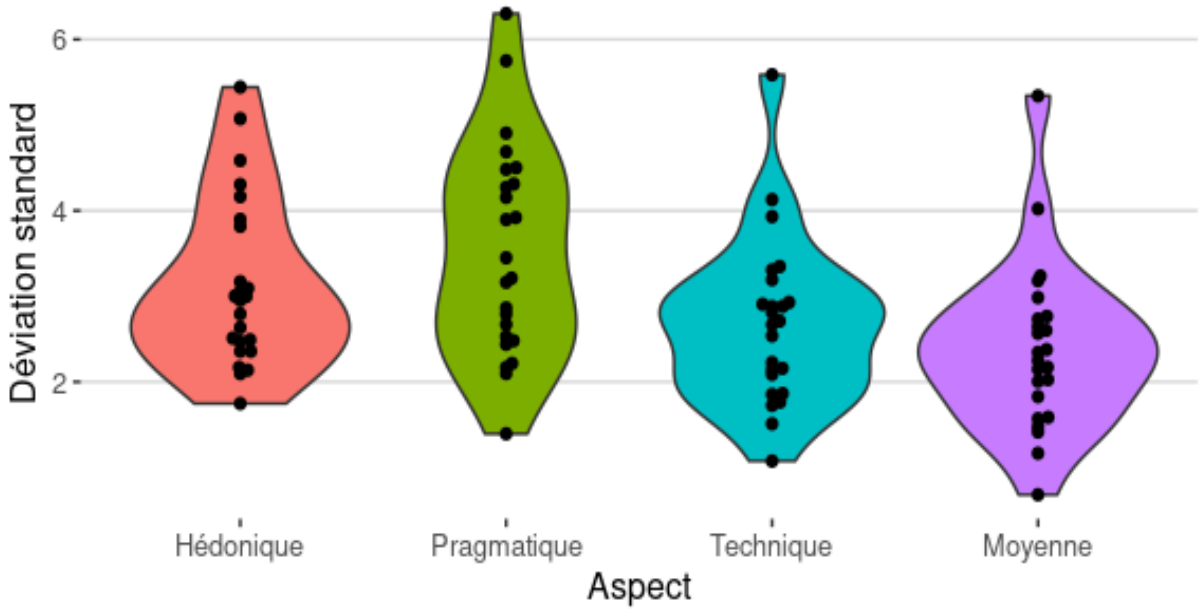


Figure 1: Analyse de l'écart-type

Deuxièmement, les critères des aspects techniques sont assez factuels, alors que les deux autres aspects font intervenir des perceptions subjectives. Par exemple, pour l'évaluation de l'esthétique du prototype A, deux évaluateurs portent un avis relativement différent : quand l'un utilise des modalisateurs positifs : « le jeu est honnêtement beau, l'interface est bien faite », l'autre considère que « son design est correct ». Sur d'autres prototypes, les avis restent similaires. Par exemple, cinq évaluateurs sur sept pensent que l'objectif d'apprentissage du prototype B est plutôt clair : « le jeu est facile à comprendre, on sait tout de suite ce qu'il faut faire » ; « le concept rapidement compris, objectif clair » ; « il respecte les objectifs d'apprentissages » ; « le jeu est facile à prendre en main, la mission est assez claire »; « la mission est facile à comprendre ». Il y a donc une subjectivité naturelle qui dépend des prototypes du jeu, et qui est bien démontrée au travers de la divergence des évaluations. Nous avons combiné les notes d'évaluations de trois aspects pour obtenir la moyenne finale (quatrième colonne), qui semble réduire certaines subjectivités selon nous.

Enfin, la Figure 2 montre l'écart-type des moyennes pour chaque prototype du jeu en fonction de la moyenne. Nous avons pu observer que plus la moyenne est haute, plus la déviation est basse. Nous

faisons l'hypothèse qu'il existe une réticence chez les étudiants-évaluateurs à rédiger de mauvaises évaluations et cela, y compris pour un prototype peu réussi. C'est la raison pour laquelle pour un « prototype réussi », les étudiants sont amenés à mettre une bonne note, alors que pour un « prototype peu réussi », certains distribuent des notes plus objectives que les autres, ce qui explique ce résultat : plus la note est basse, plus l'écart-type est grand.

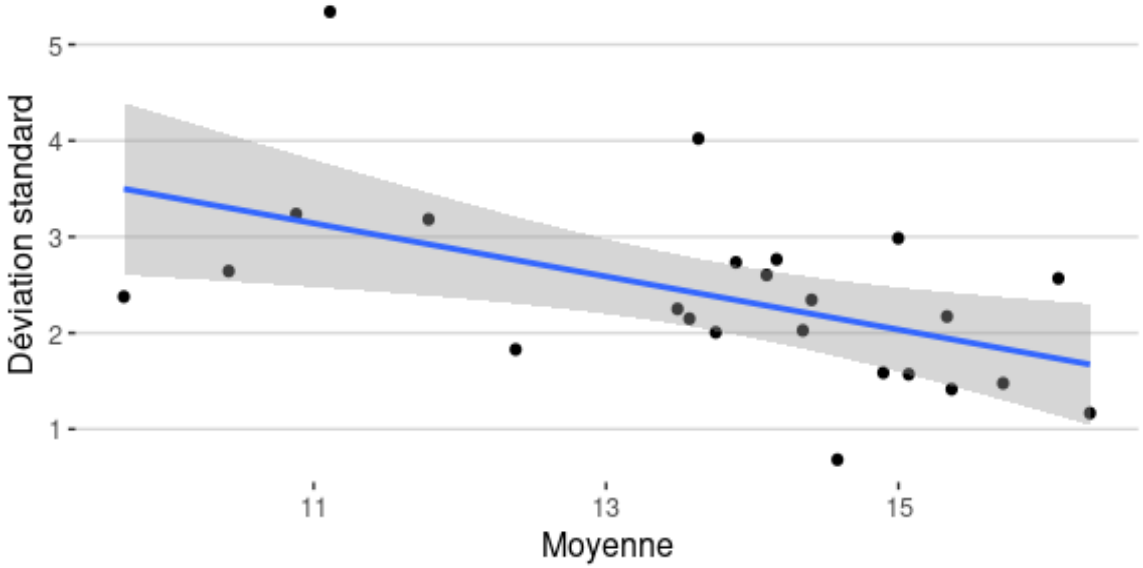


Figure 2: Corrélation entre l'écart-types et la moyenne

Pour conclure, nous avons manuellement vérifié que les notes issues de ce processus d'évaluation étaient bien justes, dans le sens où aucun prototype n'obtient une note inférieure à un prototype de moins bonne qualité, et où les écarts de notation reflétaient bien une réalité dans le travail accompli. Nous n'avons trouvé aucune note à corriger manuellement du point de vue de l'enseignant-coordinateur. Nous estimons cependant que les notes obtenues par ce processus sont globalement supérieures à celles qui auraient été données par un enseignant-coordinateur.

5 Retours d'expérience et conclusion

Cet article présente une méthode d'évaluation des jeux sérieux ainsi que l'application d'une grille d'évaluation critériée pour évaluer les travaux des étudiants dans une formation informatique. Bien que l'expérience de jeu soit très variée selon les différents apprenants-évaluateurs, la méthode proposée permet de donner un aperçu global sur les apprentissages et l'expérience vécue des apprenants au regard du développement et de l'expérience du jeu. La grille développée porte sur trois grands aspects : hédonique, pragmatique et technique. Les critères pourraient être modifiés en fonction des objectifs d'apprentissage visés dans chaque contexte. La grille proposée pourrait être également généralisée pour l'évaluation d'autres types de jeux.

L'enseignement d'aujourd'hui devrait être centré sur l'apprenant et son apprentissage. Pour citer Rege Colet et Berthiaume (2013, p. 242), « les pratiques d'enseignement invitent à articuler les activités d'enseignement avec les activités d'apprentissage et les modalités d'évaluation ». L'évaluation par les pairs à l'aide d'une grille conçue en collaboration avec les utilisateurs de cette dernière, semble être un moyen plus objectif pour évaluer l'apprentissage des étudiants car les évaluations se basent sur des critères définis. Par ailleurs, les étudiants disent avoir apprécié la méthode d'évaluation lors de la discussion entre l'enseignant-coordinateur et les étudiants, parce que la grille d'évaluation les rassure quant à l'évaluation, ce qui leur permet de travailler plus sereinement.

Références

Alvarez, J. (2007). *Du jeu vidéo au jeu sérieux. Approches culturelles, pragmatique et formelle*. Thèse de doctorat, Université de Toulouse-le-Mirail, France.

Lavigne, M. (2013). Pertinence et efficacité des serious games. Enquête de réception sur neuf serious games. *Revue des Interactions Humaines Médiatisées (RIHM)*, Europia.

Ávila-Pesántez, D., Rivera, L. A. et Alban, M. S. (2017). Approaches for Serious Game Design: ASystematic Literature Review. *Computers in education journal*, 8(3), pages 1-10.

Sweetser, P., & Wyeth. P. (2005). GameFlow: A Model for Evaluating Player Enjoyment in Games. ACM Computers in Entertainment. 3(3), 1-24.

IJsselsteijn, W. A., de Kort, Y. A. W., & Poels, K. (2013). The Game Experience Questionnaire. Eindhoven : Technische Universiteit Eindhoven.

Hassenzahl, M., Burmester, M., & Koller, F. (2003). AttrakDiff: Ein Frage-bogen zur Messung wahrgenommener hedonischer und pragmatischer Qualität. *In J. Ziegler, & G. Szwillus (Eds.), Mensch & Computer 2003*. Interaktion in Bewegung, pages 187–196. Stuttgart, Germany: B.G.Teubner.

Berthiaume, D. et Rege Colet, N. (2013). Chapitre 18. Comment développer une grille d'évaluation des apprentissages ? Dans Berthiaume D. et Rege Colet N. (dir.), *La pédagogie de l'enseignement supérieur : repères théoriques et applications pratiques / Tome 1 – Enseigner au supérieur*, pages 269-283. Berne : Peter Lang.

Rege Colet, N. et Berthiaume, D. (2013). Chapitre 16. Comment choisir des méthodes d'évaluation adaptées ? Dans Berthiaume D. et Rege Colet N. (dir.), *La pédagogie de l'enseignement supérieur : repères théoriques et applications pratiques / Tome 1 – Enseigner au supérieur*, pages 241-254. Berne : Peter Lang.

Liu, Y-D. (en cours). *Évaluer l'expérience d'apprentissage pour un nouveau design des Serious Games sur appareils mobiles*. Thèse de doctorat, Université de Strasbourg.

PseuToPy: Vers un langage de programmation naturel

Yassine Gader[1,2] Charles Lefever[1] Patrick Wang[1]
[1] ISEP – Institut Supérieur d'Électronique de Paris
10 rue de Vanves, Issy les Moulineaux, 92130 France
[2] ESPRIT – École Supérieure Privée d'Ingénierie et de Technologie
1, 2 rue André Ampère – 2083 – Pôle Technologique – El Ghazala, 2083 Ariana, Tunisie
{yassine.gader, charles.lefever}@eleve.isep.fr
patrick.wang@isep.fr

RÉSUMÉ

L'apprentissage de la programmation repose souvent sur la présentation de concepts algorithmiques puis sur leur mise en application avec un langage de programmation. Lorsqu'un langage de programmation textuel est utilisé (en opposition à un langage de programmation par blocs), l'apprentissage de la grammaire et de la syntaxe de ce langage peuvent constituer une difficulté supplémentaire pour l'apprenant. D'autre part, les langages de programmation reposent souvent sur un vocabulaire tiré de la langue anglaise. Cet aspect peut constituer un obstacle pour un public d'apprenants non anglophones puisqu'ils manipuleraient des mots et construiraient des instructions sans en comprendre leurs sens. Dans cet article, nous identifions les caractéristiques des langages de programmation qui peuvent être à l'origine de difficultés pour les apprenants. Puis, nous présentons la conception et l'implémentation de PseuToPy, un langage de programmation naturel qui permet la construction d'instructions proches de la langue naturelle de l'apprenant. Ce travail étant encore dans un état précoce, nous formulons l'hypothèse qu'un tel langage est utile dans l'apprentissage de la programmation et facilite la transition vers un langage comme Python.

ABSTRACT

PseuToPy : Towards a natural programming language

Learning to program frequently relies on the introduction of algorithmic notions followed by their application using a programming language. However, such text-based programming languages (as opposed to block-based ones) can be difficult to learn because of their unnatural grammar and syntax rules. Moreover, most programming languages use English words which can represent an added difficulty for non-English speakers who wish to learn to program as they might not understand the actual meaning of the instructions they are writing. In this paper, we identify the features of programming languages that can lead to learning difficulties. Then, we introduce the design and implementation of PseuToPy, a programming language that allows for instructions that resemble natural language sentences. Because this work is still in an early stage, we hypothesize that such a language can be useful when learning to program and can also ease the transition towards another programming language such as Python.

MOTS-CLÉS : Langages de programmation, Langage naturel, Grammaire formelle, Pseudocode.

KEYWORDS: Programming language, Natural language, Formal grammar, Pseudocode.

Atelier « Apprendre la Pensée Informatique de la Maternelle à l'Université », dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH), Fribourg, Suisse, 07 juin 2021, pages 87–95,

1 Introduction

L'apprentissage de la programmation présente de nombreuses difficultés. Bien souvent, cet apprentissage comporte deux composantes distinctes. D'un côté, il y a l'apprentissage des concepts importants de l'informatique permettant la conception d'algorithmes (par exemple, les variables ou les structures de contrôle). De l'autre, il y a l'apprentissage d'un langage de programmation et de ses règles de syntaxe et de sémantique afin de pouvoir exécuter ces algorithmes sur une machine.

Dans cet article, nous nous intéressons aux difficultés liées à l'apprentissage d'un langage de programmation textuel (en opposition aux langages de programmation par blocs tels que Scratch). Du Boulay (1986) avait déjà identifié cet obstacle qui oblige les apprenants à maîtriser la syntaxe et la grammaire d'un langage de programmation pour pouvoir écrire les programmes même les plus simples. La question de recherche que nous nous posons est la suivante : "Quelles sont les caractéristiques des langages de programmation *textuels* qui peuvent être à l'origine de difficultés d'apprentissage chez les débutants en programmation ?"

Dans cet article, nous cherchons à fournir des éléments de réponse à cette question de recherche avec, en Section 2, une description des caractéristiques grammaticales, syntaxiques, et sémantiques des langages de programmation et qui peuvent donc engendrer des difficultés d'apprentissage. En tenant compte de ces caractéristiques, nous travaillons sur la conception d'un langage de programmation *presque naturel*, intitulé PseuToPy, dont les objectifs sont justement de gommer un maximum ces caractéristiques. La conception et l'implémentation de ce langage de programmation (qui est toujours en cours de réalisation) est décrit en Section 3. Dans la Section 4, nous expliquons en quoi notre démarche se distingue des approches trouvées dans l'état de l'art portant sur l'apprentissage de la programmation. Enfin, nous concluons cet article en décrivant dans quelles situations nous comptons utiliser PseuToPy afin de fournir une réponse définitive à la question de recherche énoncée précédemment. Nous lançons également un appel à la communauté à contribuer à la conception de PseuToPy et de sa grammaire.

2 Caractéristiques d'un langage de programmation

2.1 Grammaire et syntaxe

Une grammaire formelle permet de spécifier les règles d'écriture des instructions d'un langage de programmation. Ces grammaires sont ensuite utilisées par un analyseur syntaxique afin d'assurer la conformité d'une instruction. Or, en spécifiant les instructions valides ou non, ces grammaires peuvent freiner l'apprentissage d'un langage de programmation de multiples façons.

Une première façon concerne l'aspect "peu naturel" des instructions écrites avec un langage de programmation. En effet, la résolution d'un problème passe souvent par une phase de conception d'un algorithme. Il semble évident qu'un apprenant va réfléchir et utiliser sa langue maternelle pour concevoir au préalable son algorithme. Une fois cette étape réalisée, il devra ensuite retranscrire cette réflexion en utilisant un langage de programmation dont les constructions et règles de grammaire sont probablement différentes de celles de sa langue naturelle. Cette étape nécessaire de retranscription peut donc être source de difficultés pour l'apprenant.

De plus, la grammaire d'un langage de programmation définit précisément la construction d'une

instruction. Par exemple, en Python, il est nécessaire d'avoir une valeur booléenne après le mot-clé `if`, quelle que soit la forme prise par cette valeur booléenne (un nom de variable booléenne, le résultat d'une comparaison, la valeur `True`, etc.). Cet apprentissage des règles de grammaire est indispensable, mais constitue donc un frein à l'élaboration de programmes, même les plus simples.

Enfin, les règles de grammaire ne laissent pas de place aux erreurs de syntaxe. En Python, l'oubli d'une indentation directement après une structure conditionnelle va générer une erreur de syntaxe. Il en va de même avec l'oubli d'un point-virgule à la fin d'une instruction en Java. Ces erreurs de syntaxe font partie des erreurs les plus communément retrouvées dans les programmes des apprenants. En effet, Denny et al. (2011) ont constaté la présence d'erreurs de syntaxe dans les rendus de près de 73% des étudiants considérés comme "faibles". De même, Altadmri and Brown (2015) ont analysé 37 millions de compilations en Java et ont constaté que l'erreur la plus fréquente, retrouvée près de 800 000 fois, correspondait à un oubli de parenthèses fermantes.

Ces caractéristiques montrent donc l'importance de mettre l'accent sur les règles de grammaire et de syntaxe d'un langage de programmation lors de son apprentissage. Les méthodes d'enseignement reposent souvent sur la présentation des concepts, mais d'autres approches mériteraient d'être étudiées. Par exemple, Portnoff (2018) suggère qu'il faudrait enseigner un langage de programmation comme une langue étrangère : en se focalisant justement sur la grammaire et l'orthographe de ce langage.

2.2 Sémantique

La sémantique d'un langage de programmation fait référence à la signification des mots et des symboles utilisés dans ce langage. Comme tout langage naturel, chaque langage de programmation donne aussi un sens particulier à l'ensemble des mots et symboles qu'il utilise, ce qui entraîne parfois des confusions chez l'apprenant.

Un exemple classique relevé par Qian and Lehman (2017) concerne le symbole "=". En effet, ce symbole en mathématiques est ordinairement utilisé pour représenter une égalité entre deux grandeurs. Mais le sens de ce symbole dans certains langages de programmation comme Java ou Python est complètement différent puisqu'il signifie l'affectation d'une valeur à une variable. Un autre exemple est celui des parenthèses, crochets, et accolades en Python qui permettent de construire respectivement des tuples, des listes, ou des ensembles ou dictionnaires. Un troisième exemple reprend les erreurs de syntaxe mentionnées précédemment : en Python, les indentations ont un réel sens dans la construction d'un algorithme puisqu'elles permettent de spécifier des blocs d'instructions d'un même niveau. On constate donc que ces indentations peuvent générer aussi bien des erreurs de syntaxe que des erreurs sémantiques. En plus de la syntaxe à maîtriser, l'apprenant doit donc aussi connaître le sens des différents symboles utilisés par le langage de programmation.

Une autre source de difficultés pour les apprenants résulte du fait que les langages de programmation les plus courants utilisent souvent un vocabulaire tiré de la langue anglaise. Or, en 2021, l'anglais est la langue maternelle d'environ seulement 5% de la population mondiale [1]. Ce nombre passe à 18% si l'on compte également les personnes pour qui l'anglais est une deuxième langue. Dès lors, comment pouvons-nous espérer qu'une personne non anglophone apprenne sans difficulté un langage de programmation dont elle ne comprendrait même pas le sens des mots ? D'ailleurs, une corrélation positive a été trouvée par Qian and Lehman (2016) entre le niveau d'anglais et les résultats obtenus dans un cours d'introduction à la programmation dispensé dans un collège en Chine. Pour remédier

1. https://en.wikipedia.org/wiki/List_of_languages_by_total_number_of_speakers

à ce problème, il faudrait que l'apprenant s'initie dans un premier temps à l'anglais, ou qu'on lui fournisse un langage de programmation dans sa propre langue natale.

Dans la suite de cet article, nous présentons notre approche qui consiste donc à concevoir et proposer aux apprenants un langage de programmation *naturel* et qui pourrait se décliner en plusieurs langues.

3 PseuToPy : un langage de programmation presque naturel

3.1 Conception et grammaire formelle du langage

Dans cet article, nous présentons le travail que nous menons actuellement sur PseuToPy, un langage de programmation destiné aux apprenants et présentant les caractéristiques suivantes : (1) la grammaire formelle de PseuToPy est conçue pour produire des instructions qui ressemblent à des phrases d'une langue naturelle sans pour autant être forcément grammaticalement correctes dans la langue cible, (2) plusieurs versions de la grammaire permettrait également de définir des instructions dans des langues autres que l'anglais, et (3) certaines instructions possèdent plusieurs syntaxes valides afin d'offrir plus de flexibilité à l'apprenant. Puis, PseuToPy propose aussi de convertir les instructions ainsi écrites en code Python équivalent. Avec ce langage, nous espérons faciliter l'apprentissage d'un langage de programmation en permettant aux apprenants d'écrire un algorithme dans un langage *presque* naturel, de visualiser leurs programmes en Python, et de les exécuter.

La conception de PseuToPy passe par plusieurs phases. La première consiste à identifier les instructions qu'un débutant pourrait être amené à écrire dans le cadre de son apprentissage de la programmation. Lorsque ces instructions sont identifiées, il s'agit ensuite de définir les règles de la grammaire formelle correspondante afin de spécifier le langage en cours de construction. Ce travail pour PseuToPy est encore en cours de réalisation ; en effet, nous cherchons à présent à trouver des formulations alternatives avec deux objectifs visés : le premier concerne la possibilité de construire des instructions proches du langage naturel cible, le second concerne la possibilité d'ajouter de la sémantique aux instructions en explicitant certains symboles pouvant générer de la confusion chez les apprenants. Enfin, lorsque la grammaire sera complètement spécifiée, la dernière étape de la conception de PseuToPy passe par l'implémentation de fonctions qui traduisent le code écrit en PseuToPy en du code Python équivalent.

Pour réaliser ce travail de spécification de la grammaire, et comme nous allons le présenter plus en détail dans la Section 3.2, nous avons pris la décision de nous baser sur la grammaire formelle du langage de programmation Python et de modifier ces règles en définissant de nouveaux symboles terminaux qui aideront ensuite l'apprenant à construire des instructions proches de sa langue naturelle. Ce choix est justifié par la nécessité d'écrire des instructions ayant une sémantique non ambiguë (ce qui est rendu possible grâce à la grammaire formelle) et par la possibilité de créer des instructions dont les constructions ressembleront au langage Python (puisque basées sur la même grammaire) avec l'espoir que cette ressemblance facilite la transition du langage PseuToPy vers Python.

L'approche que nous adoptons présente donc de multiples intérêts. Elle permet de désambiguïser certains symboles utilisés en Python, de générer des instructions proches du langage naturel, mais aussi de gérer de façon indépendante plusieurs langues naturelles dans un soucis d'inclusivité. Bien que PseuToPy reste un travail en cours de réalisation, nous formulons l'hypothèse que ces caractéristiques faciliteront l'apprentissage de ce langage par des apprenants débutant en programmation.

3.2 Implémentation du langage

Pour l'implémentation de PseuToPy, nous utilisons la bibliothèque Lark [2]. Cette bibliothèque permet de spécifier une grammaire en utilisant le formalisme *Extended Backus-Naur Form* (EBNF), et de procéder à une analyse syntaxique des instructions en respect des règles de grammaire spécifiées au préalable. Dans le cadre de PseuToPy, notre travail consiste donc à spécifier les règles de grammaire du langage que nous souhaitons créer puis de transformer l'arbre de syntaxe produit par Lark en un programme Python.

Pour ce qui est de la grammaire du langage, nous nous sommes basés sur la grammaire de Python 3.8 [3] à laquelle nous avons ajouté nos propres règles et mots-clés, et ce, dans la langue naturelle de notre choix (que ce soit par exemple en anglais ou en français). Une conséquence de cette décision est qu'il est donc possible d'analyser la syntaxe d'instructions écrites aussi bien en Python qu'en PseuToPy. Une autre conséquence est qu'il est donc possible de mélanger des instructions écrites en Python ou dans la langue naturelle cible puisqu'il n'y a qu'une seule et unique grammaire formelle.

Par exemple, la Figure 1 montre les modifications que nous avons apportées à la règle `assign` pour y ajouter une instruction en français permettant d'assigner une valeur à une variable. La première ligne indique le nom de la règle spécifiée. Les deux lignes suivantes permettent de définir cette règle en deux temps (le symbole "`|`" correspond à un "ou" logique) : soit pour spécifier une assignation en Python, soit pour spécifier une assignation utilisant des mots en français.

```
?assign:
  (testlist_star_expr ("=" (yield_expr|testlist_star_expr))*)
  | ("assigner" "à" testlist_star_expr ("la" "valeur" (yield_expr | testlist_star_expr))*)
```

FIGURE 1 – Modification apportée à la règle d'assignation de valeur à une variable.

Avec cette règle modifiée, les instructions utilisées en Figure 2 sont équivalentes comme l'illustrent les arbres de syntaxe identiques générés par Lark.

```
Instruction en PseuToPy:
assigner à ma_variable la valeur vrai
**********
Arbre de syntaxe généré:
file_input
  assign
    var ma_variable
    const_true
```

```
Instruction en Python:
ma_variable = True
**********
Arbre de syntaxe généré:
file_input
  assign
    var ma_variable
    const_true
```

FIGURE 2 – Arbres de syntaxe générés pour deux instructions équivalentes, l'une écrite en PseuToPy et l'autre en Python.

La conception des règles alternatives liées au langage naturel reste pour l'instant inachevée, car il nécessite un travail important de spécification. Par exemple, l'instruction donnée en Figure 2 utilise les mots-clés `assigner`, `à`, `la`, `valeur`, et `vrai`. Or, l'assignation d'une valeur à une variable

2. https://github.com/lark-parser/lark
3. La grammaire de Python 3.8 est disponible ici : https://docs.python.org/3.8/reference/grammar.html.

pourrait se formuler d'une autre façon, par exemple "`mettre ma_variable à vrai`". Quelle instruction devrions-nous autoriser ? Devrions-nous modifier la grammaire pour autoriser ces deux instructions ? Ces questions restent en suspens, et notre réponse initiale est de penser que, dans l'exemple ci-dessus, les instructions sont équivalentes et qu'il serait intéressant de pouvoir proposer plusieurs syntaxes pour une même règle. Cela peut d'ailleurs être fait en rajoutant simplement une autre possibilité à cette règle pour spécifier l'instruction précédente. Ainsi, la grammaire offrirait un peu plus de flexibilité dans les instructions que peut écrire un apprenant. Le travail réalisé pour la règle `assign` nécessite ensuite d'être aussi fait pour toutes les autres règles spécifiant des instructions qu'un apprenant débutant en programmation pourrait utiliser lors de son apprentissage.

Une fois ce travail d'extension de la grammaire réalisé, il restera un travail de développement illustré ci-après en Figure 3 et consistant à implémenter une fonction prenant en entrée une instruction écrite en PseuToPy et qui (1) procède à l'analyse syntaxique de cette instruction puis produit un arbre de syntaxe comme celui illustré en Figure 2 et (2) construit et retourne l'instruction Python équivalente à partir de cet arbre de syntaxe.

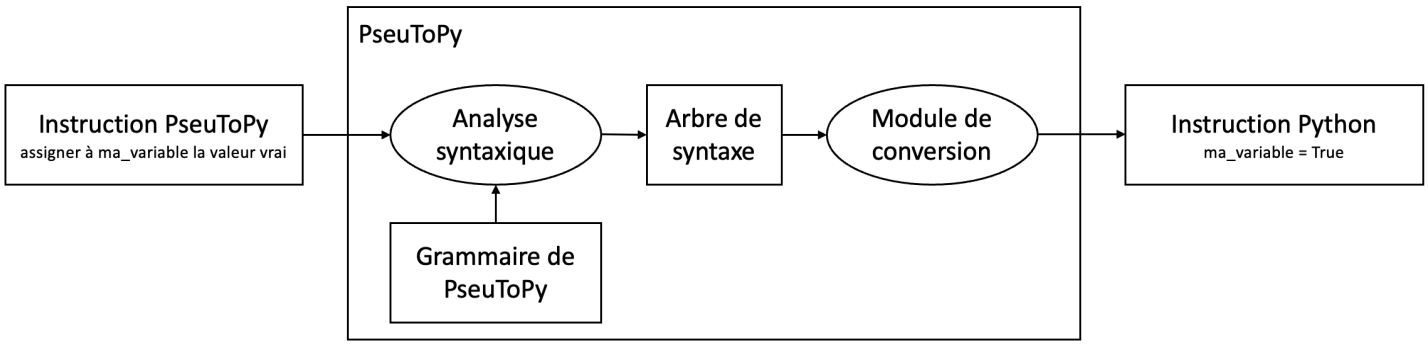


FIGURE 3 – Implémentation technique de PseuToPy.

Enfin, ce travail devra ensuite être répété pour les différentes langues naturelles que nous souhaitons proposer à la communauté d'apprenants et d'enseignants. Pour le moment, nous nous focalisons simplement sur le français et l'anglais en espérant plus tard ajouter d'autres langues. En guise d'illustration, voici les instructions de la Figure 2 écrites en arabe ou en chinois. Cette opportunité est d'autant plus intéressante qu'il est donc possible d'utiliser des caractères encodés en UTF-8 pour définir la grammaire de Lark et aussi d'utiliser des langages s'écrivant de droite à gauche.

```
Instruction en PseuToPy:
ضع x صحيح بقيمة
************************
Arbre de syntaxe généré:
file_input
  assign
    const_true
    var x
```

```
Instruction en PseuToPy:
设置 x 等于正确
************************
Arbre de syntaxe généré:
file_input
  assign
    var x
    const_true
```

FIGURE 4 – Exemples d'assignation de variables écrites en arabe et en chinois.

Le résultat de ce projet sera ensuite mis à disposition de la communauté sous deux formes différentes :

une bibliothèque Python qui sera disponible sur la plateforme PyPI [4], et une application Web proposant un éditeur permettant d'écrire des instructions en PseuToPy, de voir les instructions équivalentes en Python, et d'exécuter et visualiser le résultat de ces instructions.

4 Positionnement et état de l'art

La littérature en didactique de l'informatique présente de nombreux langages et environnements conçus spécifiquement pour l'apprentissage de la programmation. Brusilovsky et al. (1994) ont par exemple identifié trois approches utilisées pour l'enseignement de la programmation.

La première approche est qualifiée d'*incrémentale* et consiste à proposer un langage dans lequel l'apprenant construit au fur et à mesure les éléments de ce langage (par exemple DrScheme, dont le nouveau nom est désormais Racket (Findler et al., 2002; Felleisen et al., 2015)). Or, cette approche repose sur le paradigme de la programmation fonctionnelle et nécessite des connaissances préalables et n'est donc pas particulièrement adaptée à des débutants en programmation.

La seconde approche propose de simplifier un langage de programmation en n'en fournissant qu'une sous-partie adaptée aux besoins des apprenants. Par exemple, MiniJava (Roberts, 2001) propose une simplification syntaxique du langage Java, connu pour être verbeux. Cependant, le langage proposé n'étant qu'une sous-partie du langage initial, les différents symboles restent inchangés dans la grammaire, malgré qu'ils soient souvent à l'origine des erreurs les plus communément rencontrées en Java (Brown and Altadmri, 2017). Cette approche ne traite donc pas spécifiquement des difficultés d'apprentissage du langage de programmation et de ses règles de grammaire. Dans cette même optique de simplification, certains travaux cherchent à concevoir un environnement de développement adapté aux besoins des apprenants, comme par exemple BlueJ (Kölling et al., 2003) qui propose une interface permettant de construire un programme Java en définissant un diagramme UML de classe.

La dernière approche consiste à utiliser un langage construit spécifiquement pour l'initiation à la programmation. Récemment, les environnements de programmation par blocs comme Scratch (Maloney et al., 2010) ont été conçus pour focaliser l'attention de l'apprenant sur l'aspect algorithmique et limiter les erreurs de syntaxe en fournissant des blocs pré-construits. De nombreux travaux mettent ainsi en avant ce type d'environnements (Bau et al., 2015; Weintrop and Wilensky, 2017) mais très peu de langages de programmation textuels ont été conçus pour des débutants.

Toutes ces approches ne mettent pas nécessairement l'accent sur l'apprentissage des règles de syntaxe d'un langage de programmation. Dans le cas des environnements de programmation par blocs, cet aspect est même complètement mis de côté puisque des blocs prédéfinis sont utilisés pour remplacer les instructions textuelles. Bien que des travaux ont été menés pour étudier la transition entre un langage de programmation par blocs à un langage textuel (Bau et al., 2015; Weintrop and Holbert, 2017), aucune conclusion n'a pu être tirée sur l'utilité du premier pour l'apprentissage des règles de syntaxe et de grammaire du second.

PseuToPy se distingue donc de tous ces exemples en se focalisant sur les règles de grammaire des langages de programmation avec pour objectif de construire des instructions proches du langage naturel. À notre connaissance, peu de travaux ont été conduits sur cette problématique particulière. Nous pouvons par exemple mentionner MicroAlg (Gragnic, 2015) qui est un langage fonctionnel utilisant des mots de la langue française. Mais celui-ci repose sur un système de parenthèses similaire

4. Python Package Index : https://pypi.org/. Une version existe déjà, mais est en train d'être retravaillée en profondeur.

au langage Lisp, ce qui conduit à des instructions difficiles à lire pour des débutants en programmation, en plus d'être éventuellement déroutant. Nous pouvons également mentionner les travaux de Hermans ayant pour objectifs (1) de définir une *phonologie* des langages de programmation afin de déterminer précisément comment un programme peut se lire à voix haute (Hermans et al., 2018) et (2) de concevoir un langage de programmation (nommé Hedy) destinés à de jeunes enfants et dont les règles de grammaire présentent différents niveaux de complexité pour s'adapter à l'apprenant (Hermans, 2020). En particulier, les éléments syntaxiques comme les parenthèses ne sont introduits que dans les derniers niveaux pour ne pas surcharger les apprenants avec des syntaxes trop complexes dès le départ. PseuToPy se distingue de Hedy en proposant une compatibilité complète avec Python.

5 Discussion et travaux futurs

Dans cet article, nous avons analysé les caractéristiques syntaxiques et sémantiques des langages de programmation qui peuvent engendrer des difficultés lors de l'apprentissage de la programmation. Pour atténuer ces difficultés, nous proposons de construire un langage de programmation dont les instructions ressembleraient à des phrases issues d'une langue naturelle. Bien que nous travaillions en ce moment sur une version française, nous souhaitons proposer PseuToPy en plusieurs langages pour justement casser la barrière de la langue qui peut exister lors de l'apprentissage de la programmation.

À ce titre, la suite de nos travaux consiste en la construction de la grammaire française de PseuToPy pour laquelle nous souhaitons solliciter l'aide de la communauté francophone de chercheurs, d'éducateurs, et d'apprenants afin d'identifier les futurs mots-clés composant cette grammaire. Puis, nous étudierons l'utilisation de cet outil pour l'apprentissage de la programmation avec comme objectif de déterminer si PseuToPy facilite cet apprentissage du fait de sa ressemblance avec une langue naturelle.

Remerciements

Les auteurs de cet article souhaitent remercier Marin Godechot, Florian Comte, et Eric Sombroek pour leurs contributions initiales à la conception de PseuToPy, ainsi que Bastien Marais, Mathieu Valentin, Sébastien Viguier, et Laurent Yu pour leurs contributions à la nouvelle version de PseuToPy et pour leurs relectures de cet article.

Références

Altadmri, A. and Brown, N. C. (2015). 37 Million Compilations : Investigating Novice Programming Mistakes in Large-Scale Student Data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 522–527, New York, NY, USA. Association for Computing Machinery.

Bau, D., Dawson, M., and Pickens, C. S. (2015). Pencil code : Block code for a text world. In *Proceedings of the 14th International Conference on Interaction Design and Children*, IDC '15, pages 445–448, New York, NY, USA. Association for Computing Machinery.

Brown, N. C. C. and Altadmri, A. (2017). Novice Java Programming Mistakes : Large-Scale Data vs. Educator Beliefs. *ACM Transactions on Computing Education*, 17(2) :7 :1–7 :21.

Brusilovsky, P., Kouchnirenko, A., Miller, P., and Tomek, I. (1994). Teaching Programming to Novices : A Review of Approaches and Tools. In *World Conference on Educational Multimedia and Hypermedia*, Vancouver, BC.

Denny, P., Luxton-Reilly, A., Tempero, E., and Hendrickx, J. (2011). Understanding the syntax barrier for novices. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, pages 208–212, New York, NY, USA. Association for Computing Machinery.

Du Boulay, B. (1986). Some Difficulties of Learning to Program. *Journal of Educational Computing Research*, 2(1) :57–73.

Felleisen, M., Findler, R. B., Flatt, M., Krishnamurthi, S., Barzilay, E., McCarthy, J., and Tobin-Hochstadt, S. (2015). The Racket Manifesto. In Ball, T., Bodik, R., Krishnamurthi, S., Lerner, B. S., and Morrisett, G., editors, *1st Summit on Advances in Programming Languages (SNAPL 2015)*, volume 32 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 113–128, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Findler, R. B., Clements, J., Flanagan, C., Flatt, M., Krishnamurthi, S., Steckler, P., and Felleisen, M. (2002). DrScheme : A programming environment for Scheme. *Journal of Functional Programming*, 12(2) :159–182.

Gragnic, C. (2015). MicroAlg, un langage de programmation pour débutants - Les nouvelles technologies pour l'enseignement des mathématiques. *MathémaTICE*, 46.

Hermans, F. (2020). Hedy : A Gradual Language for Programming Education. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*, ICER '20, pages 259–270, New York, NY, USA. Association for Computing Machinery.

Hermans, F., Swidan, A., and Aivaloglou, E. (2018). Code phonology : An exploration into the vocalization of code. In *Proceedings of the 26th Conference on Program Comprehension*, ICPC '18, pages 308–311, New York, NY, USA. Association for Computing Machinery.

Kölling, M., Quig, B., Patterson, A., and Rosenberg, J. (2003). The BlueJ System and its Pedagogy. *Computer Science Education*, 13(4) :249–268.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., and Eastmond, E. (2010). The Scratch Programming Language and Environment. *ACM Transactions on Computing Education*, 10(4) :16 :1–16 :15.

Portnoff, S. R. (2018). The introductory computer programming course is first and foremost a *language* course. *ACM Inroads*, 9(2) :34–52.

Qian, Y. and Lehman, J. (2017). Students' Misconceptions and Other Difficulties in Introductory Programming : A Literature Review. *ACM Transactions on Computing Education*, 18(1) :1 :1–1 :24.

Qian, Y. and Lehman, J. D. (2016). Correlates of Success in Introductory Programming : A Study with Middle School Students. *Journal of Education and Learning*, 5(2) :73–83.

Roberts, E. (2001). An overview of MiniJava. *ACM SIGCSE Bulletin*, 33(1) :1–5.

Weintrop, D. and Holbert, N. (2017). From Blocks to Text and Back : Programming Patterns in a Dual-Modality Environment. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, pages 633–638, New York, NY, USA. Association for Computing Machinery.

Weintrop, D. and Wilensky, U. (2017). Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. *ACM Transactions on Computing Education*, 18(1) :3 :1–3 :25.

Une approche méta-design du jeu sérieux pour l'enseignement de l'informatique à l'école élémentaire

Xavier Nédélec[1], Bertrand Marne[2], Mathieu Muratet[3, 4], Karim Sehaba[5], Jean Lapostolle[6]
(1) ICAR UMR 5191, Université Lumière Lyon 2, France
(2) ICAR UMR 5191, Université Lumière Lyon 2, France
(3) Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
(4) INS HEA, 92150 Suresnes, France
(5) Université de Lyon. Université de Lyon 2. LIRIS, CNRS, UMR5205
(6) ICAR UMR 5191, Université Lumière Lyon 2
xavier.nedelec08@gmail.com, bertrand.marne@ens-lyon.fr,
mathieu.muratet@lip6.fr, karim.sehaba@liris.cnrs.fr, lapostolle.jean@gmail.com

RESUME

Le méta-design est une approche de conception participative dans laquelle des utilisateurs finaux sont rendus concepteurs des ressources, y compris durant les phases d'utilisation. C'est à partir de cette méthode que nous souhaitons développer notre projet autour du jeu sérieux Blockly Maze visant à permettre aux enseignants du primaire d'enseigner la pensée informatique en classe. Blockly Maze est un jeu de programmation par blocs centré sur l'apprentissage de l'algorithmique. Nous voulons que les enseignants puissent l'adapter en fonction de leurs besoins d'enseignement, des parcours d'apprentissage des élèves et des usages observés sur le terrain. À cette fin, nous nous inscrivons dans une optique de recherche collaborative orientée par la conception dans laquelle des enseignantes de cycle 3 sont parties prenantes du projet. Elles contribuent à part entière au processus d'adaptation du jeu sérieux Blockly Maze et à l'élaboration de méthodes de méta-design qui se veulent génériques.

ABSTRACT

A meta-design approach to learning games for teaching computer science in primary school. Meta-Design is a participatory design approach in which end users are enabled to design an artifact, even during the use stage. It is from this method that we wish to develop our project around the serious game Blockly Maze aiming to allow primary school teachers to teach computational thinking in class. Blockly Maze is a block programming game focused on learning algorithms. We want teachers to use it to teach computational thinking and therefore be able to adapt it to their teaching needs, to the learning paths of their students and to uses observed in the field. To this end, we are using a Design Based Research approach in which teachers in primary school are involved in the project. We believe they contribute adapting Blockly Maze to their needs and to provide methods for meta-design that are intended to be generic.

MOTS-CLES : recherche collaborative orientée par la conception, genèse instrumentale, praxéologie, méta-design, jeux sérieux, pensée informatique.
KEYWORDS: design based research, instrumental genesis, praxeology, meta-design, serious games, computational thinking.

Atelier « Apprendre la Pensée Informatique de la Maternelle à l'Université », dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH), Fribourg, Suisse, 07 juin 2021, pages 96–106,

1 Introduction

Meta-DeCT (*Meta-Design for Computer Science Teaching*) est un projet de recherche qui a pour objectif de favoriser la participation active des enseignants dans la conception et l'élaboration de jeux sérieux dans une optique d'enseignement de la pensée informatique. La méthode mise en œuvre pour élaborer ce projet s'appuie sur le méta-design (Fischer et al. 2004), une méthode intégrant l'utilisateur final dans les phases de conception et d'utilisation du jeu. Inscrit dans une démarche de recherche collaborative orientée par la conception (Sanchez et Monod-Ansaldi 2015), ce travail a pour objectif d'être un support d'aide à la conception pour les professeurs des écoles dans le but d'enseigner la pensée informatique à des élèves de cycle 3. Nous explorons plus particulièrement la contribution d'enseignants dans la (re)conception d'un jeu sérieux sur la pensée informatique. En étudiant comment intégrer des professeurs des écoles au processus de (re)conception.

Avant de développer notre approche, nous présenterons dans un premier temps notre positionnement face à certains travaux scientifiques sur la pensée informatique objet d'enseignement, sur les jeux sérieux d'apprentissage, ainsi que sur la genèse instrumentale et le méta-design. Dans un deuxième temps, nous présentons les choix méthodologiques de cette étude. Enfin, nous présenterons les premiers résultats de notre recherche sur le terrain.

2 Positionnement scientifique

2.1 La pensée informatique

De la programmation en LOGO impulsée dans les années 80 à la maîtrise d'un socle de compétences né dans les années 2000, décliné en B2i puis en C2i et dans lequel l'informatique est envisagée comme un produit utilitaire, les autorités publiques prennent peu à peu conscience des enjeux pour l'école (Baron et Drot-Delange 2016). Ainsi, la question d'un enseignement de l'informatique davantage disciplinaire a mis au jour la notion de pensée informatique. À présent, il est devenu nécessaire de comprendre ce qui se passe derrière l'écran de la machine (Baron et Bruillard 2008). Ainsi, les approches didactiques se développent et se diversifient avec Scratch (Resnick et al. 2009), l'informatique débranchée (Bell, Witten, et Fellows 1998) ou encore la robotique pédagogique (Spach 2017).

Qu'entend-on précisément par « *pensée informatique* » ? Jeannette Wing le rappelle (Wing 2006) « *la pensée informatique est un ensemble d'attitudes et d'acquis universellement applicables que tous, et pas seulement les informaticiens, devraient apprendre et maîtriser* » [1].

La réflexion de Wing a été étayée et certains ont décliné la pensée informatique en une série de compétences structurant le domaine en cinq parties qui permettant de résoudre un problème : (1) pensée algorithmique, (2) abstraction, (3) évaluation, (4) décomposition, (5) généralisation (Brunet et al. 2020).

La question de la pensée informatique fait toujours débat quant à une définition précise et acceptable pour pouvoir l'enseigner. Gilles Dowek estime qu'il faut nécessairement transmettre l'image d'une discipline la plus fidèle possible et propose ainsi de décliner l'informatique en quatre concepts : algorithme, langage, machine et information (Dowek 2011). Alors que d'autres considèrent

[1] *It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.*

qu'enseigner l'informatique avec le moteur Scratch permet de développer une pensée algorithmique (Tchounikine 2017).

En nous fondant sur tous ces travaux, nous proposons une description de la pensée informatique qui sera mise en œuvre dans notre étude. Nous pensons qu'il s'agit tout d'abord d'identifier un problème en le reformulant de façon simple et claire. Vient ensuite sa décomposition. Cela consiste à organiser et analyser les données : à savoir décomposer le problème en sous-problèmes pour proposer une amorce de solution par étapes. Cette phase de décomposition suggère également une phase d'abstraction du problème afin d'identifier un système de représentation dont le but est d'élaborer une modélisation du système. Cette modélisation s'appuie ensuite sur le choix des représentations des éléments du système, des modélisations et du choix des interactions. Une fois le problème abstrait, il s'agit d'exprimer une solution algorithmique en langage naturel puis de la traduire en langage machine. L'itération de la solution permet d'analyser les réussites et échecs. Il convient à cet égard d'identifier les problèmes rencontrés, de comparer la proposition à d'autres solutions. Le test est une épreuve qui ouvre ensuite sur la généralisation du problème à d'autres situations à savoir : (1) identifier les conceptions sous-jacentes au nouveau problème ; (2) reconnaître des cas identiques ou légèrement différents ; (3) généraliser l'algorithme pour des cas similaires.

Avec cette vision de la pensée informatique, notre ambition est de permettre son apprentissage grâce à un jeu sérieux exploitable par des enseignants non experts du domaine. Notre problématique est de développer des instruments adaptés et adaptables à leurs capacités et à leurs besoins.

Dans la sous-section suivante, nous inscrivons nos travaux dans l'état de l'art sur les jeux sérieux d'apprentissage.

2.2 Les jeux sérieux d'apprentissage

Le jeu sérieux d'apprentissage est un outil qui combine une activité hédonique avec un message « sérieux » (Djaouti 2011). Par conséquent, il peut être un élément motivationnel pour l'apprenant, car il intègre l'apprentissage dans un cadre ludique (Garris, Ahlers, et Driskell 2002).

Pour notre projet nous nous appuyons sur le jeu Blockly Maze (voir FIGURE 1), un Logiciel Libre développé par Google avec un code source disponible et réutilisable, sensiblement inspiré de Scratch (Resnick et al. 2009). Les ressorts et les mécaniques du jeu Blockly Maze visent un apprentissage de la programmation, de l'algorithmique et du déplacement dans l'espace par la résolution de problèmes. Blockly Maze est considéré comme un jeu sérieux d'apprentissage, car les problèmes à résoudre sont posés par le biais d'une interface ludique : il s'agit pour le joueur de faire sortir un avatar d'un labyrinthe grâce à de la programmation par blocs. Le jeu permet des essais multiples. Avec une interface simple à prendre en main, où les informations visuelles ne sont pas abondantes, le joueur est rapidement concentré sur la tâche à résoudre. Le joueur est mis au défi dans une optique motivationnelle où la progression se fait à travers dix niveaux, avec des contraintes progressives, par exemple, un nombre restreint de blocs à utiliser.

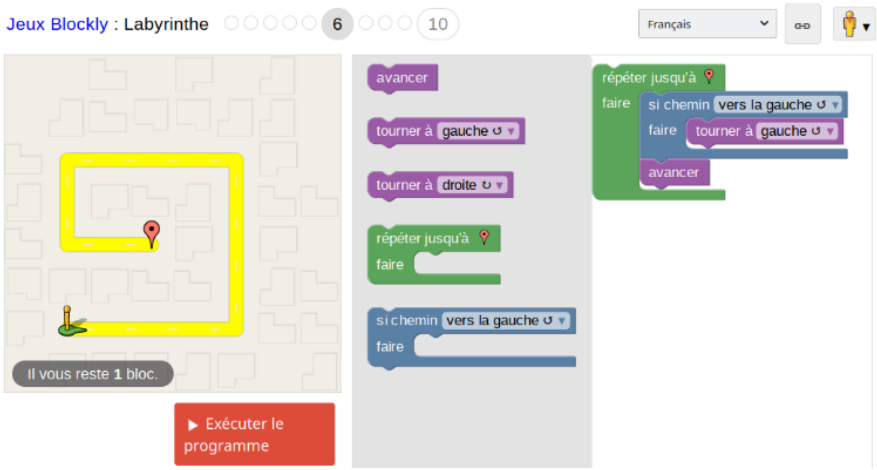


FIGURE 1 : capture d'écran du niveau 6 de Blockly Maze avec le bloc de programmation du parcours de l'avatar

Cependant, ces 10 niveaux nous paraissent trop restreints au regard des possibilités de travail sur la pensée informatique. Nous estimons qu'il faut diversifier l'offre de contenu, de parcours à travailler afin d'englober le plus largement possible la notion de pensée informatique et donc étoffer le nombre de niveaux. Nous estimons également que si nous voulons que l'enseignant soit partie prenante de la conception, il faut lui donner la possibilité de choisir, modifier et orchestrer ses niveaux dans des scénarios adaptés à ses objectifs pédagogiques et au niveau de l'élève. De plus, l'intégration d'un jeu sérieux d'apprentissage comme Blockly Maze pose des questions d'appropriation par les enseignants (Sardone 2018).

Ainsi, pour favoriser une large adoption et une bonne utilisation de Blockly Maze, nous fondons notre travail sur les concepts de genèse instrumentale et de méta-design.

2.3 Genèse instrumentale et méta-design

La genèse instrumentale génère un double processus de conception : l'instrumentalisation orientée vers l'artefact où l'utilisateur confère à l'objet de nouvelles attributions pour répondre à ses besoins, puis l'instrumentation centrée sur le sujet (l'utilisateur) où ce dernier utilise « *des schèmes* » déjà existants pour en comprendre l'usage (Rabardel 1995).

Le méta-design c'est « *concevoir le processus de conception* » (Fischer et al. 2004) : ce processus permet aux utilisateurs de résoudre eux-mêmes des problèmes rencontrés en cours d'utilisation d'un artefact en leur offrant la possibilité d'être impliqués dans le processus de conception initiale et dans une démarche de conception dans l'usage. Cette démarche de conception dans l'usage peut être suscitée grâce à l'*« underdesign* » (Fischer et al. 2004) : il s'agit de proposer des artefacts très modulaires non finalisés laissant la possibilité aux utilisateurs de les adapter selon leurs besoins et leurs usages. L'approche par le méta-design est donc une méthode d'instrumentation, voire d'instrumentalisation, elle est par ailleurs est étudiée et mise en œuvre dans le contexte des jeux d'apprentissage (Marne 2014; Vermeulen 2018).

Par conséquent, notre méthodologie de recherche s'inscrit dans une réciprocité entre le chercheur et l'enseignant-utilisateur où chacun se nourrit des apports de l'autre au cours des travaux de conception menés sur Blockly Maze.

3 Méthodologie de recherche

Nos travaux de recherche s'inscrivent dans une démarche de Recherche Collaborative Orientée par la Conception (ROC) (Sanchez et Monod-Ansaldi 2015). Notre but, est de favoriser l'utilisation d'un jeu sérieux d'apprentissage comme Blockly Maze par des enseignants du primaire pour enseigner la pensée informatique par une démarche de méta-design fondée sur l'underdesign.

Dans notre démarche de ROC, nous avons recours à un *« broker »* (Sanchez et Monod-Ansaldi 2015), c'est-à-dire une personne jouant un rôle d'intermédiaire entre les chercheurs (et également experts sur la pensée informatique et les jeux sérieux) et les enseignants (plus ou moins expérimentés).

Avec l'aide du « *broker* » et afin d'ancrer notre démarche de méta-design, nous envisageons un travail collaboratif en trois grandes phases. Dans un premier temps, nous recrutons des enseignants volontaires, puis en les questionnant sur leur parcours professionnel, sur leur expérience de l'informatique, et sur leur enseignement de la discipline en classe nous dressons un portrait de leur expérience professionnelle ainsi que de leur rapport à l'informatique et à son enseignement. Dans un second temps, nous nous servons de la construction d'une carte heuristique de la pensée informatique avec les enseignants pour explorer et comprendre quels sont leurs préconceptions, leurs usages et leurs besoins dans l'enseignement de ce domaine. Enfin, dans un dernier temps, nous élaborerons avec eux les modifications à apporter à Blockly Maze pour qu'il corresponde à leurs besoins pédagogiques et qu'il puisse être utilisé, et adapté par d'autres enseignants.

Lors de la seconde phase, inscrite dans le cadre d'une ROC, le recours à un *« broker »* permet de tenir un discours commun entre chaque partie : universitaires et enseignants. Concrètement, la praxéologie que nous souhaitons mettre en œuvre prend la forme de la conception d'une carte heuristique sur la pensée informatique élaborée avec les enseignants. Ici, le *« broker »* joue un rôle d'intercesseur entre les deux parties en aidant notamment les enseignants *« à accoucher »* leurs idées, leurs propositions dans le cadre défini par notre étude.

Cependant, la carte heuristique[2] n'est principalement qu'un outil de discussion, de réflexion de la notion de pensée informatique construite conjointement. L'objectif n'est aucunement de définir ce que doit être la pensée informatique, mais plutôt de servir d'élément discursif, nécessaire à l'étayage du concept et également d'être un support de conception et d'intercession entre le chercheur et le praticien. Cette carte sera en outre un support de formation pour les enseignants novices sur les questions de pensée informatique, un support d'indexation de compétences qui figureront dans les outils auteurs.

Dans la troisième phase de la recherche, nous allons utiliser un outil auteur de scénarios adapté au méta-design : APPLiq (Marne 2014). Les enseignants ont la capacité d'y agir directement sur le jeu en proposant un parcours de niveaux à l'apprenant. À ces niveaux sont indexés des prérequis en entrée et des objectifs en sortie, de sorte que l'enseignant choisisse quelles compétences l'apprenant va travailler.

L'agilité du dispositif rend possible de changer de niveau en fonction des performances du joueur. Il n'est pas nécessaire de suivre la progression dans le jeu de manière linéaire. Par exemple, si

[2]Voici un exemple de carte heuristique produite avec des enseignantes, selon les mêmes principes, dans une étude préliminaire de celle présentée ici : https://mycore.core-cloud.net/index.php/s/F4FyN9tbJJxOkKq/download (vu le 21/05/2021)

l'enseignant estime que l'apprenant remplit les prérequis et les objectifs d'un niveau, il peut lui attribuer de jouer sur un nouveau niveau. L'enseignant décide librement de suivre ou de modifier le parcours de chaque apprenant. APPLiq indique aussi à l'enseignant-utilisateur si des incohérences pédagogiques apparaissent au cours de la séquence de jeu (icônes triangulaires jaunes sur les liens entre les niveaux présentés dans la FIGURE 2). Ces incohérences sont liées à des prérequis n'entrant pas dans les objectifs des niveaux précédemment joués. Toutefois, pour respecter sa pleine liberté pédagogique, l'enseignant peut passer outre ce signal et confirmer son choix.

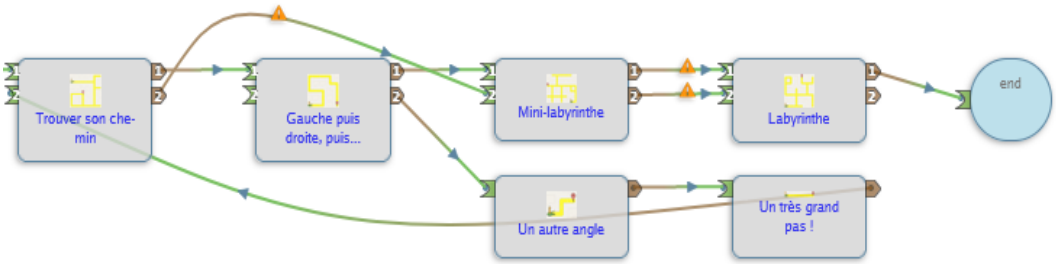


FIGURE 2 : capture d'écran d'un exemple de scénario de Blockly Maze modélisé dans APPLiq

Notre ambition est évidemment d'offrir la possibilité aux enseignants de prendre en main APPLiq afin qu'ils puissent choisir par eux-mêmes les niveaux du jeu travaillés par les apprenants. Même si son utilisation demeure assez simple, il faut néanmoins qu'il soit compris des utilisateurs non experts, comme pourraient l'être certains enseignants. Le recours à un *« broker »* jouant un rôle d'intermédiaire entre l'expert et le non-expert vise à favoriser la réussite du projet.

Toutefois, si APPLiq permet la construction de progressions non-linéaires dans le jeu, il ne permet pas de modifier l'organisation interne de chaque niveau. Pour cela, l'enseignant devrait donc intervenir directement sur le code source du jeu. C'est pourquoi, dans le contexte de notre démarche de méta-design, nous souhaitons aussi élaborer avec les enseignants volontaires un outil auteur de niveaux, complémentaire d'APPLiq, permettant de faciliter ces modifications.

Dans la section suivante, nous proposons les premiers résultats de nos travaux concernant la première phase (les entretiens de profilage et la prise en main de Blockly Maze) et le début de la seconde phase (la construction de la carte heuristiques).

4 Premiers travaux sur le terrain

4.1 Phase 1 : Recrutement et profilage des volontaires

C'est le « *broker* », lui-même enseignant de CM2, formé à l'ingénierie pédagogique dans le contexte des EIAH, qui a recruté les enseignantes volontaires de notre étude. Elles enseignent en cycle 3, l'une en CM1, les deux autres en CM2, toutes en classe ordinaire. Elles ont accepté de participer au projet, car elles y voient une possibilité de formation continue et une montée en compétences. Nous présentons plus loin des extraits de leurs entretiens au sujet de leur perception de l'informatique, de leur usage en classe et de leur retour à la suite de la prise en main du jeu Blockly Maze.

Dans un premier temps, afin de dresser un tableau descriptif du profil de chaque volontaire, nous avons mené un entretien individuel en évitant d'interférer ou biaiser certaines de leurs réponses. Nous les avons interrogées sur leur parcours professionnel, leur rapport personnel puis professionnel aux outils numériques et à l'informatique et sur leur enseignement de l'informatique.

Dans un second temps, les enseignantes ont pris en main le jeu Blockly Maze, sans information au préalable. À l'issue de cette partie qui a été enregistrée (audio et écran), un débriefing à chaud nous a permis de sonder leurs impressions et jugements à propos de Blockly Maze. Puis, quelques jours plus tard, lors d'un second débriefing, nous les avons questionnées plus en profondeur sur ce qu'elles avaient appris du jeu et sur les dispositifs didactiques qu'elles utiliseraient en classe.

La première des participantes impliquées dans le projet enseigne en CM1. Elle a 40 ans et a 16 ans d'ancienneté dans le métier. Même si elle a voulu au départ devenir professeure de portugais, elle a passé le concours de professeur des écoles et a toujours connu la classe. C'est une enseignante expérimentée qui connaît très bien le CM1 puisque cela fait dix ans qu'elle enseigne à ce niveau.

Les deux autres enseignantes sont quant à elles moins expérimentées. L'une, âgée de 52 ans, a trois ans d'expérience, l'autre, âgée de 43 ans, a sept ans d'expérience. Toutes deux découvrent le CM2. Elles ont aussi en commun d'avoir un parcours professionnel diversifié, notamment hors du professorat. L'un des points communs que nous identifions chez ces trois enseignantes concerne la perception et l'appréhension qu'elles ont de l'informatique. La vision qu'elles en ont est confuse et elles se sentent dépassées par l'objet « *ordinateur* » : « *Je me sens vieille un peu et c'est un outil avec lequel je ne suis pas très aguerrie* », *« je me sens perdue* », « *on va dire que mon fils de 13 ans en connaît beaucoup plus que moi et je fais souvent appel à lui* ».

Elles posent un regard distant et perçoivent l'informatique exclusivement comme un outil de travail professionnel destiné à la pratique de la bureautique pour préparer leurs cours et faire des recherches sur internet. L'école dans laquelle elles travaillent est équipée de TNI et tablettes. Elles utilisent le TNI pour la projection des cours, mais ne se servent pas du logiciel accompagnant son usage. Concernant l'enseignement de l'informatique en classe, seule l'enseignante de CM1 estime avoir déjà enseigné la discipline à ses élèves, mais dans une démarche d'informatique exclusivement utilitaire, comme se servir d'un traitement de texte.

Leur témoignage soulève une demande forte de formation à l'outil informatique : « *(...) c'est vrai que l'on n'a pas vraiment eu de formation sur l'informatique. C'est difficile, il faut chercher par nous-même* », « *j'ai envie d'être formée, de découvrir pour pouvoir enseigner l'informatique plus tard, de prendre les choses en main* ».

Seule l'enseignante de CM1 a reçu une formation en robotique « (…) *on nous a montré un robot-souris qui se déplaçait sur un tapis et qui avait été programmé par des élèves au préalable* ». Mais l'équipement est inexistant et les concepts (programmation, fonctionnement d'une machine) sous-jacents à cette formation n'ont pas été évoqués.

Nous pouvons alors nous interroger sur la formation : quel genre de formation souhaitent-elles suivre ? Une formation à l'usage du numérique ? Ou bien une formation destinée à l'apprentissage de l'informatique ? Les réponses recueillies montrent que cette distinction reste floue dans leur esprit.

À propos de la question sur la pensée informatique, les trois enseignantes disent ne jamais avoir entendu parler de cette notion. Deux d'entre elles connaissent Scratch, découvert à l'occasion du suivi de travail de leurs enfants au collège, mais nullement dans le cadre de leur métier. Enfin, elles énoncent clairement la difficulté de se projeter dans un enseignement de l'informatique, ce qui est évidemment compréhensible compte tenu de leur niveau de connaissance du domaine.

En respectant notre postulat de départ d'intégrer l'utilisateur à la conception de notre projet, la première étape a été de mettre entre les mains des trois enseignantes le jeu Blockly Maze. Le but était double : (1) Observer la manipulation du jeu et leur progression ; (2) Recueillir leurs impressions sur les dimensions ludiques et pédagogiques.

Pour les trois enseignantes, dès le niveau 1 les objectifs ludiques du jeu n'étaient pas identifiés : la manipulation des blocs ainsi que le glisser-déposer dans l'espace réservé à l'écriture du programme. Il fallait un guidage oral pour débloquer la situation. Il a fallu deux minutes pour réussir le niveau 1 pour deux enseignantes et plus de quatre minutes pour la troisième. Par la suite, pour deux d'entre elles, la progression s'est faite sans trop de difficultés. Il ne leur a fallu qu'entre quatre et cinq minutes pour réaliser les niveaux 2, 3, 4, 5. En revanche pour une enseignante de CM2 la progression a été plus compliquée. Quinze minutes ont été nécessaires pour réussir les niveaux 2, 3, 4, 5. Enfin, pour toutes les trois, le niveau 6 n'a pas été réussi. Nous avons arrêté au bout d'un quart d'heure d'essai.

À propos des objectifs pédagogiques, elles ont compris qu'il fallait faire cheminer l'avatar vers une cible (sortie du labyrinthe).

Nous en concluons, que la progression dans le jeu a été hésitante pour chacune d'elle. Programmer une séquence d'instructions a été correctement exécuté. L'utilisation de la boucle « *repeat until* » a nécessité de nombreux essais sur les niveaux 3, 4, 5, mais au niveau 6, aucune n'a su imbriquer ce bloc dans un autre bloc du programme. D'ailleurs, c'est aussi à ce niveau qu'elles ont toutes les trois été bloquées à cause notamment de l'apparition de la condition « *if/else* » qu'elles n'ont pas su employer.

Le jeu met évidemment en exergue des compétences de résolution de problèmes liées à l'algorithmique. À partir de leurs essais-erreurs, de leurs interrogations sur leurs échecs, de leurs réussites, nous avons observé leur démarche de raisonnement et les procédures employées pour résoudre les problèmes de chaque niveau. Même si la programmation par bloc leur est inconnue, le raisonnement adopté pour résoudre un niveau est souvent similaire : décomposer le cheminement de l'avatar, planifier les étapes du trajet, associer instructions et/ou imbrications d'instructions dans une boucle. Cependant, lors de leur propre retour d'expérience, aucune n'a su faire le lien entre le jeu et l'apprentissage de l'informatique ni perçu des contenus sous-jacents de la pensée informatique de chaque niveau.

En d'autres termes, personne n'a su dire quelles compétences d'informatique étaient travaillées. Nous expliquons ce constat tout simplement par le fait qu'aucune n'a conscience de posséder de connaissances ou compétences en informatique. Néanmoins, le point qui a soulevé leur attention, et que nous n'avions pas envisagé, concerne la question de la représentation et de l'orientation dans l'espace du plan du jeu et l'orientation d'un objet par rapport à un autre. L'enseignante de CM1 a relié cette caractéristique à un apprentissage en cours dans les niveaux de CM1 et CM2 qui s'inscrit en géométrie. Cette remarque nous semble importante et nous devons la prendre en compte pour proposer un artefact qui doit lever toutes formes d'obstacle à l'apprenant-joueur.

Enfin, même si elles n'ont pas encore conscience des concepts de programmation que permet d'apprendre Blockly Maze, les trois enseignantes s'accordent à dire que ce jeu peut être un bon outil à utiliser en classe. Elles estiment que proposer un tel objet constituerait un nouveau support ludique, source de motivation.

4.2 Phase 2 : ébauche de construction d'une carte heuristique

Pour la seconde phase du projet, celle de la construction de la carte, nous avons opté pour une démarche praxéologique à savoir la constitution d'un lexique pouvant définir des concepts de la pensée informatique et de leurs liens.

Nous souhaitons mettre en œuvre un nouveau protocole d'observation (avec enregistrement audio et vidéo) des enseignantes jouant à Blockly Maze. Mais, cette fois-ci, il leur sera demandé de vocaliser chacune de leurs actions, chacun de leurs raisonnements au cours de la partie, de sorte que l'on puisse dresser un vocabulaire et un raisonnement type sur lequel nous pourrions nous appuyer pour étayer la carte heuristique.

Nous n'avons pas encore réalisé cette expérience en leur présence. Toutefois, nous nous sommes essayés à la pratique en vocalisant nos parties. Deux observations peuvent être soulignées : tout d'abord, l'exercice de *« penser à voix haute »* n'est pas aussi aisé qu'il parait. Ensuite, le lexique basique de programmation est présent (instruction, boucle, *repeat*/*until*, les conditions) et les raisonnements sont clairement identifiés : la question de l'anticipation, la décomposition, l'essai-erreur, le test de la solution, la généralisation d'une solution. Si de notre côté, les résultats sont ceux auxquels nous nous attendons, qu'en sera-t-il des enseignantes ? Probablement, que le lexique spécifique de la programmation ne sera pas prononcé, mais des mots s'en apparentant pourront émerger. D'autre part, la façon de raisonner ne sera peut-être pas aussi claire, mais nous pourrons observer des points de convergence. Dans tous les cas, ce premier travail constitue pour elles un début de formation à la pensée informatique, car il sera suivi d'un retour réflexif sur leur méthode de résolution et d'un apport de vocabulaire spécifique.

Ce premier travail doit nous amener vers la production d'un outil commun de pensée, de réflexion et de construction du concept de pensée informatique. Puis, dans un second temps, il doit permettre la constitution d'un socle pour co-construire à partir de l'expérience des enseignantes et de leur connaissance des compétences des élèves des niveaux de jeu plus adaptés.

5 Conclusion

Le méta-design ancre l'idée d'engager activement les utilisateurs-enseignants dans la conception de jeux sérieux afin que ces derniers puissent les intégrer dans leurs parcours pédagogiques et favoriser ainsi l'apprentissage des élèves. Faire participer des enseignants *« novices »* en informatique à un projet de conception d'un jeu sérieux nécessite d'emblée de planifier un accompagnement formatif. Le recours à une personne jouant le rôle de *« broker »* nous permet d'avoir un retour pragmatique des enseignantes sur les conditions d'usage d'un tel dispositif pour elles-mêmes et en classe, en situation d'enseignement avec des élèves. Les enseignantes nous apportent quant à elles leur expertise de pédagogue sur les connaissances et compétences des élèves de cycle 3 afin d'adapter au mieux Blockly Maze.

Notre démarche de recherche collaborative orientée par la conception vise à prendre en compte intégralement les compétences et connaissances des enseignantes en matière d'informatique, sans contournement, tel qu'elles se présentent en condition écologique. Ce choix nous impose de surcroit un accompagnement de proximité pour outiller les enseignantes afin d'optimiser leur participation et leur adhésion au projet. Nous avons conscience du problème de connaissance du domaine de la pensée informatique, propre d'ailleurs à beaucoup d'enseignants du primaire, en raison d'un manque de formation. Nous estimons que l'apport d'un « *broker* », servant de pont entre la recherche et le terrain,

ne peut que bénéficier aussi bien au projet qu'aux enseignantes elles-mêmes. Et l'apport d'une carte heuristique sur la pensée informatique est justifié d'une part par le besoin en formation du domaine, d'autre part par celui d'établir un socle commun sur lequel construire la démarche de méta-design.

Fondés sur ces premiers pas, nos travaux en cours et futurs portent sur le développement de nouveaux niveaux de Blockly Maze et leur intégration dans APPLiq pour la scénarisation ainsi que sur la mise en place des indicateurs de suivi pour permettre aux enseignants de comprendre les usages de leurs élèves. Toutes ces actions se font en collaboration avec les enseignantes sur le terrain avec qui nous continuons les entretiens et expérimentations.

Références

Baron, Georges-Louis, et Eric Bruillard. 2008. « Technologies de l'information et de la communication et indigènes numériques : quelle situation ? » *STICEF* 15 (195): 12 pages.

Baron, Georges-Louis, et Béatrice Drot-Delange. 2016. « L'informatique comme objet d'enseignement à l'école primaire française? Mise en perspective historique ». *Revue française de pédagogie. Recherches en éducation*, nº 195: 51-62.

Bell, Tim, Ian H. Witten, et Mike Fellows. 1998. *Computer Science Unplugged: Off-line activities and games for all ages*. Citeseer.

Brunet, Olivier, Amel Yessad, Mathieu Muratet, et Thibault Carron. 2020. « Vers un modèle de scénarisation pour l'enseignement de la pensée informatique à l'école primaire ». In *Didapro 8 – DidaSTIC*. Lille, France. https://hal.archives-ouvertes.fr/hal-02496191.

Djaouti, Damien. 2011. « Serious Game Design : considérations théoriques et techniques sur la création de jeux vidéo à vocation utilitaire ». http://thesesups.ups-tlse.fr/1458/.

Dowek, Gilles. 2011. « Les quatre concepts de l'informatique ». In , 21-29. Athènes : New Technologies Editions. https://edutice.archives-ouvertes.fr/edutice-00676169.

Fischer, Gerhard, Elisa Giaccardi, Yunwen Ye, Alistair G. Sutcliffe, et Nikolay Mehandjiev. 2004. « Meta-design: a manifesto for end-user development ». *Communications of the ACM* 47 (9): 33-37.

Garris, Rosemary, Robert Ahlers, et James E. Driskell. 2002. « Games, motivation, and learning: A research and practice model ». *Simulation & gaming* 33 (4): 441-67.

Marne, Bertrand. 2014. « Modèles et outils pour la conception de jeux sérieux : une approche meta-design ». Thèse de Doctorat en Informatique, Paris: Université Pierre et Marie Curie (UPMC).

Rabardel, Pierre. 1995. *Les hommes et les technologies : une approche cognitive des instruments contemporains*. U. Série Psychologie. Paris, France: Armand Colin. http://ergoserv.psy.univ-paris8.fr/Site/Groupes/Modele/Articles/Public/ART372105503765426783.PDF.

Resnick, Mitchel, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Eastmond Eastmond, Karen Brennan, Amon Millner, et al. 2009. « Scratch: Programming for All ». *Communications of the ACM* 52 (11): 60-67. https://doi.org/10.1145/1592761.1592779.

Sanchez, Éric, et Réjane Monod-Ansaldi. 2015. « Recherche collaborative orientée par la conception. Un paradigme méthodologique pour prendre en compte la complexité des situations d'enseignement-apprentissage ». *Éducation et didactique*, nº 9-2 (septembre): 73-94. https://doi.org/10.4000/educationdidactique.2288.

Sardone, Nancy B. 2018. « Attitudes Toward Game Adoption: Preservice Teachers Consider Game-Based Teaching and Learning ». *International Journal of Game-Based Learning (IJGBL)* 8 (3): 1-14. https://doi.org/10.4018/IJGBL.2018070101.

Spach, Michel. 2017. « Activités robotiques à l'école primaire et apprentissage de concepts informatiques: quelle place du scénario pédagogique? Les limites du co-apprentissage ». Thèse de doctorat de Sciences de l'éducation, Université Sorbonne Paris Cité.

Tchounikine, Pierre. 2017. « Initier les élèves à la pensée informatique et à la programmation avec Scratch ».

Vermeulen, Mathieu. 2018. « Une approche meta-design des learning games pour développer leur usage ». Theses, Sorbonne Université, Faculté des Sciences et Ingénierie. https://tel.archives-ouvertes.fr/tel-01871048.

Wing, Jeannette M. 2006. « Computational Thinking ». *Communications of the ACM* 49 (3): 33-35.