8e Atelier "Apprentissage de la pensée informatique de la maternelle à l'Université (APIMU 1000)" @ EIAH 2025

Julien Broisin, Christophe Declercq, Sébastien Jolivet, Yannick Parmentier, Yvan Peter, Yann Secq

EIAH

2025 Lille

Environnements Informatiques pour l'Apprentissage Humain

8e Atelier "Apprentissage de la pensée informatique de la maternelle à l'Université (APIMU 1000)" @ EIAH 2025 [1]

Julien BROISIN, Christophe DECLERCQ, Cédric FLUCKLINGER, Sébastien JOLIVET, Yannick PARMENTIER, Yvan PETER, Yann SECQ (Éds.)

Villeneuve d'Ascq, France, 10 juin 2025

1. https://apimu.gitlabpages.inria.fr/site/ateliers/eiah25/

Table des matières

Actes du 8e Atelier "Apprentissage de la pensée informatique de la maternelle à l'Université (APIMU 1000)" @ EIAH 2025
Villeneuve d'Ascq, France, 10 juin 2025

« *Dirlo sous l'eau* » : développer une culture générale en informatique avec un jeu sérieux d'investigation

Sonia Agrebi [1], Jamil Bakhouy [2], Yann Secq [1]

(1) Centre LEARN, École Polytechnique Fédérale de Lausanne, Suisse
(2) Etablissement Primaire et Secondaire d'Oron la Ville, Suisse

sonia.agrebi@epfl.fr, jamil.bakhouy@edu-vd.ch, yann.secq@epfl.ch

Résumé

Cette contribution présente une modalité de jeu sérieux conçue pour enseigner différents concepts d'Éducation Numérique et de Science Informatique dans un cadre scolaire. Cette activité débranchée est un jeu de cartes impliquant des élèves sur une démarche d'investigation soutenue par des mécanismes ludiques. Une déclinaison sur le thème de l'intelligence artificielle a été expérimentée en contexte écologique. La synthèse des retours élèves et enseignant·e·s illustre l'impact de la modalité sur l'implication des élèves et la richesse des dialogues induits lors de la phase de structuration au cours et à l'issue de l'activité.

Abstract

This contribution presents a serious game modality designed to teach various Digital Education and Computer Science concepts in a school setting. This unplugged activity is a card game that engages students in an inquiry-based learning supported by game mechanics. A variation on the theme of artificial intelligence was tested in an ecological classroom context. The synthesis of feedback from both students and teachers highlights the modality's impact on student engagement and the richness of dialogues generated during and after the structuring phase of the activity.

Mots-clés : Activité débranchée, Jeux sérieux, Éducation numérique, Pédagogie active.
Keywords: Unplugged activity, serious games, digital education, active learning.

1 Introduction

Le développement du numérique et de ses usages confrontent les citoyen·ne·s et les élèves à des environnements relevant parfois de la « magie » pour un trop grand nombre d'entre eux. Récemment, l'accès du grand public à des outils d'intelligence artificielle relativement efficaces pour de la génération de texte, d'images, voire de vidéos induit de nouvelles problématiques en termes de compréhension des possibilités et limites de ces outils et de leurs multiples impacts sociaux et environnementaux. De nombreux pays ayant réintroduit des éléments d'Éducation Numérique et de Science Informatique dans les programmes scolaires, il est possible de s'appuyer

sur ce cadre pour démythifier ces outils en expliquant à la fois le comment (s'en servir à bon escient) et le pourquoi (cela fonctionne ainsi).

Sur des éléments portant sur une culture technique générale de l'informatique, il est intéressant de mobiliser des approches issues des pédagogies actives pour engager les élèves dans la découverte et l'apprentissage de notions qui nécessitent de s'approprier un contexte complexe, combinant une compréhension technique minimale et une réflexion sur les enjeux sociaux associés. Pour cela, nous proposons un type de jeux sérieux reposant sur une modalité tangible impliquant une démarche d'investigation réalisée en petits groupes pour la découverte des notions et des phases discursives réalisées collectivement pour la structuration.

Cette contribution présente dans un premier temps le cadre théorique dans lequel cette modalité a été développée, puis expose les principes de cette modalité et l'une de ses instanciations, avant de synthétiser les résultats d'une expérimentation réalisée avec environ 80 élèves de 13 à 16 ans en milieu écologique.

2 État de l'art

Les définitions récentes des jeux sérieux sont assez fortement influencées par le développement de l'informatique et des jeux vidéo (Sawyer & Rejeski, 2002). Nous préférons nous référer à la définition plus ancienne et générale proposée par (Abt, 1970) : « *Nous considérons comme serious games les jeux explicitement et intentionnellement conçus à des fins éducatives, et qui ne sont pas principalement destinés au divertissement. Cela n'implique aucunement que les serious games ne soient pas, ou ne doivent pas, être amusants.* ». Mobilisé dans un cadre scolaire, ce type d'activité relève d'une démarche de pédagogie active, c'est-à-dire d'après (Prince, 2004) « *any instructional method that engages students in the learning process. In short, active learning requires students to do meaningful learning activities and think about what they are doing.* ». L'impact des jeux sérieux sur la motivation et les apprentissages a été étudié, notamment dans la méta-analyse (Wouters & al, 2013) étudiant les effets cognitifs et motivationnels. Les auteurs ont observé des résultats positifs en terme d'efficacité d'apprentissage et de rétention, mais étonnamment pas nécessairement sur une différence de motivation. L'importance de certains facteurs comme le nombre de sessions et la réalisation de l'activité en groupe est aussi soulignée. Notre contribution s'inscrit pleinement dans cette approche des jeux sérieux pour un usage pédagogique en contexte scolaire avec la mobilisation du travail de groupe sur différents périmètres.

3 Une modalité de jeu d'investigation

Cette section présente d'abord les principes structurant cette modalité de jeu d'investigation et les points d'attention sur sa mise en œuvre en classe, puis introduit le jeu « *Dirlo sous l'eau* » ayant fait l'objet de l'expérimentation détaillée en section 4.

3.1 Principes de la modalité

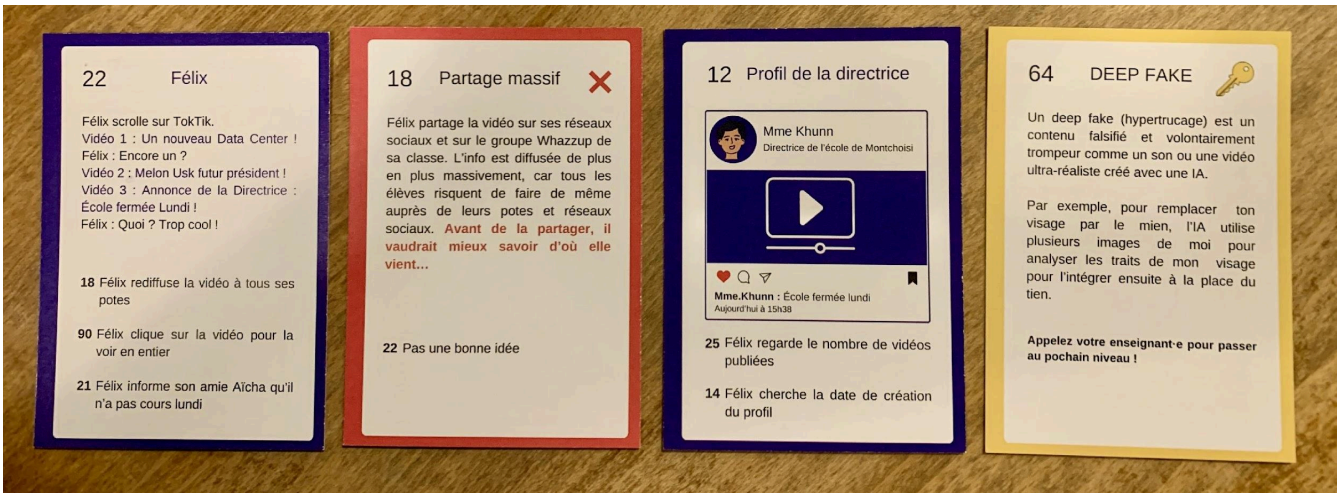
Les principaux objectifs de conception de cette modalité sont à la fois de permettre aux élèves de découvrir de nouvelles notions, induisant la nécessité de lire[1], et aussi de développer un esprit

[1] ou accéder à des ressources numériques, mais nous souhaitons minimiser cet aspect nécessitant une infrastructure technologique.

critique par une sensibilisation à quelques réflexes de bonne hygiène numérique. Pour cela nous avons conçu une activité débranchée sur la base d'un jeu de cartes détournant le mécanisme d'un type de livre (« *livres dont vous êtes le héros* ») en vogue dans les années 80-90 reposant sur un principe narratif non linéaire. L'activité se structure autour d'une alternance de phases de découverte en petit groupe (2 à 3 élèves) avec la manipulation des cartes et des phases d'échanges collectifs (en classe complète) pour synthétiser les éléments découverts.

Cette modalité place les élèves dans une démarche d'investigation, nourrie par l'imaginaire d'un scénario intrigant et induisant le souhait de compréhension de la situation déclenchante (carte n°22 dans la figure 1). Chaque carte propose un certain nombre de choix que les élèves discutent avant de choisir l'une des pistes possibles. Le parcours des cartes correspond à un graphe avec des impasses et retours en arrière parfois. Deux cartes spécifiques portent les apprentissages en explicitant les concepts visées, soit en terme de bonne hygiène numérique (Carte n° 18), soit en termes de concepts à découvrir (Carte 64).

Figure 1: Carte initiale (22) et ses trois alternatives possibles, une carte « risque » (18), une carte illustrée (12) et une carte « clé » (64) portant le concepts à découvrir sur le premier niveau



Le jeu sérieux est structuré en plusieurs niveaux : le premier, ancré dans le quotidien des élèves, les plonge dans une situation initiale dont ils découvrent un premier élément de réponse ; les niveaux suivants introduisent des concepts plus complexes, en lien avec celui du premier niveau. À l'issue de chaque niveau, un temps d'échange collectif est réalisé afin de partager les éléments découverts (les cartes « *clés* » et « *risques* »). Ce moment d'institutionnalisation est important pour reformuler et étayer les concepts visés et aussi partager l'ensemble des éléments à découvrir car il est possible de choisir différents chemins ne menant pas tous au même ensemble de cartes « *clés* » dans les niveaux ultérieurs.

Cette modalité a été développée sous plusieurs déclinaisons thématiques comme la notion d'identité numérique et les médias sociaux, la cybersécurité ou encore des premiers éléments d'intelligence artificielle qui fait justement l'objet de l'expérimentation détaillée dans la section suivante.

3.2 Déclinaison de la modalité sur des notions d'IA

L'activité « *Dirlo sous l'eau* » décline cette modalité pour la découverte de notions de base sur l'intelligence artificielle. Le premier niveau est placé dans un contexte proche du quotidien des élèves et permet de découvrir la notion de *deep fake* et aussi d'initier une démarche de prise de recul critique face à ce type de vidéo. Le second niveau développe différents impacts sociaux de l'usage d'outils d'intelligence artificielle et présente quelques éléments concernant leur fonctionnement. Les cartes « clés » qui présentent les concepts à découvrir, portent sur la notion de deep fake au niveau 1, et sur le cyberharcèlement, l'impact des *deep fakes* sur la démocratie, l'usage de l'IA pour reproduire des voix, ainsi que sur des notions d'apprentissage automatique et de réseaux de neurones profonds au niveau 2. Les cartes « *risques* » contenant des règles de bonne conduite numérique portent sur la notion d'amplification sur les réseaux sociaux (niveau 1) et sur le droit à l'image (niveau 2).

4 Expérimentation écologique de « *Dirlo sous l'eau* »

Nous avons pu participer à une expérimentation à moyenne échelle dans le cadre de la réalisation d'une journée de la citoyenneté organisée dans l'établissement primaire et secondaire d'Oron-la-Ville. Durant cette journée, tous les élèves (environ 120) du secondaire 1 (13 à 16 ans) ont été répartis en 6 groupes d'âges hétérogènes et ont réalisé différents ateliers en lien avec la citoyenneté. L'un des ateliers portait sur le jeu d'investigation « *Dirlo sous l'eau* » et a été animé par 8 enseignant·e·s qui avaient pu s'initier à l'activité lors d'une session de 2h réalisée par les auteurs et autrices quelques jours auparavant. Juste après l'activité, les élèves ont répondu à un questionnaire interactif (*Wooclap*) et un autre questionnaire a été proposé quelques jours plus tard aux enseignant·e·s.

4.1 Synthèse des retours au niveau des élèves

Le questionnaire portait à la fois sur le ressenti des élèves par rapport à l'activité et sur des questions d'apprentissage. Les tableaux 2 et 3 ci-dessous synthétisent les résultats portant sur environ 80 élèves. On constate que globalement l'activité est évaluée positivement sur différentes dimensions (tableau 2). Concernant l'auto-évaluation des apprentissages (Q2. Avez-vous appris de nouvelles choses sur l'IA ?), 60 % des élèves estiment avoir appris de nouvelles choses (48 % un peu, 12 % beaucoup), tandis que 40 % ont répondu « pas vraiment ». Il serait intéressant de réaliser une expérimentation complémentaire avec des groupes de niveau scolaire homogène avec un pré-test en amont de l'activité et un post-test plusieurs semaines après afin d'affiner la répartition des apprentissages auto-évalués. Ces éléments confortent les observations réalisées en classe sur le degré d'implication et le climat de classe dynamique et positif durant la réalisation de l'activité.

Tableau 2: Ressenti des élèves par rapport à l'activité (notation de 1 à 5)

	laide ↔ jolie	sans imagination ↔ créative	ennuyeuse ↔ captivante	scolaire ↔ ludique	complexe ↔ simple
Q1. Qu'avez-vous pensé de cette activité ?	3,3	3,4	3,0	3,0	4,1

Un second ensemble de questions portait sur les apprentissages visés dans l'activité et le tableau 3 synthétise les résultats qui illustrent une rétention « instantanée » satisfaisante.

Tableau 3: Apprentissage des élèves par rapport à l'activité (pourcentage de réponse valide)

Q3. Pour Ralph le principal danger du *deep fake* c'est	Q4. Un *deep fake* c'est	Q5. Aïcha a des doutes sur la vidéo car	Q6. Le droit à l'image permet de	Q7. Un réseau de neurones c'est
90%	82%	92%	55%	82%

4.2 Synthèse des retours au niveau des enseignant·e·s

Le questionnaire pour les enseignant·e·s visait à recueillir leurs impressions sur l'activité (tableau 4 similaire à celui des élèves) et surtout leur appréciation sur l'appropriation et la mise en œuvre de l'activité avec leurs élèves (tableau 5). Sur ces aspects critiques, les retours sont positifs et semblent confirmer que cette modalité est pertinente pour un usage en contexte scolaire. Le kit permettant de réaliser l'activité restant dans l'établissement, on peut raisonnablement supposer que l'activité sera remobilisée dans les années scolaires à venir. Il est important de noter cependant qu'avec seulement 50% de retour, ces éléments portent sur un très faible échantillon (4 personnes).

Tableau 4: Ressenti des enseignant·e·s par rapport à l'activité (notation de 1 à 5)

	déplaisante ↔ attrayante	sans imagination ↔ créative	ennuyeuse ↔ captivante	scolaire ↔ ludique	complexe ↔ simple
Q1. Qu'avez-vous pensé de cette activité ?	3,8	4	3,5	3,8	4

Tableau 5: Retours sur l'appropriation et la mise en œuvre du point de vue enseignant

Appréciations pédagogiques sur cette activité	*Notation de 1 à 5*
Pertinence pour des apprentissage en éducation numérique	4
Facilité de prise en main de l'activité	4,5
Facilité de mise en œuvre avec les élèves	4,5
Degré d'implication des élèves lors de la phase de découverte	3,5
Degré d'implication des élèves dans la phase d'échange collectif	3,8
Qualité de la phase d'échanges collectifs	3,5

5 Conclusion et perspectives

Cette contribution présente une modalité de jeu sérieux visant à développer une culture générale en informatique et apporte des premiers éléments sur sa pertinence et son efficience, à la fois au niveau des élèves et de leurs enseignant·e·s. Cette modalité repose sur une démarche d'investigation réalisée en petit groupe à l'aide d'un jeu de cartes et de phases d'échanges collectifs pour structurer les

concepts découverts. Cette modalité a été mobilisée dans de multiples formations continues d'enseignant·e·s du secondaire, majoritairement impliqués dans l'Éducation Numérique.

Une expérimentation réalisée en milieu écologique avec une dizaine d'enseignant·e·s de différentes disciplines et un peu moins de 100 élèves a permis de quantifier quelques premiers éléments sur la pertinence et l'efficience de l'activité. Les résultats au niveau des élèves confirment un impact fort sur la motivation et l'implication dans l'activité ainsi que des taux de réussite élevés sur les concepts visés. Au niveau des enseignants, l'appréciation est similaire à celle des élèves, mais le point le plus important est la facilité d'appropriation et de mise en œuvre en classe qui constitue un point critique pour l'ancrage d'une pratique de pédagogie active et sa mobilisation récurrente.

Nous souhaiterions poursuivre des expérimentations avec des niveaux de classes homogènes et une mesure des apprentissages sur un temps long afin de quantifier l'impact à long terme des notions découvertes avec cette modalité.

Remerciements

Nous souhaitons remercier M. Jamil Bakhouy qui a proposé de mobiliser ce jeu sérieux dans le cadre de la journée de la citoyenneté organisée dans son établissement, ainsi que ses collègues qui ont participé à la courte séance de formation en amont de la mise en œuvre avec leurs élèves. Sans leur implication, cette contribution n'aurait pu être possible.

Références

Abt C. (1970). *Serious Games*. Viking Press.

Wouters, P., van Nimwegen, C., van Oostendorp, H., & van der Spek, E. D. (2013). *A meta-analysis of the cognitive and motivational effects of serious games*. Journal of Educational Psychology, 105(2), 249–265.

Prince M. (2004). *Does Active Learning Work? A Review of the Research*. Journal of Engineering Education, 93(3), 223-231.

Sawyer B., Rejeski D. (2002). *Serious Games: Improving Public Policy Through Game-based Learning*. Woodrow Wilson International Center for Scholars.

Actes du 8e Atelier "Apprentissage de la pensée informatique de la maternelle à l'Université (APIMU 1000)" @ EIAH 2025
Villeneuve d'Ascq, France, 10 juin 2025

Typologie d'erreurs en programmation et détection automatique dans un code Python

Eva Chouaki[1] Badmavasan Kirouchenassamy[2] Sébastien Jolivet[1, 2, 3] Amel Yessad[2]
(1) IUFE, UNIGE, Genève, Suisse
(2) Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
(3) TECFA, UNIGE, Genève, Suisse
eva.dechaux@unige.ch, sebastien.jolivet@unige.ch, prenom.nom@lip6.fr

RÉSUMÉ
Cette contribution s'inscrit dans un contexte plus large de décision de rétroaction dans l'environnement AlgoPython. Pour décider ces rétroactions, nous devons identifier et catégoriser finement des erreurs dans des codes Python produits par des novices en programmation. Après avoir précisé nos besoins et les avoir mis en regard de la littérature, nous exploitons un référentiel de types de tâches déjà existant pour construire notre propre typologie d'erreurs. Puis nous présentons un processus, réifié dans une librairie Python, que nous avons développé pour identifier les erreurs de notre typologie dans des codes erronés.

ABSTRACT

This contribution is part of a wider context of feedback decision in the AlgoPython environment. To decide on feedback, we need to identify and finely categorize errors in Python code produced by programming novices. After specifying our needs and comparing them with the literature, we exploit an existing repository of types of tasks to build our own typology of errors. We then present a process, reified in a Python library, that we have developed to identify the errors of our typology in erroneous codes.

MOTS-CLÉS : Python, erreur en programmation, types de tâches.

KEYWORDS: Python, programming error, types of tasks.

1 Introduction

Cette contribution se focalise sur un enjeu particulier d'un projet d'implémentation de rétroactions épistémiques adaptées dans l'environnement AlgoPython [1] (Jolivet et al., 2025). Parmi les éléments définissant l'état utilisé pour décider la rétroaction, deux nécessitent de pouvoir identifier et catégoriser les erreurs dans un code proposé. Tout d'abord, lors de la soumission d'un code erroné [2], un des éléments contribuant à la décision de la rétroaction est la ou les erreurs identifiées dans ce code. Par ailleurs, nous construisons aussi, au fil de l'utilisation de l'environnement, un profil de l'apprenant prend notamment en compte son rapport aux différentes erreurs (apparition, fréquence...).

1. http://algopython.fr
2. Dans notre contexte il s'agit d'un code syntaxiquement correct mais qui ne résout pas le problème donné

Pour répondre à ces deux besoins, nous devons, sur la base d'un code erroné, pouvoir identifier et positionner par rapport à une typologie, les erreurs dans ce code. C'est ce double objectif que nous abordons dans cette contribution.

Dans un premier temps (section 2), pour définir notre grille d'analyse de la littérature, nous précisions les besoins issus de notre contexte de travail. Nous proposons alors une synthèse de différentes approches proposées dans la littérature relativement aux erreurs dans le domaine de la programmation. Face au constat de l'absence d'une typologie permettant de satisfaire nos besoins, nous proposons la définition d'une typologie d'erreurs (section.3). Elle est basée sur un référentiel de types de tâches déjà exploité par ailleurs dans le projet.

Dans un deuxième temps (section.4) nous présentons notre moyen de lier notre typologie d'erreurs et les codes erronés des apprenants. Il est basé sur une analyse des codes AST et est implémenté au moyen d'une librairie Python.

2 Définition et classification des erreurs en programmation

De nombreux travaux mettent en évidence la place et l'importance de l'erreur dans l'apprentissage en général (Astolfi, 1997). Dans le cadre des rétroactions la prise en compte de l'erreur est aussi pointée comme un critère d'efficacité (Brooks et al., 2019) ; (Small and Lin, 2018). Afin d'orienter notre analyse de la littérature relative aux erreurs dans la programmation nous commençons par spécifier nos besoins en termes de détection et d'analyse des erreurs. Il est important, à ce stade, de préciser que l'EIAH *AlgoPython* est destiné à des apprenants totalement, ou largement, novices en programmation et qu'ils sont amenés à y résoudre des problèmes simples (affichage, labyrinthes, reproduction de dessins) ne demandant qu'un nombre limité de lignes de codes (entre 3 et une dizaine au maximum) pour parvenir à une solution (ceci limite le nombre de solutions possibles pour un problème donné). De plus, dans l'environnement, nous analysons le code de l'apprenant uniquement s'il est syntaxiquement correct et s'il ne résout pas le problème donné. Nous considérons donc comme erreur un écart entre le code de l'apprenant et un (ou plusieurs) code(s) permettant effectivement de résoudre le problème donné.

2.1 Définition de nos besoins en terme de détection et classification d'erreur

Nous proposons des rétroactions sur des codes courts et erronés (même s'ils peuvent être proches d'une solution). Pour avoir la possibilité de proposer des rétroactions les plus adaptées possible à l'apprenant nous avons besoin de pouvoir identifier et classifier précisément la ou les erreurs présentes dans son code. Ceci implique de pouvoir distinguer des différences précises entre deux codes, sans se limiter à des analyses globales du type *"Problème sur la boucle"* ou *"La structure proposée n'est pas bonne"*. Précisons que l'identification des erreurs est un des éléments, parmi d'autres, guidant la décision de la rétroaction. Ceci ne signifie ni que l'erreur identifiée soit explicitement signalée dans la rétroaction ni que la rétroaction vise précisément, et encore moins exclusivement, à corriger cette erreur. La décision de la meilleure rétroaction, au regard du contexte et des objectifs visés (réussite de la tâche et apprentissage), n'est donc pas uniquement fondée sur les erreurs mais prend aussi en compte par exemple le profil de l'apprenant, la tâche à réaliser, plus d'éléments sur ce processus de décision sont présentés dans (Jolivet et al., 2025).

Afin d'identifier dans la littérature relative à l'erreur en programmation les éléments pouvant nous servir nous regardons donc :

1. ce qui est détecté : manifestation de l'erreur (i.e. problème dans le code) ou cause de l'erreur (i.e. élément relatif à l'apprenant)
2. finalité pour laquelle l'erreur est détectée ;
3. contexte du travail (langage particulier, dans un EIAH ou non) ;
4. existence d'une classification ;*a priori* des erreurs ou la production d'une classification uniquement sur la base des productions des apprenants ;
5. existence d'une méthode de détection des erreurs (absente, manuelle, automatique).

Ces éléments précisés nous présentons maintenant les éléments essentiels issus de la littérature et les conséquences que nous en tirons.

2.2 Etude de la littérature

Pour réaliser cette étude nous avons travaillé de la manière suivante. Nous avons analysé plusieurs articles traitant des erreurs en programmation. Cela nous a permis d'extraire certaines caractéristiques communes à ces articles notamment sur les mots clés utilisés dans les résumés. En effet, le mot comme "error" combiné aux mots clés "programming", "programmer", "algorithm", "student" revenaient systématiquement dans les premiers articles étudiés. Nous avons ensuite cherché dans de grandes bases de données comme HAL plusieurs requêtes combinant ces différents mots clés, en filtrant sur le domaine relatif à l'informatique. Les quatre contributions suivantes ont retenues notre attention, car les autres ne proposaient pas de classification des erreurs en programmation.

Il s'agit des articles suivants :

A. **Types of Errors in Block Programming : Driven by Learner, Learning Environment** de Ben-Yaacov et Hershkovitz, *Journal of Educational Computing Research*, 2023
B. **Logic Error Detection Algorithm for Novice Programmers based on Structure Pattern and Error Degree** de Yuto Yoshizawa et Yutaka Watanobe, *iCAST*, 2018
C. **A New Look at Novice Programmer Errors** de Davin McCall et Michael Kölling, King's College London, *ACM Transactions on Computing Education*, 2019
D. **Novice Java Programming Mistakes : Large-Scale Data vs. Educator Beliefs** de Neil C. C. Brown & Amjad Altadmri, *ACM Transactions on Computing Education*, 2017

Dans le tableau 1, nous comparons ces quatre contributions au regard des critères identifiés dans la section précédente.

On constate que, dans l'ensemble des articles étudiés, les erreurs sont décrites en fonction de ce qui est observé dans les codes des apprenants ou dans les messages d'erreurs générés lors de la compilation et non pas relativement à une typologie d'erreurs préexistante. Il apparaît également que deux objectifs principaux motivent les auteurs dans leur volonté de classifier ces erreurs. Le premier, à visée pédagogique, vise à mieux comprendre les erreurs des apprenants afin d'adapter les stratégies d'enseignement. Le second objectif concerne la détection automatique des erreurs, dans le but de développer des outils capables d'identifier ces erreurs de manière autonome, par exemple dans le but d'adapter un EIAH (les activités, les rétroactions, etc.).

Concernant la manière dont les erreurs sont détectées, l'article A repose sur une analyse manuelle

	(Ben-Yaacov and Hershkovitz, 2022)	(Yoshizawa and Watanobe, 2018)	(McCall and Kölling, 2019)	(Brown and Altadmri, 2017)
Manifestation ou cause de l'erreur	Manifestation	Manifestation	Manifestation	Manifestation
Finalité de la détection	Pédagogique	Détection	Pédagogique	Pédagogique et détection
Contexte du travail	Plateforme Kodetu (programmation par blocs)	Aizu Online Judge (java : évaluation automatique)	BlueJ (java)	BlackBox pour collecter des soumissions en Java à partir d'autres environnements (BlueJ, Eclipse, etc.)
Classification des erreurs	Basés sur production d'apprenants	Basés sur production d'apprenants	Basés sur production d'apprenants	Basés sur production d'apprenants
Méthode de détection	Analyse manuelle et catégorisation	Utilisation d'AST	Analyse automatisée des messages du compilateur	Analyse automatisée des messages du compilateur

TABLE 1 – Comparaison de quatre approches de détection et classification des erreurs

des codes. Pour les articles C et D il s'agit d'exploiter une analyse automatique des messages du compilateur, qui permet d'identifier le nombre d'erreurs similaires afin de déterminer une fréquence des erreurs. Enfin, l'article B utilise des algorithmes basés sur la représentation en Abstract Syntax Trees (AST) pour détecter automatiquement les erreurs.

On peut retenir en particulier des articles étudiés que nous n'avons identifié aucune typologie des erreurs en programmation, d'un niveau un peu plus fin que *erreurs syntaxiques* ; *erreur logique* ; *erreur d'exécution*. Par ailleurs, d'éventuelles classifications sont construites de manière dynamique au fur et à mesure de l'analyse de codes d'apprenants.

Or dans notre contexte nous devons pouvoir mettre en relation les erreurs observées avec d'autres éléments tels que l'énoncé de l'exercice et l'apprenant. Il est donc souhaitable pour nous de disposer d'un ensemble d'erreurs prédéfini. C'est la construction d'un tel ensemble que nous abordons dans la section suivante.

3 Proposition de typologie d'erreurs

Dans le projet au sein duquel s'inscrit ce travail, on exploite déjà un référentiel de types de tâches [3], présenté par exemple dans (Jolivet et al., 2023), pour décrire les exercices et modéliser les rétroactions. Un extrait du référentiel des types de tâches relatif aux boucles est proposé figure 1.

Afin de conserver une cohérence globale aux différentes composantes de notre environnement nous

3. `https://link.infini.fr/ref_prog`

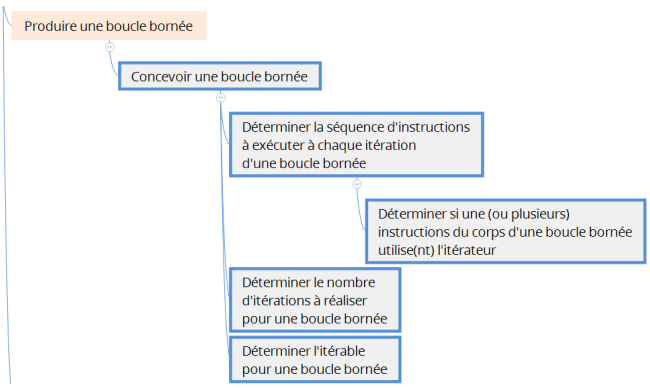


FIGURE 1 – Extrait du référentiel de types de tâches relatif aux boucles

avons donc décidé de construire notre typologie d'erreur sur la base de ce même référentiel. Pour cela, pour cela nous avons identifié les erreurs possibles associées à un ou plusieurs de ces types de tâches. Cette typologie est structurée de la même manière que le référentiel de type de tâches autour des thèmes suivants : Programmation et algorithme ; Variable ; Instruction conditionnelle ; Fonction ; Boucle ; Expression. Elle est présentée en intégralité dans l'Annexe 1 et un extrait qui contient les erreurs relatives aux boucles est proposé dans le tableau 2.

ERREUR	**DETAILS OU EXEMPLES**	**TYPE(S) DE TÂCHES**
BOUCLES		
(LO-Iterator-Usage-Error) Erreur dans l'utilisation de l'itérateur de la boucle	L'apprenant s'est trompé dans les valeurs prises par l'itérateur (valeur de départ par exemple)	B. Déterminer si une (ou plusieurs) instruction(s) du corps d'une boucle bornée utilise(nt) l'itérateur
(LO-FOR-Number-Iteration-Error-Under2) Erreur nombre d'itérations de la boucle FOR (à un près)	La boucle s'exécute une fois de trop ou une fois de moins	B. Déterminer le nombre d'itérations à réaliser pour une boucle bornée
(LO-FOR-Number-Iteration-Error) Erreur nombre d'itérations de la boucle FOR (>2)	La boucle ne s'exécute pas le bon nombre de fois	B. Déterminer le nombre d'itérations à réaliser pour une boucle bornée
(LO-WHILE-Number-Iteration-Error-Under2) Erreur nombre d'itérations de la boucle WHILE (à un près)	La boucle s'exécute une fois de trop ou une fois de moins	B. Déterminer le nombre d'itérations à réaliser pour une boucle bornée
(LO-WHILE-Number-Iteration-Error) Erreur nombre d'itérations de la boucle WHILE (>2)	La boucle ne s'exécute pas le bon nombre de fois	B. Déterminer le nombre d'itérations à réaliser pour une boucle bornée

(suite page suivante)

ERREUR	DETAILS OU EXEMPLES	TYPE(S) DE TÂCHES
(LO-Exit-Condition-Error-Start) Erreur condition d'arrêt de la boucle : départ	Erreur liée à la condition de départ	B. Déterminer la condition d'arrêt d'une boucle non bornée
(LO-Exit-Condition-Error-Evolution) Erreur condition d'arrêt de la boucle : évolution	Erreur liée à une instruction qui n'est pas faite pour arrêter la boucle	B. Déterminer l'instruction modifiant la valeur de la condition d'arrêt d'une boucle non bornée
(LO-Body-Missing-Not-Present-Anywhere) Erreur dans le corps de la boucle : Instruction manquante et non présente	Instruction absente dans le corps de la boucle et dans le programme	B. Déterminer les instructions à exécuter à chaque itération d'une boucle bornée B. Déterminer les instructions à exécuter à chaque itération pour une boucle non bornée
(LO-Body-Misplaced) Erreur dans le corps de la boucle : Instruction manquante et présente ailleurs dans le programme	Instruction présente ailleurs dans le programme	B. Déterminer les instructions à exécuter à chaque itération d'une boucle bornée B. Déterminer les instructions à exécuter à chaque itération pour une boucle non bornée
(LO-Body-Error) Erreur dans le corps de la boucle : Instruction erronée	Instruction présente proche de celle manquante	B. Déterminer les instructions à exécuter à chaque itération d'une boucle bornée B. Déterminer les instructions à exécuter à chaque itération pour une boucle non bornée
(LO-For-Missing) Boucle FOR manquante	Absence d'une boucle FOR	B. Produire une boucle bornée
(LO-While-Missing) Boucle WHILE manquante	Absence d'une boucle WHILE	B. Produire une boucle non bornée

TABLE 2 – Typologie d'erreurs, extrait relatif aux boucles

Nous présentons dans la section suivante l'exploitation de cette typologie dans le cadre de notre problème de détection des erreurs dans AlgoPython.

4 De la typologie d'erreurs à l'analyse de codes d'élèves

Dans le contexte de notre recherche, nous disposons du code erroné d'un apprenant et d'un (ou plusieurs) codes corrects. Pour identifier les erreurs présentes dans le code erroné nous avons défini le processus suivant : représentation du code erroné et du ou des codes corrects sous forme d'un arbre syntaxique abstrait (AST). Cette représentation, qui capture les principales constructions d'un programme tout en abstrayant les détails syntaxiques, a été fréquemment utilisée dans le domaine des EIAH (Piech et al., 2012; Broisin and Hérouard, 2019; Lazar et al., 2017) ainsi que dans d'autres domaines tels que le test logiciel ou la correction automatique de code (Le Goues et al., 2019; Shen and Chen, 2020). Ensuite, nous nous appuyons sur l'algorithme de Zang et Shasha (Zhang and Shasha, 1989) (AZS) de programmation dynamique pour apparier l'AST de l'élève avec l'AST du code correct le plus proche. Cet algorithme permet de chercher une séquence d'opérations d'édition élémentaires et ordonnées (insertion, suppression et mise à jour de nœuds) permettant de transformer l'AST de l'élève en celui du code correct, tout en minimisant le coût cumulé de ces opérations (c'est-à-dire le nombre

d'édition).

```
for i in range(4):
    droite(2)
bas(2)
```

Listing 1 – Soumission d'un élève

```
for k in range(3):
    droite(2)
    bas(1)
```

Listing 2 – Solution correcte possible

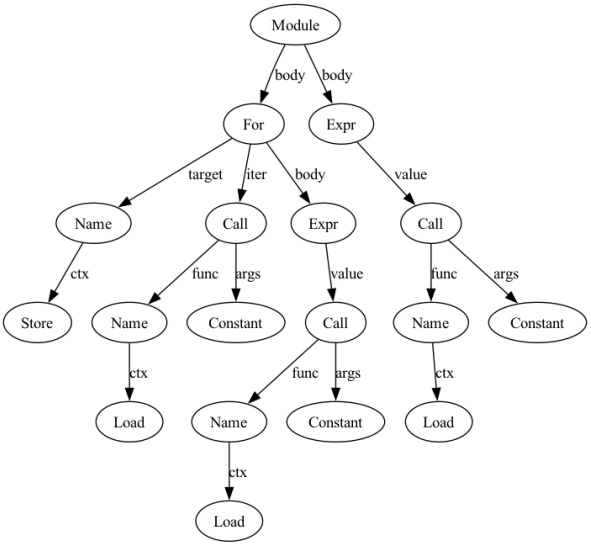


FIGURE 2 – AST sous sa forme originale du code de l'élève

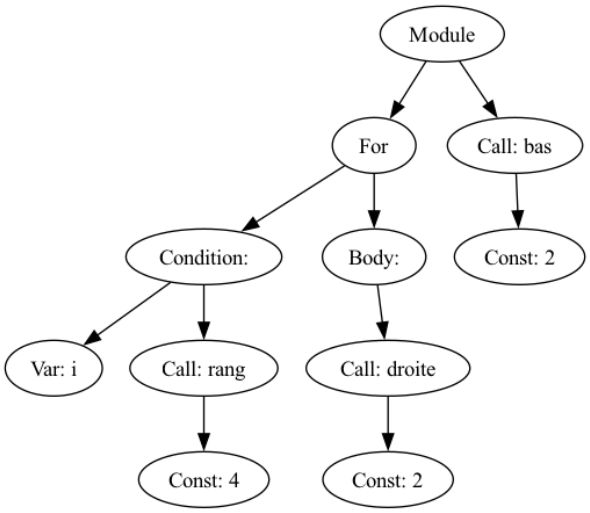


FIGURE 3 – AST modifié et contextualisé du code de l'élève

Nous avons modifié l'implémentation standard de l'AST afin de permettre l'extraction de la localisation absolue du nœud concerné par l'opération d'édition. Cela permet ainsi de repérer précisément la modification dans le code de l'élève. Par exemple, cette amélioration facilite l'identification d'une instruction erronée lorsqu'elle apparaît à deux endroits différents dans le code de l'élève. Enfin, nous construisons des motifs à partir de sous-séquences d'opérations d'édition contextualisées, que nous associons aux erreurs de la typologie. Un exemple de pattern est celui où un nœud et ses enfants dans l'AST de l'élève sont supprimés d'un endroit de l'AST puis ajoutés sous un nœud *For* de l'AST. Cela peut indiquer l'erreur $LO - Body - Misplaced$: *Erreur dans le corps de la boucle : Instruction manquante et présente ailleurs dans le programme*. La librairie qui permet d'identifier dans le code d'un élève des erreurs de notre typologie est accessible ici : `https://pypi.org/project/asterrdetection/`

Le Listing 3 montre les erreurs identifiées, grâce à la librairie développée, dans le code erroné d'un élève (Listing 1 représenté par l'AST modifié de la Figure 3), le code correct attendu est celui du Listing 2. L'analyse du code de l'élève a permis d'identifier trois erreurs : une erreur $(LO - Body - Misplaced)$ *"Erreur dans le corps de la boucle"* et deux erreurs $(F - Call - Error)$ *"Erreur dans l'appel de la fonction"*.

```
{'type': 'update', 'path': ['Module', 'For[0]', 'Condition:[0]', 'Var: i[0]'], 'current': 'Var: i', 'new': 'Var: k'},
{'type': 'update', 'path': ['Module', 'For[0]', 'Condition:[0]', 'Call: range[1]', 'Const: 4[0]'], 'current': 'Const: 4', 'new': 'Const: 3'},
{'type': 'match', 'path': ['Module', 'For[0]', 'Condition:[0]', 'Call: range[1]'], 'current': 'Call: range', 'new': 'Call: range'},
{'type': 'match', 'path': ['Module', 'For[0]', 'Condition:[0]'], 'current': 'Condition:', 'new': 'Condition:'},
{'type': 'match', 'path': ['Module', 'For[0]', 'Body:[1]', 'Call: droite[0]', 'Const: 2[0]'], 'current': 'Const: 2', 'new': 'Const: 2'},
{'type': 'match', 'path': ['Module', 'For[0]', 'Body:[1]', 'Call: droite[0]'], 'current': 'Call: droite', 'new': 'Call: droite'},
{'type': 'insert', 'path': ['Module', 'For[0]', 'Body:[1]', 'Call: bas[1]', 'Const: 1[0]'], 'current': None, 'new': 'Const: 1'},
{'type': 'insert', 'path': ['Module', 'For[0]', 'Body:[1]', 'Call: bas[1]'], 'current': None, 'new': 'Call: bas'},
{'type': 'match', 'path': ['Module', 'For[0]', 'Body:[1]'], 'current': 'Body:', 'new': 'Body:'},
{'type': 'match', 'path': ['Module', 'For[0]'], 'current': 'For', 'new': 'For'},
{'type': 'delete', 'path': ['Module', 'Call: bas[1]', 'Const: 2[0]'], 'current': 'Const: 2', 'new': None},
{'type': 'delete', 'path': ['Module', 'Call: bas[1]'], 'current': 'Call: bas', 'new': None},
{'type': 'match', 'path': ['Module'], 'current': 'Module', 'new': 'Module'}
```

Listing 3 – Liste des opérations d'édition pour passer de l'AST de l'élève à l'AST du code correct le plus proche. Les opérations surlignées définissent des patterns qui permettent d'identifier les erreurs `LO-FOR-Number-Iteration-Error-Under2`,`LO-Body-Misplaced`.

5 Conclusion et perspectives

Dans cette contribution nous proposons deux résultats principaux :

- Une typologie des erreurs en programmation, dans le contexte des premiers apprentissages, basée sur un référentiel de types de tâches de programmation ;
- Un processus, et une librairie associé, permettant d'analyser un code erroné, au regard d'un code correct, et d'identifier les erreurs de la typologie présentes dans le code erroné.

L'exploitation de cette typologie dans la modélisation et la production de rétroaction est développée dans (Jolivet et al., 2025).

Enfin, si dans notre contexte ce travail vise à contribuer à la décision de rétroactions dans l'environnement AlgoPython, il est très certainement exploitable et adaptable dans d'autres contextes.

Ainsi, la typologie proposée est adaptable soit, par exemple, en réduisant le niveau de finesse pour un public d'apprenants plus expert, ou en introduisant des erreurs spécifiques à un autre contexte. Elle peut aussi être exploitée pour des finalités diverses, comme la recommandation d'activités adaptées, l'évaluation et la notation automatiques ou encore l'élaboration d'indicateurs comportementaux à destination des enseignants pour les aider dans leurs stratégies de remédiation. Ce travail pourrait également être utile à des chercheurs qui s'intéressent à l'apprentissage et à l'enseignement de la programmation pour un public débutant, ainsi qu'à la compréhension des phénomènes cognitifs sous-jacents.

De même, le processus proposé autour de l'analyse des AST de différents codes pourrait être déployé et adapté dans différents contextes, par exemple la comparaison de différents codes d'élèves, éventuellement tous corrects, pour mettre en exergue les différences et la diversité des stratégies souvent possibles dans un contexte de programmation. Il s'agirait alors de s'appuyer sur une typologie de différences et non pas une typologie d'erreurs.

Références

Astolfi, J.-P. (1997). *L'erreur, un outil pour enseigner*. Pratiques et enjeux pédagogiques. ESF éditeur. Publisher : ESF Paris.

Ben-Yaacov, A. and Hershkovitz, A. (2022). Types of Errors in Block Programming : Driven by Learner, Learning Environment. *Journal of Educational Computing Research*, page 073563312211023.

Broisin, J. and Hérouard, C. (2019). Design and evaluation of a semantic indicator for automatically supporting programming learning. In *12th International Conference on Educational Data Mining (EDM 2019)*, pages 270–275.

Brooks, C., Carroll, A., Gillies, R. M., and Hattie, J. (2019). A matrix of feedback for learning. *Australian Journal of Teacher Education (Online)*, 44(4) :14–32.

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) :1–21.

Jolivet, S., Dechaux, E., Gobard, A.-C., and Wang, P. (2023). Construction et exploitation d'un référentiel de types de tâches d'apprentissage de la programmation. In Iksal, S., Pélissier, C., Gilliot, J.-M., and Marzin-Janvier, P., editors, *Actes du colloque EIAH2023 : 11ème Conférence sur les Environnements Informatiques pour l'Apprentissage Humain*, pages 166–177, Brest.

Jolivet, S., Kirouchenassamy, B., Yessad, A., and Luengo, V. (2025). Modélisation et décision de rétroactions épistémiques dans l'environnement AlgoPython. In *Acte de la 10e conférence sur les Environnements Informatiques pour l'Apprentissage Humain, EIAH25*, Lille.

Lazar, T., Možina, M., and Bratko, I. (2017). Automatic extraction of ast patterns for debugging student programs. In *Artificial Intelligence in Education : 18th International Conference, AIED 2017, Wuhan, China, June 28–July 1, 2017, Proceedings 18*, pages 162–174. Springer.

Le Goues, C., Pradel, M., and Roychoudhury, A. (2019). Automated program repair. *Communications of the ACM*, 62(12) :56–65.

McCall, D. and Kölling, M. (2019). A New Look at Novice Programmer Errors. *ACM Transactions on Computing Education*, 19(4) :1–30.

Piech, C., Sahami, M., Koller, D., Cooper, S., and Blikstein, P. (2012). Modeling how students learn to program. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 153–160.

Shen, Z. and Chen, S. (2020). A survey of automatic software vulnerability detection, program repair, and defect prediction techniques. *Security and Communication Networks*, 2020(1) :8858010.

Small, M. and Lin, A. (2018). Instructional feedback in mathematics. In Lipnevich, A. and Smith, J., editors, *The Cambridge handbook of instructional feedback*, pages 169–190. Cambridge University Press.

Yoshizawa, Y. and Watanobe, Y. (2018). Logic Error Detection Algorithm for Novice Programmers based on Structure Pattern and Error Degree. In *2018 9th International Conference on Awareness Science and Technology (iCAST)*, pages 297–301, Fukuoka. IEEE.

Zhang, K. and Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing*, 18(6) :1245–1262. Publisher : SIAM.

6 Annexe 1

Cette annexe présente l'intégralité des erreurs identifiées classifiées selon la structuration du référentiel de types de tâches.

ERREUR	DETAILS OU EXEMPLES	TYPE(S) DE TÂCHE
VARIABLE		
(VA-Declaration-Initialization-Error) Erreur de déclaration et initialisation variable	Mauvais choix du type dans la déclaration de la variable ou dans la valeur lors de l'initialisation	VA. Produire une variable (et ses fils)
(VA-Missing-Increment) Oubli incrémentation variable	Pour un compteur par exemple ; soit la variable n'est pas modifiée, soit elle est réinitialisée à chaque fois	—
(VA-Invalid-Name) Erreur dans choix du nom de la variable	Caractère interdit, mot réservé	VA. Choisir un nom valide (ne pas utiliser caractère interdit, nom réservé...)
INSTRUCTION CONDITIONNELLE		
(CS-Incorrect-Number-Branches) Erreur nombre de branches IC	Oubli d'un cas ou mauvais recouvrement des branches	IC. Déterminer les branches nécessaires pour une IC IC. Déterminer l'ensemble des expressions de manière à définir une partition de l'ensemble des cas d'une instruction conditionnelle
(CS-Body-Error) Erreur dans les instructions d'une IC	Une instruction est manquante ou erronée	IC. Déterminer les instructions à exécuter pour chaque branche d'une instruction conditionnelle
(CS-Body-Misplaced) Erreur dans le corps de l'IC : Instruction manquante et présente ailleurs dans le programme	Une instruction ne se trouve pas dans la bonne branche ou se trouve en dehors de l'IC	IC. Déterminer les instructions à exécuter pour chaque branche d'une instruction conditionnelle
(CS-Missing) Instruction conditionnelle manquante	Absence de l'IC	IC. Produire une IC
FONCTIONS		
(F-Definition-Error-Arg) Erreur définition de la fonction sur les paramètres	Liée aux consignes de l'énoncé ou à la gestion possible avec le reste du code : - oubli de paramètres ou mauvais choix de paramètres - erreur sur les préconditions des paramètres	F. Déterminer les paramètres (noms, types, valeurs par défaut) d'une fonction F. Déterminer les préconditions d'exécution (par exemple : tableau trié dans une recherche dichotomique...)

(suite page suivante)

ERREUR	DETAILS OU EXEMPLES	TYPE(S) DE TÂCHE
(F-Definition-Error-Return) Erreur définition de la fonction sur les valeurs retournées	Liée aux consignes de l'énoncé ou à la gestion possible avec le reste du code : - erreur sur ce qui est retourné ou non	F. Déterminer, le cas échéant, les éléments à retourner par une fonction
(F-Call-Error) Erreur dans l'appel de la fonction	Appel avec les mauvais paramètres ou pas d'affectation du résultat de la fonction	—
(F-Name-ParameterName-Error) Erreur nom de la fonction ou des paramètres	Caractère interdit, mot réservé	F. Déterminer le nom d'une fonction F. Déterminer les paramètres (noms, types, valeurs par défaut) d'une fonction
(F-Definition-Missing) Definition d'une fonction absente	Absence d'une fonction	F. Produire une fonction
BOUCLES		
(LO-Iterator-Usage-Error) Erreur dans l'utilisation de l'itérateur de la boucle	L'apprenant s'est trompé dans les valeurs prises par l'itérateur (valeur de départ par exemple)	B. Déterminer si une (ou plusieurs) instruction(s) du corps d'une boucle bornée utilise(nt) l'itérateur
(LO-FOR-Number-Iteration-Error-Under2) Erreur nombre d'itérations de la boucle FOR (à un près)	La boucle s'exécute une fois de trop ou une fois de moins	B. Déterminer le nombre d'itérations à réaliser pour une boucle bornée
(LO-FOR-Number-Iteration-Error) Erreur nombre d'itérations de la boucle FOR (>2)	La boucle ne s'exécute pas le bon nombre de fois	B. Déterminer le nombre d'itérations à réaliser pour une boucle bornée
(LO-WHILE-Number-Iteration-Error-Under2) Erreur nombre d'itérations de la boucle WHILE (à un près)	La boucle s'exécute une fois de trop ou une fois de moins	B. Déterminer le nombre d'itérations à réaliser pour une boucle bornée
(LO-WHILE-Number-Iteration-Error) Erreur nombre d'itérations de la boucle WHILE (>2)	La boucle ne s'exécute pas le bon nombre de fois	B. Déterminer le nombre d'itérations à réaliser pour une boucle bornée
(LO-Exit-Condition-Error-Start) Erreur condition d'arrêt de la boucle : départ	Erreur liée à la condition de départ	B. Déterminer la condition d'arrêt d'une boucle non bornée
(LO-Exit-Condition-Error-Evolution) Erreur condition d'arrêt de la boucle : évolution	Erreur liée à une instruction qui n'est pas faite pour arrêter la boucle	B. Déterminer l'instruction modifiant la valeur de la condition d'arrêt d'une boucle non bornée

(suite page suivante)

ERREUR	DETAILS OU EXEMPLES	TYPE(S) DE TÂCHE
(LO-Body-Missing-Not-Present-Anywhere) Erreur dans le corps de la boucle : Instruction manquante et non présente	Instruction absente dans le corps de la boucle et dans le programme	B. Déterminer les instructions à exécuter à chaque itération d'une boucle bornée B. Déterminer les instructions à exécuter à chaque itération pour une boucle non bornée
(LO-Body-Misplaced) Erreur dans le corps de la boucle : Instruction manquante et présente ailleurs dans le programme	Instruction présente ailleurs dans le programme	B. Déterminer les instructions à exécuter à chaque itération d'une boucle bornée B. Déterminer les instructions à exécuter à chaque itération pour une boucle non bornée
(LO-Body-Error) Erreur dans le corps de la boucle : Instruction erronée	Instruction présente proche de celle manquante	B. Déterminer les instructions à exécuter à chaque itération d'une boucle bornée B. Déterminer les instructions à exécuter à chaque itération pour une boucle non bornée
(LO-For-Missing) Boucle FOR manquante	Absence d'une boucle FOR	B. Produire une boucle bornée
(LO-While-Missing) Boucle WHILE manquante	Absence d'une boucle WHILE	B. Produire une boucle non bornée
EXPRESSIONS		
(EXP-Error-Conditional-Branch) Erreur dans l'expression booléenne d'une des branches d'une IC	Test si une variable est $<$ au lieu de $\leq$ par exemple	IC. Déterminer l'expression booléenne correspondant à une branche d'une instruction conditionnelle
(EXP-Error-Assignment-Variable) Erreur dans l'expression affectée à une variable	Oubli d'une partie de l'expression, ou mauvais choix du signe	VA. Affecter la valeur d'une expression à une variable
PROGRAMME ET ALGORITHME		
(PA-Problem-Decomposition-Algorithmmic-Strategy) Erreur de décomposition du problème ou mauvaise stratégie algorithmique	Le programme ne fait pas appel aux structures de contrôle attendues	P. Concevoir un algorithme permettant de réaliser une tâche t A. Décomposer la tâche en sous-tâches t1…tn A. Déterminer les instructions (affectations et structure de contrôles) A. Ordonner et articuler les instructions (séquentialité et imbrication)
(PA-Misunderstanding-Task-Instructions) Erreur liée à la compréhension de la consigne	Programme correct mais fait complètement autre chose que ce qui est demandé (exemple : exécute le code de l'exemple donné)	—
(PA-Incomplete) Programme non terminé	Morceau de programme correct mais il manque une succession d'instructions	A. Déterminer les instructions (affectations et structure de contrôles)

(suite page suivante)

ERREUR	DETAILS OU EXEMPLES	TYPE(S) DE TÂCHE
(PA-Not-Optimized) Programme non optimisé	Fait la tâche demandée, mais de manière plus longue (redondance, non utilisation de boucles ou de fonctions)	P. Optimiser un programme A. Modifier un algorithme existant pour répondre à des contraintes particulières

Actes du 8e Atelier "Apprentissage de la pensée informatique de la maternelle à l'Université (APIMU 1000)" @ EIAH 2025
Villeneuve d'Ascq, France, 10 juin 2025

Conception et analyse d'un instrument non genré pour un apprentissage de la programmation Python à la transition collège-lycée

Christophe Declercq[1] Sébastien Hoarau[1]
(1) LIM, Université de La Réunion
christophe.declercq@univ-reunion.fr, seb.hoarau@univ-reunion.fr

RÉSUMÉ

Nous présentons la conception d'un instrument d'apprentissage de la programmation par blocs en Python dédié à la programmation de dessins vectoriels pour la découpe laser. Nous justifions les choix de conception par l'objectif de créer un environnement non genré et adapté à la transition collège lycée. Nous montrons que l'environnement obtenu allie la simplicité d'usage de Turtle et la puissance du format SVG. Nous détaillons ensuite la première expérimentation de cet environnement, lors d'un stage de 3e réservé aux collégiennes, avec un scénario d'apprentissage dédié. L'analyse de l'activité des élèves permet de montrer leur engagement dans l'activité de programmation créative. Cela permet de valider la pertinence de l'instrument proposé dans ce contexte. L'analyse permet aussi de proposer des améliorations à la fois au scénario d'apprentissage et à l'instrument lui-même. Nous concluons avec des perspectives de recherche et de nouveaux usages de l'instrument proposé.

ABSTRACT

Design and analysis of an instrument for gender-neutral learning of Python programming at the middle school-high school transition

We present the design of a Python block programming learning tool dedicated to programming vector drawings for laser cutting. We justify the design choices by the objective of creating a gender-neutral environment adapted to the middle school-high school transition. We show that the resulting environment combines the ease of use of Turtle and the power of the SVG format. We then detail the first experimentation of this environment, during a 3rd year internship reserved for middle school girls, with a dedicated learning scenario. The analysis of the students' activity shows their engagement in the creative programming activity. This validates the relevance of the proposed tool in this context. The analysis also makes it possible to propose improvements to both the learning scenario and the tool itself. We conclude with research perspectives and new uses of the proposed instrument.

MOTS-CLÉS : Apprentissage de la programmation, Python, Analyse de traces, Programmation créative, Genre et informatique.

KEYWORDS: Learning Programming, Python, Trace Analysis, Creative Programming, Gender and computing.

1 Introduction

Ce travail a pris sa source dans la proposition de l'université de Lille (Bolka-Tabary et al., 2020; Everaere et al., 2024) de stages d'observation pour des collégiennes de troisième, destinés à leur faire découvrir l'informatique en déconstruisant l'image de cette discipline par des activités de programmation créative. Le choix de proposer des activités créatives pour l'apprentissage de la programmation permet d'éviter la programmation de jeux souvent liée à la figure du "geek" ou la résolution d'énigmes cryptographiques trop liée à l'image du "hacker". La prévention des stéréotypes de genre incite à proposer un instrument tenant compte des biais de genre identifiés par I. Collet (Collet, 2020) qui montre, pour la même tâche, une réussite différente pour les filles et les garçons selon qu'elle est présentée comme un test de géométrie ou de dessin. F. Tort montre aussi une différence de réussite, selon le genre, aux tâches du concours Castor selon leur présentation (Tort et al., 2017). Les activités créatives offrent une grande variété de réalisations possibles, sans biais de genre connu. Nous avons ainsi choisi de proposer un instrument de dessin vectoriel pouvant déboucher sur la production d'un objet tangible par découpe et gravure laser.

Ayant auparavant conçu et expérimenté, dans le cadre de la transition collège-lycée (Declercq and Nény, 2020) un environnement de programmation par blocs en python, qui avait montré son intérêt pour faciliter la transition d'un langage par blocs (Scratch) à un langage textuel (Python), nous avons choisi de lui ajouter une catégorie d'instructions élémentaires dédiées au dessin vectoriel. Ce choix était aussi destiné à montrer que les filles sont capables d'apprendre le langage Python dès la classe de 3e, leur donnant ainsi l'occasion d'une découverte en avance sur le programme de la classe de seconde. Le défi à relever était que l'interface de programmation de dessin en Python (module `turtle`) inspirée de la tortue Logo génère des images matricielles, alors que la commande d'une machine à découpe laser nécessite comme format d'entrée une image vectorielle au format SVG (*Scalable Vector Graphics*).

Nous présentons successivement la conception de notre instrument – a priori non genré – pour un apprentissage de la programmation Python, puis l'analyse de son usage dans le cadre d'un stage proposé aux collégiennes de l'académie de La Réunion en décembre 2024. Ce stage a été proposé et choisi en tant que stage d'observation par quinze collégiennes de l'académie.

2 Conception d'un instrument de dessin vectoriel en programmation par blocs

Le micro-monde de la tortue Logo de Seymour Papert (Papert, 1989) a depuis longtemps montré son intérêt pour l'apprentissage de la programmation. Nous voulons maintenant montrer qu'il a un intérêt particulier pour la commande de machines à découpe laser, car son paradigme repose sur la génération d'un trait continu suivant les déplacements programmés de la tortue. C'est précisément un trait continu qui est nécessaire pour effectuer une découpe.

2.1 Une interface inspirée de Turtle pour générer une image SVG

Le module `turtle` distribué avec le langage Python, génère au fur et à mesure une image matricielle dans un `canvas` en utilisant la bibliothèque `tkinter`. Les lignes sont approximées par des suites

de pixels et les cercles sont approximés par des polyèdres réguliers. L'export dans un format d'image matricielle ne permet pas de commander une machine à découpe laser car les suites de pixels produisent des lignes discontinues. La solution la plus simple à ce problème est de générer directement des figures vectorielles au format SVG à partir des instructions de déplacement de la tortue en remplaçant le module `turtle` par un module dédié. La tortue possède à tout moment une position donnée par son abcisse et son ordonnée dans le repère de l'image (point 0,0 en haut à gauche), ainsi qu'une orientation ('heading' en anglais) donnée par un angle de 0 à 360 degrés.

— L'orientation de la tortue peut être contrôlée par l'instruction `heading` qui fixe le cap de manière absolue, ou par l'instruction `rotate` qui tourne la tortue d'un angle donné relativement à son orientation courante.
— Le déplacement de la tortue peut être contrôlé par l'instruction `moveto` qui la téléporte à une position donnée sans dessiner, ou par l'instruction `forward` qui la fait avancer en dessinant une ligne, c'est à dire en générant un chemin de type ligne dans le SVG.
— Le déplacement en courbe est contrôlé par l'instruction `turn` qui fait parcourir à la tortue un arc de cercle de rayon donné et d'angle donné. Cette instruction génère un chemin en arc dans le SVG.
— Le contrôle d'une machine à découpe et gravure laser nécessite de pouvoir choisir les portions de dessin à découper et celles à graver. Les instructions de la tortue usuelle permettant de descendre et remonter le stylo ont été remplacées par les instructions `draw_on`, `draw_off` et `cut_on`, `cut_off` permettant de contrôler respectivement la gravure et la découpe.

La traduction de toutes ces instructions est directe en SVG sauf pour l'instruction `turn` qui nécessite quelques calculs géométriques. En effet, l'instruction `turn` génère un arc partant de la position courante selon la direction courante, tournant d'un angle donné selon un arc de rayon donné. L'arc en SVG est quant à lui défini par la position de départ, la position d'arrivée, le rayon de l'arc et deux indicateurs précisant le sens de l'arc et le choix entre le petit et le grand arc (voir figure 1). La position d'arrivée de l'arc doit donc être calculée après avoir calculé son centre pour pouvoir générer l'arc en SVG.

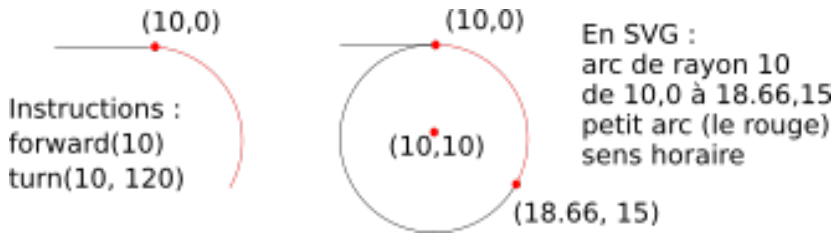


FIGURE 1 – Traduction de l'instruction turn en arc SVG

L'écriture en Python des primitives graphiques a été réalisée dans l'environnement Brython (Quentel, 2020), un interprète Python dans le navigateur fournissant des accès en consultation et en création au contenu de la page web courante. Brython a permis de simplifier l'écriture des primitives graphiques en générant directement le SVG dans le document courant.

2.2 Intégration à un environnement de programmation par blocs en Python

L'instrument conçu spécialement pour la découpe laser est une adaptation de l'environnement de programmation Block2Py, et est disponible en ligne à l'adresse : `https://iremi974.gitlab.io/block2py/laser.html`

L'interface (voir figure 2) comporte un espace de travail au milieu où la programmeuse vient assembler ses blocs de programme, un menu à gauche pour sélectionner les blocs, et un espace de rendu à droite où est visualisée l'image vectorielle lors de l'exécution du programme. Toutes les instructions spécifiques pour le dessin vectoriel sont rassemblées dans le menu "instructions de dessin". Les autres menus permettent d'accéder aux instructions simples (`print` et affectations) aux instructions composées (`for`, `if` et `while`) et aux expressions et variables de différents types (entier, booléen et chaînes). L'ensemble constitue un sous-ensemble du langage Python suffisant pour les programmeurs débutants, avec une interface de saisie prévenant les erreurs de syntaxe et de types.

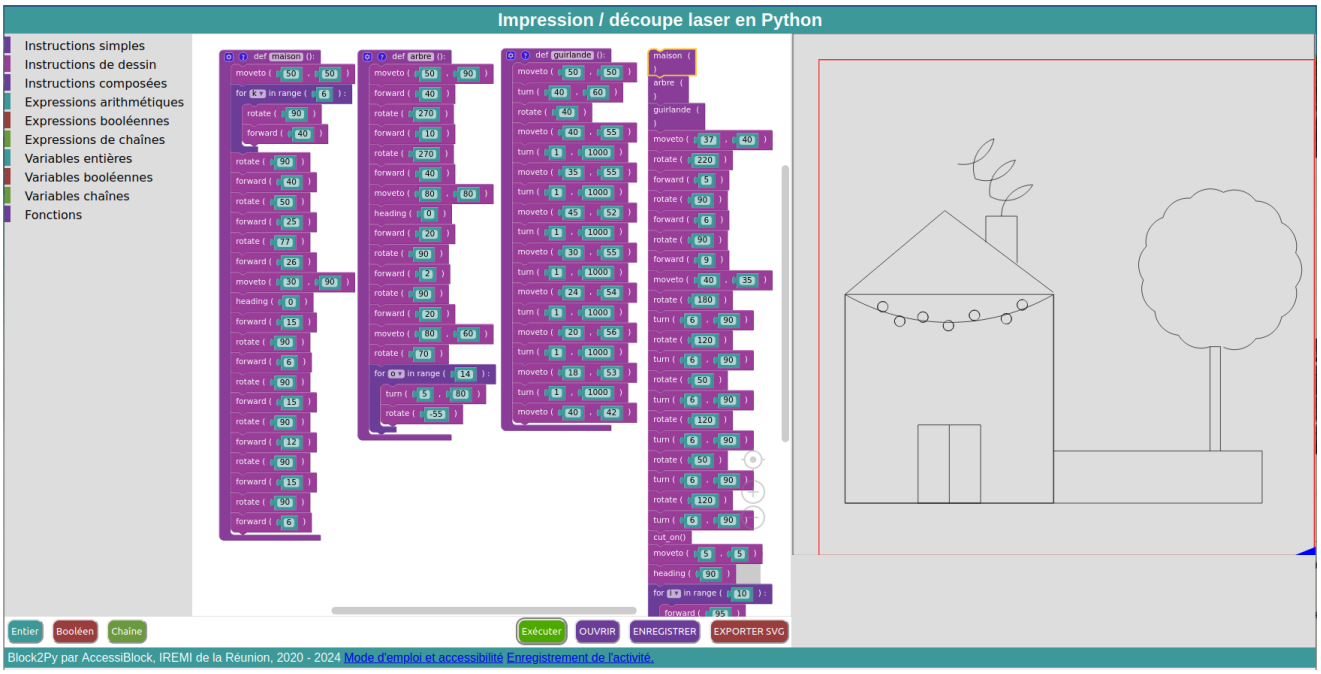


FIGURE 2 – Interface de l'environnement Block2Py/Laser

La figure 2 montre un programme écrit par le binôme numéro 2 lors du stage avec le dessin vectoriel en résultant.

3 Expérimentation lors du stage "Informatique au féminin"

La première expérimentation de l'instrument Block2Py/Laser a eu lieu pendant le stage d'observation de troisième qui s'est déroulé du lundi 16 décembre au jeudi 19 décembre 2024 à la Faculté des Sciences et Technologies de l'Université de La Réunion. Les activités de programmation ont été planifiées sur quatre demi-journées (une par jour) du lundi après-midi au jeudi matin pour 15 collégiennes originaires de deux collèges de la ville de Saint-Denis : le collège des Mascareignes et le collège Mahé de La Bourdonnais. Les séances de 2h se sont toutes déroulées avec ce même environnement et

l'ensemble des activités ont été enregistrées automatiquement pour analyse ultérieure des traces. Une dernière séance le jeudi après-midi a permis les découpes laser.

3.1 Scénario pédagogique et déroulement

La progression que nous avons proposée s'inspire de la méthode PRIMM (Predict-Run-Investigate-Modify-Make) (Sentance and Waite, 2017) et propose successivement des programmes fournis permettant de découvrir le sens des instructions simples qu'ils contiennent, puis des programmes à modifier (voir figure 3) et enfin des idées de création.

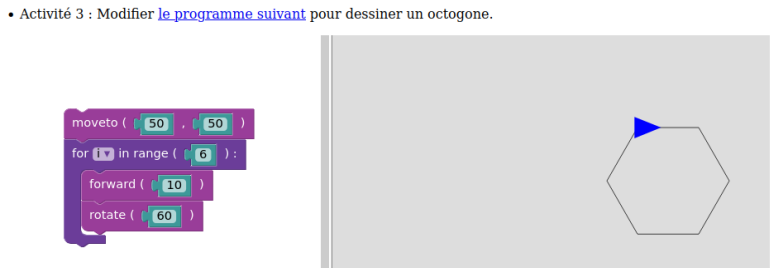


FIGURE 3 – Une tâche de type "Modify"

L'ensemble de la progression comporte une vingtaine de tâches réparties en quatre séquences : la première consacrée à la découverte des instructions de dessin et de la boucle `for`, la seconde à la réalisation de frises et de motifs, la troisième à la réalisation de motifs imbriqués de taille variable, et la dernière à la création libre d'une oeuvre personnelle pour découpe laser. Les tâches d'apprentissage sont toutes de complexité limitée, aucun des programmes proposés pour évaluation ou modification ne dépassant vingt lignes de code. Deux temps d'institutionnalisation ont été aménagés pour préciser les concepts abordés à savoir les boucles et la notion de fonction.

3.2 Analyse de l'activité des élèves

L'activité des élèves a été enregistrée par l'application dans le navigateur (`localstorage`). A chaque exécution, le texte du programme a été enregistré ainsi que l'heure d'exécution. Les graphiques (voir figure 5 et 6) en annexe montrent pour chaque instant d'exécution, le nombre de lignes du programme qui vient d'être exécuté. Un code couleur permet de plus de savoir si ce programme contient une ou plusieurs boucles et une ou plusieurs fonctions.

La grande quantité d'informations enregistrées, plus de 1000 exécutions de programmes en moyenne par binôme pendant la durée du stage (10h), montre un engagement important des collégiennes dans l'activité. Selon les binômes, cela représente une à trois exécutions par minute en moyenne. On constate aussi une complexité importante des programmes écrits, leur taille maximum variant de 70 à 190 lignes, alors que les programmes fournis ne dépassaient pas 20 lignes.

Une analyse plus détaillée, en consultant les programmes écrits, permet de relativiser ces nombres et de les expliquer. L'évolution temporelle des nombres de lignes de programme exécutés montrent à de nombreuses reprises des progressions linéaires rapides (voir par exemple binôme 1 mardi, binôme 2 lundi, binôme 4 jeudi, binôme 5 mardi). Dans ces quatre cas, ces courbes sont caractéristiques

de stratégies de programmeur pas à pas (Declercq and Tort, 2018) : l'élève repète le schème "ajout d'une instruction / exécution", jusqu'à obtenir la figure souhaitée. Les instructions ajoutées sont des instructions simples en séquence. Dans le cas du binôme 4, on constate un maintien de cette stratégie jusqu'au dernier jour, avec une amélioration de l'anticipation, car les instructions ne sont plus ajoutées une par une mais par paquets de quatre ou cinq.

L'usage des fonctions semble provoqué par la complexité (en terme de nombre de lignes) des programmes. Pour le binôme 1, c'est à partir de 83 lignes, pour le binôme 2 à partir de 64. Cette analyse doit cependant rester prudente, car l'usage de fonctions a pu être recommandé par les encadrant.e.s, lorsque l'élève commençait à avoir des difficultés avec un programme trop long. Globalement ce sont six binômes sur huit, qui ont utilisé les fonctions de manière pertinente pour nommer des parties de programme dans leur projet de création. Aucun n'a utilisé de fonction avec paramètres. On a pu constater une instrumentalisation du bloc fonction pour y ranger les instructions à ne pas exécuter, lors de la mise au point d'un programme en pas à pas.

Concernant l'usage des boucles dans les créations, il semble essentiellement lié à la nature des dessins. Tous les dessins répétitifs par nature (tous sauf le papillon, voir figure 4) ont bien été programmés en utilisant la boucle `for` de Python. On note des usages pertinents de la boucle dans six projets de création sur huit.

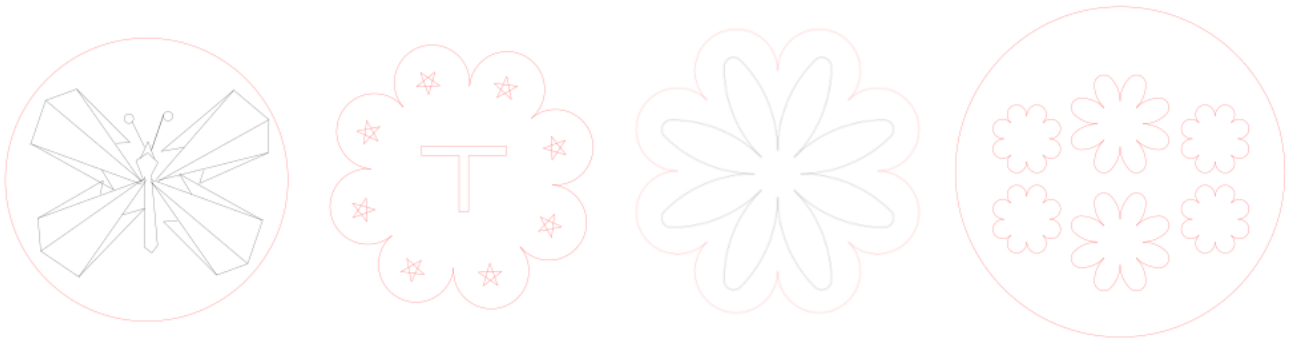


FIGURE 4 – Quelques réalisations originales

Globalement, parmi les huit binômes, quatre ont montré leur maîtrise de la boucle et des fonctions sans paramètres, deux de la boucle seulement et deux des fonctions seulement. Les deux mécanismes ont été utilisés principalement pour maîtriser la complexité des dessins visés, la boucle pour des dessins répétitifs, et la fonction pour des dessins séquentiels trop longs. Les réalisations ont été jugées originales, car elles diffèrent largement des exemples de programmes fournis. Elles peuvent en reprendre certaines parties en les combinant différemment dans un programme plus complexe.

Concernant l'environnement utilisé, son appropriation nous a semblé rapide. La plupart des collégiennes avaient déjà utilisé Scratch, la transition vers Python s'est effectuée en douceur, aucune remarque n'a été formulée sur le fait que les instructions étaient écrites en anglais. Les difficultés rencontrées avec les instructions de base (en particulier `rotate` et `turn`), semblent hériter de lacunes en géométrie. Des feuilles de papier millimétré avec une rose des vents imprimée, ont pu pallier ces difficultés. On a pu ainsi observer deux stratégies pour le dessin d'une forme élémentaire incluant des lignes et des courbes (par exemple un coeur) : dessin sur papier pour noter les longueurs les rayons et les angles, ou tatonnement avec l'environnement informatique.

La découpe des dessins conçus par programmation n'a posé aucune difficulté, les fichiers SVG générés

par Block2Py/Laser étant directement acceptés par la découpeuse laser mise à notre disposition. Les seules surprises apparues au moment de la découpe ont été provoquées par des confusions entre les traits devant être coupés et ceux devant être gravés.

Le fait que les quinze collégiennes aient pu, après 10h d'activité, toutes repartir de leur stage avec un objet découpé selon le programme qu'elles avaient créé, nous suffit pour démontrer que l'outil proposé est pertinent dans le contexte pour lequel il a été créé. Il reste cependant perfectible comme le scénario d'apprentissage utilisé.

3.3 Retours sur le scénario d'apprentissage

Par rapport au scénario initial envisagé en quatre séquences, l'activité des collégiennes a été plus rapide que prévu. En effet, l'ensemble des tâches étant présentes sur le serveur de la salle où avait lieu le stage, la plupart d'entre elles ont rapidement réalisé la vingtaine de tâches prévues, avant de s'engager dès la fin du premier jour dans l'activité de création. Les questions des collégiennes les jours suivants ont montré qu'elles n'avaient pas nécessairement pris le temps de comprendre chacune des tâches introductives avant de passer à leur projet de création. Nous avons alors pris le temps d'y remédier dans les séances suivantes en particulier lors des deux temps d'institutionnalisation dédiés respectivement à la boucle et à la notion de fonction. Nous avons laissé beaucoup d'autonomie aux collégiennes et les avons encouragées pour leur projet de création qui était manifestement plus source de motivation qu'une tâche pouvant sembler scolaire.

Au vu de l'analyse de leur activité, le réinvestissement des activités introductives dans leur projet de création nous semble tout à fait satisfaisant pour les deux premières séquences consacrées aux boucles et aux frises. Par contre la troisième séquence, portant sur l'imbrication de motifs de taille variable, n'a été réinvestie par aucune des collégiennes. En effet, aucune variable n'a été définie dans leurs projets, ni aucun paramètre dans leurs fonctions. Cette partie du scénario pourrait donc être supprimée sans que cela ne nuise aux réalisations possibles. Le temps libéré pourrait être utilement mis à profit pour mieux comprendre les primitives géométriques en systématisant le passage du dessin sur papier au dessin sur ordinateur. On pourrait ainsi réorganiser le scénario autour de quatre séquences, la première dédiée aux instructions géométriques de base, la seconde aux boucles et aux frises, la troisième aux fonctions et la dernière à la création.

3.4 Préconisations d'évolutions de l'instrument

La principale évolution nécessaire de l'instrument concerne la mise en place d'un mécanisme d'exécution pas à pas. Le fait que les collégiennes soient restées dans la posture de "programmeuse pas à pas" montre leur besoin d'un retour instrumental sur l'exécution de chaque instruction. On peut espérer, avec un mode d'exécution pas à pas, que les programmeuses oseront ajouter plusieurs instructions d'un coup, sachant que l'exécution pourra leur donner un retour sur chacune. La mise en place d'une exécution en pas à pas dans l'environnement Blockly est facilitée par la structure de blocs du langage : en effet il est possible de visualiser l'instruction en cours d'exécution tout en affichant son effet sur le curseur ou le dessin. Ce pas à pas devra être mis en place seulement au niveau instruction - et pas au niveau des expressions - pour éviter une exécution trop longue et fastidieuse. Il devrait aussi permettre une meilleure compréhension de l'ordre d'exécution des instructions dans le cas d'un appel de fonction.

Le travail du chercheur sur l'analyse de trace pourra être facilité par la généralisation de la trace à tous les événements y compris les actions `ouvrir`, `enregistrer` et `exporter svg`. En effet seule l'action `exécuter` était tracée dans la version actuelle, ce qui a obligé à comparer les programmes exécutés à tous ceux qui avaient été fournis dans le scénario, pour savoir si le programme exécuté avait été donné par l'enseignant ou créé par l'élève.

Enfin, tout en cherchant à garder une interface minimaliste, l'instrument gagnerait à être doté d'une action `importer` permettant d'ajouter un autre programme à l'intérieur d'un programme en cours d'édition. Le besoin est apparu lorsque des collégiennes ont voulu réutiliser un dessin déjà programmé à l'intérieur de leur projet en cours.

4 Conclusions

Nous avons conçu et proposé un nouvel instrument dédié à l'apprentissage de la programmation dans un contexte de transition collège lycée en adoptant une approche non genrée. L'expérimentation de son usage dans le contexte d'un stage non mixte nous semble démontrer que l'outil est pertinent dans ce contexte. L'engagement des collégiennes et leur réussite à créer des objets par programme en est la preuve tangible.

L'objectif du stage était de leur donner de l'informatique l'image d'une discipline accessible dans laquelle elles peuvent réussir. Pour connaître l'impact à moyen terme de ce stage et en particulier des instruments utilisés, il faudrait pouvoir évaluer l'influence des acquis en programmation sur leur réussite ultérieure en particulier en classe de SNT (Sciences Numériques et Technologie) en seconde, et l'influence de l'expérience de stage sur leur représentation des métiers de l'informatique et leurs souhaits d'orientation. Cette recherche reste à mener et est critique pour mieux comprendre les stéréotypes de genre et en corriger les effets néfastes.

Concernant l'instrument "Block2Py/Laser", nous prévoyons de l'expérimenter dans d'autres contextes en classe de troisième et en classe de seconde pour continuer à étudier la transition de la programmation par blocs à la programmation Python. Cette étude devra inclure, outre les traces d'exécution, l'analyse des interactions entre élèves, des interventions de l'enseignant et toutes autres variables pouvant influer sur les apprentissages. L'instrument a dès l'origine été conçu en évitant a priori tout biais de genre. Pour démontrer qu'il est effectivement exempt de tels biais, il reste à mener une étude à plus large échelle permettant de mesurer l'engagement et la réusite d'élèves garçons et filles en groupe mixte et non mixte pour tester l'indépendance de l'engagement et de la réussite par rapport au genre.

Nous avons aussi pour perspective d'étudier l'usage du dessin vectoriel et de la découpe laser à d'autres niveaux de classe en programmation textuelle avec Python. Dans cet objectif nous avons publié une bibliothèque `laser-turtle` disponible sous `Pypi` à l'adresse `https://pypi.org/project/laser-turtle/`. Cette bibliothèque est utilisable à différents niveaux d'enseignement – en particulier en terminale et premier cycle universitaire – pour l'apprentissage de la récursivité.

Références

Bolka-Tabary, L., Pupin, M., and Secq, Y. (2020). Stage d'observation de troisième : "Wi Code, Wi Build". In *Didapro 8 – DidaSTIC*, Lille, France.

Collet, I. (2020). Genre et numérique : pratiques égalitaires, dispositifs inclusifs. conférence invitée. In *Didapro 8 – DidaSTIC*, Lille, France.

Declercq, C. and Nény, F. (2020). Block2Py, un éditeur de blocs pour l'apprentissage du langage Python. In *Didapro 8 – DidaSTIC*, Lille, France.

Declercq, C. and Tort, F. (2018). Organiser l'apprentissage de la programmation au cycle 3 avec des activités guidées et/ou créatives. In *RJC EIAH 2018*, Besançon, France.

Everaere, P., Marquet, P., Pupin, M., and Secq, Y. (2024). Dispositifs pour favoriser la venue des adolescentes dans les filières informatiques. In *Diversité, Réussite[s] dans l'Enseignement Supérieur (2024)*, Nantes, France. Nantes Université.

Papert, S. traduit par Vassallo-Villaneau, R. (1989). *Jaillissement de l'esprit : ordinateurs et apprentissage*. Flammarion.

Quentel, P. (2020). Brython, Une implémentation de Python 3 pour la programmation web côté client. `https://brython.info/index.html`. Consulté le : 30 janvier 2025.

Sentance, S. and Waite, J. (2017). Primm : Exploring pedagogical approaches for teaching text-based programming in school. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, WiPSCE '17, page 113–114, New York, NY, USA. Association for Computing Machinery.

Tort, F., Drot-Delange, B., and Mano, M. (2017). Filles et informatique : qu'en est-il du concours castor. In *L'informatique et le numérique dans la classe. Qui, quoi, comment*, pages 69–84. Presses Universitaires de Namur.

5 Annexes

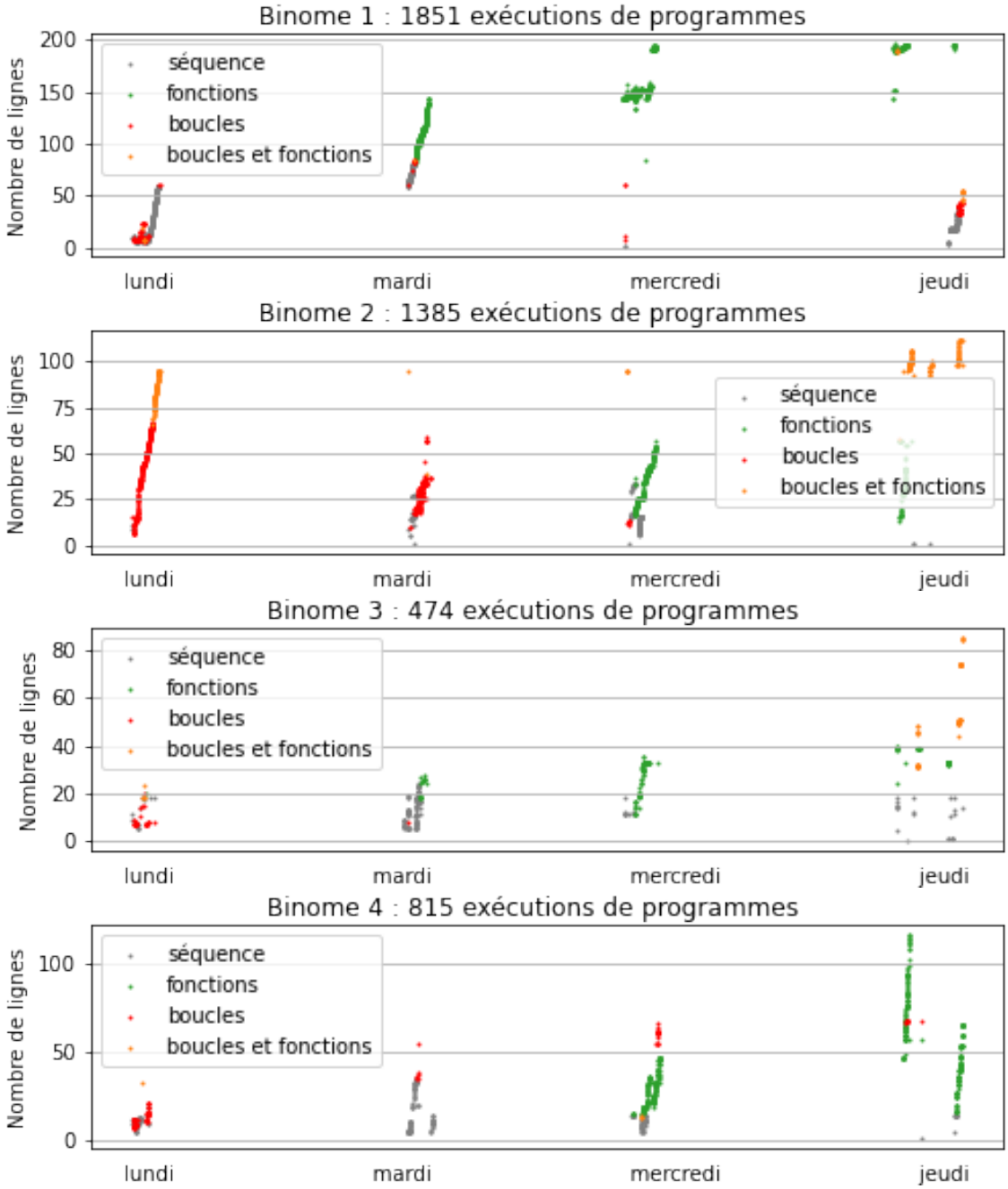


FIGURE 5 – Analyse de l'activité des binômes 1 à 4

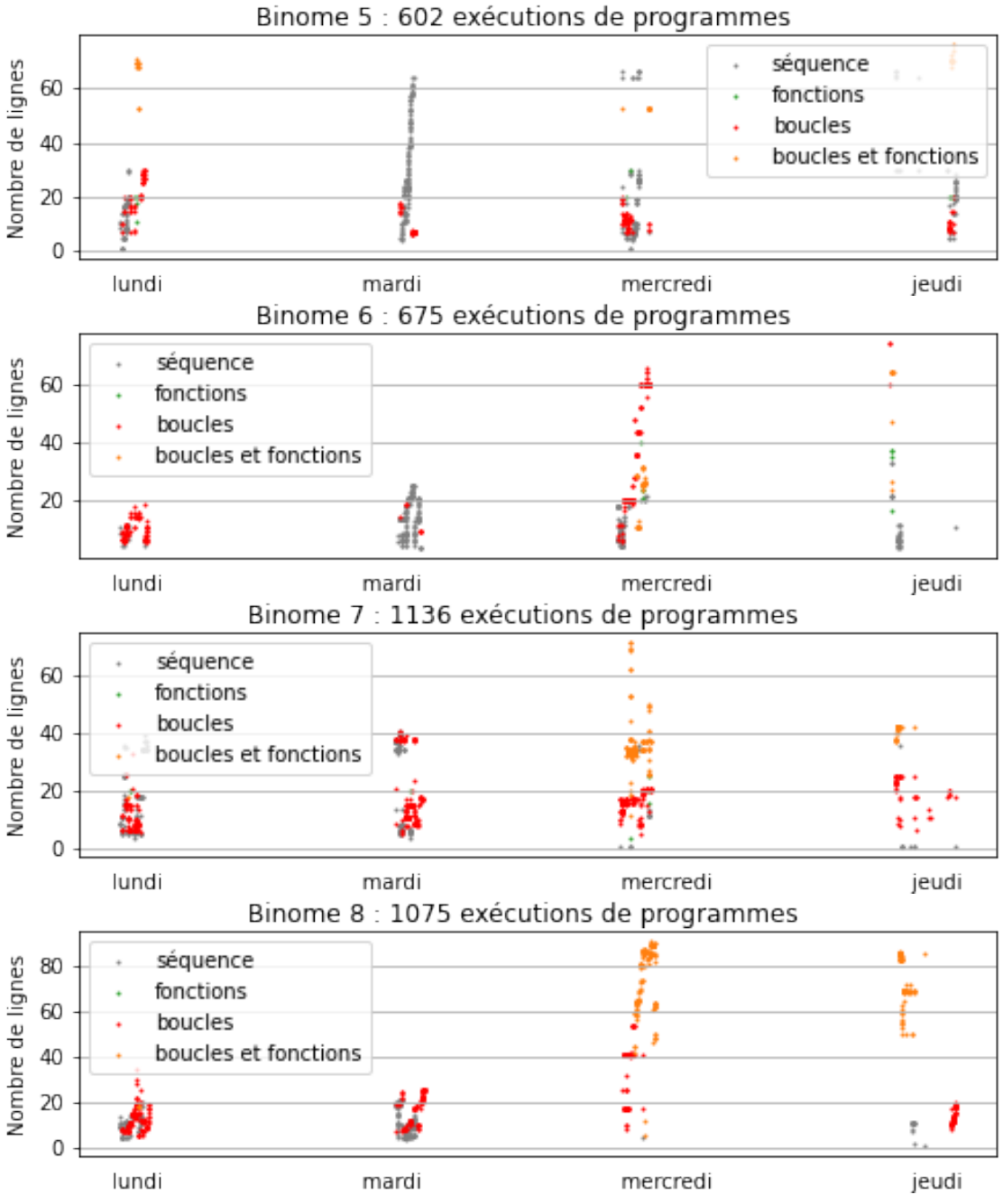


FIGURE 6 – Analyse de l'activité des binômes 5 à 8

Actes du 8e Atelier "Apprentissage de la pensée informatique de la maternelle à l'Université (APIMU 1000)" @ EIAH 2025
Villeneuve d'Ascq, France, 10 juin 2025

Introduction des scénarios UML en première NSI

Emmanuel Desmontils[1,2] Jean-Anne Colombel[3]
(1) Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, F-44000 Nantes, France
(2) Nantes Université, Centre de recherche en éducation de Nantes (CREN), UR 2661, F-44000 Nantes, France
(3) Lycée Saint-Joseph, Machecoul, France
emmanuel.desmontils@univ-nantes.fr,
Jean-Anne.Colombel@saintjoseph-machecoul.fr

RÉSUMÉ

Nous présentons une expérimentation faite en début de première, spécialité NSI, dans le cadre d'un projet en groupe. L'objectif est d'initier les élèves à la modélisation et à la structuration de projet en intégrant UML, notamment à travers les diagrammes de scénario. Un protocole expérimental a été mis en place sur trois années, avec pour support la programmation de jeux de société simplifiés. L'analyse des productions et le résultat d'une enquête montrent l'intérêt de l'introduction des outils de modélisation dès le lycée pour développer des compétences fondamentales en pensée informatique.

ABSTRACT

Introduction to UML Scenarios in First Year of NSI

In this article, we present an experiment carried out at the start of the first year of NSI specialty as part of a group project. The objective is to introduce students to modeling and project structuring by integrating the UML language, particularly through scenario diagrams. An experimental protocol was implemented over three years, supported by the programming of simplified board games. The analysis of the productions and the results of a student survey show the benefit of integrating modeling tools from high school to develop fundamental skills in computational thinking.

MOTS-CLÉS : Didactique de l'informatique, Projet en groupe, Modélisation du comportement, NSI, Scénario UML.

KEYWORDS: Computer science teaching, group project, behavior modeling, NSI, UML scenario.

1 Introduction

L'objectif de ce travail, né d'une collaboration entre un enseignant-chercheur et un enseignant de NSI, est l'introduction de techniques de gestion de projet informatique professionnelles dans la spécialité NSI. La part des projets est importante dans les enseignements de NSI. Le Bulletin Officiel pour le programme de première NSI (MENJ, 2019), en page 2, indique :

> *Une part de l'horaire de l'enseignement d'au moins un quart du total en classe de première doit être réservée à la conception et à l'élaboration de projets conduits par des groupes de deux à quatre élèves.*
>
> *Les projets réalisés par les élèves, sous la conduite du professeur, constituent un apprentissage fondamental tant pour la compréhension de l'informatique que pour l'acquisition de compétences...*

Nous nous sommes donc intéressés à introduire des outils dédiés à la conception et la gestion de projet.

Nous pensons que de tels outils sont susceptibles de faire progresser les élèves sur des compétences fondamentales comme l'abstraction et la généralisation, mais aussi la modélisation. De plus, nous pensons, mais c'est complexe à évaluer, que des méthodes rigoureuses issues du monde professionnel permettent à certains élèves de mettre une distance entre l'approche "geek" et la réalité du métier d'informaticien. Dans le cadre de cette publication, nous présentons l'utilisation du langage de modélisation UML (Unified Modeling Language) pour concevoir un logiciel. Nous allons présenter le diagramme de scénario utilisé dans cette approche (section 2), puis nous développerons notre démarche (section 3), évoquerons nos premières expérimentations (section 4) et analyserons quelques résultats (section 5), puis nous conclurons par un bilan et les évolutions que nous mettrons en place pour les prochaines expérimentations.

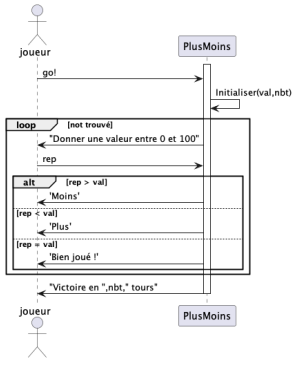
2 Cadre

UML [1] est un langage de modélisation graphique utilisé pour représenter les différents aspects d'un système logiciel. Il s'agit d'un standard international publié par l'Object Management Group (OMG). UML est utilisé pour améliorer la communication et la compréhension des systèmes logiciels à différents stades du cycle de vie du développement logiciel, allant de la conception à la documentation, en passant par les tests (Fowler, 2018). Il peut être utilisé par des analystes, des développeurs, des testeurs et des utilisateurs. UML comprend un ensemble de diagrammes qui peuvent être utilisés pour représenter différents aspects du logiciel. Les diagrammes les plus courants sont :

- diagrammes de cas d'utilisation et de scénario (figure 1b) : modélisent les interactions entre les acteurs et le système, en mettant l'accent sur les fonctionnalités offertes par le système ;
- diagramme de classes : modélise les classes d'un système logiciel et leurs relations ;
- diagramme d'état : modélise les différents états d'un objet et les transitions entre les états ;
- diagramme de séquence : modélise dans le temps les messages échangés entre les objets ;
- diagramme d'activité : modélise le flux de contrôle d'un système logiciel.

```
@startuml
skinparam BackgroundColor transparent
    actor joueur
    participant PlusMoins
    activate PlusMoins
    joueur->PlusMoins : go!
    PlusMoins->PlusMoins:Initialiser(val,nbt)
    loop not trouvé
        PlusMoins->joueur:"Donner une valeur entre 0 et 100"
        joueur->PlusMoins:rep
        PlusMoins->PlusMoins:MàJ(rep,nbt)
        alt rep > val
            PlusMoins->joueur:'Moins'
        else rep < val
                PlusMoins->joueur:'Plus'
        else rep = val
                PlusMoins->joueur:'Bien joué !'
        end
    end
    PlusMoins->joueur : "Victoire en ",nbt," tours"
    deactivate PlusMoins
@enduml
```

(a) Scénario en PlantUML.



(b) Représentation graphique.

FIGURE 1 – Diagramme de scénario pour le jeu "Plus-Moins" avec PlantUML

Un diagramme de scénario est une représentation graphique de l'algorithme qui modélise les interactions, par échanges de messages, entre les acteurs (utilisateurs ou systèmes externes) et un système logiciel ou un processus. Il est souvent associé à un diagramme de cas d'utilisation et prend la forme d'un diagramme de séquence simplifié. Il met en évidence les différentes actions ou étapes qu'un

1. https ://www.omg.org/spec/UML/

acteur peut effectuer avec le système, ainsi que les réponses du système en fonction des actions de l'acteur. Il est souvent utilisé au début de la phase de conception pour comprendre les exigences du système et identifier les fonctionnalités principales qu'il devrait fournir. Il sert aussi de référence, en fin de projet, dans la phase des tests fonctionnels, pour vérifier que le logiciel produit répond bien aux attentes des futurs utilisateurs. Les diagrammes de scénario (séquence), de classe et d'activité, sont les diagrammes UML les moins difficiles à introduire (Rivera-Lopez et al., 2009). UML est généralement présenté en premier cycle universitaire, d'abord en initiation à la programmation objet (diagrammes de classe), puis plus tard, lors des enseignements en ingénierie des exigences. Cependant, nous pensons possible de commencer à introduire quelques notions dès la spécialité NSI.

L'idée est de mettre en œuvre une activité ludique sous forme de projet en groupe, dans le cadre de (MENJ, 2019), avec comme principaux objectifs de :

- amener les élèves à réfléchir sur la modélisation des informations manipulées et sur les algorithmes à mettre en place avant de se lancer dans la phase de programmation ;
- introduire des éléments des méthodes agiles de type Scrum, en évitant, au début, le "jargon" traditionnel de ces méthodes informatiques pour ne pas perdre les élèves ;
- introduire le diagramme UML de scénario pour l'analyse et les tests ;
- aider les élèves dans le découpage en tâches et l'intégration des programmes ;
- aider les élèves à tester leurs programmes...

Dans notre travail, nous nous intéresserons à l'introduction des scénarios pour la programmation de jeux de société simplifiés. Cette approche nous permet de mobiliser les compétences fondamentales de la pensée informatique (Selby and Woollard, 2013; Dagiene et al., 2017; Declercq, 2021). La conception d'un scénario fait travailler :

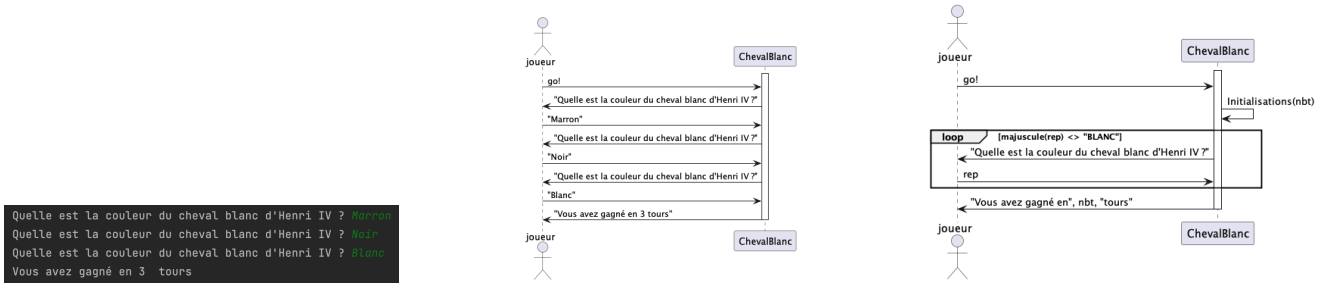
- anticiper : se projeter dans le fonctionnement du logiciel à venir ;
- généraliser : passage de la pratique et des règles du jeux à un scénario cohérent ;
- abstraire : gestion de la ligne de vie en discernant ce qui est explicité et ce qui ne l'est pas ;
- décomposer : par la division d'un scénario complexe en scénarios plus simples ;
- évaluer : vérifier les scénarios, mais aussi les utiliser pour les tests fonctionnels.

À cela, nous proposons aussi la compétence de modélisation, ajoutée par (Chane-Lune et al., 2024), qui nous semble fondamentale. Le programme NSI adresse explicitement cette compétence (MENJ, 2019) : «*[NSI] permet de développer des compétences : - analyser et modéliser un problème en termes de flux et de traitement d'informations ; [...]*» Comme le signale (Chane-Lune et al., 2024), la notion de modélisation est fortement polysémique. Nous nous intéressons à la modélisation des comportements dont les scénarios sont une représentation possible. Les élèves doivent expliciter les échanges entre le(s) joueur(s) et l'application qui met en œuvre le jeu. Cette compétence est présente dans l'action de transformer des règles ou des situations de jeu en un échange formalisé entre le jeu et les joueurs. Cette compétence est difficile à acquérir et à mettre en œuvre.

3 Conception

Nous avons choisi le jeu comme support de ce projet. Plus précisément, nous demandons aux élèves de créer une version numérique et simplifiée d'un jeu de société. Cette approche nous permet de partir d'un support concret qui permettra une phase en débranché. Le projet se fait en groupes d'au moins 4 élèves pour développer les compétences de travail en groupe. Nous avons conçu la séquence en trois parties : une phase d'analyse avec les scénarios UML, une phase de développement en Python et une présentation orale du travail.

La phase d'analyse se fait en débranché (crayon-papier), éventuellement assistée par PlantUML [2] pour faire les diagrammes. La figure 1a montre un exemple de scénario pour le jeu du "Plus Moins" dans ce langage et la figure 1b la représentation graphique. Cette phase est découpée en deux parties : (1) une initiation aux scénarios (1 séance) et (2) l'application au projet (3 séances). L'initiation aux scénarios est progressive à travers deux jeux simplistes. Dans un premier temps, nous présentons le jeu du Cheval-Blanc en donnant une trace (figure 2a). Puis, nous montrons comment passer de la trace à une description de l'échange à l'aide des scénarios (figure 2b). Enfin, nous montrons comment généraliser un échange pour une séquence de jeu quelconque (figure 2c). Dans un second temps, nous nous appuyons sur le jeu du Plus-Moins (rechercher une valeur dans un intervalle), qu'ils ont déjà vu en programmation, en leur donnant à nouveau une trace. Après leur avoir laissé du temps pour produire un scénario, nous construisons une solution possible à travers des échanges avec la classe pour obtenir un scénario satisfaisant (figure 1b). Ces deux activités nous permettent d'introduire les deux structures essentielles du scénario : la boucle et l'alternative. Nous complétons la présentation par l'introduction de l'opérateur de note et de l'opérateur de référence (pour permettre de décomposer les scénarios). Ensuite, nous réinvestissons la méthode sur le projet (3 séances). D'abord, les groupes se mettent au travail en jouant au jeu proposé. Ceci leur permet de mieux maîtriser les règles. Ensuite, ils produisent les scénarios du projet. Bien sûr, nous les accompagnons dans cette étape. A l'issue de ces séances, nous leur demandons un premier livrable avec leurs scénarios que nous corrigeons avec eux afin qu'ils puissent entamer la phase de programmation en confiance sur cette partie.



(a) Trace (b) Mise en scénario (c) Généralisation

FIGURE 2 – Exemple du jeu du Cheval Blanc.

La phase de programmation est sur plusieurs semaines. Au début de chaque séance (une heure par semaine), les groupes font un point sur ce qui a été fait la séance précédente (et éventuellement entre les séances) et sur ce qu'ils vont faire sur cette séance. C'est le "daily meeting" des méthodes agiles. Ils évoquent aussi les difficultés et les solutions possibles. Ils s'entendent sur les structures de données communes et sur les données échangées entre les parties du projet.

Finalement, lors de la phase de restitution, les élèves présentent leur projet. Parmi les attendus, les élèves doivent exploiter les scénarios pour valoriser leur programmation (tests fonctionnels).

L'objectif de notre approche est de proposer des projets, dès la classe de première, qui s'appuient sur des notions utilisées dans le monde professionnel, dans l'espoir de permettre aux élèves de mieux aborder les projets de cette discipline, de montrer qu'il existe des méthodologies pour mener des projets et pour, le cas échéant, qu'ils puissent avoir plus confiance en eux dans le développement de projet en classe ou durant les stages en entreprise à venir. Bien sûr, il n'est pas question de prendre ces notions dans leur intégralité. Introduire les scénarios en début de NSI peut sembler un peu délicat, voire risqué. En effet, en plus d'apprendre le Python, les élèves doivent apprendre le langage

2. `https://plantuml.com/`

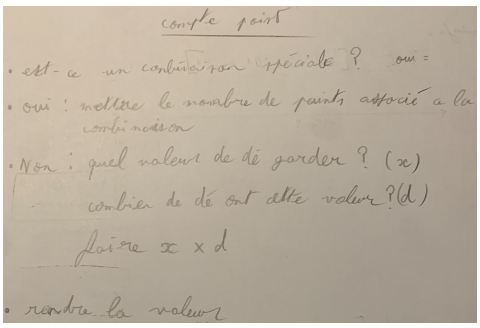
graphique des scénarios, voire la version textuelle de PlantUML. Cependant, nous pensons que le langage est assez simple et, finalement, assez proche d'un langage algorithmique standard.

L'activité proposée permet aux élèves un travail sur les compétences fondamentales. L'intention première était de travailler la modélisation. Effectivement, l'enseignant de NSI se trouvait en difficulté pour travailler cette compétence en première. À cette fin, la proposition de l'enseignant-chercheur était donc d'utiliser UML comme outil didactique. Par la suite, nous voyons que les diagrammes ont été un outil d'abstraction intéressant. C'est probablement ce qui met le plus en difficulté les élèves, mais qui est aussi très intéressant, car ils sont « forcés » de faire une analyse *a priori* de leur production. Enfin, nous aimerions que l'activité permette davantage de travailler la compétence "évaluer" lors de la phase finale du projet, même si ce dernier point reste encore à travailler.

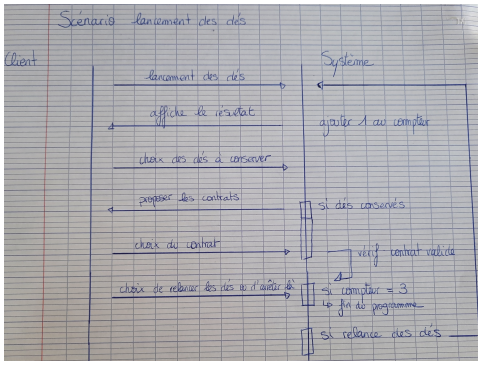
4 Expérimentations et analyse

L'expérimentation présentée dans cette publication s'est déroulée entre 2022 et 2025, dans le lycée Saint-Joseph à Machecoul (Loire-Atlantique, France) avec des élèves de première en spécialité NSI. Elle porte sur le second projet de l'année. Les jeux proposés sont simplifiés dans leurs règles et sont en ligne de commande pour ne pas surcharger le développement. Certaines séances devant les élèves ont fait l'objet de co-animation.

La première expérimentation s'est déroulée en 2022-2023 avec 13 élèves. Le projet consistait à programmer un jeu de Yam's. Nous avions 2 groupes avec respectivement 6 et 7 élèves. Chaque groupe était divisé en deux sous-groupes. Les sous-groupes A travaillaient sur l'échange avec les joueurs alors que les sous-groupes B programmaient le calcul des contrats. L'introduction des scénarios s'est faite juste par une présentation du langage graphique et de PlantUML. Nous n'avions pas encore, cette année-là, la séquence d'introduction.Globalement, les élèves ont été contents de travailler sur ce projet et, pour être seulement le second projet en informatique, les travaux rendus ont été plutôt intéressants et aboutis. Pour la phase 1, le travail a été un peu compliqué pour les élèves. Sur l'ensemble de la phase, beaucoup ont été frustrés de ne pas se mettre à coder de suite. Il est apparu nécessaire de mieux introduire les outils de modélisation. Il est à noter que, malgré des conditions difficiles de mise en route des modèles, certains élèves ont renvoyé des modèles de scénarios assez intéressants. La figure 3 présente deux exemples de scénarios proposés cette année-là. L'hétérogénéité des notations souligne la nécessité d'introduire PlantUML pour faciliter le travail sur ces diagrammes.



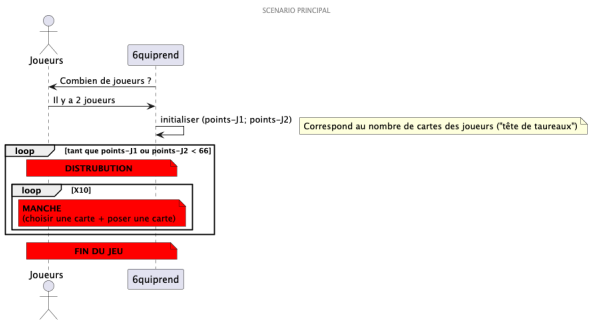
(a) Un scénario proposé par le groupe 1B.



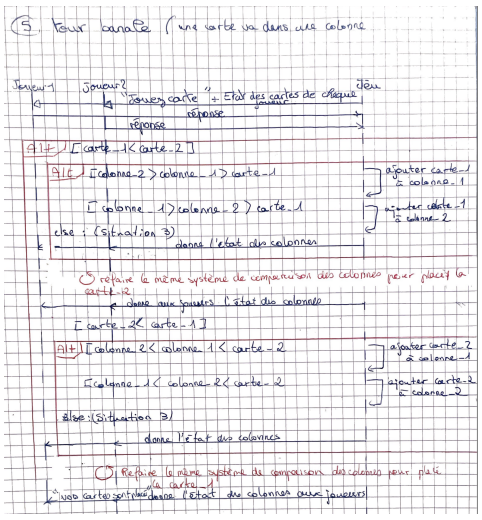
(b) Un scénario proposé par le groupe 2A.

FIGURE 3 – Exemples de scénarios produits pour le jeu de Yam's.

Pour l'année 2023-2024, nous avons proposé le "6 qui prend". Nous avions 4 groupes de 4 élèves. Compte tenu des difficultés des élèves à prendre en main les scénarios, nous avons ajouté la phase d'introduction de ces schémas avec les deux petits jeux pour illustrer les principaux concepts. Afin de consolider le travail des élèves, nous avons choisi de demander un livrable intermédiaire accompagné d'un retour individuel afin d'ajuster les scénarios et de pouvoir mieux s'appuyer dessus lors de la restitution orale de fin de projet. La figure 4 présente deux exemples de scénarios proposés cette année-là. On constate que les productions sont de meilleure qualité sur la syntaxe des scénarios.



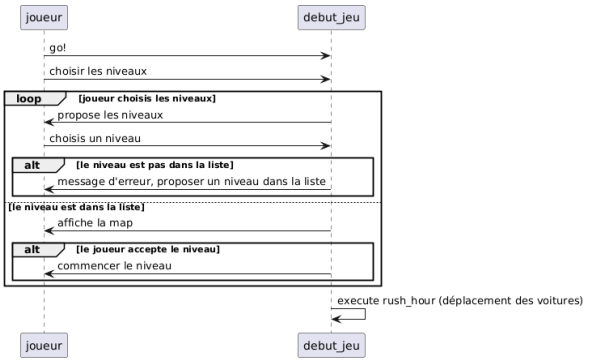
(a) Un scénario proposé par le groupe 2.



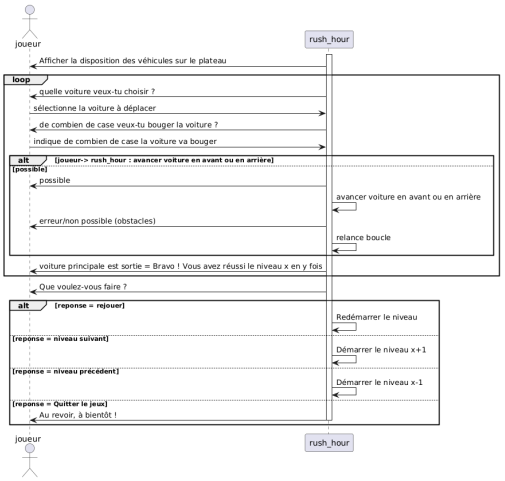
(b) Un scénario proposé par le groupe 3.

FIGURE 4 – Exemples de scénarios produits pour le jeu 6-qui-prend.

En 2024-2025, nous avons choisi le jeu Rush Hour. Nous avions 4 groupes de 4 élèves et un de 3 élèves. Nous avons, comme l'année passée, demandé un livrable intermédiaire. Le retour sur cette première production a constitué une séance de co-animation. À l'issue de ce retour, nous avons donné la possibilité aux élèves de modifier leurs scénarios avant le rendu définitif. La figure 5 présente deux exemples de scénarios proposés cette année-là.



(a) Un scénario proposé par le groupe 1.

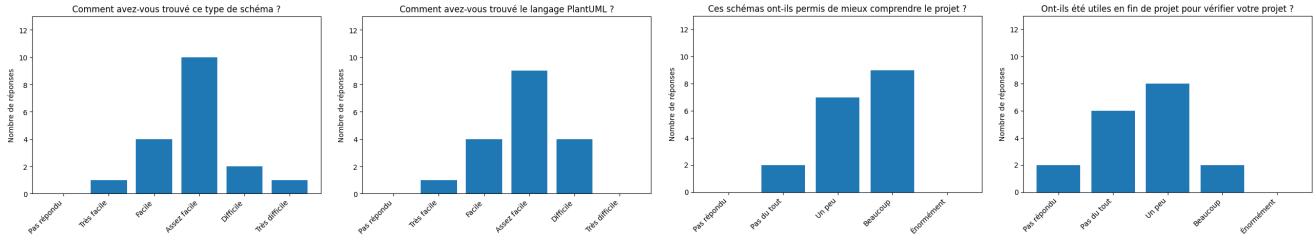


(b) Un scénario proposé par le groupe 2.

FIGURE 5 – Exemples de scénarios produits pour le jeu Rush Hour.

Nous avons, pour l'instant, expérimenté notre idée sur 3 années, avec 48 élèves (14 groupes). Après la première année, nous avons constaté que les modèles proposés n'étaient pas très bons dans l'ensemble. Nous avons donc ajouté une séance préliminaire permettant d'introduire les diagrammes de scénario. Cette séance s'est révélée très importante et la qualité des rendus s'est grandement améliorée.

En 2024-2025, nous avons proposé, aux élèves, une enquête sur les outils mis en place. Nous avons eu 18 répondants sur 19 élèves. En ce qui concerne les scénarios, les élèves ont trouvé ces schémas plutôt faciles à manipuler (figures 6a et 6b). C'est confirmé par la qualité de ce qui a été produit cette année (figure 5). Les élèves ont eu l'impression de comprendre l'intérêt de cet outil. 14 élèves pensent avoir compris l'objectif des scénarios (6 totalement). Ils ont trouvé que les scénarios les ont aidés un peu pour la compréhension du projet (figure 6c), mais n'ont pas tous compris l'intérêt pour la vérification de programme (figure 6d). Ceci s'est vérifié par le peu d'usage des scénarios en validation lors des soutenances. La séance de présentation initiale n'a donc pas suffisamment soulignée l'importance les tests fonctionnels. Malgré cela, 16 élèves envisagent de les utiliser pour de futurs projets. Finalement, même si les élèves doivent acquérir de nouveaux langages, ils se les approprient plutôt bien. L'enquête montre bien que la plupart des élèves n'ont pas été effrayés.



(a) Difficulté des scénarios (b) PlantUML (c) Compréhension (d) Tests fonctionnels

FIGURE 6 – Extraits de l'enquête auprès des élèves (année 2024-2025).

La qualité des scénarios produits est assez inégale (figure 3a vs. 3b), mais pas vraiment plus qu'un algorithme. Avec la séance de préparation, les diagrammes proposés sont plus travaillés, souvent avec PlantUML. Dans certains cas, ils sont assez riches (figures 4b ou 5a par exemple), parfois trop, malgré la présentation de la possibilité de composition de diagrammes. Les élèves ont en effet parfois produit des diagrammes très longs et complexes ou plusieurs diagrammes sans lien entre eux. Nous sommes évidemment dans la difficulté, pour un débutant, de décomposer un problème en sous-problèmes de manière cohérente (par exemple figure 4b). Certains groupes, tout de même, ont bien compris la décomposition. Par exemple, figure 4a propose une bonne décomposition, même si les outils des scénarios ne sont pas bien utilisés. Pour la construction des scénarios, la généralisation est souvent d'une bonne qualité (figure 5a par exemple, même si le système n'a pas de ligne de vie), mais pas toujours (figures 3b, 4b ou 5b). Pour l'abstraction, c'est plus inégal. Il est complexe pour les élèves de considérer le jeu comme une boite noire (fig. 4b).

La capacité "évaluer" apparaît lors de la phase de programmation et lors de la présentation orale. Nous nous apercevons, lors de la phase de programmation, que les élèves rencontrent toujours des problèmes de modélisation, dans le choix des objets et des structures à utiliser. C'est une difficulté naturelle pour des élèves débutant en programmation. Pour les débloquer, nous les avons parfois renvoyés vers leurs scénarios pour un "regard croisé" des deux démarches. Parfois, ce retour n'est pas concluant et le groupe s'aperçoit que c'est le scénario qui est mal conçu. Nous encourageons alors les élèves à en parler et à l'expliquer dans leur soutenance finale. De même, nous attendons des élèves qu'ils évaluent leur développement au regard des scénarios. Au cours des soutenances,

nous nous apercevons que l'utilisation des scénarios comme outil de validation du programme final est limitée. Les élèves exposent leur(s) scénario(s) comme demandé, mais les comparent peu aux programmes. Ils admettent volontiers que le programme ne suit pas le scénario initial, mais ne vont pas plus loin dans l'analyse. Quand on les interroge sur ces décalages, ils ont tendance à remettre en cause le scénario plutôt que le programme.

5 Conclusion

Afin de mieux cadrer les projets des élèves et pour éviter qu'ils "se jettent sur le clavier", nous avons introduit une phase d'analyse de l'application à travers les diagrammes de scénario UML. Cette activité nous a permis aussi d'introduire la notion de vérification fonctionnelle. Trois années d'expériences nous ont montré de bons résultats. Les compétences visées sont effectivement travaillées. Notre approche est donc à prolonger.

Notre projet se situe très tôt dans l'apprentissage de la programmation des élèves. Ils sont donc moins influencés par des réflexes parfois liés à la syntaxe d'un langage. Cependant, ce manque d'expérience cause aussi des difficultés dans la modélisation *a priori* imposée par les scénarios. Il serait intéressant d'expérimenter le même type de projet sur des élèves plus avancés dans leur apprentissage de la programmation et de comparer les résultats. De plus, afin de mieux les accompagner côté langages, nous allons proposer une fiche aide-mémoire pour retrouver facilement les concepts principaux (même s'ils ne sont pas nombreux). Enfin, il nous semble important de mieux expliquer l'usage que l'on peut faire des scénarios comme outil pour les tests fonctionnels. Pour cela, nous pensons ajouter une séance durant laquelle nous leur fournirons les scénarios correspondant au premier projet de l'année. L'exercice portera alors sur la validation de leur production au regard de ces scénarios. Une perspective de poursuite du projet en terminale serait aussi d'introduire un autre diagramme UML en lien avec la programmation objet : le diagramme de classe (sans doute en version simplifiée).

Références

Chane-Lune, S., Declercq, C., and Hoarau, S. (2024). Un référentiel de compétences en programmation pour identifier les difficultés des débutants et différencier les activités. In Mens, K. and Goletti, O., editors, Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC. Volume 1, pages 53–63, Louvain-La-Neuve, Belgium.

Dagiene, V., Sentance, S., and Stupuriené, G. (2017). Developing a two-dimensional categorization system for educational tasks in informatics. Informatica, 28(1) :23–44.

Declercq, C. (2021). Didactique de l'informatique : une formation nécessaire. STICEF (Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation), 28(3).

Fowler, M. (2018). UML Distilled : A Brief Guide to the Standard Object Modeling Language. Addison-Wesley Professional.

MENJ (2019). Programme d'enseignement de spécialité de numérique et sciences informatiques de la classe de première de la voie générale. Bulletin officiel spécial 1, Ministère de l'Éducation nationale et de la Jeunesse.

Rivera-Lopez, R., Rivera-Lopez, E., and Rodriguez-Leon, A. (2009). Another approach for the teaching of the foundations of programming using UML and Java. In WSEAS Inter. Conf. Proceedings. Mathematics and Computers in Science and Engineering. World Scientific and Engineering Academy and Society.

Selby, C. and Woollard, J. (2013). Computational thinking : the developing definition. Univ. of Southampton.

Actes du 8e Atelier "Apprentissage de la pensée informatique de la maternelle à l'Université (APIMU 1000)" @ EIAH 2025
Villeneuve d'Ascq, France, 10 juin 2025

De l'intérêt de machines notionnelles adaptées à l'apprentissage de la programmation en Python

Sébastien Hoarau[1] Christophe Declercq[1]
(1) LIM, Université de La Réunion
sebastien.hoarau@univ-reunion.fr, christophe.declercq@univ-reunion.fr

RÉSUMÉ

Nous présentons deux études exploratoires montrant d'une part l'imprécision ou le caractère erroné des modèles mentaux de l'exécution d'un programme Python chez des apprenants débutants, et d'autre part la faible utilisation par les enseignants de machines notionnelles adaptées. La première étude a été réalisée auprès d'étudiants entrants en licence d'informatique. La seconde étude a été menée auprès des enseignants de spécialité NSI de l'académie. En nous référant aux travaux académiques sur les machines notionnelles et sur les outils de visualisation, nous formulons l'hypothèse que ces observations peuvent être liées. Nous postulons donc l'intérêt de définir et partager des machines notionnelles adaptées, capables d'induire des modèles mentaux conformes chez les élèves. Nous concluons avec des perspectives d'élaboration de machines notionnelles adaptées à chaque niveau d'apprentissage, à savoir un modèle d'ardoise pour les débutants - en classe de seconde et première - et un modèle de références capable de distinguer mutables et non mutables pour les programmeurs avancés - en classe de terminale ou en première année à l'université.

ABSTRACT

The value of notional machines adapted to learning Python programming

We present two exploratory studies showing, on the one hand, the imprecision or erroneous nature of mental models of the execution of a Python program among beginner learners, and on the other hand, the low use by teachers of adapted notional machines. The first study was conducted among incoming computer science undergraduate students. The second study was conducted among NSI specialty teachers in the academy. Referring to academic work on notional machines and visualization tools, we hypothesize that these observations may be linked. We therefore postulate the interest of defining and sharing adapted notional machines, capable of inducing conforming mental models among students. We conclude with perspectives for developing notional machines adapted to each level of learning, namely a slate model for beginners and a references-based model capable of distinguishing mutable and non-mutable for advanced programmers.

MOTS-CLÉS : didactique de l'informatique, sémantique des langages de programmation, machine notionnelle, modèle de mémoire.

KEYWORDS: computer science teaching, semantics of programming languages, notional machine, memory model.

1 Introduction

Les difficultés des débutants face à l'apprentissage de la programmation ont été largement documentées (Arsac, 1980; Rogalski and Vergnaud, 1987; Sorva, 2013). Les travaux récents sur les modèles de mémoire (Exibard et al., 2024) rejoignent ceux sur les machines notionnelles (Du Boulay et al., 1981; Du Boulay, 1986) pour démontrer leur intérêt pour comprendre le fonctionnement d'un programme et se forger un modèle mental conforme au modèle conceptuel de ces machines (Sorva, 2013; Fincher et al., 2020; Greca and Moreira, 2000). Les machines notionnelles sont comme les modèles conceptuels de machines, des "machines abstraites", mais à visée pédagogique justifiant une simplification des modèles. Une machine notionnelle doit permettre d'expliquer la sémantique des constructions du langage pour pouvoir interpréter un programme.

La machine notionnelle peut être accompagnée d'une méthode de visualisation mise en oeuvre par un outil (Sorva et al., 2013). Cependant une visualisation – par exemple avec Python Tutor (Guo, 2013) – ne peut se substituer à un modèle permettant à l'apprenant d'être actif en construisant sa visualisation (Naps et al., 2002).

Nous commençons par présenter une expérimentation menée en février 2024 avec des étudiants de première année de licence d'informatique montrant des modèles mentaux imprécis ou erronés. C'est ce premier constat qui a motivé la seconde étude menée avec des enseignants de la spécialité NSI de l'académie en décembre 2024. Dans cette deuxième expérimentation, nous questionnons les enseignants sur leur manière d'expliquer aux élèves deux programmes écrits en Python : le premier avec des types élémentaires, des boucles et une fonction, le second faisant appel à des structures de données *mutables*. Nous poursuivons en montrant l'intérêt de machines notionnelles partagées entre enseignants (Dickson and Dragon, 2021) et en proposant quelques pistes vers la construction de telles machines.

2 Des modèles mentaux imprécis ou erronés chez les étudiants

Une expérimentation a été menée avec une population de 89 étudiants de première année de licence d'informatique au début du second semestre en février 2024, après un cours d'initiation à Python dans lequel l'outil Python Tutor avait été utilisé. Six programmes Python de difficulté croissante leur ont été présentés : d'abord une simple affectation, un programme comportant deux affectations, une séquence plus longue d'affectations, une affectation de liste, et enfin deux programmes manipulant des fonctions sur des types élémentaires puis sur des types mutables. Pour chaque programme (voir annexe) la consigne était de "donner une visualisation de l'exécution de ce programme" dans l'objectif de capturer le modèle mental de l'étudiant.

Dès le premier programme, on remarque que les étudiants ont une grande variété de modèles mentaux. Le plus fréquemment utilisé est le modèle dit *de boites* qui représente plus du tiers des réponses. La figure 1 montre deux exemples de réponses appartenant à cette catégorie. Pour un quart des réponses, on retrouve les explications textuelles (e.g. *x vaut* 1 ou *x est égal à* 1). Un autre quart des réponses utilise un modèle de flèche, *à la mode Python Tutor*, dont un exemple est présenté figure 2. La dernière catégorie, regroupe l'ensemble des autres réponses : les mélanges (fléches + texte), les non-réponses, les réponses plus *exotiques* avec, par exemple, des réponses qui tentent de se raccrocher à des représentations connues dans le registre mathématique (voir figure 2).

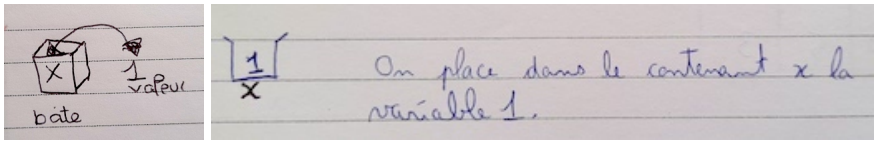


FIGURE 1 – Exemples de représentations de : $x = 1$ par *boite*.

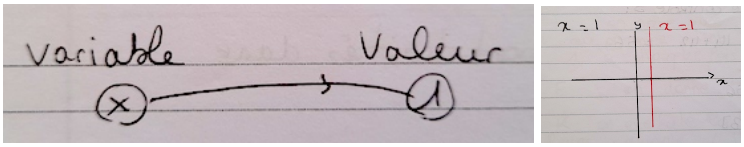


FIGURE 2 – Deux exemples de représentations de : $x = 1$ avec *flèche* ou mathématique.

Pour le deuxième programme, 45% des réponses utilisent une représentation claire et correcte (voir par exemple la figure 3).

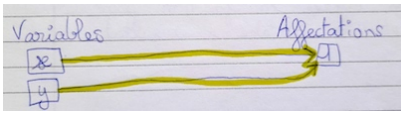


FIGURE 3 – Un exemple de représentation de : `x = 1 ; y = x` par *flèche*.

La majorité restante n'arrive pas à représenter correctement cette situation. On retrouve alors des réponses *erronées* aussi bien chez les utilisateurs du modèle de boites que chez ceux se servant de flèches. Quelques uns de ces exemples sont présentés dans la figure 4.

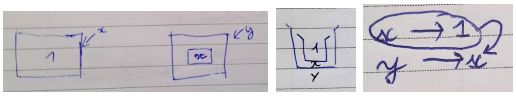


FIGURE 4 – Exemples de représentations erronées de : `x = 1 ; y = x`.

Lorsque l'on introduit la notion de tableau le nombre de représentations correctes diminue encore pour ne plus atteindre que 22% des réponses. La figure 5 illustre les erreurs les plus fréquentes : disparition de la notion d'indice, de la structure séquentielle, de la variable, variable avec plusieurs valeurs simultanément.

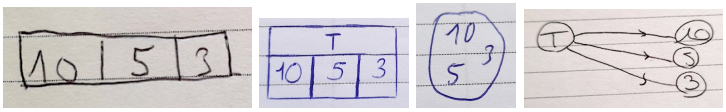


FIGURE 5 – Exemples de représentations erronées d'un tableau simple : `T = [10, 5, 3]`.

C'est en cherchant à explorer les origines de ces modèles mentaux chez les étudiants, que nous avons décidé d'interroger les enseignants afin de recenser les machines notionnelles utilisées au lycée. Notons cependant qu'une partie seulement des étudiants de première année de licence ont suivi la spécialité informatique au lycée.

3 Des machines notionnelles faiblement utilisées par les enseignants

L'expérimentation proposée aux enseignants de SNT et NSI de l'académie consiste à se mettre dans la situation d'expliquer aux élèves, l'exécution d'un programme en utilisant un modèle de représentation (voir la consigne en annexe figure 12). Le programme à expliquer est d'un niveau début de classe de première laissant la possibilité d'utiliser un modèle de type boites.

Nous avons pu classer les 28 réponses en 8 catégories de pratiques. En retirant les catégories marginales (à 1 ou 2 réponses ou contenant des réponses difficiles à interpréter), il reste 3 catégories principales : les approches algébriques (4 réponses), l'utilisation d'un tableau de valeurs (4 réponses) et la réécriture de code avec instanciation des variables et/ou dépliage de la boucle (13 réponses).

La *réécriture du code* est une méthode d'explication qui ne repose pas sur une machine notionnelle. La figure 6 montre un exemple où le code de chaque appel à la fonction `etage` est réécrit et instancié. On note un mélange de registres : lorsque l'enseignant écrit : `i = 1`, ou encore : `total = 0 + 1`, il se place dans le registre informatique avec un symbole = qui représente l'affectation. Mais avec : `0 + 1 = 1`, il se place dans le registre mathématique avec un symbole = qui représente cette fois l'égalité usuelle des mathématiques. Si ce passage d'un registre à l'autre ne pose pas de difficulté pour un expert, il peut favoriser les confusions chez l'apprenant débutant. En particulier, la confusion de registres peut amener l'élève à appliquer de manière erronée sur une séquence d'affectations, le principe de substitution utilisable en mathématiques dans des équations.

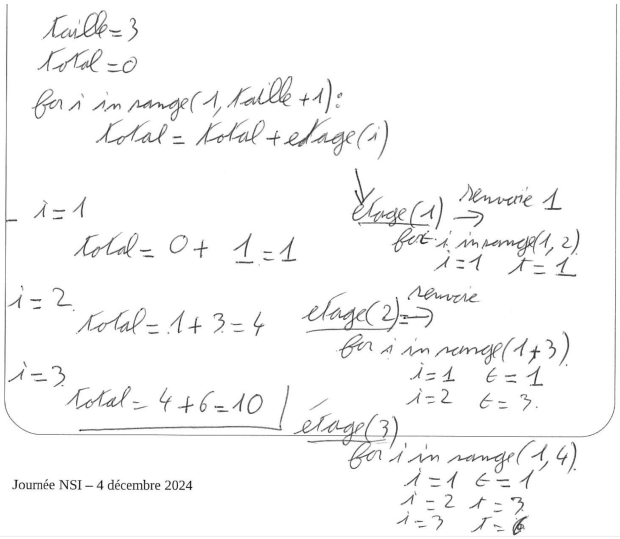


FIGURE 6 – Une approche *réécriture de code* pour l'explication d'un programme.

La figure 7 montre une explication commençant par un début de tableau d'évolution. En haut de la

capture on peut lire le nom des variables du programme principal `taille`, `total` mais l'évolution des variables n'apparait pas et on voit un passage à une formule mathématique ($1+1+2+\ldots$). Cette formulation algébrique apparait plus explicitement pour expliquer un appel à la fonction `etage` : `etage(n)` est équivalent à $\frac{n(n+1)}{2}$.

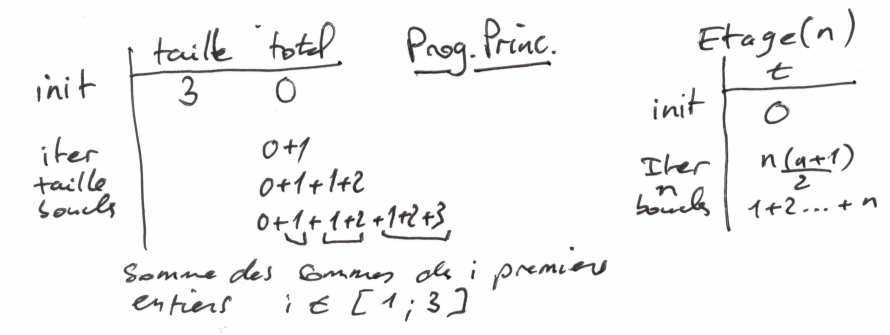


FIGURE 7 – Une approche algébrique de l'explication d'un programme.

La figure 8 est un représentant de la catégorie des tableaux d'évolution des variables. Dans cet exemple, nous pouvons voir un tableau pour le déroulement du programme principal et 3 autres pour les appels à la fonction `etage`. On observe l'influence du registre mathématique : les tableaux sont présentés en ligne à l'image des tableaux des valeurs d'une fonction en mathématiques et il manque l'identification de l'instruction qui a été exécutée.

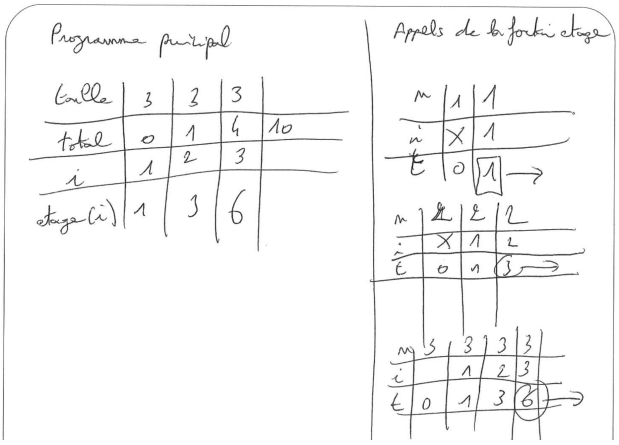


FIGURE 8 – Des tableaux de valeurs des variables.

Pour synthétiser l'analyse des réponses des enseignants pour la première partie, on retient que :

1. les pratiques sont extrêmement variées et n'utilisent que rarement une machine notionnelle ;
2. il y a une forte influence du registre mathématique ;
3. l'usage d'un modèle de boite (ou d'une variante) ou d'un tableau d'évolution des variables est marginal.

On peut postuler que la faible utilisation d'une machine notionnelle adaptée par les enseignants de lycée peut être une des causes de la faiblesse des modèles mentaux des étudiants de première année,

pour expliquer des programmes très simples, constitués uniquement de séquences d'affectations. Cependant, une étude à plus grand échelle serait nécessaire pour conclure en distinguant bien les élèves ayant suivi un enseignement d'informatique au lycée et ceux n'en ayant pas suivi.

La figure 13 en annexe montre la consigne pour la partie 2 de l'expérimentation : deux versions d'un programme censé modifier les valeurs d'un tableau passé en paramètre sont présentés. L'un est erroné. La tâche pour l'enseignant est de donner à l'écrit les explications pour convaincre les élèves que le programme P_1 est erroné.

Un modèle à base de références permet de montrer l'erreur faite dans le programme P_1 : ce n'est pas la bonne référence qui est mise à jour. Ainsi, dans les programmes P_1 et P_2 de l'expérimentation, `t[0]` et `valeurs[0]` sont la même référence. De même, on visualise, dans le cas de P_1 que l'affectation : `elt = 2 * elt`, modifie la référence qui lie `elt` à sa valeur et n'a aucun impact sur la référence qui lie par exemple : `valeurs[0]`, à sa valeur. C'est ce que montre la figure 9 : à gauche, on a l'état de la mémoire à l'entrée du premier passage dans la boucle `for`, à droite cet état juste après l'affectation.

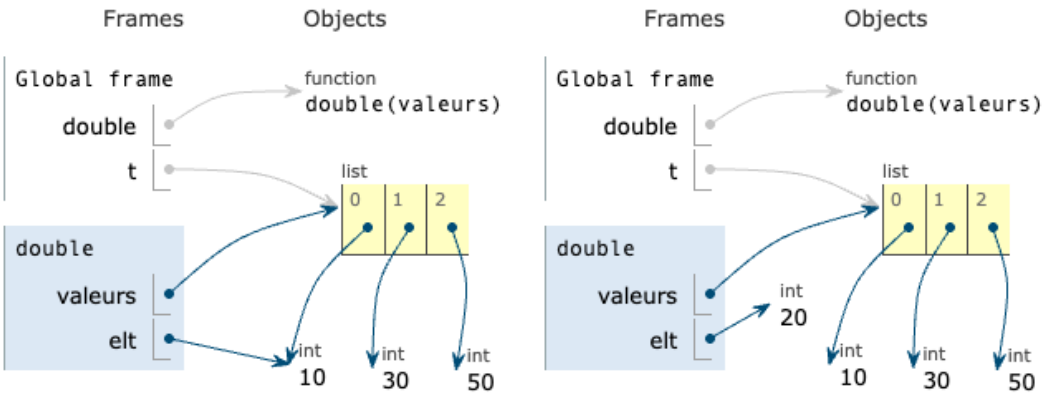


FIGURE 9 – Visualisation d'une étape du programme P_1.

Sur les 19 résultats analysés, on constate l'absence quasi-générale d'un modèle de références. La seule réponse utilisant ce modèle est présentée figure 10. Dix des réponses n'utilisent pas de modèle et se contentent d'affirmer (en utilisant par exemple des débuts de phrase comme *on voit...*) que : `elt = 2 * elt`, ne modifie pas le tableau `t` alors que : `valeurs[i] = 2 * valeurs[i]`, recopie la valeur dans la case du tableau. Aucune explication n'est donnée sur le lien entre la variable globale `t` et la variable locale `valeurs` ni pourquoi : `elt = 2 * elt`, ne modifie pas le tableau.

Parmi ces 10 réponses, 2 affirment que l'erreur du programme P_1 vient de la modification de la variable de boucle `elt` à l'intérieur de la boucle.

7 réponses proposent un début de modélisation mais mélangent une modélisation par cases et par flèches avec des flèches qui n'ont pas toujours la même sémantique. La figure 11 montre un exemple de ce type avec un début de modélisation par des flèches.

Le passage aux *mutables* de Python pose des difficultés aux enseignants pour proposer des explications convaincantes du comportement de programmes. Des 28 répondants de la phase 1, il ne reste que 9 tentatives de modélisation et 1 seule réellement convaincante.

On déduit de cette expérimentation, que l'usage de machines notionnelles par les enseignants pour

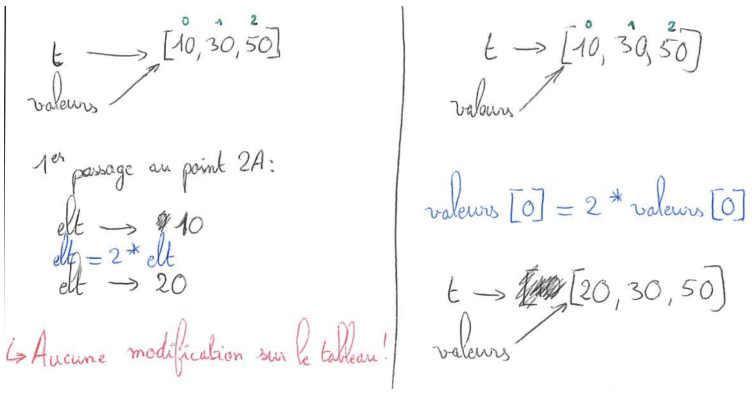


FIGURE 10 – Un modèle de références pour expliquer le programme erroné.

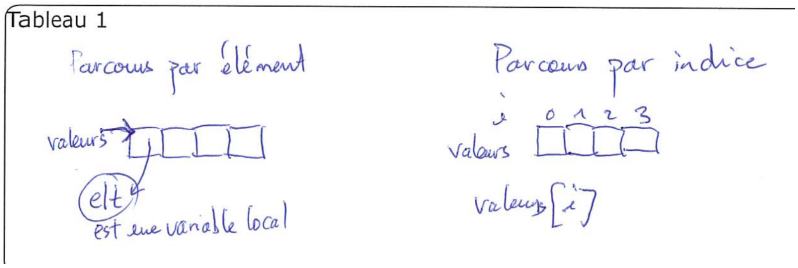


FIGURE 11 – Un modèle hybride : case + flèches ; ambigüité sur le sens de certaines flèches.

expliquer le comportement de programmes est peu développé dans le cas de programme simples nécessitant uniquement un modèle de boites, et est quasi-inexistant pour des programmes plus complexes nécessitant un modèle basé sur les références.

4 Intérêt de machines notionnelles adaptées

Les méthodes d'explication utilisées par les enseignants utilisent majoritairement soit le registre algébrique - souvent difficile d'accès aux lycéens ne suivant pas la spécialité mathématiques - soit la réécriture de code. Cette dernière méthode experte est difficile à manier pour les élèves et ne dispense d'ailleurs pas de donner aux élèves une machine notionnelle capable d'exécuter une séquence d'instructions affectant des variables, pour comprendre le caractère dynamique de l'évolution de leurs valeurs.

Nous défendons donc la proposition de formuler au plus tôt et de manière explicite, un modèle de machine permettant d'expliquer le comportement de chacune des instructions du langage. Cette proposition fait d'ailleurs consensus dans la communauté scientifique qui a montré l'importance pour l'apprenant d'avoir une modèle mental fiable et que ce modèle est en partie construit à partir de la machine notionnelle présentée par l'enseignant (Sorva, 2013). De nombreuses machines notionnelles ont été recensées par (Fincher et al., 2020) et sont disponibles en ligne à l'adresse : `https://notionalmachines.github.io/notional-machines.html`.

Par ailleurs, il nous semble raisonnable de vouloir construire des machines notionnelles communes à

un groupe d'enseignants et adaptées aux différents niveaux d'apprentissage. Cette idée d'uniformiser une pratique dans ce contexte n'est pas nouvelle. Elle a été, par exemple, mise en œuvre à l'université d'Ithaca de New-York où les enseignants ont adopté et présenté à leurs étudiants la même machine notionnelle (nommée IDMs) (Dickson and Dragon, 2021) permettant d'expliquer des programmes écrits dans trois langages : Python, C++ et Java. Cette machine est de ce fait complexe et ne nous semble pas adaptée au contexte du lycée en France où seul le langage Python est utilisé.

Pour les programmeurs Python débutants en classe de seconde ou de première, les machines notionnelles *variable as box* ou *Python beginner* semblent suffisantes. Nous préférons cependant utiliser la métaphore de l'*ardoise effaçable* pour représenter une variable plutôt que celle de la boite, le versement d'une boîte dans une autre ne correspondant pas à la réalité de l'affectation.

Pour les programmeurs Python avancés, en classe de terminale ou à l'université, l'introduction des objets et des structures de données mutables nécessite une machine notionnelle à base de références, pour expliquer les phénomènes d'alias, le passage de paramètres et le partage de références à un même objet mutable. Les modèles d'adresse ou de référence actuellement disponibles ont cependant le défaut de ne pas distinguer les structures mutables et non mutables du langage. Les outils de visualisation comme Python Tutor (Guo, 2013) ne les distinguent pas non plus représentant aussi bien les tuples que les listes par des références.

5 Conclusion et perspectives

Constatant les difficultés des étudiants dans l'apprentissage de la programmation, nous avons voulu comprendre quelles représentations ils se faisaient de l'exécution d'un programme Python. Constatant des faiblesses dans les différents modèles mentaux, nous avons voulu inciter des enseignants à exhiber leurs machines notionnelles en les plaçant dans une situation concrète : expliquer des programmes écrits en langage Python comme s'ils étaient face à des élèves. Il ressort de cette étude un usage très faible de machines notionnelles par les enseignants interrogés.

Nous avons discuté l'intérêt de définir et présenter des machines notionnelles adaptées aux apprentissages et que ces machines soient largement partagées entre enseignants. Nous avons identifié deux machines notionnelles que nous nommons *modèle à ardoises* et *modèle de références* qui peuvent être dérivées des machines existantes pour mieux les adapter aux publics visés, à savoir les débutants en classe de seconde et première et les programmeurs avancés en terminale ou à l'université.

Nous travaillons actuellement à expliciter ces machines notionnelles en vue de les partager avec la communauté académique et celle des enseignants. Pour le *modèle à ardoise*, nous travaillons à l'étendre aux tableaux vus comme des collections de variables et aux fonctions avec un modèle multi-ardoises. Concernant le *modèle de références* autorisant la manipulation de structures mutables, y compris comme paramètres de fonctions, nous élaborons une représentation permettant de distinguer les non-mutables et les mutables. Cette machine en cours de formalisation aura l'avantage de disposer d'une règle unique pour expliquer l'affectation.

Une perspective de ce travail sera aussi de sélectionner, ou développer si besoin, des outils de visualisation conformes aux machines notionnelles définies. L'objectif à terme est bien d'aligner, pour chaque niveau d'enseignement, un sous-ensemble du langage Python à enseigner, une machine notionnelle permettant d'expliquer la sémantique des instructions du langage, un outil permettant d'en visualiser l'exécution.

Références

Arsac, J. (1980). *Premières leçons de programmation*. Cedic.

Dickson, P. E. and Dragon, T. (2021). A memory diagram for all seasons. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, ITiCSE '21, pages 150–156, 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.

Du Boulay, B., O'Shea, T., and Monk, J. (1981). The black box inside the glass box : presenting computing concepts to novices. *International Journal of man-machine studies*, 14(3) :237–249.

Exibard, L., Francis, N., Meyer, A., and van den Bogaard, M. (2024). Modèles de mémoire pour l'enseignement de la programmation. In Mens, K. and Goletti, O., editors, *Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC. Volume 1*, pages 33–43, Louvain-La-Neuve, Belgium.

Fincher, S., Jeuring, J., Miller, C. S., Donaldson, P., du Boulay, B., Hauswirth, M., Hellas, A., Hermans, F., Lewis, C., Mühling, A., Pearce, J. L., and Petersen, A. (2020). Notional machines in computing education : The education of attention. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '20, pages 21–50, New York, NY, USA. Association for Computing Machinery.

Greca, I. M. and Moreira, M. A. (2000). Mental models, conceptual models, and modelling. *International Journal of Science Education*, 22(1) :1–11.

Guo, P. J. (2013). Online python tutor : embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 579–584, New York, NY, USA. Association for Computing Machinery.

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., and Velázquez-Iturbide, J. A. (2002). Exploring the role of visualization and engagement in computer science education. *SIGCSE Bull.*, 35(2) :131–152.

Rogalski, J. and Vergnaud, G. (1987). *Didactique de l'informatique et acquisitions cognitives en programmation*.

Sorva, J. (2013). Notional machines and introductory programming education. *ACM Transactions on Computing Education*, 13 :8 :1–8 :31.

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).

Annexe A : Expérimentation avec les étudiants

Dans la suite, nous considérons des instructions et des programmes en langage Python 3. Donner une visualisation de l'exécution de ces programmes :

1.
```
x = 1
```

2.
```
x = 1
y = x
```

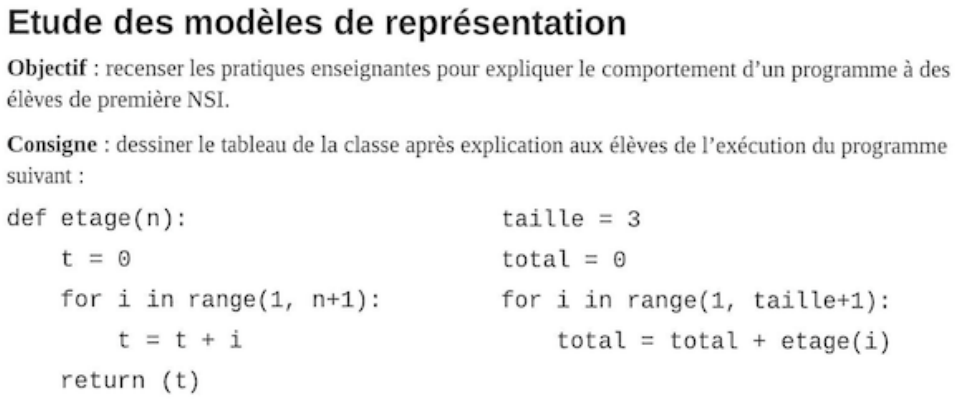
3. Pour le programme 3, donner la visualisation à chacun des points de contrôle
```
a = 42
b = 56
# point de contrôle 1
a = b - a
b = b - a
# point de contrôle 2
a = a + b
# point de contrôle 3
```

4.
```
T = [10, 5, 3]
```

5. Pour le programme 5, donne l'ordre dans lequel les points de contrôle seront exécuté puis une visualisation de l'exécution à chacun de ces points :
```
def swap(a, b):
    # point de contrôle D
    a, b = b, a
    # point de contrôle T
a = 1
b = 10
# point de contrôle E
swap(a, b)
print(a, b)
# point de contrôle K
```

6. Programme 6 (même consigne que pour le programme 5) :
```
def ajoute(t):
    # point de contrôle D
    t[0] = t[0] + 1
    # point de contrôle T
valeurs = [4, 9, 7]
# point de contrôle E
ajoute(valeurs)
# point de contrôle K
```

Annexe B : Expérimentation avec les enseignants



Etude des modèles de représentation

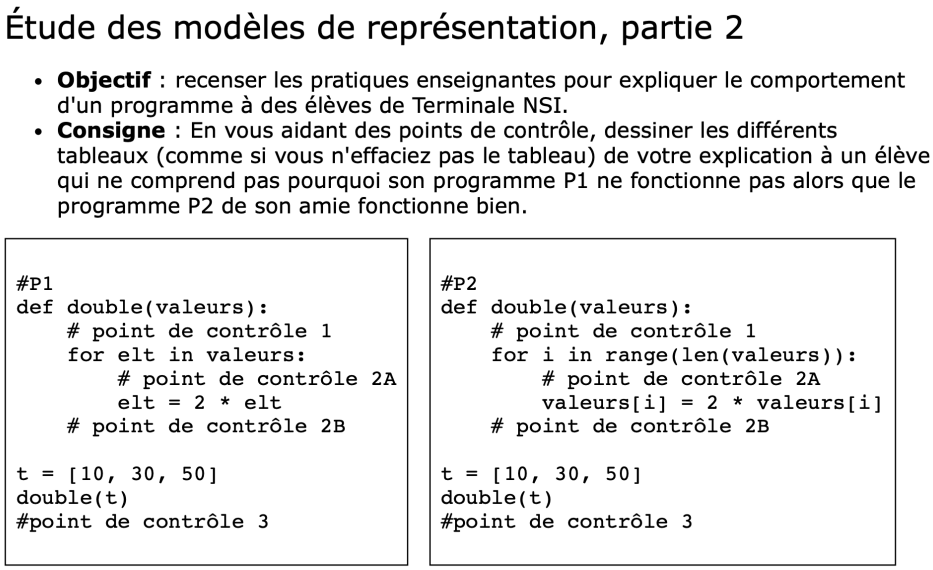
Objectif : recenser les pratiques enseignantes pour expliquer le comportement d'un programme à des élèves de première NSI.

Consigne : dessiner le tableau de la classe après explication aux élèves de l'exécution du programme suivant :

```
def etage(n):
    t = 0
    for i in range(1, n+1):
        t = t + i
    return (t)
```

```
taille = 3
total = 0
for i in range(1, taille+1):
    total = total + etage(i)
```

FIGURE 12 – La consigne de la partie 1 de l'expérimentation enseignants.



Étude des modèles de représentation, partie 2

- **Objectif** : recenser les pratiques enseignantes pour expliquer le comportement d'un programme à des élèves de Terminale NSI.
- **Consigne** : En vous aidant des points de contrôle, dessiner les différents tableaux (comme si vous n'effaciez pas le tableau) de votre explication à un élève qui ne comprend pas pourquoi son programme P1 ne fonctionne pas alors que le programme P2 de son amie fonctionne bien.

```
#P1
def double(valeurs):
    # point de contrôle 1
    for elt in valeurs:
        # point de contrôle 2A
        elt = 2 * elt
    # point de contrôle 2B

t = [10, 30, 50]
double(t)
#point de contrôle 3
```

```
#P2
def double(valeurs):
    # point de contrôle 1
    for i in range(len(valeurs)):
        # point de contrôle 2A
        valeurs[i] = 2 * valeurs[i]
    # point de contrôle 2B

t = [10, 30, 50]
double(t)
#point de contrôle 3
```

FIGURE 13 – La consigne de la partie 2 de l'expérimentation enseignants.