"Didactique de l'informatique : la méthodologie PRIMM"

Debongnie, Cyrille

ABSTRACT

Actuellement, dans l'enseignement de la programmation, après une introduction théorique sur un nouveau sujet, les étudiants doivent trop souvent produire du code directement. Cette manière de faire peut poser problème pour des débutants, qui peuvent par exemple ne pas savoir par où commencer, être frustrés par l'ampleur des concepts à assimiler, où encore ressentir l'angoisse de la faute de syntaxe. Des études ont soulevé l'utilité de rajouter différentes approches pédagogiques pour faciliter l'apprentissage de nouveaux concepts. Ces études se sont penchées sur l'amélioration de l'accompagnement de l'élève dans son apprentissage. Parmi ces études est née la méthodologie PRIMM, qui est au cœur de ce sujet. L'idée derrière PRIMM, qui est l'acronyme de "Predict, Run, Investigate, Modify, Make" ou "Prédire, Exécuter, Enquêter, Modifier, Faire", est de permettre à l'élève de se familiariser avec un nouveau concept en échafaudant progressivement sa maîtrise. Selon PRIMM, les élèves pourraient bénéficier à passer du temps à lire du code déjà existant, discuter entre eux de ce qu'ils comprennent et ne comprennent pas, scruter ligne par ligne le fonctionnement du programme, essayer de prédire le résultat, mais aussi exécuter le programme pour vérifier leurs prédictions. J'ai pris et analysé une séquence de cours non-PRIMM, dont la majorité des exercices est orientée Make, je l'ai ensuite modifiée pour l'adapter à la méthodologie PRIMM. J'ai personnellement soumis cette séquence à une classe d'étudiants. Une enseig...

CITE THIS VERSION

UCLouvain

epl

École polytechnique de Louvain

Didactique de l'informatique : la méthodologie PRIMM

Auteur: **Cyrille Debongnie**
Promoteurs: **Kim Mens, Olivier Goletti**
Lecteur: **Ludovic Taffin**
Année académique 2021–2022
Master [60] en sciences informatiques

Résumé

Actuellement, dans l'enseignement de la programmation, après une introduction théorique sur un nouveau sujet, les étudiants doivent trop souvent produire du code directement. Cette manière de faire peut poser problème pour des débutants, qui peuvent par exemple ne pas savoir par où commencer, être frustrés par l'ampleur des concepts à assimiler, où encore ressentir l'angoisse de la faute de syntaxe. Des études ont soulevé l'utilité de rajouter différentes approches pédagogiques pour faciliter l'apprentissage de nouveaux concepts. Ces études se sont penchées sur l'amélioration de l'accompagnement de l'élève dans son apprentissage. Parmi ces études est née la méthodologie PRIMM, qui est au cœur de ce sujet. L'idée derrière PRIMM, qui est l'acronyme de *"Predict, Run, Investigate, Modify, Make"* ou "Prédire, Exécuter, Enquêter, Modifier, Faire", est de permettre à l'élève de se familiariser avec un nouveau concept en échafaudant progressivement sa maîtrise. Selon PRIMM, les élèves pourraient bénéficier à passer du temps à lire du code déjà existant, discuter entre eux de ce qu'ils comprennent et ne comprennent pas, scruter ligne par ligne le fonctionnement du programme, essayer de prédire le résultat, mais aussi exécuter le programme pour vérifier leurs prédictions.

J'ai pris et analysé une séquence de cours non-PRIMM, dont la majorité des exercices est orientée *Make*, je l'ai ensuite modifiée pour l'adapter à la méthodologie PRIMM. J'ai personnellement soumis cette séquence à une classe d'étudiants. Une enseignante à Paris a testé la séquence avec ses élèves. Ces mises en situations réelles ont produit une série de données sur laquelle je me base pour tenter d'évaluer si PRIMM délivre ses promesses.

Remerciements

Je tiens à remercier l'ensemble des personnes m'ayant accordé leur temps et leur soutien tout au long de ces études.
Merci au professeur Kim Mens pour son aide, ses relectures et ses conseils tout au long de l'année.
Merci à Olivier Goletti pour sa supervision, ses suggestions et ses nombreux coups de pouces.
Merci à Ludovic Taffin pour ses explications et son aide.
Merci à Carine Grancher pour sa collaboration, son investissement et ses conseils.
Merci à Clémence Lorent pour sa relecture, sa patience et son indéfectible soutien, sans qui ces études n'auraient probablement jamais abouties.

Table des matières

Chapitre 1

Introduction

1.1 Contexte

L'apprentissage de l'informatique dans les écoles primaires et secondaires en Belgique francophone est pour l'instant assez lacunaire, malgré une utilisation de plus en plus omniprésente d'appareils numériques et un besoin de plus en plus grand de compétences liées à l'informatique, comme publié par la fédération technologie Agoria [1] qui annonce 541 000 postes vacants liés à l'informatique d'ici à 2030. À ce jour, un jeune diplômé du secondaire en Wallonie n'aura reçu au mieux qu'une formation basique et principalement orientée bureautique, comme révélé dans l'état des lieux des études menées par N. Joris et J. Henry [2, 3].

Dans l'optique de faire évoluer l'enseignement de la programmation en Belgique, une réforme de l'éducation est en train d'être mise en place [1]. Une partie des mesures de la réforme ont pour but d'introduire des compétences digitales à partir de la troisième année primaire (8-9 ans). La section de la réforme relative à l'informatique, qui devrait être graduellement instaurée de 2022 pour la troisième année primaire jusqu'en 2028 pour la troisième année du secondaire, présentera aux élèves dès la sixième primaire (11-12 ans) des concepts tels que la logique, l'algorithmique et les programmes informatiques.
Certaines des compétences qui seront attendues d'eux sont les suivantes : qu'ils soient capables de résoudre des problèmes en sachant lire, comprendre et rédiger un algorithme, mais aussi lire, comprendre et concevoir des programmes.

Des outils sont développés pour faciliter l'apprentissage de la programmation aux débutants, telle que la programmation par blocs, de plus en plus utilisée. Par

1. https://rfie.ares-ac.be/boite-a-outils/referentiels-duTC

exemple le site Code.org [2] avec ses nombreux cours, la publication "Beyond curriculum: The exploring computer science program" par J. Goode et al. [4], ou encore "The CS principles course" par O. Astrachan et A. Briggs [5], utilisent tous différents outils de programmation par blocs. On peut expliquer en grande partie l'adoption de la programmation par blocs dans l'éducation grâce à l'aspect visuel attrayant. La forme et la couleur des blocs fournissent des indices visuels à l'apprenant, ainsi que les mécanismes de manipulation des blocs qui facilitent la composition de code. Il existe un certain nombre de langages de programmation par blocs, parmi les plus connus figurent **Scratch** [3] disponible depuis 2007 et développé par le *MIT media lab* [4], **Snap !** disponible depuis 2011 et développé à l'Université de Berkeley [5] ainsi que **Blockly** disponible depuis 2012 et développé par Google. Ce dernier est le langage utilisé pour ce mémoire, que je couvre plus en détail dans le chapitre 2.

Trop souvent, dans la manière dont on enseigne la programmation aujourd'hui, les étudiants sont rapidement livrés à eux-mêmes. Les exercices demandent aux élèves de coder directement un concept qu'on vient seulement de leur introduire. Des études ont soulevé l'utilité de rajouter différentes approches pédagogiques pour faciliter l'apprentissage de nouveaux concepts [6, 7, 8]. Ces études cherchent à fournir des outils pédagogiques aux professeurs pour mieux accompagner l'élève. L'idée est de permettre à l'élève de se familiariser avec un nouveau concept en échafaudant progressivement sa maîtrise. Ces études ont amené à la confection de différentes méthodologies d'enseignement telles que **PRIMM**, qui est l'acronyme de *"Predict, Run, Investigate, Modify, Make"* ou "Prédire, Exécuter, Enquêter, Modifier, Faire". Pour améliorer l'enseignement en programmation, ces méthodologies s'appuient sur des stratégies pédagogiques telles que la lecture du code avant d'écrire, la discussion au sein de la classe et le *tracing* [9]. (*Tracing* : l'action d'analyser les valeurs contenues dans les variables et suivre leur évolution à travers l'exécution du programme). **PRIMM** est expliqué en détail dans le chapitre 2.

Suite à la création en 2015 et aux développements récents de l'exerciseur d'INGInious [10], il nous est maintenant possible de créer ainsi qu'évaluer des séquences de cours et des exercices Blockly à l'Université Catholique de Louvain. INGInious est une plateforme web de cotation automatique créée au cœur de l'UCLouvain, et celle-ci supporte plusieurs formats d'exercices tels que du code de programmation ou de la programmation par blocs. INGInious est expliqué en détail dans le chapitre 2.

2. https://code.org/
3. https://scratch.mit.edu/
4. https://www.media.mit.edu/
5. https://snap.berkeley.edu/

1.2 Objectifs

C'est donc dans cet environnement que ce mémoire s'inscrit. L'objectif de ce sujet est d'élaborer une séquence d'exercices qui ont pour but d'introduire des concepts en programmation au moyen de la programmation par blocs, plus précisément Blockly. La séquence est développée suivant la méthodologie **PRIMM** sur la plateforme INGInious. La séquence a pour objectif d'être testée et validée en classe avec des élèves en situation réelle, si possible avec l'aide d'un professeur externe. La population visée est les élèves du secondaire n'ayant pas eu de formation en programmation au préalable.

1.3 Approche

J'ai analysé une séquence existante du point de vue de PRIMM. Lors de cette analyse, à part un exercice à choix multiple qui pousse l'élève à prédire et enquêter la solution, j'ai pu observer que cette séquence était dans sa grande majorité orientée *Modify/Make*. La séquence ne donne pas assez l'opportunité à l'élève de se poser des questions sur le fonctionnement en détail du code. Il n'est pas poussé à passer du temps à lire et scruter du code déjà existant, aussi il n'a pas l'occasion d'exécuter une solution fournie pour tester les limites de sa compréhension. J'ai modifié cette séquence pour décomposer les concepts abordés selon les phases de PRIMM. Les concepts principaux sont divisés en sous-séquences, cette optique a pour but de permettre à l'élève de bien faire la distinction dans les concepts abordés. Sur base de ces modifications, on espère que les promesses de PRIMM soient délivrées.

La séquence adaptée à PRIMM a été testée en situation réelle à deux reprises. Premièrement, par moi-même sur une classe de rhétorique en Belgique. Deuxièmement, par madame Carine Grancher, professeur d'informatique à Paris, dans sa classe de quatrième (système français). Il s'agit de l'équivalent de la deuxième secondaire en Belgique. Les élèves ayant participé ont reçu un questionnaire pour tenter d'évaluer si les différentes phases de PRIMM ont eu un impact positif sur leur apprentissage, mais aussi leur ressenti, leurs difficultés et les concepts qu'ils ont retenus. J'essaie d'évaluer l'impact de PRIMM basé sur les critères suivants :

- L'observation des résultats. Par exemple : jusqu'où ils ont été dans la séquence, combien de temps ça leur a pris, les problèmes rencontrés, le nombre de tentatives par exercice.
- L'opinion de C. Grancher. Par exemple : ses observations durant la classe, ce qu'elle en a pensé, quel type de questions ont été posées.
- Mon ressenti personnel. Par exemple : mes observations durant la classe, quel type de questions ont été posées, les difficultés que j'ai pu observer.
- Le retour des élèves, grâce aux questionnaires qu'ils ont remplis.

Par exemple : "Est-ce que l'exercice dans lequel tu devais prédire le résultat, puis exécuter le code pour vérifier ta prédiction, t'a aidé à comprendre le concept de positions dans les chaînes de caractères ?".

Après avoir donné les séquences, collecté puis analysé les résultats, il semblerait que PRIMM ait eu un impact positif pour faciliter l'introduction de nouveaux concepts.

1.4 Contribution

Un aspect intéressant que mes expériences ont mis à jour est qu'il est très important, avant de donner les exercices en situation réelle, de les faire tester par une personne extérieure n'ayant pas de notions en programmation. Procéder de cette manière apporte un regard novice qui soulève de nombreux détails utiles, permettant d'éviter potentiellement de nombreux problèmes pour les élèves.

Les élèves ayant suivi la séquence PRIMM semblent avoir apprécié la thématique, qui est d'identifier le coupable dans le cadre d'une enquête scientifique. Rajouter une thématique semi-ludique tout au long des exercices semble être une plus-value pour les élèves, ils ont un sentiment de satisfaction à l'aboutissement de l'histoire. Je recommande donc l'utilisation d'une histoire pour motiver les élèves.

Cependant, il incombe, au professeur décidant d'intégrer une histoire, de ne pas négliger deux aspects importants :

- Cette histoire ne doit pas prendre le dessus sur la matière enseignée. J'ai pu observer dans mes résultats que des élèves ont mal compris certains concepts, considérant que l'extraction de chaîne ADN était un type de donnée à part entière. Je recommande d'essayer de trouver une thématique de fond, qui maintient la motivation de l'élève, tout en faisant clairement la distinction entre la matière enseignée et l'histoire.
- Il est ressorti à plusieurs reprises que beaucoup d'élèves ne lisaient que partiellement voire pas du tout les énoncés. Il est donc important, même en l'absence d'une thématique ludique, de faire une distinction évidente et structurée dans l'énoncé entre les différentes informations. L'élève doit pouvoir repérer rapidement ce qui est du domaine de l'histoire et de la théorie, mais surtout ce qu'il est attendu d'eux dans l'exercice. Cette distinction doit être respectée dans l'entièreté des exercices, laissant le choix à l'élève d'apprécier ou non l'histoire.

J'ai pu aussi observer qu'il faut éviter le plus possible d'utiliser des concepts n'ayant jamais été introduits. L'exercice final de la séquence non-PRIMM, contenant plusieurs nouveaux concepts non expliqués, a totalement déstabilisé les étudiants.

1.5 Structure du mémoire

Le chapitre 2, “Contexte théorique” explique l'environnement et les outils utilisés pour mettre en place les séquences d'exercices. J'explique en détail la méthodologie PRIMM. J'y explique quelles sont les origines de PRIMM, sur quelles recherches Sue Sentance s'est basée pour fonder cette méthodologie, ainsi que les objectifs pédagogiques qu'elle a voulu transmettre. Je décris en détail chaque phase de PRIMM et je les illustre, images à l'appui, à l'aide des exercices de la séquence que j'ai développé. J'explique comment fonctionne la plateforme de cotation automatique INGInious, pourquoi est-elle utilisée dans ce mémoire, sur quelles technologies repose la plateforme, comment INGInious permet la création et l'utilisation d'exercices. J'explique aussi l'intérêt d'utiliser la programmation par blocs, les avantages ainsi que les inconvénients qui en découlent. Enfin, je parle de Blockly et de son fonctionnement.

Le chapitre 3, “Les séquences d'exercices et leurs différentes approches”, couvre les deux séquences d'exercices, leur contenu respectifs, les analyse et les compare. La première séquence introduit des concepts en suivant une méthodologie non-PRIMM. À l'aide d'une introduction théorique, elle fournit aux élèves un seul exercice par concept, puis une tâche dans laquelle l'élève doit résoudre le problème en codant. La deuxième séquence, créée selon les phases de la méthodologie PRIMM, aborde les mêmes concepts que la première séquence, mais les divises en plusieurs sous-séquences. Elle permet à l'élève de lire le code pour assimiler le plus possible, se poser des questions en enquêtant sur des puzzles ou des erreurs volontaires, mais aussi vérifier ses suppositions en exécutant un code déjà existant. Enfin, ce chapitre discute des forces et des faiblesses de chaque séquence pour essayer d'établir l'impact de cette adaptation à PRIMM.

Le chapitre 4 discute de la validation des séquences, des différents questionnaires soumis aux élèves, des résultats obtenus grâce aux différentes séances. Ce chapitre termine par une analyse qualitative des résultats pour évaluer l'apport de PRIMM.

Le chapitre 5 apporte une conclusion sur le travail effectué, parle des objectifs atteints ainsi que des objectifs futurs soulevés dans le cadre de ce travail.

Chapitre 2

Contexte théorique

Ce chapitre aborde la méthodologie PRIMM, les recherches sur lesquelles Sue Sentance, la créatrice de PRIMM, s'est appuyée, ainsi que ses objectifs pédagogiques et ses différentes phases. Chaque phase est expliquée en détail puis illustrée par un exemple tiré de la séquence PRIMM réalisée. Ensuite, sont décrits les différents outils utilisés qui ont permis la mise en place des deux séquences d'exercices. Terminant l'explication des outils, la programmation par blocs et Blockly sont décrits plus en détail.

2.1 PRIMM

PRIMM [11], l'acronyme de *"Predict, Run, Investigate, Modify, Make"*, est une approche structurée pour enseigner la programmation. C'est une méthodologie qui combine plusieurs techniques et différents niveaux d'abstractions. Cette méthodologie a pour objectif de structurer au mieux l'introduction et la découverte de nouveaux concepts aux élèves. PRIMM [12] permet aux élèves de s'approprier et de maîtriser de nouveaux concepts informatiques complexes, ce grâce à des exercices aux approches variées qui vont graduellement accompagner l'élève. Cette méthodologie, introduite en 2017 par Sue Sentance [1] [13] s'inspire d'autres recherches dans l'éducation, notamment la méthodologie *"Use-Modify-Create"* [6], l'utilité du *tracing*, et le *Block model* [8, 12].

"Use-Modify-Create" est une méthodologie introduite en 2011 par Lee et al. dans une étude [6] qui se base sur l'idée qu'un élève apprend d'abord à lire avant d'écrire, ils ont appliqué cette idée à l'apprentissage du code. L'objectif est d'aider l'élève à s'améliorer tout en diminuant son anxiété. À la phase *"Use"*, les apprenants ont un rôle de consommateur, ils reçoivent un programme qu'ils doivent regarder, analyser, essayer de comprendre, mais ils ne doivent rien créer. À la phase *"Modify"*, les

1. https://www.cst.cam.ac.uk/people/ss2600

apprenants, qui se sont familiarisés avec le programme fourni à la phase précédente, sont encouragés à le modifier, d'abord par des petits détails triviaux, puis par des tâches qui augmentent graduellement en complexité. Enfin, lors de la phase *"Create"*, ayant accumulé de l'expérience aux phases précédentes, les apprenants sont plus à l'aise d'écrire leurs propres programmes. Ces trois phases, comme visuellement représentées dans la figure 2.1, permettent une appropriation du code pour l'apprenant. Dans les deux premières phases, l'apprenant est dans une optique de *Not mine* - pas à moi - il passe ensuite à *Mine* - à moi - dans la dernière phase. PRIMM, avec ses différentes phases, a récupéré et adapté ce concept d'appropriation par l'élève. Dans la figure 2.8 page 18, on observe ce flux d'appropriation par l'élève en fonction de la phase à laquelle il se trouve.

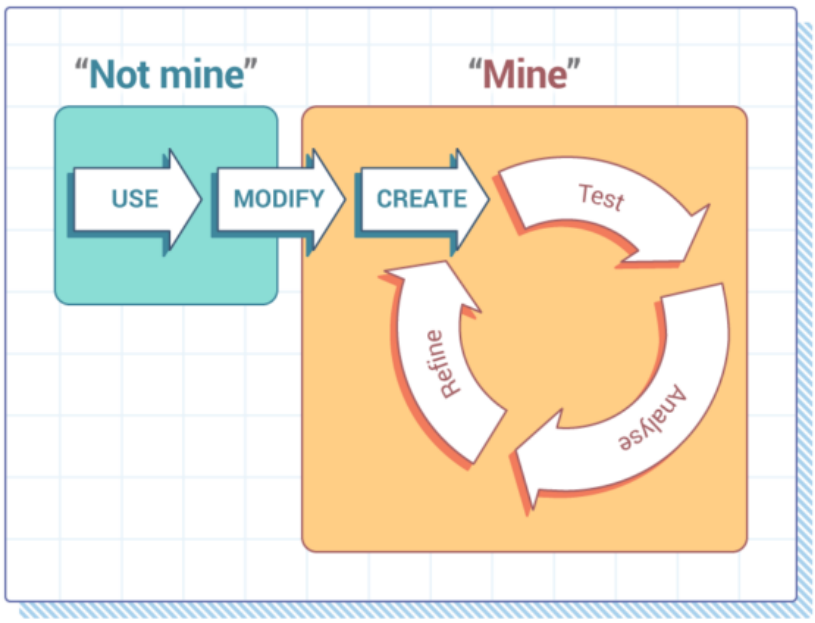


FIGURE 2.1 – Use - Modify - Create : Appropriation.

Le *Block model* est introduit en 2008 par C. Schulte et al. [8] dans lequel ils s'intéressent aux différents niveaux de compétences nécessaires aux débutants en programmation. Le *Block Model* est représenté dans la figure 2.2 par une grille divisée en deux. La partie de gauche représente la taille de l'élément considéré, la partie droite fait la distinction entre la structure, l'exécution et la fonction d'un programme.

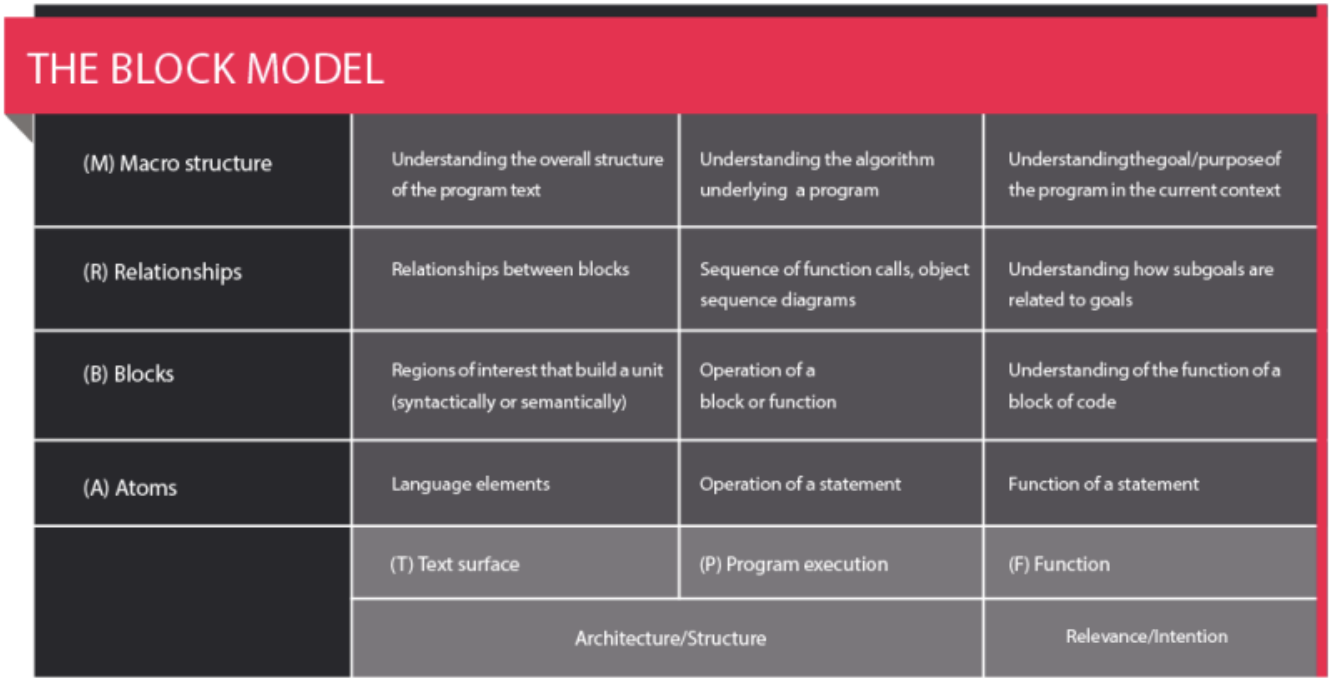


FIGURE 2.2 – Le Block model.

PRIMM est basée sur trois grands axes que nous détaillerons ci-dessous [14, 15, 16] :

1. **Tracing**
   Le premier point principal, basé sur les recherches de R. Lister [17, 18, 19] et al., porte sur la lecture et le *tracing* du code, facilitant ainsi la compréhension de l'élève avant d'avoir à coder seul. De nombreuses recherches portent sur l'utilité et l'efficacité de lire avant de coder. Notamment, une étude menée en 2014 par S. Sentance et al. [20] a démontré que les enseignants utilisaient diverses stratégies pour accompagner l'apprentissage de la programmation, telles que : le *tracing*, le *debugging* ainsi que modifier des morceaux de code fournis par le professeur. Ces stratégies ont pour but de réduire la frustration et l'exaspération qui peut résulter d'avoir commis des erreurs de syntaxe. Les recherches de R. Lister [17, 18, 19], ont démontré que les novices ont besoin d'être capable d'avoir un score de 50% au *tracing* de code avant de pouvoir coder seuls. L'élève améliore ainsi ses compétences en vocabulaire, en syntaxe, en analyse et en logique.
2. **Encourager la discussion au sein de la classe**
   Le deuxième point principal, basé sur une étude de A. Walqui [21] porte sur l'aspect social et la communication verbale des élèves à propos du programme.

En lien avec le point précédent, l'étude de R. Lister [19] a confirmé qu'un élève qui performe mal au *tracing* de code a du mal à expliquer le code, inversement un élève à l'aise avec le *tracing* sera plus apte à verbaliser, car il comprend mieux le code. Cette verbalisation a trois niveaux d'utilité :

- L'élève est forcé d'utiliser le bon vocabulaire ou la bonne terminologie pour arriver à formuler sa pensée.
- Le fait de verbaliser force à structurer la pensée pour arriver à ce que les autres comprennent notre point de vue, mais apporte aussi de la clarté d'esprit, car les élèves doivent décomposer le problème et s'attarder sur des petits détails.
- Au travers du dialogue, on peut poser des questions, répondre à des questions et apprendre des autres.

3. **Ne pas commencer de zéro**
   Le troisième point principal porte sur l'appropriation et la responsabilisation du code, basé sur la méthodologie *Use-Modify-Create* mentionnée plus haut [6]. Les élèves doivent initialement travailler avec un code qui n'est pas le leur, ils pourront progressivement se l'approprier en modifiant le programme au fil des exercices. Le fait de travailler sur un programme déjà fourni par une personne externe retire à l'élève la responsabilité d'erreurs potentielles, diminuant ainsi la charge émotionnelle de l'élève quand le programme ne s'exécute pas correctement. C'est une manière de protéger les élèves des sentiments de frustration, d'anxiété, de doute et de déception qui peuvent surgir lors de l'apprentissage en programmation. Fournir un programme de départ représente également un gain de temps, recopier le code peut prendre beaucoup de temps à l'élève qui n'est pas familier avec la syntaxe.

La section ci-dessous couvre plus en détail chaque phase de PRIMM avec des exemples concrets d'exercices.

Predict: Les élèves sont confrontés à un programme qu'ils doivent lire puis essayer d'en prédire le résultat. *Predict* permet aux élèves de travailler leur vocabulaire, comme recommandé par la verbalisation à la base de PRIMM décrite plus haut. Cette phase se prête bien à travailler par paires ou en petits groupes, ce qui stimule la discussion et donc la verbalisation. Les élèves sont encouragés à chercher des indices dans le code qui vont les aider à comprendre quelle est la fonction du code fourni. Un exemple d'exercice *Predict* est donné à la figure 2.3. Dans cet exercice, l'élève est confronté à un nouveau bloc dont il doit essayer de prédire le résultat. Ce bloc permet d'extraire une sous-chaîne avec les positions définies. Avec les exercices précédents qui expliquent les chaînes de caractères et leurs positions, l'élève dispose de toutes les informations nécessaires pour trouver la bonne réponse dans les choix proposés.

FIGURE 2.3 – Un des exercices *Predict* de la séquence PRIMM.

Run: Les élèves exécutent un code déjà écrit pour eux ce qui évite qu'ils doivent le recopier. Ne pas devoir recopier a plusieurs avantages : ça leur permet de se focaliser sur le concept abordé sans s'encombrer avec les erreurs potentielles de syntaxes, mais aussi étant donné que le code vient du professeur, on élimine la responsabilisation de l'élève en cas d'erreurs déjà présentes. À cette étape-ci, l'élève considère le code comme *Not mine* (observable dans la partie gauche dans la figure 2.8 page 18). Un exemple d'exercice *Run* est donné à la figure 2.4 page 14. Dans cet exercice, un nouveau bloc est introduit à l'élève, ce bloc a pour but de trouver l'occurrence d'une sous-chaîne de caractères. L'élève peut directement observer un exemple qui utilise ce concept.

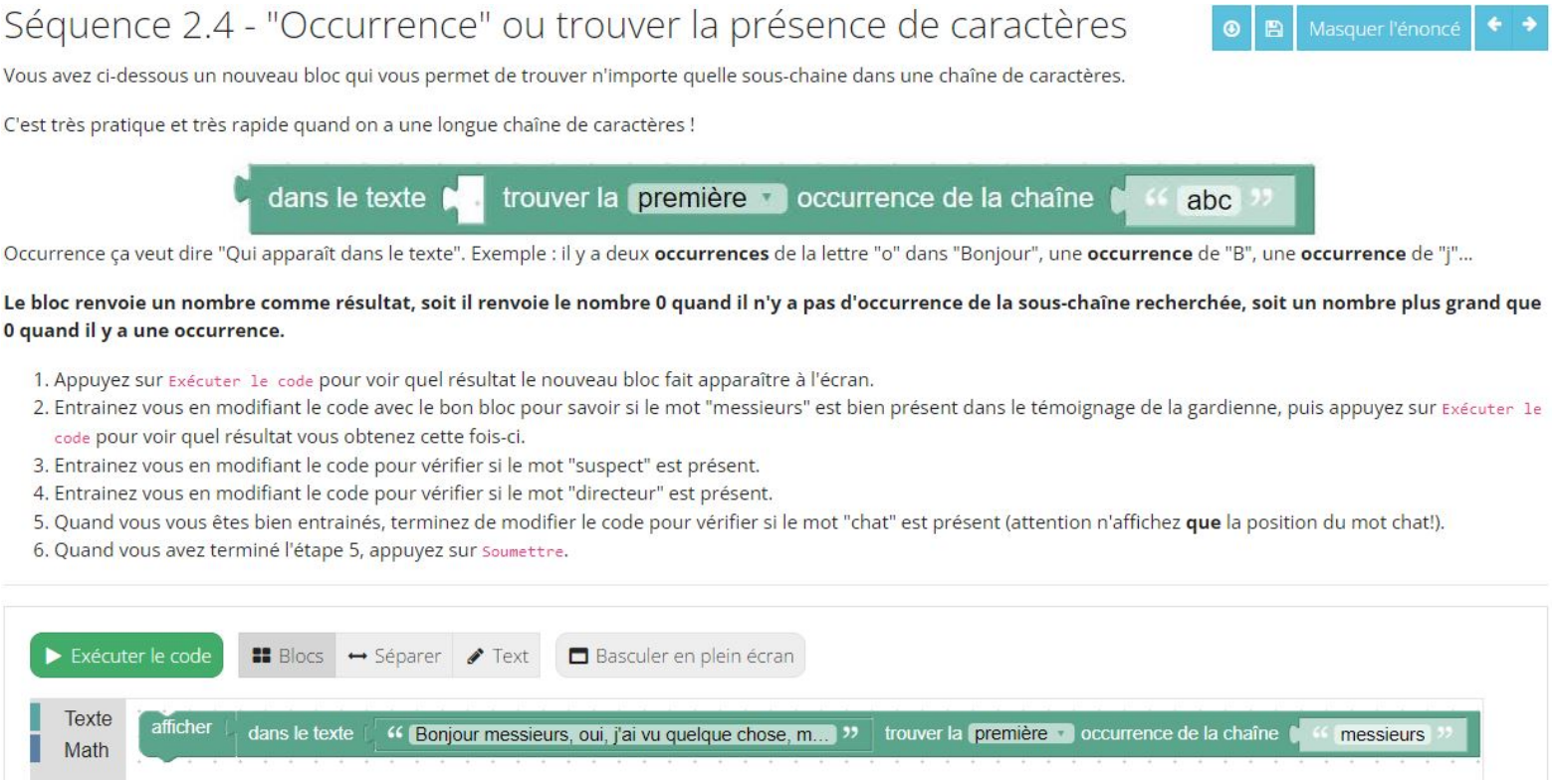


FIGURE 2.4 – Un des exercices *Run* de la séquence PRIMM.

Investigate: Cette phase permet à l'élève de s'attarder sur les concepts et de les analyser plus en profondeur, il a l'occasion de se pencher sur la structure du code, de regarder ligne par ligne ce que le code représente. Le professeur a un large choix de variations d'exercices pour cette phase. Il est important de bien accompagner l'élève dans sa découverte, car selon l'expérience de P. Bagge [22], livrés à eux-mêmes, certains élèves ne vont pas analyser le code avec attention ou essayer de formuler leurs propres questions. Dans cette phase, les élèves peuvent avoir à :

- Commenter le code. Les élèves commentent ligne par ligne ce que le programme fait, leur apportant une vision du flux de l'information.
- Expliquer le code, en pairs, ce qui aide à mémoriser le vocabulaire.
- Faire du *tracing*.
- Répondre à des questions.
- Dessiner le flux de contrôle, ce qui amène l'élève à analyser étape par étape le programme, améliorer sa logique, etc.
- Trouver des erreurs introduites volontairement.
- Résoudre des puzzles de **Parsons** [7]. Un puzzle de **Parsons** est un

programme que l'élève reçoit. Ce programme contient par exemple un ensemble de blocs de code mis dans le désordre et l'élève doit trouver comment recomposer le programme correctement.

Un exemple d'exercice *Investigate* est donné à la figure 2.5. Dans cet exercice, l'élève doit faire attention aux types de blocs utilisés. Une erreur volontaire est insérée et le but pour l'élève est qu'il doit se rendre compte qu'il n'est pas possible d'incrémenter sur une chaîne de caractères.

FIGURE 2.5 – Un des exercices *Investigate* de la séquence PRIMM.

Modify: Les élèves doivent dans cette phase modifier du code qu'ils ont déjà eu l'occasion d'analyser durant les exercices précédents. Les élèves commencent par une tâche simple, puis continuent une série de modifications du programme initial au travers de tâches de plus en plus complexes. Grâce à cette phase, les élèves vont progressivement passer d'une vision du code de *Not mine* à *Partly mine*, en effet le fait de modifier morceau par morceau le code va permettre une appropriation du code par l'élève, comme représenté à gauche et au milieu de la figure 2.8 page 18. Un exemple d'exercice *Modify* est donné à la figure 2.6. Dans cet exercice, l'élève reçoit tous les blocs nécessaires pour résoudre le problème, il doit modifier le contenu et l'emplacement des blocs pour arriver à la solution.

FIGURE 2.6 – Un des exercices *Modify* de la séquence PRIMM.

Make: Ici, les élèves ont à résoudre un nouveau problème en concevant leur propre programme. Ce problème utilise les mêmes concepts et la même structure qu'ils ont appris dans les phases précédentes. Ils peuvent donc appliquer les compétences assimilées dans un cadre familier. L'objectif ici est de permettre à l'élève de planifier et construire l'algorithme qui va résoudre le problème. Cette étape peut être complexe, car l'élève doit faire preuve de créativité et se rappeler son apprentissage lors des phases précédentes. En arrivant à créer une solution, l'élève se prouve qu'il s'est approprié les concepts ainsi que le code, il peut donc passer d'une vision du code de *Partly mine* à *All mine*, partie droite de la figure 2.8 à la page 18.
Un exemple d'exercice *Make* est donné à la figure 2.7. Dans cet exercice, on peut observer que la tâche n'est pas exactement telle que décrite ci-dessus. En prenant en considération la réalité du terrain, c'est-à-dire le peu de temps disponible dans les classes et la quantité de concepts couverts, les élèves ont eu 1h30 pour résoudre l'entièreté de la séquence. Il aurait été illusoire de vouloir donner une tâche *Make*, où ils doivent résoudre un nouveau problème de zéro dans ce laps de temps. C'est pourquoi, afin de fluidifier la séquence

et espérer qu'ils puissent tout couvrir dans le temps imparti, la plupart des tâches *Make* de la séquence sont des adaptations avec du *Modify* dedans.

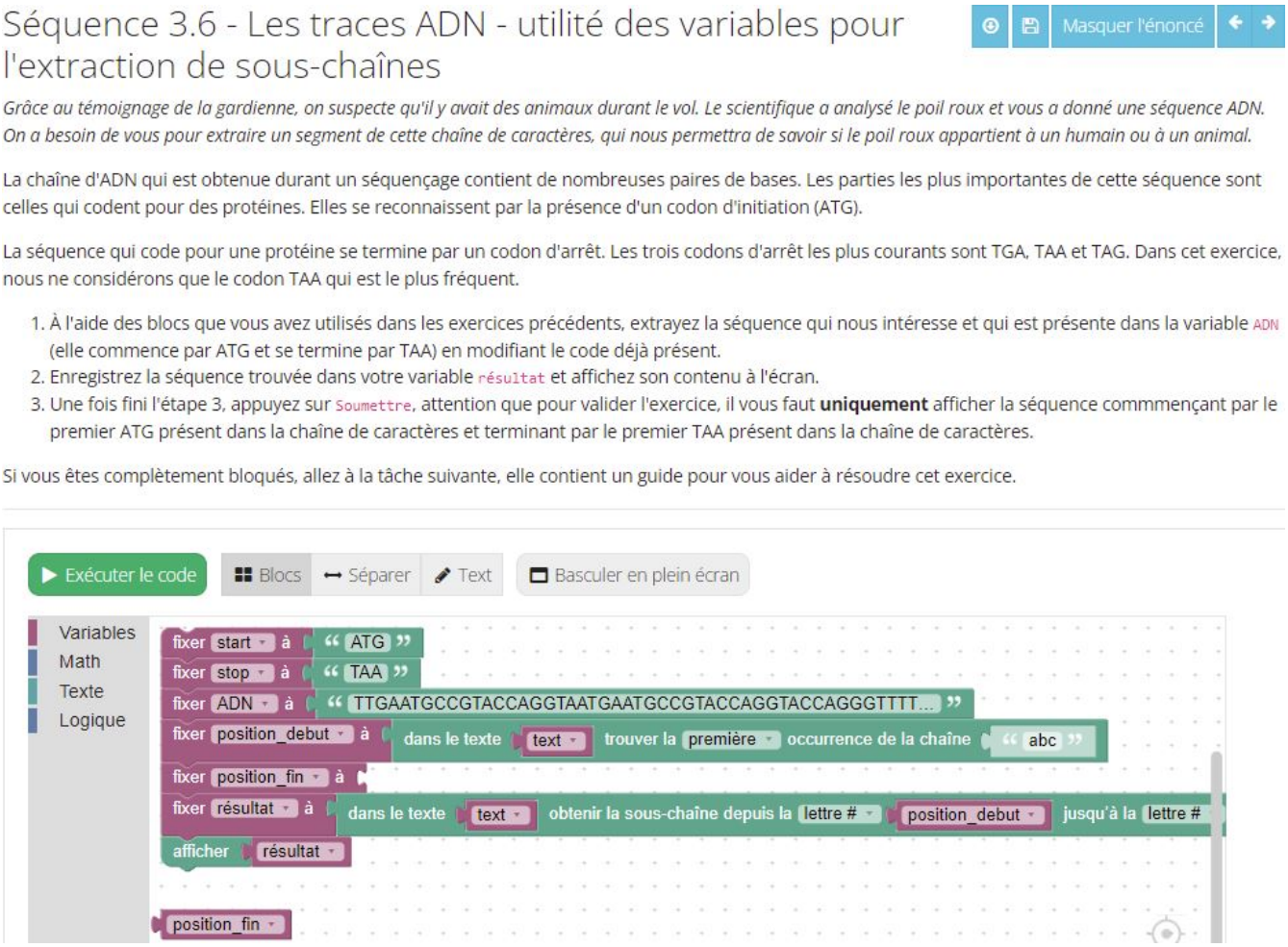


FIGURE 2.7 – Un des exercices *Make* de la séquence PRIMM.

Une caractéristique intéressante de PRIMM est que la méthodologie donne de la liberté d'adaptation au professeur, qui peut au mieux évaluer la progression des élèves et répondre aux besoins du terrain. Le temps passé sur chaque phase va dépendre des élèves, du professeur et du sujet abordé. Comme expliqué par Sue Sentance [16, 11], il est possible de passer une période de cours entière par phase si nécessaire, de faire des allers-retours entre les cycles si les élèves ont besoin de plus d'explications ou encore de faire un cours qui couvre toutes les phases. Une série d'exemples d'exercices créés par Sue Sentance peuvent être consultés ici [23, 24].

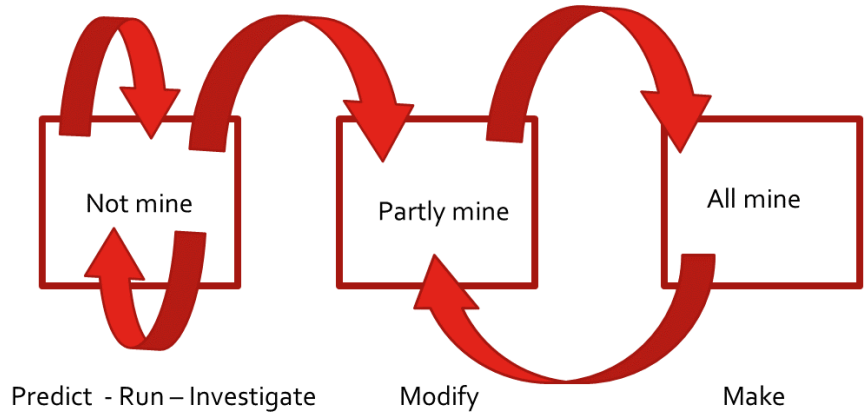


FIGURE 2.8 – Flux des étapes PRIMM.

2.2 Outils

Pour pouvoir enseigner la programmation, plusieurs moyens conventionnels existent tels que les livres de théorie ou les cours en classe. Cependant, pour des raisons physiques et matérielles, l'accès est potentiellement limitatif. Une solution rendue possible par Internet est d'avoir des cours en ligne, accessibles à tous ceux qui ont une connexion et un ordinateur à travers le monde. Certaines de ces formations sont payantes et restreintes, d'autres sont gratuites et ouvertes à tous. Les cours disponibles et gratuits s'appellent des *MOOC*[2], *MOOC* est l'acronyme de *Massive Open Online Courses* ou *formation en ligne ouverte à tous*. Pour pouvoir dispenser de manière autonome ces cours, il faut que l'élève puisse vérifier ses réponses directement sur la plateforme, sans attendre la correction d'une personne tierce. C'est maintenant rendu possible par les *autograder*, des logiciels de correction automatiques qui vont évaluer la réponse de l'élève, puis lui afficher une réponse en fonction du résultat. Parmi ces plateformes en ligne qui hébergent des cours avec *autograder*, il existe INGInious, décris à la section suivante.

2. https://www.mooc.org/

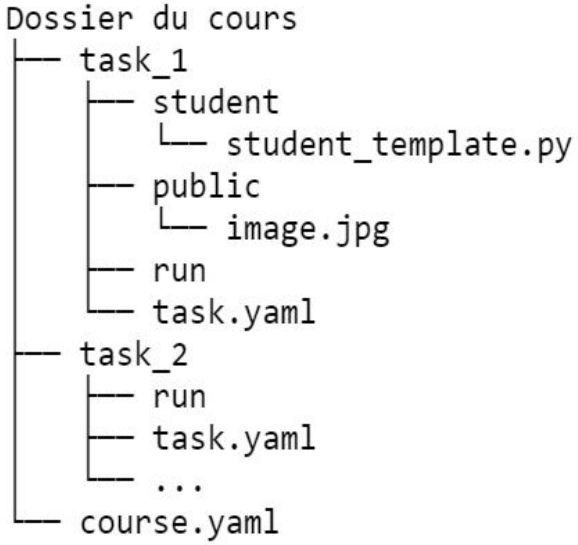
2.2.1 INGInious

INGInious [10] est une plateforme web développée par des élèves de l'UCL en 2014. Elle est ensuite reprise par une équipe de *sysadmins*, qui à ce jour, continuent son développement et sa maintenance active. Cette plateforme permet aux professeurs de concevoir et mettre à disposition des cours en ligne. Les élèves peuvent tester leurs réponses directement dessus, ils reçoivent ensuite une correction immédiate et automatique. INGInious est capable de gérer un nombre important de soumissions en parallèle grâce à Docker. Docker [25] est une plateforme de conteneurisation open source utilisée pour développer, déployer et gérer des applications dans des environnements virtualisés légers appelés conteneurs.

Parmi les caractéristiques intéressantes d'INGInious, il est possible pour le professeur de récupérer une série d'informations sur toutes les soumissions des élèves, telles que le nombre de tentatives, la solution proposée et le temps passé sur un exercice. Toutes ces informations donnent une visibilité au professeur, permettant d'analyser en profondeur la performance des élèves. Comme mentionné plus haut, une autre caractéristique intéressante est que l'élève reçoit un retour instantané et automatisé sur sa réponse, il a aussi accès à un historique de ses tentatives.
La structure type d'un cours sur INGInious est divisée en plusieurs fichiers et sous fichiers, observable dans la figure 2.9 :

- Un fichier **course.yaml** qui définit les différents paramètres du cours : le nom, les administrateurs, la liste des tâches qu'il contient, etc.
- Un dossier **public** qui va contenir les images et fichiers communs à la tâche.
- Un fichier **task.yaml** qui sert de configuration pour la tâche, il contient les différents paramètres : le nom de la tâche, le type d'exercice, les blocs contenus dans la tâche (quand il s'agit d'une tâche Blockly), l'environnement de test, etc.
- Un fichier **Python** qui récupère le code de l'élève, ce fichier contient une série de tests unitaires pour vérifier que l'élève ait répondu correctement, il envoie ensuite un message au fichier **run**.
- Un fichier **run** qui va récupérer le message du fichier **Python** et retourner un message positif ou négatif sur l'interface visuelle d'INGInious en fonction du message reçu.

FIGURE 2.9 – Structure d'un cours INGInious.



2.2.2 Programmation par blocs

Pourquoi utiliser la programmation par blocs ? Une étude de D. Weintrop et U. Wilensky [26], ayant pour but d'essayer d'évaluer l'impact d'apprendre la programmation au moyen de la programmation par blocs, a soulevé des points très intéressants. Pour mener cette étude, ils ont divisé leurs élèves en deux groupes, les deux groupes ont suivi exactement le même cursus, dans la même école, à la même vitesse et pendant le même intervalle de temps. Ils ont posé une série de questions aux élèves avant et après les cours. Leurs résultats ont montré que les élèves qui avaient suivi le cours avec la programmation par blocs ont :

- Montré significativement plus de gains d'apprentissages que les élèves qui suivaient le cours textuel.
- Exprimé un gain de confiance en eux par rapport à la programmation après avoir suivi le cours, ce qui n'était pas le cas pour les élèves qui suivaient le cours textuel.
- Performé mieux pour l'ensemble des critères d'évaluation établis que les élèves qui suivaient le cours textuel.
- Mieux compris les concepts de logique conditionnelle et logique itérative que les élèves qui suivaient le cours textuel.
- Exprimé un gain d'intérêt pour continuer à apprendre la programmation. Quant aux élèves qui suivaient le cours textuel, ils ont exprimé une diminution d'intérêt.

– Montré le même niveau d'appréciation durant la classe que les élèves qui suivaient le cours textuel.

Un des professeurs ayant participé à l'étude s'est exprimé sur son ressenti durant l'expérience, selon lui : *"Les blocs créent une atmosphère différente, ils ont l'air attrayants, c'est quelque chose que tu peux utiliser directement. Grâce à ça l'ambiance dans la classe est différente, les enfants sont plus enclins à discuter avec leurs voisins, plus enclins à plaisanter sur la matière".*

Une autre étude de D. Weintrop [27] a montré que les élèves ont mieux performé sur des questions traitant de la logique conditionnelle, les fonctions et les boucles définies. Il n'y a pas eu de différence pour les questions en lien avec les variables, les boucles indéfinies et la compréhension du programme. Encore une autre étude de TW. Price et T. Barnes [28] a trouvé peu de différence dans les résultats d'apprentissage, mais que les élèves en programmation par blocs complétaient plus rapidement les exercices. Ces deux études suggèrent que bien que les deux approches permettent aux élèves d'apprendre la même chose, ils peuvent progresser plus vite en programmation par blocs.

En prenant comme exemple la figure 2.10 qui contient un bloc *print* avec un bloc *text* à gauche et son équivalent textuel JavaScript à droite, certains avantages de la programmation par blocs deviennent évidents :

– La prise en main ainsi que l'utilisation des blocs sont faciles et rapides. Il suffit de les tirer, de les amener sur l'espace de travail et les coller entre eux.
– Les blocs sont plus lisibles et compréhensibles qu'une ligne de code textuelle équivalente. Dans l'exemple, les deux blocs s'avèrent visuellement plus rapides à comprendre que le code **window.alert() ;**.
– Ils demandent moins de charges cognitives pour l'élève. L'élève doit retenir moins de vocabulaire, de syntaxe et de commandes. On peut directement couvrir le concept voulu sans passer par beaucoup plus d'explications. Cette simple ligne de code **window.alert() ;** contient déjà plusieurs règles syntaxiques à absolument respecter sous faute d'erreur. L'élève ne doit pas faire de fautes de frappe, il lui faut comprendre que du texte va entre un guillemet de début et un guillemet de fin, il doit clôturer sa ligne par un point-virgule, il va peut-être se demander pourquoi il y a un point avant et une parenthèse après *alert*, etc.

FIGURE 2.10 – Bloc print et la syntaxe équivalente.

Bien que la programmation par blocs semble fonctionner efficacement pour des concepts simples ainsi que des petits programmes, là où le bât blesse, c'est que cet outil semble peu adapté pour des sujets plus complexes ou des projets de plus grande ampleur. L'étude de D. Weintrop [26] souligne que l'on ne sait pas encore comment la programmation par blocs aide la transition vers des langages de programmation plus conventionnels, tels que Python ou Java.
Des études ont testé **Scratch** [29] et **Alice**[30, 31, 32], deux langages de programmation par blocs similaires à **Blockly**.
Pour **Scratch**, qui a pour publique cible des jeunes en dessous de 18 ans, ils ont observé un investissement et un engouement chez les novices, de bons résultats dans l'ensemble, mais ont noté que la faible granularité du langage donnait comme résultat que les élèves ont développé de mauvaises habitudes de programmation. Ils n'ont pas une bonne vision d'ensemble ni une utilisation correcte des structures de programmation abordées durant l'étude.
Pour cet aspect-là, la programmation textuelle a donné de meilleurs résultats. Ils considèrent que **Scratch** est potentiellement plus efficace pour introduire des concepts de programmation, mais qu'il ne semble pas universellement meilleur que les langages textuels.
Pour **Alice**, qui vise plus les étudiants en sciences informatiques, au-dessus de 18 ans, ils ont observé une bonne rétention des concepts abordés, les élèves ont trouvé que la programmation par blocs était plus engageante et agréable à prendre en main.
Leur conclusion est que dans l'ensemble avec **Scratch**, la programmation par blocs permet un apprentissage plus rapide pour les novices, mais pas universellement plus efficaces que des langages de programmations comparables. La conclusion pour **Alice** est qu'**Alice** est plus attrayant et engageant que les alternatives textuelles, mais aussi que les élèves ont une meilleure rétention des concepts.

On peut donc déduire de ce qui est expliqué ci-dessus que la programmation par blocs présente des avantages non négligeables pour enseigner la programmation

à des débutants. Le fait que les élèves du groupe de programmation par blocs aient mieux performé dans l'ensemble des critères d'évaluation de l'étude de D. Weintrop [26], est une preuve supplémentaire que la programmation par blocs fournit des supports d'apprentissage pour l'introduction de concepts à des novices.

2.2.3 Blockly

Il existe dans INGInious, la plateforme de cours avec cotation automatique décrite plus haut, un plug-in qui permet de créer et manipuler des séquences de cours Blockly directement dessus. Blockly est une librairie logicielle JavaScript créée par Google, permettant de coder avec des blocs visuels sans s'encombrer de l'apprentissage syntaxique. Ces blocs visuellement explicites peuvent être tirés et déplacés à l'aide du curseur de la souris. Dans l'éventualité où un exercice requiert un bloc qui n'existe pas dans la librairie de base, il est possible pour le développeur de créer des blocs sur mesure en fonction de ses besoins.

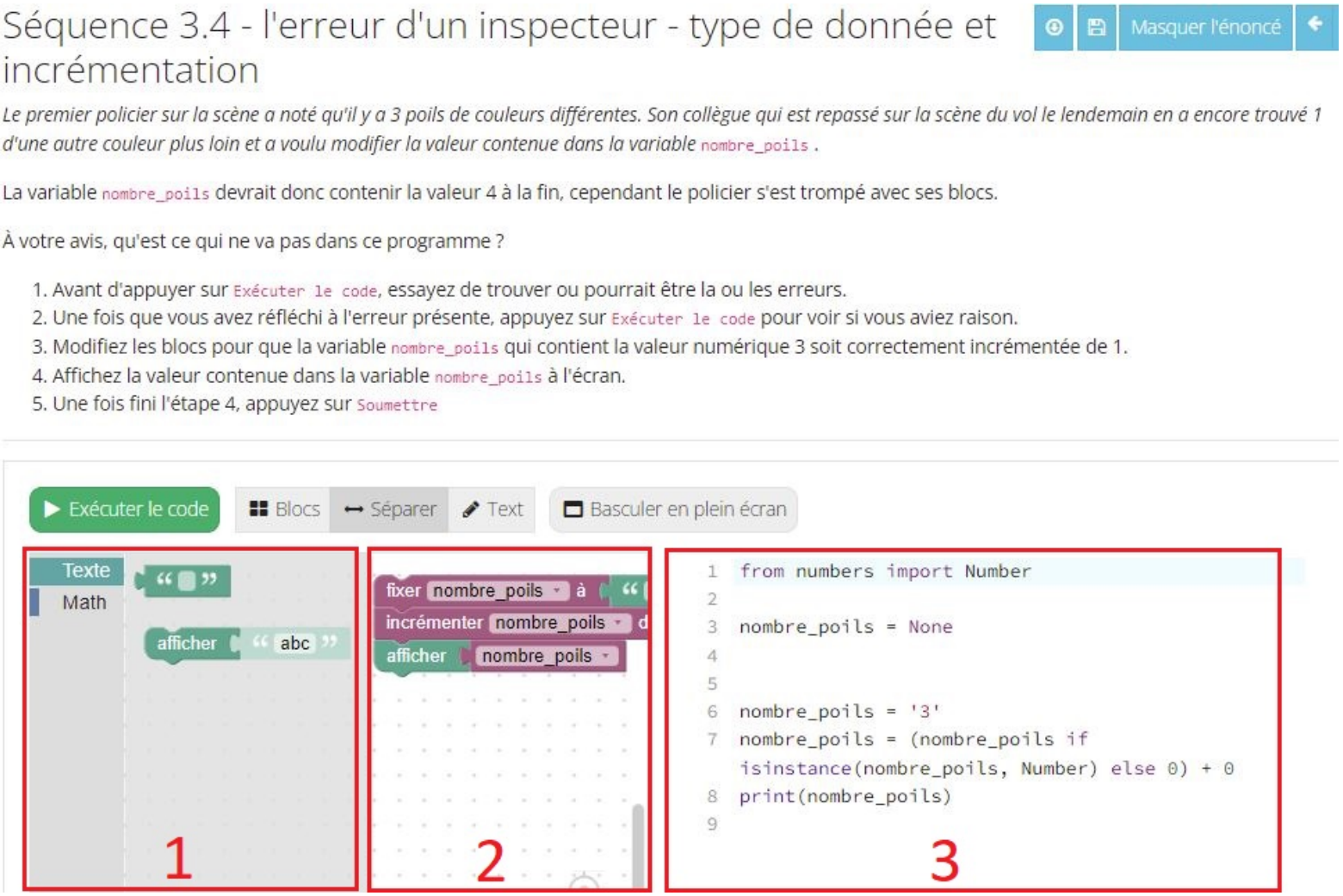


FIGURE 2.11 – Exemple d'un programme Blockly sur INGInious.

L'interface Blockly intégrée dans INGInious, observable dans la figure 2.11, est divisée en trois sections distinctes :

- Dans l'espace à gauche en rouge numéroté 1, il s'agit de la *toolbox* ou boite à outils, qui fournit à l'élève l'entièreté des blocs nécessaires à l'accomplissement de l'exercice. C'est lors de la création de la tâche que le professeur décide quels blocs il met à disposition dans la *toolbox*, car ça peut porter à confusion pour l'élève d'avoir accès à l'entièreté des blocs existants.
- Dans l'espace au centre en rouge numéroté 2, il s'agit du *workspace* ou l'espace de travail, dans lequel le code est écrit grâce aux blocs visuels. Il est possible d'avoir une série de blocs déjà mis en place pour l'élève.

– Dans l'espace à droite en rouge numéroté 3, il s'agit d'une interface qui affiche en temps réel la syntaxe correspondante aux blocs mis en place, ici en JavaScript, mais il est possible à l'utilisateur de changer le langage correspondant en Python, PHP et autre. Cette interface n'est pas éditable par l'utilisateur. L'élève ne peut donc pas commencer à coder directement dedans pour espérer voir les blocs correspondants apparaître.

Chapitre 3

Les séquences d'exercices et leurs différentes approches

Ce chapitre couvre la concrétisation des différentes séquences, leurs contenus et les concepts abordés. J'y décris comment, sur base de la séquence non-PRIMM, j'implémente ma séquence en suivant les différentes phases de la méthodologie PRIMM. Je parle des avantages et désavantages de chaque séquence, de la population qui a testé les séquences et ce qu'ils en ont pensé. Enfin, j'essaie d'évaluer au mieux l'impact que les modifications vers PRIMM a eu dans l'apprentissage des élèves.

3.1 Séquence non-PRIMM : Introduction à la bio-informatique

Cette première séquence, créée dans un objectif pédagogique ne suivant pas la méthodologie PRIMM, sert de point de référence pour le développement d'une séquence PRIMM. Elle se trouve à l'adresse [33].

3.1.1 But

Dans le cadre d'une activité semi-ludique semi-éducative appelée "Les biologistes mènent l'enquête", cette séquence créée par Olivier Bonaventure, professeur à l'UCLouvain, a pour but de donner envie à des élèves du secondaire de commencer un bachelier en bio-informatique. L'aboutissement de la séquence permet aux élèves de comparer une trace ADN trouvée sur un cadavre avec une liste de chaînes ADN, leur permettant d'identifier le coupable.

3.1.2 Contenu

Cette séquence traite les concepts suivants :

1. Manipulation de chaînes de caractères.
2. Recherche de la position d'une sous-chaîne.
3. Extraction d'une sous-chaîne.
4. Utilisation de variables.
5. Utilisation d'instructions conditionnelles.
6. Comparaison d'une sous-chaîne avec une liste de sous-chaînes.

3.1.3 Structure de la séquence - Analyse du point de vue PRIMM

Cette séquence contient au total sept exercices. La majorité des exercices est dans une optique de *Modify/Make*, que je décris brièvement ci-dessous :

- Après une brève introduction dans la première tâche pour permettre à l'élève de se familiariser avec la plateforme, comment manipuler les blocs et ce qu'est une chaîne de caractère, l'élève est rapidement confronté à une tâche qui contient plusieurs nouveaux concepts. Dans cette seconde tâche, on lui présente le concept de variable ainsi que brièvement le type de donnée, expliquant la différence entre le bloc de chaîne de caractères et le bloc de nombre. Cette tâche est dans une approche *Modify* puisqu'il y a déjà une partie du code en place dans l'espace de travail, l'élève doit le compléter avec les blocs de la *toolbox*.
- La troisième tâche couvre plusieurs concepts tels que la longueur d'une chaîne, la position d'un caractère dans la chaîne, l'occurrence d'une sous-chaîne, puis termine par l'extraction d'une sous-chaîne. On peut observer ici que l'exercice est orienté *Modify/Make*. *Modify* puisque pour résoudre la tâche, l'élève doit modifier des blocs déjà disponibles, mais principalement *Make* puisqu'il doit utiliser tous les nouveaux concepts cités pour arriver à extraire une sous-chaîne.
- La quatrième tâche qu'on peut observer à la figure 3.1 n'introduit pas de nouveaux concepts mais demande à l'élève de réutiliser ce qu'il a appris précédemment. Cette tâche est orientée *Modify/Make*, car l'élève doit trouver comment résoudre le problème avec les blocs de la *toolbox*, ainsi qu'une base de code déjà écrite dans le *workspace*.

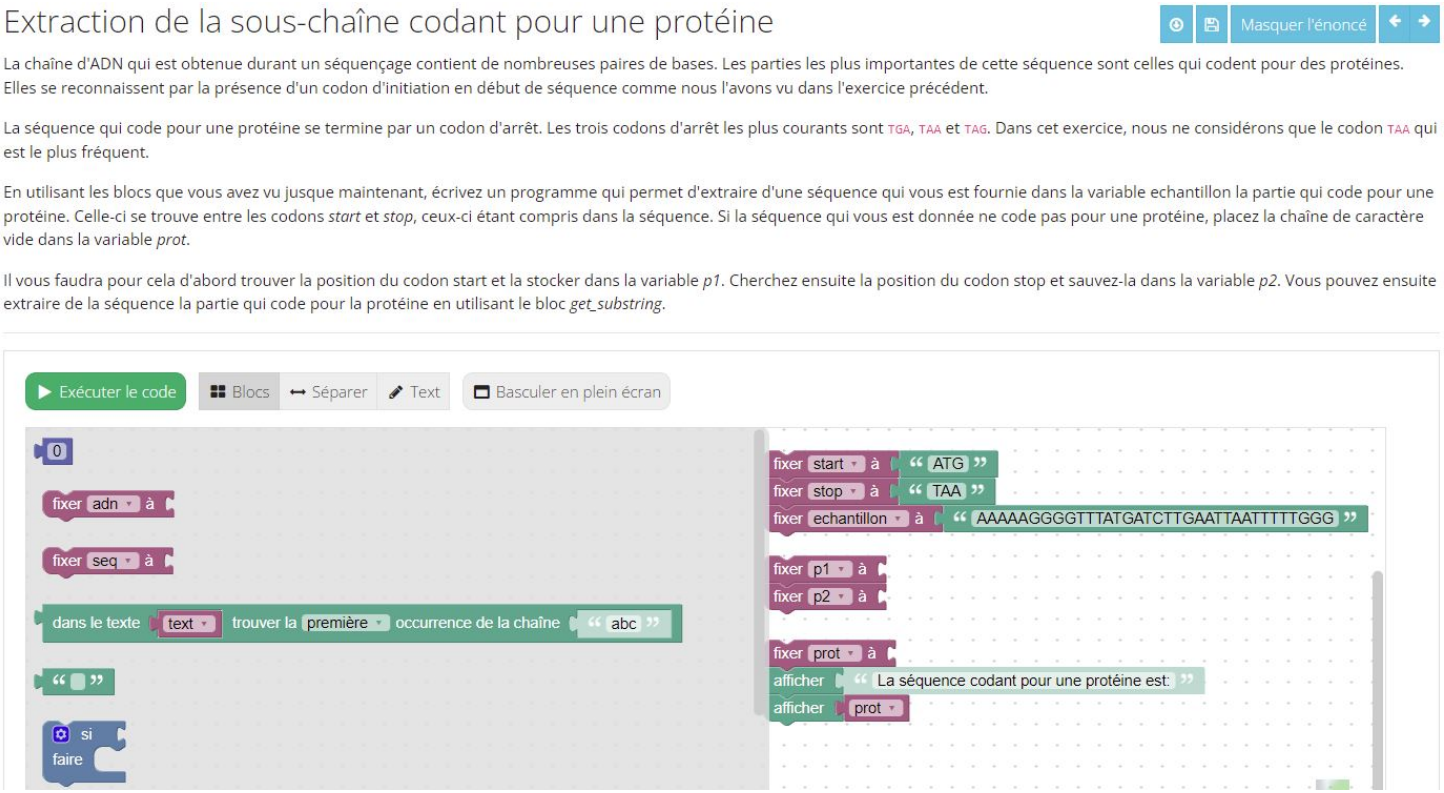


FIGURE 3.1 – Séquence non-PRIMM - Quatrième exercice.

- La cinquième tâche est ici orientée *Predict/Investigate* puisqu'il s'agit de questions à choix multiples. Cette tâche introduit le concept de comparaison logique, ainsi que les instructions de choix.
- Les sixième et septième tâches n'expliquent pas de nouveaux concepts, elles sont toutes les deux orientées *Modify/Make*, les élèves doivent ici résoudre des problèmes qui regroupent tous les concepts vus auparavant.

On peut donc observer que dans sa majorité, cette séquence fournit des exercices *Modify/Make*. Les exercices sont presque tous identiques dans leur résolution. Elle ne donne pas l'occasion à l'élève de lire ou d'exécuter du code déjà fonctionnel, l'élève a peu d'occasions d'enquêter sur des petits détails pour approfondir sa compréhension. Cependant, la difficulté qui augmente graduellement rejoint ici l'optique de la méthodologie PRIMM avec la transition du *Modify* vers le *Make*, l'élève modifie des tâches de plus en plus complexes pour arriver à un exercice final, où il lui est demandé de résoudre un nouveau problème.

3.1.4 Population

La population ayant participé à cette activité est un groupe de 30 élèves de rhétorique (17-18 ans). Une petite portion des élèves avait des bases en programmation, néanmoins la majorité était débutante. Les élèves étaient par groupes de deux. Le temps alloué pour la séquence est d'une heure.

3.1.5 Retour des élèves

La complexité pour les élèves résidait principalement dans la clarté des objectifs, ayant tendance à ne pas lire les énoncés, ils ne savaient pas ce qu'il fallait faire. Une minorité d'élèves est arrivée très vite à l'exercice final. Aucun groupe d'élèves n'a réussi à finir l'exercice final. Néanmoins, le ressenti des élèves était positif dans l'ensemble.

3.1.6 Conclusion

Le but initial de cette séquence était de permettre à des élèves d'avoir un aperçu de ce qu'il est possible de faire avec la programmation, de leur donner envie de s'inscrire au bachelier en bio-informatique. Les élèves ont pu avoir accès à une résolution de l'entièreté des exercices, ils ont pu recevoir l'information sur l'ADN pour trouver le coupable de leur enquête. Beaucoup d'entre eux ont soulevé des questions intéressantes et ont montré de l'intérêt durant l'atelier. Cet atelier ainsi que les élèves ayant participé ont apporté beaucoup d'informations pour servir de point de comparaison avec la séquence PRIMM.

3.2 Séquence PRIMM : Les bio-informaticiens mènent l'enquête

Comme expliqué précédemment, les exercices qui ont pour objectif d'enseigner la programmation sont trop souvent orientés *Make*. La séquence discutée ci-dessus illustre ce problème, sur les sept tâches, une seule n'est pas *Modify/Make*. La séquence discutée dans cette section est développée sur base de la séquence précédente, les concepts abordés sont similaires, mais ils sont segmentés en rajoutant des exercices orientés *Predict/Run/Investigate*. Cette séquence se trouve à l'adresse [34].

3.2.1 But

Les objectifs sont de créer une séquence selon la méthodologie PRIMM, d'arriver à analyser ce que la méthodologie PRIMM apporte de concret par rapport à une méthodologie plus conventionnelle. Une optique secondaire est que la séquence soit facile à prendre en main, tant pour le professeur qui doit donner la leçon que pour l'élève qui doit la recevoir. Suivant les objectifs pédagogiques de PRIMM, l'idée est de permettre aux élèves d'arriver à s'approprier facilement de nouveaux concepts, d'être accompagnés progressivement dans leur apprentissage, et si possible, de retenir un maximum d'informations sur le long terme.

3.2.2 Contenu

Comme expliqué plus haut, cette séquence est volontairement similaire à la précédente dans les concepts traités, cependant elle contient beaucoup plus d'exercices pour éviter d'avoir plusieurs nouveaux concepts dans une seule tâche. Là où la séquence précédente incorporait tous les concepts discutés en sept exercices, ici la séquence en contient vingt-deux. L'idée est de permettre à l'élève de bien faire la distinction dans la matière abordée :

1. Manipulation de chaînes de caractères.
2. Recherche de la position d'une sous-chaîne.
3. Extraction d'une sous-chaîne.
4. Utilisation de variables.
5. Utilisation d'instructions conditionnelles.

Chacune des sous-séquences s'efforce de décomposer les concepts abordés en incorporant des *Predict/Run/Investigate*. Considérant le nombre d'exercices, afin d'éviter un aspect routinier qui pourrait diminuer l'intérêt de l'élève, j'ai voulu éviter de suivre systématiquement une structure qui commencerait par un *Predict*, puis un *Investigate*, suivi par un *Run*, continuant sur plusieurs *Modify*, terminant sur un *Make*.

Séquence 1 : Introduction à Blockly

Dans cette sous-séquence de quatre exercices, l'élève reçoit une introduction pour se familiariser avec la plateforme, les blocs à déplacer, les boutons sur lesquels appuyer. Du point de vue PRIMM, l'élève doit *Predict/Investigate* à la tâche 2 sur le comportement séquentiel du code. L'idée est d'attirer son attention sur le fait que le code est lu de haut en bas, que chaque instruction "afficher" déclenche un message. L'élève peut tester à la troisième tâche le fonctionnement d'un nouveau bloc, qui permet de modifier une chaîne de caractère en la mettant en majuscule ou

en minuscule. Cette tâche, commençant par un *Run* puis continuant sur du *Modify*, est représentée à la figure 3.2. La séquence d'introduction termine par un exercice simple *Modify/Make* dans lequel l'élève réutilise les blocs et les concepts abordés.

FIGURE 3.2 – Séquence PRIMM - Troisième exercice de l'introduction.

Séquences 2 à 4

Chaque sous-séquence contient six exercices. Pour plus de lisibilité, les concepts abordés et l'approche PRIMM de chaque sous-séquence sont décrits plus en détail respectivement dans les tableaux 3.6 page 35, 3.7 page 36 et 3.8 page 37. Des exemples visuels sont disponibles aux figures 3.3 page 32, 3.4 page 33 et 3.5 page 34.

FIGURE 3.3 – Séquence PRIMM - Les caractères d'une chaîne et leurs positions - *Predict/Investigate*.

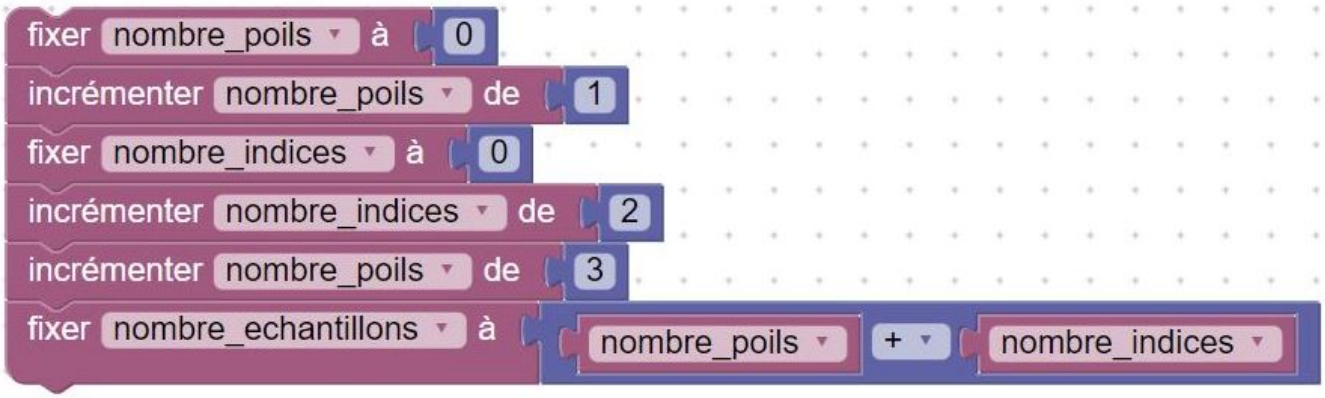
Séquence 3.5 - les variables et leur contenu

Masquer l'énoncé

Le laboratoire voudrait bien savoir combien de tests il va devoir faire au total, c'est pourquoi il fait la somme des indices et des poils récoltés pour ses échantillons. (mais il a besoin de garder une trace du nombre de poils et du nombre d'indices séparément pour le dossier.)

Les variables sont très pratiques pour ne pas devoir écrire plusieurs fois une information, on aura tendance à les réutiliser à travers tout un programme. Vous pouvez par exemple prendre deux variables, `nombre_indices` et `nombre_poils`, additionner leur contenu puis mettre le résultat dans une troisième variable `nombre_echantillons`.

1. À votre avis, combien d'échantillons au total le laboratoire recevra-t-il ?



Le nombre d'échantillons affiché sera :

- ○ 2
- ○ 0
- ○ 1
- ○ 6
- ○ 4

FIGURE 3.4 – Séquence PRIMM - *Tracing* du contenu de variables - *Predict/Investigate*.

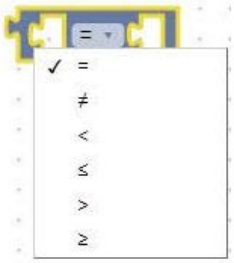
Séquence 4.1 - introduction aux conditions

Masquer l'énoncé

En informatique, on utilise le concept de **condition** très souvent.

Avec le bloc ci-dessous, vous pouvez évaluer si une condition est vraie (**TRUE** en anglais) ou fausse (**FALSE** en anglais), ça nous permet de prendre une décision en fonction du résultat. Attention que le résultat sera en anglais

Dans la liste déroulante de ce bloc, vous avez accès à plusieurs comparateurs tels que "égal", "différent de", "plus petit que", ...



Question 1: À votre avis, quelle valeur sera affichée ici ?

○ False

○ True

Question 2: À votre avis, quelle valeur sera affichée ici ?

○ True

○ False

FIGURE 3.5 – Séquence PRIMM - Introduction aux conditions - *Predict/Investigate*.

Exercices	PRIMM	Contenu
2.1	Run - Modify	L'élève peut appuyer sur **exécuter directement pour tester un nouveau bloc**, ce bloc calcule la longueur d'une chaîne de caractères. Ensuite, il doit modifier les blocs existants, lui permettant de se familiariser et de s'approprier le concept.
2.2	Modify	L'élève doit réutiliser le bloc précédent dans une situation légèrement plus complexe.
2.3	Predict - Investigate	Introduction sur le concept de **position dans une chaîne de caractères.** L'élève doit analyser le fonctionnement d'un nouveau bloc, qui retourne le caractère dans une chaîne à la position définie. L'élève doit ensuite répondre à une série de questions à choix multiples pour vérifier sa compréhension. Cet exercice pousse l'élève à passer du temps à analyser, lire et se poser des questions sur les chaînes de caractères. Grâce à INGInious, l'élève reçoit un commentaire pour chacune des réponses cochées, qu'elle soit bonne ou mauvaise, ce qui lui donne des indices et des explications supplémentaires.
2.4	Run - Modify	Introduction sur le concept d'**occurrence d'une sous-chaîne dans une chaîne**. L'élève reçoit une introduction théorique qu'il peut observer lui-même en exécutant le code déjà écrit. L'élève doit ensuite modifier le code pour tester différentes combinaisons.
2.5	Predict - Investigate	Introduction sur le concept d'**extraction de sous-chaîne**. L'élève doit analyser le fonctionnement d'un nouveau bloc, qui extrait une sous-chaîne présente dans une chaîne en utilisant la position de début et de fin de la sous-chaîne. Après l'avoir analysé, il doit deviner le résultat d'un petit cas pratique pour vérifier sa compréhension.
2.6	Modify - Make	Il n'y a pas de nouveaux concepts ici, **l'élève doit appliquer ce qu'il a appris jusque-là.** L'élève doit trouver la position de début et de fin d'une sous-chaîne puis l'extraire.
2.6-Bis	-	Cet exercice sert d'aide de résolution pour les élèves ayant des problèmes. Le problème est décomposé en plusieurs sous-problèmes pour simplifier la compréhension.

FIGURE 3.6 – Contenu de la sous-séquence 2 sur les chaînes de caractères.

Exercices	PRIMM	Contenu
3.1	Modify	Introduction sur le concept de **variable**. L'élève doit simplement modifier des blocs déjà existants pour remplir son nom et son âge.
3.2	Make	Introduction sur le concept d'**incrémenter la valeur contenue dans une variable**. L'élève doit réutiliser ce qu'il a appris dans l'exercice précédent ainsi que le nouveau bloc.
3.3	Predict - Investigate	Introduction sur le concept de **types de données**. L'élève doit lier et analyser une série d'exemples mettant l'accent sur les nombres et les chaînes de caractères. Il vérifie sa compréhension à l'aide de questions à choix multiples.
3.4	Investigate - Run - Modify	Cet exercice repasse sur les types de données, une erreur est volontairement introduite dans un code déjà fourni dans l'espace de travail. L'élève doit enquêter sur l'origine de l'erreur, exécuter le code pour voir si sa supposition est correcte, puis l'élève doit trouver comment réparer le code.
3.5	Predict - Investigate	Cet exercice est basé sur le **tracing**. L'élève doit analyser le contenu de variables dans un programme simple, il doit trouver quelle est la valeur finale d'une variable suite à plusieurs additions et incrémentations.
3.6	Modify - Make	Comme dans la séquence 2 sur les chaînes de caractères, il n'y a pas de nouveaux concepts ici. Cet exercice cherche à appuyer sur les concepts abordés depuis le début de ce cours, dans l'espoir de stimuler la mémoire de l'élève ainsi que de créer des connexions entre les différents concepts. L'élève doit ici trouver les positions de début et de fin d'une sous-chaîne, à l'aide de sous-chaînes contenues dans des variables. L'élève doit ensuite extraire une sous-chaîne à l'aide des positions obtenues qu'il aura enregistré à l'aide de variables.
3.6-Bis	-	Cet exercice sert d'aide de résolution pour les élèves ayant des problèmes. Le problème est décomposé en plusieurs sous-problèmes pour simplifier la compréhension.

FIGURE 3.7 – Contenu de la sous-séquence 3 sur les variables.

Exercices	PRIMM	Contenu
4.1	Predict - Investigate	Introduction sur le concept de **conditions**. L'élève doit répondre à une série de questions à choix multiples.
4.2	Predict - Investigate	Suite de l'exercice précédent sur le concept de conditions. L'élève doit répondre à une série de questions à choix multiples.
4.3	Modify	Cet exercice contient tous les blocs nécessaires déjà disponibles dans l'espace de travail, cependant aucun des blocs n'est au bon endroit. Il s'agit d'un **puzzle de Parsons**, expliqué au chapitre 2. L'élève est amené à tester sa compréhension des concepts abordés lors des deux exercices précédents.
4.4	Modify	Le quatrième exercice est similaire au précédent, il contient lui aussi un puzzle de Parsons. Cet exercice repasse sur les concepts vus plus tôt dans cette séquence, mais augmente légèrement en difficulté.
4.5	Modify	Le cinquième exercice continue sur la lancée des deux précédents, il va augmenter la difficulté avec un troisième puzzle de Parsons. **Cet exercice reprend des concepts présents dans les séquences précédentes**. L'élève doit trouver s'il y a occurrence d'une sous-chaîne, contenue dans une variable, puis afficher un message en fonction du résultat à l'aide d'un bloc de condition.
4.6	Modify - Make	Le sixième et dernier exercice de ce cours **reprend l'entièreté des concepts abordés depuis le début**. L'élève doit manipuler des variables, trouver les positions d'une sous-chaîne, l'extraire, puis chercher s'il y a occurrence de cette sous-chaîne dans une autre chaîne.

FIGURE 3.8 – Contenu de la sous-séquence 4 sur les conditions.

3.2.3 Population

Deux groupes distincts ont participé à cette activité. Le premier groupe est une classe de 13 élèves de rhétorique de Tournai (17-18 ans) en option sciences fortes. Les élèves étaient seuls ou par groupes de deux sur un ordinateur, pour un total de 8 groupes. Aucun des élèves n'avait de base en programmation. Le temps alloué pour la séquence est d'une heure et demie. Le deuxième groupe est une classe de 13 élèves de quatrième (13-14 ans, système français) de Paris. Les élèves avaient un ordinateur chacun. Les élèves ont participé à une compétition de robotique - *Lego*, ils ont eu l'occasion d'apprendre à manipuler Scratch. Ils avaient donc des bases de programmation. Le temps alloué pour la séquence est de deux fois une heure, étalé sur deux semaines. Concrètement, la première ainsi que la deuxième séance n'auront duré que 40-45 minutes suite à la mise en place, la connexion ainsi que des interruptions extérieures.

3.2.4 Développement et améliorations

Un problème rencontré dans cette séquence PRIMM ainsi que dans la précédente est que les élèves ne prenaient pas le temps de lire l'énoncé. Ce cours pourrait gagner à avoir plus de lisibilité dans les énoncés. Il faudrait segmenter l'énoncé en trois parties. Ces trois parties reprendraient la théorie explicative, l'histoire et finalement ce qu'on attend de l'élève pour résoudre l'exercice. Cette structure devrait être respectée dans l'entièreté du cours pour que les élèves puissent aller directement à l'essentiel s'ils le désirent.

Un autre problème, maintenant résolu, est que par défaut, la *toolbox* de Blockly est aussi large que son plus large bloc, ce qui empiète sur le *workspace*, rendant la manipulation de certains exercices inutilement compliquée sur des petits écrans. Ce problème est résolu en mettant les blocs dans des catégories. Le cadre rouge dans la figure 3.9 représente l'espace disponible par défaut pour travailler en l'absence de catégories. Le cadre bleu montre le *workspace* qui est aussi large que le plus large bloc. Grâce aux catégories, la *toolbox* est par défaut aussi large que le bloc orange, elle ne se déploie que quand l'utilisateur va chercher un bloc depuis une catégorie.

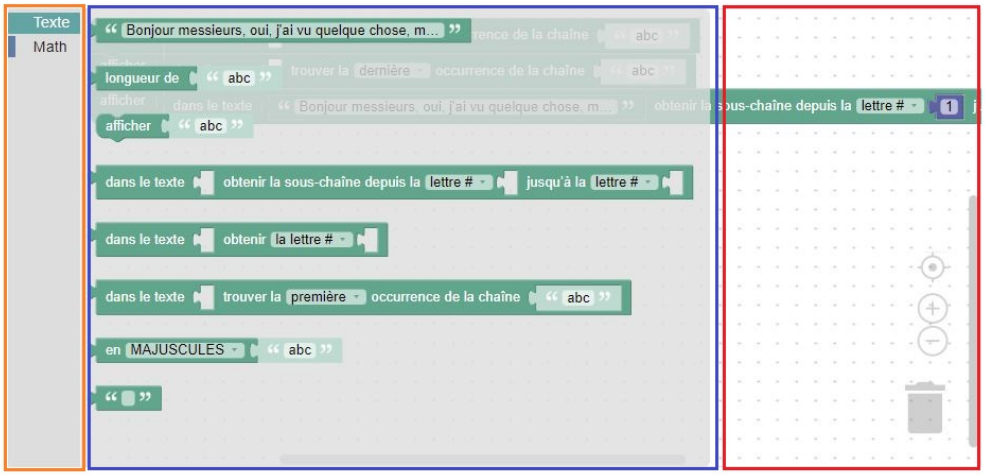


FIGURE 3.9 – Problème de largeur de la toolbox

Un dernier problème bien plus important est dans la manière dont j'ai créé les exercices. J'ai voulu éviter d'inclure le concept de fonctions en plus, car il me semblait que rajouter encore de la matière serait préjudiciable à l'apprentissage. Ne pas utiliser de fonctions, en combinaison avec le fait que l'élève doit résoudre l'exercice pour un cas précis et unique, s'est révélé être un frein pour les tests unitaires. Ces tests unitaires sont cruciaux pour contrôler si la réponse de l'élève est correcte ou non, ce qui est finalement bien plus dommageable pour l'étudiant que d'avoir une rapide introduction aux fonctions. À cause de cette erreur, l'élève a différentes solutions incorrectes pour valider certaines tâches sans jamais atteindre la bonne solution, ce qui peut va potentiellement résulter en une incompréhension, un mauvais apprentissage et peut mener à de la frustration. Certains exercices de cette séquence gagneraient donc à être modifiés dans leur structure pour rectifier ce manquement.

3.2.5 Retour des élèves

Pour tenter d'évaluer l'impact de PRIMM ainsi que le ressenti des élèves, j'ai rédigé deux itérations d'un formulaire qu'ils ont rempli après avoir participé. Je me suis rendu compte que la majorité des questions de la première itération n'étaient pas bien formulées, c'est pourquoi le formulaire a été réécrit pour la deuxième classe et que la plupart des réponses du premier formulaire ne sont pas pertinentes. Les réponses aux formulaires sont discutées au chapitre 4, section 4.1.2.

3.2.6 Conclusion

Modifier la séquence initiale non-PRIMM en une séquence PRIMM a soulevé un certain nombre de défis. Tels que :

- Arriver à fournir suffisamment d'exercices à l'élève pour qu'il ait compris en profondeur un nouveau concept.
- Trouver la bonne quantité de tâches intermédiaires permettant un échelonnage fluide pour l'élève.
- Trouver la bonne complexité de ces tâches intermédiaires.
- Garder l'intérêt de l'élève tout le long du cours, malgré le grand nombre de tâches.
- Ne pas être trop répétitif.
- Arriver à ce que l'élève sache utiliser le bon concept au bon moment.

Dans l'ensemble, j'estime qu'adapter la séquence initiale en une séquence PRIMM est une réussite. Les deux groupes ayant participé ont eu l'air d'apprécier, mais aussi d'arriver à manipuler une quantité non négligeable de nouveaux concepts en peu de temps. Je pense que la méthodologie PRIMM apporte beaucoup de points positifs pour l'apprentissage de l'élève. La variété des exercices, les exemples concrets, la possibilité de tester ses connaissances, représentent pour moi une base utile sur laquelle l'élève peut s'appuyer avant de devoir mettre en pratique.

3.3 Comparaison entre les deux séquences

Dans cette section je vais comparer les deux séquences du point de vue de PRIMM, je décris les forces et les faiblesses que j'ai pu observer dans chacune d'elles.

3.3.1 Forces et faiblesses de la séquence "Introduction à la bio-informatique" - Séquence non-PRIMM

Forces par rapport à une séquence PRIMM

- Cette séquence contient peu d'exercices en tout, une séquence PRIMM diviserait les concepts en plus d'exercices. Ici, le peu d'exercices diminue la redondance. En effet, la répétitivité peut potentiellement baisser l'intérêt des élèves les plus rapides.
- Cette séquence contient beaucoup de concepts abordés en seulement une heure, ce qui permet aux élèves d'avoir un aperçu de ce qu'il est possible de faire grâce à la programmation. PRIMM vise plus à s'attarder sur chaque concept, comme expliqué au chapitre 2.1 consacré à PRIMM.

Faiblesses par rapport à une séquence PRIMM

- Cette séquence passe peu de temps à expliquer chaque concept, mais aussi couvre beaucoup de nouvelle matière en seulement une heure. Les élèves passent rapidement d'un concept à un autre, il y a plusieurs nouveaux concepts par tâches. Les élèves ont peu de temps pour assimiler, s'approprier, comprendre et retenir correctement les concepts.
- Vu le peu d'exercices, il est facile d'être découragé ou d'abandonner pour l'élève s'il y a un concept qu'il n'a pas eu le temps d'assimiler et dont il a besoin pour résoudre la tâche suivante.
- Peu de temps est passé à expliquer l'utilité de chaque bloc. Les élèves ont eu tendance à ne pas utiliser le bon bloc dans la bonne situation, ils oublient vite leur utilité, particulièrement les variables. Les élèves réécrivaient le contenu des variables ailleurs, au lieu d'utiliser le bloc variable, suggérant une mauvaise assimilation du concept. PRIMM avec son échafaudage plus lent des concepts pourrait permettre une meilleure rétention de l'information.
- Cette séquence contient des concepts qui ne sont jamais introduits. Dans la figure 3.10 montrant le *workspace* du dernier exercice, l'élève voit pour la première fois une liste d'éléments à parcourir ainsi qu'une boucle pour évaluer chaque élément de ladite liste. Les élèves ne savaient pas comment commencer à résoudre cet exercice, ils n'osaient pas toucher à la boucle et ne comprenaient pas le concept de liste.

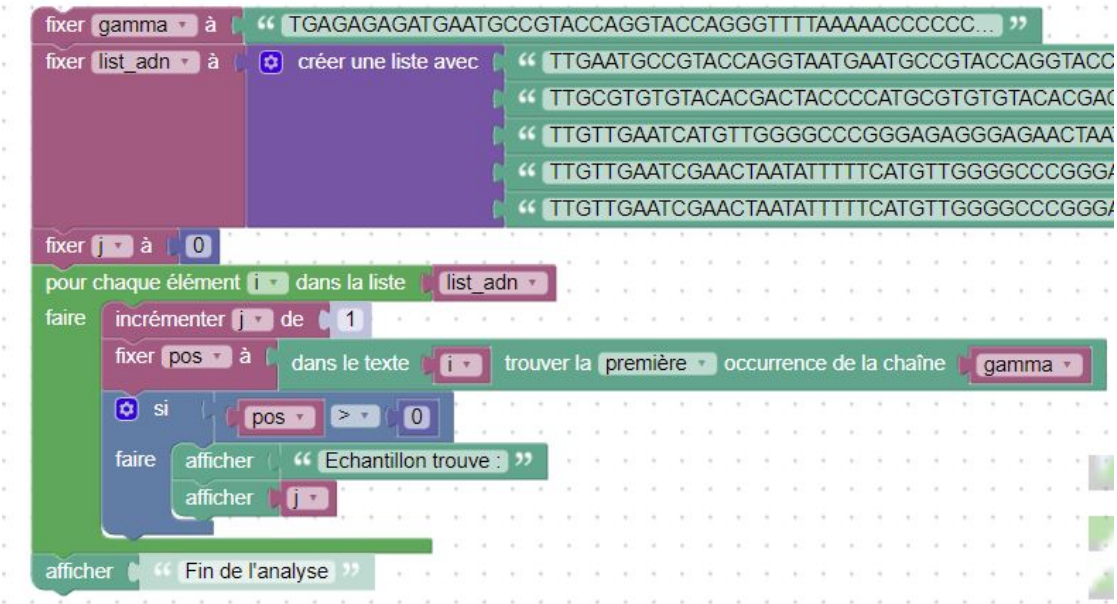


FIGURE 3.10 – Exercice final de la séquence non-PRIMM - Liste et boucle.

3.3.2 Forces et faiblesses de la séquence PRIMM

Forces par rapport à une séquence non-PRIMM

— La séquence est facilement décomposable en plusieurs séances de cours. Chaque concept (chaîne de caractères, variables et conditions) est divisé en sous-séquence, il est donc possible de s'adapter à la progression des élèves ou encore d'ignorer une séquence si le concept est déjà acquis par les élèves.

— La séquence permet aux élèves de tester ses connaissances sur un concept de plusieurs manières. La séquence fournit pour ça des questions à choix multiples, du code déjà fourni à exécuter ainsi que la possibilité de revenir à plusieurs reprises sur les mêmes concepts, avec des exercices graduellement plus complexes.

— La séquence permet aux élèves de passer du temps sur chaque concept, de s'approprier progressivement le code en devant le modifier dans des exercices simples, jusqu'à résoudre des exercices plus complexes.

Faiblesses par rapport à une séquence non-PRIMM

— La séquence contient un nombre important d'exercices, ce qui peut être ennuyant pour les élèves plus rapides qui vont trouver ça long.

— La séquence contient peu d'exercices *Make*, comme discuté précédemment. Étant donné le temps disponible, il aurait été déraisonnable de faire un exercice final *Make* où l'élève doit résoudre un nouveau problème avec un *workspace* vide.

3.3.3 Qu'est-ce que PRIMM a apporté ?

Comme discuté ci-dessus, le fait d'adapter la première séquence selon la méthodologie PRIMM semble améliorer la séquence selon les points suivants :

— En décomposant les concepts en des exercices plus simples, plus rapides, PRIMM permet un meilleur échelonnage de difficulté pour l'élève.
— En plus d'un meilleur échelonnage, l'élève est mieux accompagné dans son apprentissage, on court alors moins de risque que l'élève abandonne devant un trop grand écart de complexité entre deux exercices.
— En n'incluant pas plus d'un nouveau concept par exercice, on espère permettre à l'élève de bien compartimenter ce qu'il vient d'apprendre.
— L'élève a plus le temps de bien retenir un concept ainsi que le bloc correct qui va avec.
— L'élève doit passer du temps à lire, analyser le code, lui offrant plus d'opportunités de mémorisation et de clarté d'esprit.
— L'élève doit vérifier sa compréhension des concepts ainsi que l'utilité du bloc correspondant de plusieurs manières.
— Les exercices sont aussi plus variés dans leur approche, évitant un aspect répétitif, on espère ainsi arriver à stimuler l'élève tout au long du cours.

Passer de sept exercices à vingt-deux exercices a amené plusieurs craintes. Parmi mes craintes initiales, une en particulier était qu'il serait difficile d'inclure autant de concepts en si peu de temps. Le premier groupe ayant testé ma séquence a réussi à terminer en une heure et demie, ce qui est plus long d'une demi-heure que le temps nécessaire pour la séquence non-PRIMM. La majorité des élèves du deuxième groupe a aussi terminé en une heure et demie. Ce qui suggère que cette approche est adaptée à plusieurs tranches d'âge.

Une autre crainte était que les élèves risqueraient de trouver la séquence PRIMM trop longue ou redondante, à part pour quelques élèves, cette crainte était infondée dans l'ensemble.

Chapitre 4

Validation et résultats

Ce chapitre, toujours dans le but d'évaluer l'apport de PRIMM, décrit puis analyse les résultats obtenus grâce aux différentes séances données. Les critères d'évaluations sont les suivants :

- Le ressenti de madame Carine Grancher, professeur en informatique dans une école secondaire à Paris. Ce qu'elle a pu observer pendant la séance, les questions des élèves et ses commentaires.
- Mon ressenti. Ce que j'ai observé durant les séances, les problèmes rencontrés, les questions soulevées.
- Les résultats observables sur INGInious. Par exemple la qualité des soumissions des élèves ou encore jusqu'où sont-ils parvenus.
- Les réponses des élèves aux questionnaires. Ces questionnaires s'efforcent d'évaluer si modifier la séquence non-PRIMM vers PRIMM les a aidés dans leur apprentissage.

4.1 Résultats bruts

4.1.1 Professeur externe - Observations et ressenti - Carine Grancher

Pour pouvoir calibrer le mieux possible le contenu de la séquence, le niveau de difficulté ainsi que pour tester la séquence en situation réelle, il était nécessaire d'avoir l'aide d'un professeur externe. Madame Carine Grancher a permis de pouvoir remplir un des objectifs secondaire de ce mémoire, qui était d'obtenir la participation d'un professeur extérieur à l'UCLouvain. Madame Grancher m'a apporté son aide et donné la séquence PRIMM dans sa classe. Suite aux deux séances, j'ai pu récupérer son retour, un après chaque séance.

Première séance - Premier retour

Durant la première séance, madame Grancher a pu observer que les élèves :

— Ont apprécié la nouveauté, ils n'avaient jamais eu l'occasion de coder de cette manière.
— Ont apprécié la complexité, ils ont trouvé que les exercices n'étaient pas simples, qu'ils représentaient un défi.
— Ont eu du mal à lire les consignes, ils avaient du mal à faire la différence entre l'histoire, la théorie et ce qu'il fallait répondre dans l'exercice.
— Ont eu du mal avec le concept d'occurrence, ne connaissant pas la signification de ce mot, malgré la présence d'une définition dans l'énoncé, ce qui rejoint le point précédent.
— Ont été très autonomes, ils ont eu très peu besoin de son assistance.

Son ressenti personnel était que :

— Les élèves avaient l'air sincèrement contents de participer.
— Pour une minorité d'élèves plus rapides, la séquence était un peu lente.
— Il serait intéressant de structurer les énoncés pour que les élèves sachent clairement ce qu'il est attendu d'eux.
— Certains élèves ont été frustrés par le concept d'espace dans les chaînes de caractères. Point que madame Grancher a trouvé très utile, ayant observé des problèmes de compréhension avec ce concept dans plusieurs de ses classes.

Madame Grancher a conclu ce premier retour en expliquant qu'elle n'avait aucun doute que ses élèves aient appris, car grâce à PRIMM et ses différentes approches, ils ont pu manipuler, toucher et tester.

Deuxième séance - Deuxième retour

Les points observés durant la deuxième séance par madame Grancher sont :

— Les exercices comportent une réelle diversité ainsi qu'une réelle dynamique.
— Les exercices racontent une histoire que les élèves ont apprécié.
— Les élèves doivent occuper différents rôles / positions grâce à la structure variée des exercices.
— La progression de la difficulté est bien amenée.
— Les pré-requis / l'introduction de nouveaux concepts sont bien établis pour résoudre les exercices.

Selon madame Grancher, décomposer les concepts en petits morceaux est une bonne chose, les petits exercices étaient nécessaires pour faciliter la compréhension. Fournir des exercices préliminaires simples qui expliquent un nouveau concept lui semble très important pour aider les élèves à comprendre.

4.1.2 Résultats questionnaire

La figure 4.1 ci-dessous regroupe les réponses du premier groupe ayant testé la séquence PRIMM, auquel treize élèves ont répondu.

FIGURE 4.1 – Réponses du premier groupe au premier questionnaire.

La figure 4.2 ci-dessous regroupe les réponses du premier groupe à la question ouverte : “Quels sont les concepts de programmation que vous avez retenus ?”. Six des treize élèves ont répondu :

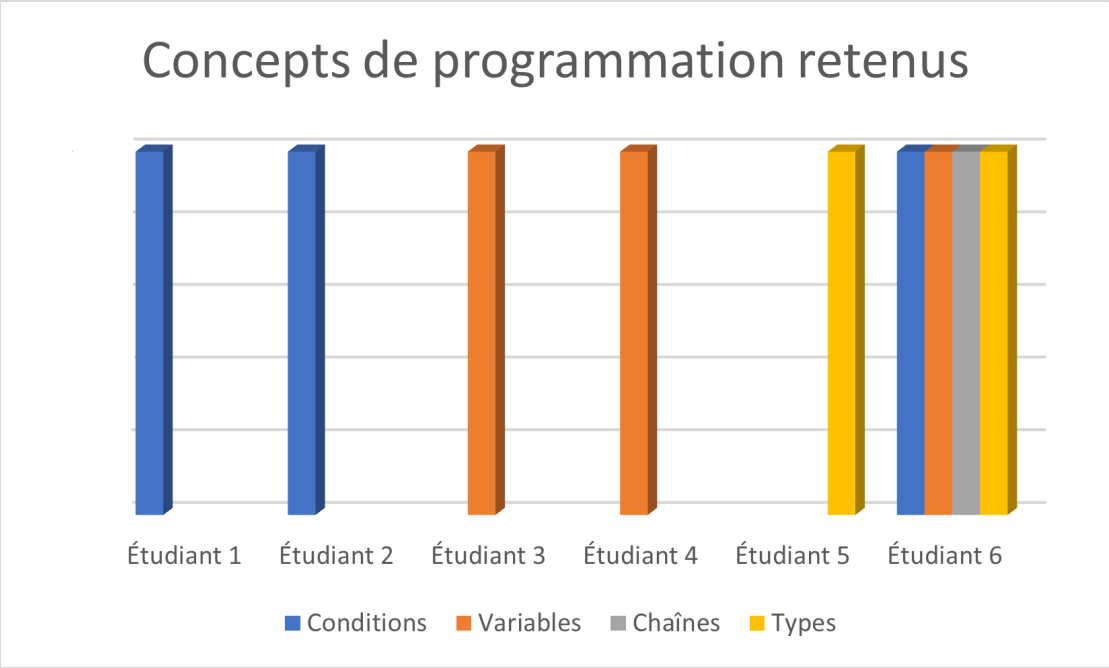


FIGURE 4.2 – Réponses du premier groupe au premier questionnaire.

La figure 4.3 ci-dessous regroupe les réponses du premier groupe à la question ouverte : “Comment vous êtes-vous senti pendant les exercices ? (intéressé, agacé, content, stressé, perdu ...)” :

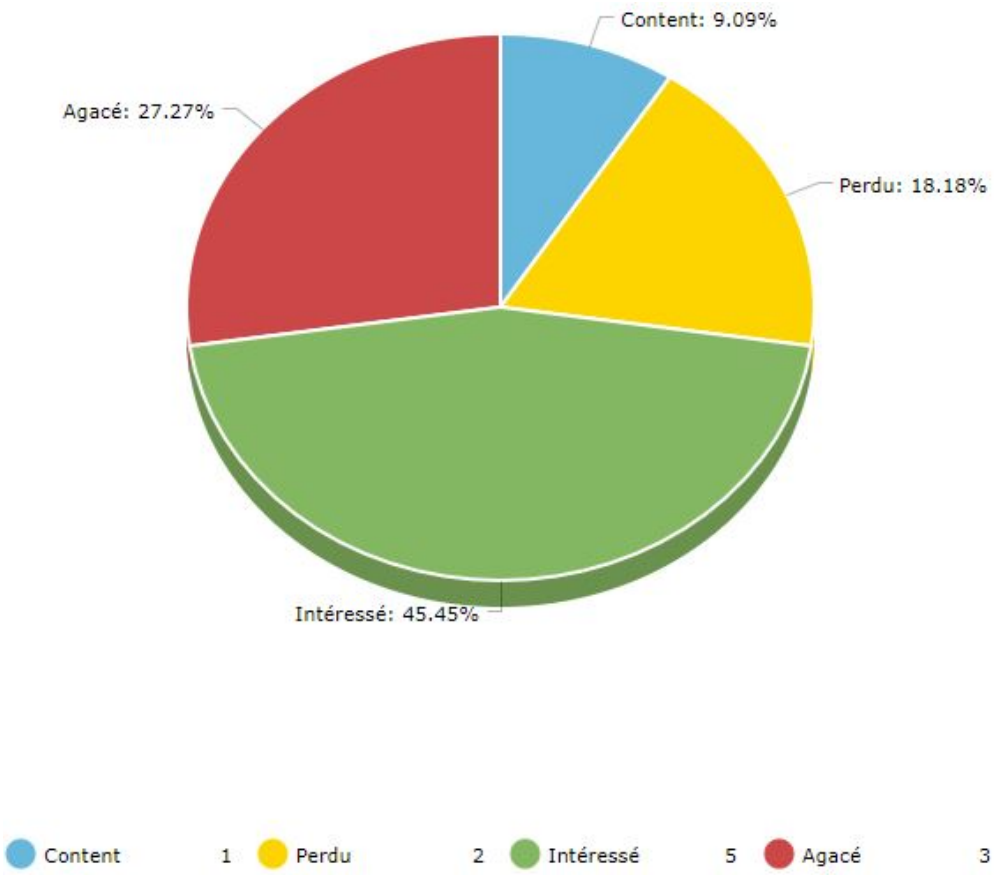


FIGURE 4.3 – Réponses du premier groupe au premier questionnaire.

À la question ouverte : “Avez-vous trouvé les exercices intéressants ? Lesquels et pourquoi ?”. Six élèves ont répondu :

— Trois élèves ont répondu qu’ils avaient tout trouvé intéressant, car ça les aidait à comprendre le fonctionnement de la programmation. Un d’eux a spécifié que l’histoire était intéressante et permettait de les encadrer tout en leur montrant l’utilisation de l’informatique dans une enquête.
— Deux élèves ont répondu que la séquence leur a permis de voir l’application pratique de l’informatique.
— Un élève a répondu que l’exercice final était particulièrement intéressant parce que ça lui a permis de revoir tous les concepts depuis le début.

— Un élève a répondu que les exercices de prédictions étaient intéressants, car ils étaient faciles à comprendre et l'aidaient bien pour la suite.

La figure 4.4 ci-dessous regroupe les réponses du deuxième groupe ayant testé la séquence PRIMM, auquel dix élèves ont répondu.

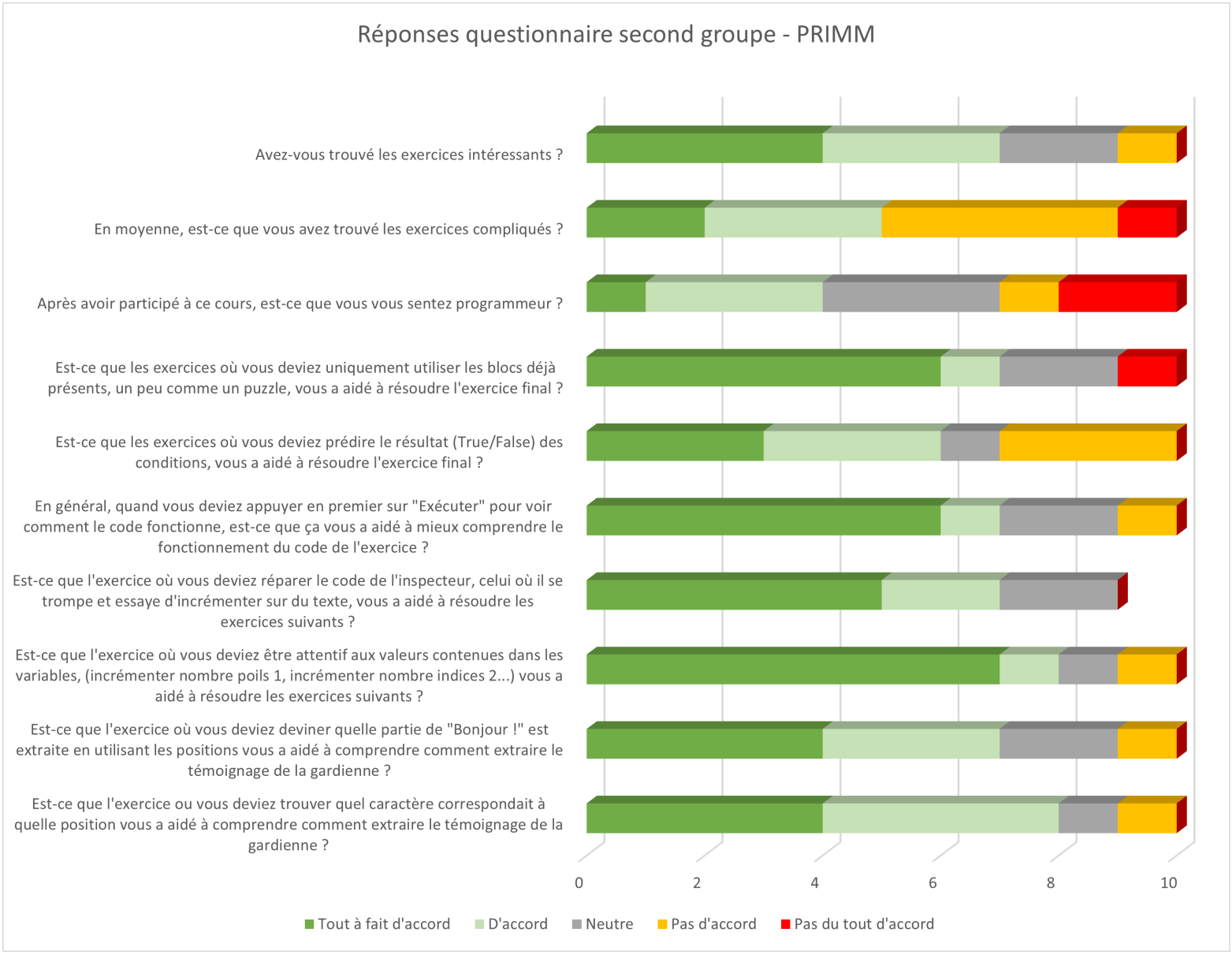


FIGURE 4.4 – Réponses du second groupe au deuxième questionnaire.

À la question ouverte :“Quels sont les concepts de programmation que vous avez retenus ?”. Trois élèves ont répondu :

— Un élève a répondu : “Extraction de chaînes de caractères”.
— Un élève a répondu : “Syntaxe Python”.
— Un élève a répondu : “Changer le contenu des variables”.

À la question ouverte :“Y a-t-il des exercices en particulier qui étaient trop compliqués ?”. Sept élèves ont répondu :

— 3 élèves ont répondu le 3.6, où ils devaient extraire une sous-chaîne avec l'aide de la majorité des blocs vus depuis le début.
— 1 élève a répondu “aucun”.
— 3 élèves ont mentionné les positions dans les chaînes et le concept d'occurrence.

Madame Grancher a rédigé puis soumis une évaluation une semaine après la deuxième séance à ses élèves. Les résultats de cette évaluation sont affichés dans la figure 4.5 ci-dessous.

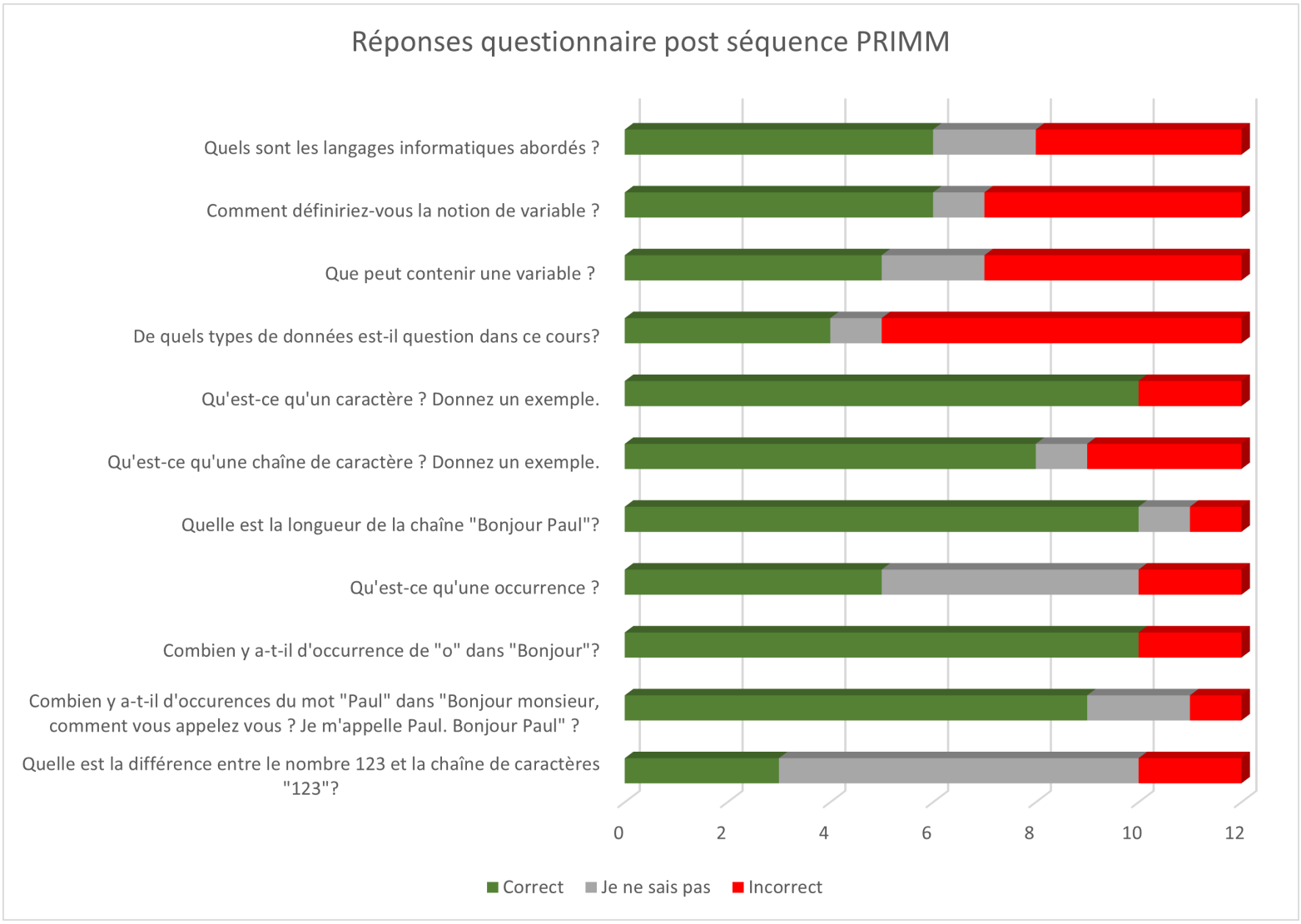


FIGURE 4.5 – Réponses du second groupe au questionnaire rédigé par madame Grancher.

4.1.3 Résultats INGInious

La graphique 4.6 reprend différentes statistiques récupérées sur INGInious. On peut y voir pour chaque exercice le nombre d'étudiants ayant tenté de répondre, le nombre d'étudiants ayant réussi, mais aussi le nombre total de tentatives pour l'ensemble des étudiants. On peut y observer que, à part pour les deux derniers exercices, les élèves ayant essayé un exercice ont réussi l'exercice. La majorité des élèves sont parvenus à finir les exercices. Le nombre total de soumissions est parfois augmenté par un ou deux élèves ayant tenté toutes les solutions possibles sans réfléchir.

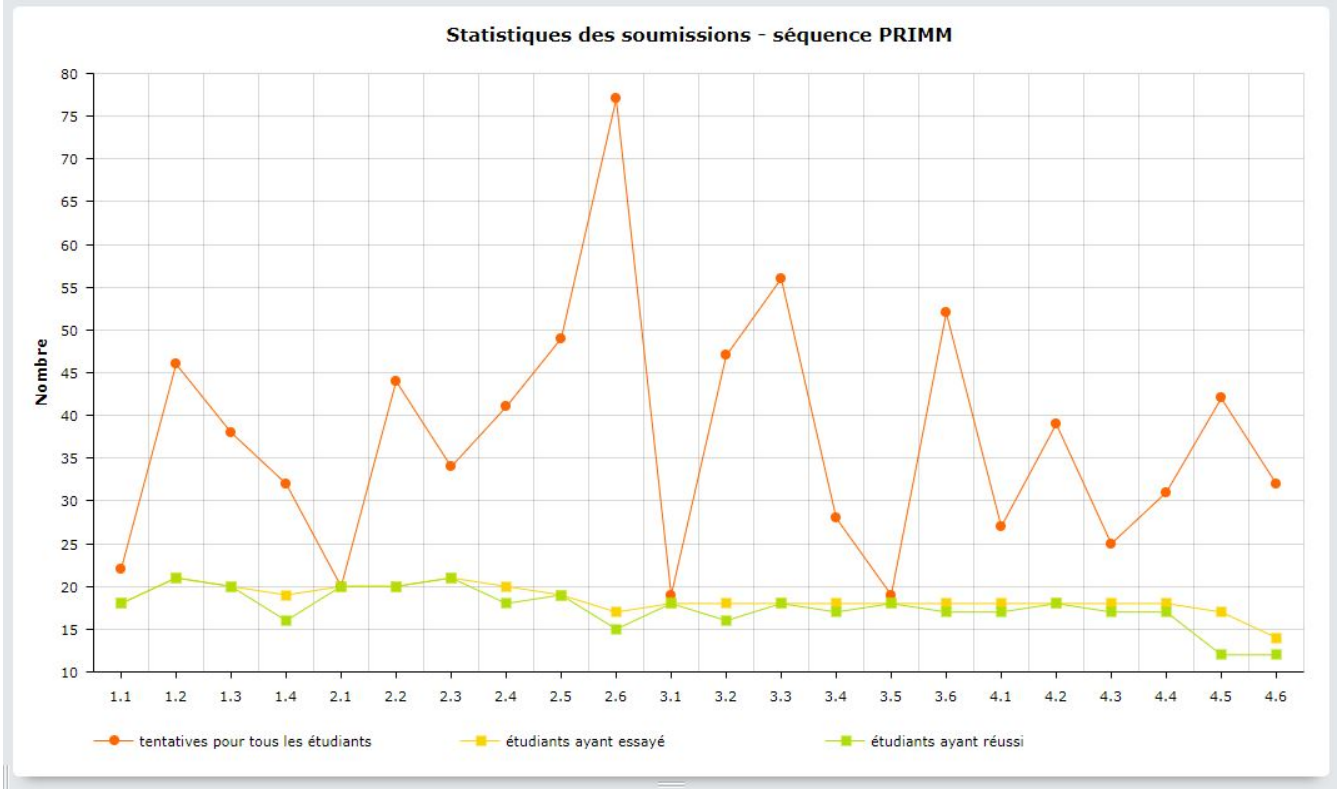


FIGURE 4.6 – Résultats INGInious.

En analysant l'ensemble des soumissions pour chaque exercice de la séquence PRIMM, j'ai pu observer les points suivants :

- Pour les exercices *Predict* - *Investigate*, qui comptabilisent sept exercices au total :
  - Les performances étaient très bonnes, tant pour le premier que pour le second groupe.
  - Les élèves des deux groupes ont presque systématiquement répondu correctement à leur première tentative.
  - Les concepts ayant légèrement posé problème aux deux groupes étaient l'ordre séquentiel dans lequel est lu le code, le type de donnée ainsi que les caractères spéciaux tels que l'espace ou la virgule.
- Pour les exercices *Run* - *Modify*, qui comptabilisent cinq exercices au total :
  - Les performances étaient très bonnes, tant pour le premier que pour le second groupe.
  - Le concept ayant largement posé problème dans les deux groupes est survenu à l'exercice sur le bloc qui permet de trouver l'occurrence d'une sous-chaîne. Les élèves n'utilisaient pas la bonne chaîne dans laquelle chercher l'occurrence d'une sous-chaîne.
- Pour les exercices *Modify*, qui comptabilisent cinq exercices au total :
  - Les performances étaient très bonnes, tant pour le premier que pour le second groupe.
  - Le concept ayant largement posé problème dans les deux groupes était, ici aussi, la recherche d'occurrence d'une sous-chaîne.
- Pour les exercices *Modify* - *Make*, qui comptabilisent trois exercices au total (pour rappel, ces exercices sont les aboutissements de chaque sous-séquence, demandant à l'élève de réutiliser les concepts vus précédemment) :
  - Exercice 2.6 - Trouver les positions d'une sous-chaîne puis l'extraire :
    - Les tests de vérifications de la solution étant imparfaits, certains élèves ont pu valider en utilisant des positions incorrectes. Parmi ces élèves, plusieurs ont tout de même fait l'exercice correctement, trouvant les bonnes positions, mais se sont contentés de n'utiliser que la position de début.
    - Certains élèves ne comprenaient toujours pas le mot "occurrence" à cette étape.
    - Un élève a trouvé la réponse en imbriquant directement tous les blocs nécessaires dans un seul bloc, au lieu d'extraire les positions, puis de remplacer les valeurs des blocs nombres contenant les positions.
    - Un élève a inversé la chaîne et la sous-chaîne, cherchant l'occurrence de la grande chaîne dans la sous-chaîne.
    - Plusieurs élèves ont fait l'exercice correctement et ont trouvé rapide-

ment la solution correcte.

— Exercice 3.6 - Trouver les positions d'une sous-chaîne, puis l'extraire, mais cette fois en utilisant des variables :
  — Plusieurs élèves ont trouvé la réponse correcte du premier coup.
  — Plusieurs élèves ne comprenaient pas qu'il fallait modifier la variable "text" vers la variable "ADN", visible à la figure 4.7 encadrée en rouge.
  — Plusieurs élèves ont copié le contenu des variables "start/stop/ADN" pour mettre la chaîne directement dans le cadre orange de la figure 4.7, montrant qu'ils ne comprennent pas suffisamment bien l'utilité des variables.
  — Plusieurs élèves utilisaient correctement les variables, mais au mauvais endroit, inversant par exemple les variables de position de début et de fin pour l'extraction, comme dans le cadre bleu de la figure 4.7.
  — Plusieurs élèves utilisaient incorrectement les variables, les collants un peu n'importe où, comme dans le cadre noir de la figure 4.7.

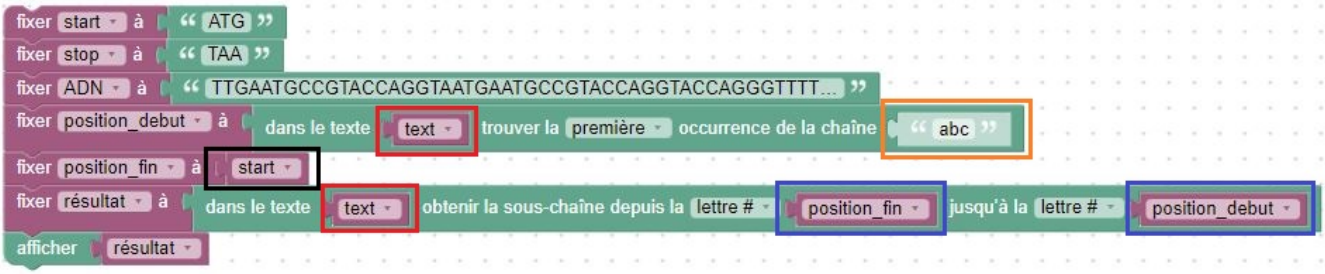


FIGURE 4.7 – exercice 3.6 - Problème avec les variables.

— Exercice 4.6 - Trouver les positions d'une sous-chaîne, l'extraire, à l'aide de variables, mais cette fois en utilisant les blocs de conditions :
  — Cet exercice était particulièrement susceptible aux réponses incorrectes. Plusieurs élèves ont validé l'exercice avec une réponse incorrecte.
  — Dans le premier groupe, seulement deux groupes d'élèves ont correctement utilisé l'extraction de sous-chaînes.
  — Dans le second groupe, uniquement un élève a correctement extrait la sous-chaîne. Un deuxième élève était sur la bonne voie.
  — Pour le reste des élèves des deux groupes, il semblerait qu'ils n'aient pas compris comment utiliser les variables ainsi que leurs contenus, ils n'ont pas compris comment utiliser les blocs d'extraction de sous-chaînes.

— Pour les exercices d'aide 2.6.bis et 3.6.bis, trois élèves du premier groupe et quatre élèves du deuxième ont répondus qu'ils en avaient eu besoin.

4.1.4 Observations et ressenti - Cyrille Debongnie

Premier groupe séquence non-PRIMM

Durant la toute première séance, j'ai pu observer les problèmes suivants en répondant aux questions :

— Plusieurs élèves ne savaient pas bien réutiliser les blocs d'une tâche à l'autre, même s'ils venaient d'en apprendre le fonctionnement.
— La majorité des élèves a très vite oublié le concept expliqué dans une tâche, ils étaient donc perdus quand ils devaient réutiliser ce concept dans la tâche suivante.
— Plusieurs élèves ne savaient pas comment créer une variable, comment réutiliser une variable.
— Plusieurs élèves n'ont pas du tout compris la différence entre les blocs de texte et les blocs de nombre.
— Plusieurs élèves ne comprenaient pas comment utiliser le bloc de condition "Si..Sinon".
— Plusieurs élèves ne lisaient que partiellement, voire pas du tout les énoncés.
— Tous les élèves ont été déstabilisés à l'exercice final, à cause de la liste et de la boucle qui ne sont jamais expliqués.

Deuxième groupe séquence PRIMM

Durant la deuxième séance, j'ai pu observer les problèmes suivants en répondant aux questions :

— Plusieurs élèves ne lisaient que partiellement, voire pas du tout les énoncés.
— Plusieurs élèves ne réutilisaient pas les bons blocs, préférant réécrire le contenu d'une variable plutôt que d'utiliser la variable.
— La majorité des élèves avançaient rapidement sur les exercices de début de séquence, mais avaient du mal avec les exercices de fin de séquence, où ils doivent réutiliser les concepts qu'ils viennent d'apprendre.

Troisième groupe séquence PRIMM

Avec le troisième et dernier groupe, n'ayant eu l'opportunité d'assister qu'à la deuxième partie, à distance, mes observations ont été limitées. Durant la première séance, la majorité des élèves sont arrivés à la fin de la séquence 2, sauf deux élèves très rapides qui sont arrivés à la fin de la séquence 3. Les élèves avaient l'air de facilement discuter entre eux, comparer leurs réponses ou encore poser des

questions à leurs voisins. L'ambiance avait l'air bonne dans l'ensemble. Certains élèves avaient l'air agacés à l'exercice 2.6, dans lequel il faut trouver les positions d'une sous-chaîne, puis l'extraire. Le terme "occurrence" a posé soucis dans leur compréhension.

4.1.5 Limites du terrain

Avant de pouvoir analyser toutes ces informations, il est important d'identifier les différentes limites rencontrées lors de cette expérience :

- Les groupes ayant participé différaient par leur âge. Il y avait deux groupes de rhétorique (17-19 ans) et un groupe de 4ème collège (système français 13-14 ans).
- Les groupes ayant participé venaient chacun d'une école différente.
- Seulement le dernier groupe aura été sujet à une évaluation après la séquence. Donnée par madame Grancher, cette évaluation vise à définir si les concepts ont été correctement assimilés.
- Aucun élève ayant testé la séquence PRIMM n'aura testé la séquence non-PRIMM, et vice-versa.
- Chaque groupe a testé la séquence dans un cadre différent, avec un intervalle de temps différent.
  1. Le premier groupe, celui qui a testé la séquence non-PRIMM, disposait d'une heure. La séquence était incluse dans un atelier qui n'était pas uniquement dédié à tester la séquence.
  2. Le deuxième groupe, celui qui a testé la séquence PRIMM en premier, disposait d'une heure et demie. Cet atelier s'est déroulé durant une journée de découverte à l'informatique, cette fois-ci l'atelier était uniquement dédié à tester la séquence.
  3. Le troisième groupe, celui qui a testé la séquence PRIMM en second, disposait de deux séances de cours d'une heure, dont seulement 45 minutes par séance sont réellement utilisées. Pour ce groupe, la séquence s'est déroulée dans le cadre d'une journée d'école classique.
- Il était important, pour les trois groupes, de permettre aux élèves de progresser durant la séance, même si un exercice était trop complexe, il fallait éviter qu'ils stagnent et ne puissent arriver au bout des exercices. J'ai fait le choix de mettre en place un horaire pour chaque classe. Basé sur une estimation du temps nécessaire pour compléter la séquence, ainsi que sur le niveau de progression en temps réel des élèves, j'ai fourni une explication de la solution pour chaque fin de sous-séquence à un temps donné. Le fait de transmettre la solution de chaque fin de sous-séquence biaise donc en partie les résultats obtenus pour les exercices finaux.

4.2 Analyse des résultats

Suite à l'analyse de la séquence non-PRIMM, discutée au chapitre 3.1, j'ai pu observer que cette séquence pourrait bénéficier d'un gain d'exercices *Predict/Run/Investigate/Modify*. Cette décomposition de la séquence en exercices intermédiaires faciliterait selon moi l'apprentissage de nouveaux concepts pour des débutants. J'espère que les modifications implémentées dans ma séquence PRIMM ont réussi cet objectif. Je vais essayer d'établir dans cette section quel impact les modifications ont eu en me basant sur les quatre points discutés ci-dessus. C'est-à-dire les performances observables sur INGInious, les réponses des élèves aux questionnaires, le ressenti de madame Grancher ainsi que mon ressenti personnel.

4.2.1 Analyse vis-à-vis de PRIMM

Les résultats INGInious de la séquence PRIMM suggèrent que les tâches intermédiaires *Predict/Run/Investigate/Modify* ont été rapidement et efficacement résolues. Les élèves ont l'air d'avoir compris, dans des situations simples, la majorité de la matière. Cette information suggère que ces exercices supplémentaires sont utiles pour introduire de nouveaux concepts. Cependant, les élèves des deux groupes ont eu plus de mal avec les tâches de fin de sous-séquence, dans lesquelles ils devaient réutiliser tous les concepts appris. Ce point peut suggérer que :

- Les élèves auraient peut-être eu besoin de plus d'exercices préliminaires avant les *Modify - Make* de fin de sous-séquence.
- Les exercices ne remplissent pas bien leur objectif avec les nouveaux concepts et les élèves ne retiennent pas, ne comprennent pas suffisamment ou ne font pas le lien avec une situation concrète dans laquelle les appliquer.
- Les exercices de fin de sous-séquence sont tout simplement trop complexes pour un cours d'introduction.
- Cette séquence dans son ensemble contient peut-être trop de concepts à assimiler en si peu de temps. Point non négligeable étant donné que tous ces concepts ne s'apprennent pas en une heure à l'UCLouvain.

Il y a eu peu de différences observables entre le groupe plus âgé et le groupe plus jeune ayant testé la séquence PRIMM. Ce qui peut suggérer que cette méthodologie est adaptée à plusieurs tranches d'âge.

Le témoignage de madame Grancher suggère que cette modification était très positive dans l'ensemble. Mis à part la structure des énoncés qui manquaient de clarté, l'élève est bien accompagné dans son apprentissage, le défi est présent, l'échelonnage des exercices est fluide, les concepts sont tous introduits. La décomposition des exercices, ainsi que l'ajout des différentes tâches *Predict/Run/Investigate/Modify*, est selon elle nécessaire pour aider les élèves à comprendre. Chose inévitable, un

petit nombre d'élèves parmi les plus rapides ont trouvé la séquence un peu longue.

Les réponses aux deux premiers questionnaires sont positives dans l'ensemble :
- Les élèves ont l'air d'avoir apprécié les exercices.
- Quelques élèves se sentent programmeurs après avoir participé.
- La complexité avait l'air bien calibrée.
- Les questions contenant un puzzle de Parsons semblent avoir aidé les élèves à comprendre l'exercice final.
- Les questions contenant du *tracing* semblent avoir été utiles pour maîtriser le concept.
- Les questions *Predict - Investigate* semblent avoir été utiles pour l'apprentissage des élèves.
- Les questions *Run* semblent avoir aidé les élèves à comprendre l'exercice final.
- Plusieurs élèves du premier groupe ont au moins retenu un concept, un élève semble avoir été particulièrement réceptif.

Il est important de garder en tête que les réponses comportent potentiellement un biais de sympathie, les élèves auront peut-être voulu être gentils dans leurs réponses.

Concernant l'évaluation rédigée par madame Grancher, on peut observer plusieurs choses à propos du second groupe :
- Variable :
  - La moitié des élèves savaient définir le concept de variable puis expliquer ce qu'une variable peut contenir. Ce qui suggère qu'ils ont compris la théorie.
  - L'autre moitié pense que les variables servent à définir une condition, plusieurs lignes de code, un algorithme ou un logiciel. Ce qui montre qu'ils n'ont pas compris.
- Chaînes de caractères :
  - La majorité des élèves semblent avoir compris ce qu'était un caractère, plusieurs élèves ont repris l'exemple de la virgule ou de l'espace dans une chaîne. Ce qui suggère que ce concept est acquis.
- Type de données :
  - Le concept de type de données ne semble pas acquis pour la majorité des élèves, seulement 3 élèves ont répondu correctement aux deux questions.
  - Le reste des élèves ont répondu que le type de données est : l'ADN, la recherche de coupable, l'extraction d'ADN, les données cellulaires.
- Occurrence :
  - La majorité des élèves semble avoir compris le concept d'occurrence, malgré que la plupart n'ont pas su restituer la définition, ils ont répondu

correctement aux deux exemples.

Les réponses à cette évaluation, qui a pris place une semaine après, suggèrent que les élèves ont pu assimiler et retenir plusieurs nouveaux concepts.

Mon ressenti est que dans l'ensemble :

- Rajouter des exercices *Predict/Run/Investigate/Modify* a permis aux élèves d'être plus à l'aise avec la réutilisation des blocs à travers les exercices, ce qui suggère qu'ils aient mieux assimilé les concepts que les élèves ayant suivi la séquence non-PRIMM.
- Les trois groupes ont montré des faiblesses sur la réutilisation des variables, préférant copier puis coller le contenu d'une variable au lieu d'utiliser la variable directement, ou bien collant des blocs variables un peu partout en espérant que ça fonctionne. Ce point suggère qu'il faudrait potentiellement plus d'exercices intermédiaires sur ce sujet, ou que le concept de variable est difficile à maîtriser en si peu de temps.
- Le fait d'introduire explicitement chaque nouveau concept semble avoir facilité leur compréhension, les élèves ne doivent jamais résoudre un exercice dans lequel un concept n'est pas expliqué au moins une fois.
- Le fait de varier les types d'exercices semble avoir stimulé les élèves.
- Le fait de monter graduellement en difficulté semble avoir aidé les élèves à progresser en douceur ainsi que maintenir leur intérêt.
- Les informations récupérées suggèrent que la séquence PRIMM est adaptée à plusieurs publics.
- Les trois groupes ont eu des difficultés à résoudre les exercices finaux qui regroupaient tous les concepts. Ce point peut suggérer que :
  - Les séquences contiennent trop de concepts à assimiler.
  - Il est difficile d'appliquer autant de nouveaux concepts, dans une nouvelle situation, juste après les avoir découverts.

Je ne vois que deux points négatifs que cette modification a amenés. Le premier point est que vu le nombre d'exercices largement plus grand, les élèves qui ont compris rapidement le concept ont trouvé ça un peu long. Ces élèves sont ici assez peu nombreux, madame Grancher semblait dire que c'était inévitable dans une classe, il est donc difficile de dire que PRIMM en est la cause. Le deuxième point, toujours sur le nombre d'exercices, est que cette séquence prend plus de temps à terminer, dans ce cas-ci la séquence PRIMM a pris une demi-heure de plus que la séquence non-PRIMM.

4.2.2 Commentaires

Plusieurs facteurs ont impacté les résultats obtenus, tels que les limites du terrain discutées plus haut, mais surtout l'absence de tests corrects dans certains exercices. Permettre à l'élève de valider un exercice avec une réponse incorrecte est

non seulement dommageable pour son apprentissage, mais limite aussi la qualité des différents résultats obtenus et en conséquence la qualité de l'analyse que je peux en tirer.
Cependant, en analysant l'ensemble des soumissions INGInious des élèves, particulièrement l'exercice 3.6 (où les élèves doivent extraire une sous-chaîne depuis une chaîne ADN), il est devenu clair, au vu des erreurs observables, que la majorité d'entre eux ne comprenaient pas comment appliquer les concepts requis dans cette situation. Je soupçonne que l'incompréhension du terme "occurrence" a une part de responsabilité. Comme discuté à plusieurs reprises, il est probable que le niveau de complexité des exercices de fin de sous-séquence est de toute façon trop élevé. Les soumissions précédant les exercices de sous-séquence, ainsi que le temps disponible, suggèrent que la majorité des élèves des deux groupes n'auraient pas pu réussir à trouver la solution correcte pour l'exercice final 4.6.
Il faut aussi prendre en compte que cette séquence d'exercice est riche sur la quantité de nouveaux concepts, mais surtout que les élèves doivent découvrir, assimiler, manipuler puis restituer tous ces nouveaux concepts en peu de temps.

4.2.3 Conclusion

En se basant uniquement sur les performances aux exercices finaux, il est difficile, pour toutes les raisons discutées plus haut, d'évaluer si PRIMM a réellement délivré ses promesses. Les élèves semblent avoir été réceptifs aux exercices *Predict/Run/Investigate/Modify*, ils ont eu l'air de pouvoir facilement manipuler et comprendre de nouveaux concepts. Il semble être difficile d'estimer si la séquence PRIMM a permis aux élèves d'être mieux préparés aux exercices finaux, car ces exercices sont compliqués, la séquence est chargée en concepts et aucun des trois groupes n'a vraiment trouvé ces exercices faciles. Il s'avère aussi compliqué de pouvoir évaluer si les élèves des différents groupes ont correctement appris et retenu les concepts sur le long terme. Néanmoins, l'évaluation de madame Grancher suggère que plusieurs concepts, tels que les variables et les chaînes de caractères, ont été correctement assimilés.
Cependant, en prenant en compte l'entièreté des données récupérées, les indicateurs qui suggèrent que PRIMM a eu un impact positif sont plus nombreux que les points négatifs. Toutes ces informations tendent vers la conclusion qualitative que modifier la séquence non-PRIMM en y rajoutant des exercices *Predict/Run/Investigate/Modify* a permis de faciliter l'introduction de nouveaux concepts aux élèves.

Chapitre 5

Conclusion

5.1 Objectifs atteints

Les objectifs de ce mémoire sont selon moi atteints. J'ai pu créer une séquence d'exercices suivant la méthodologie PRIMM, en programmation par blocs avec Blockly. J'ai eu la chance de pouvoir travailler en collaboration avec une professeur extérieure. La séquence a été testée en situation réelle à deux reprises. La séquence a permis d'introduire de nouveaux concepts en programmation à des jeunes élèves. J'ai pu récupérer plusieurs pôles de données me permettant de m'exprimer de manière qualitative sur les promesses de PRIMM.

5.2 Objectifs futurs

Il serait utile, dans cette séquence d'exercices, mais aussi de manière générale, de bien séparer l'énoncé, pour permettre aux élèves de rapidement distinguer l'histoire, la théorie et les objectifs.
Il serait intéressant de pouvoir évaluer de manière quantitative l'apport de PRIMM dans une séquence Blockly. Pour y arriver, à l'instar de l'étude de D.Weintrop [26], il faudrait deux groupes similaires, dont un apprendrait avec une séquence PRIMM et le second sans. Il faudrait tester leurs connaissances avant et après avoir suivi le cours. Au moyen de différents tests, on pourrait sonder les performances de leur apprentissage, puis comparer les résultats pour établir avec plus de certitudes l'apport bénéfique de PRIMM.
Pour permettre aux élèves d'être mieux préparés aux exercices finaux, il me semble qu'il serait intéressant de rajouter plus d'exercices intermédiaires *Predict/Run/Investigate/Modify*. Une approche serait de créer plus d'exercices mettant en lien tous les concepts, d'abord deux par deux, puis augmenter doucement la quantité de concept par tâches pour les combiner de manière crescendo, menant

graduellement à une combinaison de tous les concepts.

Une autre approche bénéfique à l'apprentissage serait d'aborder moins de concepts par séance de cours, pour éviter une surcharge cognitive chez l'élève. Ne pas laisser suffisamment de temps entre chaque concept empêche potentiellement les élèves de bien mémoriser chaque concept, qui se bousculent dans leurs têtes, diminuant la qualité de l'enseignement.
Afin de faciliter la prise en main par un professeur de la séquence PRIMM créée ici, il serait utile de rédiger un petit manuel d'utilisation. Ce manuel pourrait contenir un guide décrivant étape par étape l'inscription et la navigation d'un cours INGInious, mais aussi les figures 3.6, 3.7 et 3.8 pages 35-37, servant de récapitulatif des exercices avec leurs buts pédagogiques ainsi que l'approche PRIMM.

5.3 Conclusion

Dans un monde de plus en plus tourné vers l'informatique, la demande en talents compétents ne fait qu'augmenter. En effet, cette ère numérique dans laquelle nous évoluons aujourd'hui augmente de manière exponentielle le nombre de secteurs qui sont, ou qui seront bientôt, en demande d'employés compétents en programmation. Outre son impact sur les autres capacités des élèves (le développement de la logique, la résolution de problèmes, l'analyse d'informations, etc.), l'enseignement de la programmation prend de plus en plus de sens pour l'avenir professionnel des apprenants. C'est pourquoi il est important de continuer à chercher à l'améliorer et le rendre accessible au plus grand nombre.
Arriver à introduire des concepts informatiques à des novices n'est pas une mince affaire. Des méthodologies prometteuses comme PRIMM cherchent à répondre à ce défi. C'est avec intérêt que je vais continuer à suivre les nouveaux outils facilitant l'apprentissage qui verront le jour dans les années à venir.
De manière plus personnelle, j'ai beaucoup apprécié partager cette passion avec de jeunes élèves en présentant la programmation de manière ludique et accessible.

Mon opinion, basée sur les critères d'évaluations ainsi que sur les résultats obtenus discutés au chapitre précédent, est que dans l'ensemble, PRIMM apporte une réelle plus-value dans l'apprentissage de nouveaux concepts. C'est pourquoi je recommande l'utilisation de PRIMM dans des cours introduisant de nouveaux concepts en programmation.

Annexe A

Récapitulatif des concepts abordés, créé par madame Carine Grancher

Que retenir de ce TP ?

Langages informatiques

Blockly est un langage à blocs qui se programme comme Scratch.
Python est un des nombreux langages de programmation à texte qui existent, comme le C ou le JavaScript dont vous avez peut-être déjà entendu parler.

La notion de variables

Dans un programme informatique, une **variable** est une zone de la **mémoire** qui permet de conserver, stocker de l'information.

Cette information peut être un nombre, la valeur vrai ou faux (un **booléen**), une lettre (un **caractère**), un mot, une phrase ou un texte (une **chaîne de caractères**), ou des structures plus compliquées comme nous le verrons plus tard.

Les **variables** peuvent être expliquées comme un petit tableau noir sur lequel vous pouvez écrire à la craie une **valeur**, comme le nombre 12 ou le texte “Bonjour Paul”.
Chacune de ces ardoises est identifiée par son nom. Le programme peut alors déposer une valeur, utiliser l'information déjà écrite sur l’ardoise ou la modifier. Il lui suffit pour cela de faire référence à l’ardoise par son nom.

☝ Il ne peut y avoir qu'une et une seule valeur à la fois dans une variable !
Quand vous mettez une nouvelle valeur dans votre variable, ce qu'il y avait avant est effacé et l'ordinateur écrit dans votre variable la nouvelle valeur que vous venez d'y mettre.

Types de variables

Un concept important de l’informatique est **le type de donnée** que vous utilisez.

Si vous mettez du texte dans une variable, sa valeur est de **type texte**.
Par exemple : 

Si vous mettez des nombres dans une variable, sa valeur est de **type nombre**.
Par exemple : 

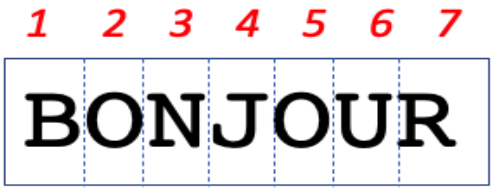
En informatique, on utilise le terme "incrémenter" pour augmenter de un la valeur d'un nombre. Le jour de l’anniversaire de Paul la variable sera incrémentée de un

☝ Attention qu'une valeur ne peut avoir qu'un seul type de donnée à la fois.

Chaîne de caractères

En informatique, on appelle le texte une **chaîne de caractères**.
Exemple: "abc" ou “Bonjour Paul” sont des chaînes de caractères.

Une chaîne de caractères est une **séquence de caractères**. Chaque caractère est placé à une position dans la séquence. Cette position est identifiée par un nombre. Dans la séquence BONJOUR, le caractère B se trouve en position 1, J en position 4, …



Tout ce qui est contenu dans le bloc vert est considéré comme une **chaîne de caractères**.
Tout ce qui est contenu dans la chaîne de caractères: "Bonjour Tom, comment allez-vous ?" est considéré comme un **caractère**. Le "B", le "o", la virgule "," et même l'espace entre "Bonjour" et "Tom" sont considérés comme des caractères.

La longueur d'une chaîne de caractères.

Les langages de programmation permettent de facilement manipuler ces chaînes de caractères et d'en extraire de l'information. On dit que grâce à l'informatique, on peut **automatiser** le **traitement** des chaînes de caractère, comme d’autres tâches d’ailleurs.

Ainsi, si on veut savoir la taille d'un texte, on ne doit pas compter caractère par caractère. Des instructions permettent de calculer automatiquement le nombre de caractères que contient une chaîne. Cela s’appelle la **longueur de la chaîne** de caractères.

Quelle est la longueur de la chaîne de caractères contenue dans la chaîne “texte1” ?
6 bien sûr ! Et “Bonjour Paul” ? 12 !

Les occurrences

Occurrence ça veut dire "le nombre de fois que quelque chose est présent dans une chaine de caractère".
Cela peut être un caractère ou même un mot donc un sous-chaîne de caractères.

Exemples :
il y a deux occurrences de la lettre "o" dans "Bonjour", une occurrence de "B", une occurrence de "j"…
“Bonjour Monsieur, comment vous appelez vous ? Je m’appelle Paul. Bonjour Paul” il y a deux occurrences du mot Paul.

Afficher une chaine de caractère

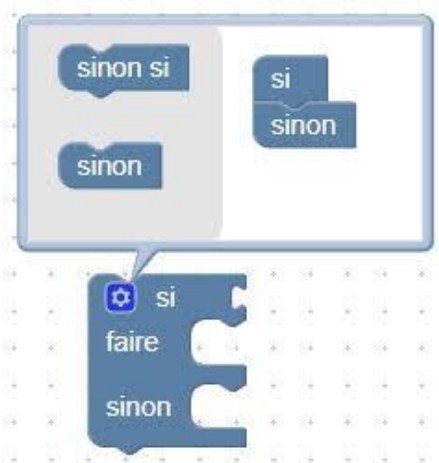
Les programmes informatiques utilisent principalement l'écran pour rendre compte du résultat du traitement. Le résultat d'un traitement est la **sortie** du programme.
Pour s'assurer de la bonne marche de l'algorithme, un programmeur a besoin d'afficher les résultats. Le mot clé associé en anglais est **PRINT**.

Les conditions branchements

Un programme informatique doit parfois prendre des **décisions** en fonction de l'environnement du traitement à faire.
Si le nombre est positif alors la valeur absolue est lui même, sinon il faut ôter le signe "-".

Pour cela, il doit pouvoir évaluer des **conditions**.
Dans les langages de programmation, on retrouve toujours des **instructions conditionnelles.** Elles permettent d'exécuter des instructions lorsqu'une condition est **vraie** ou d'autres si elle est **fausse**.
Cela permet aux algorithmes de fabriquer un branchement, orienter une action, prendre une décision en fonction du résultat de l'évaluation de la condition.

Dans le bloc si.., si la **condition évaluée** est vraie (le programme retourne la valeur **True**), le bloc va exécuter tous les blocs à l'intérieur du faire.., si la condition évaluée est fausse le programme retourne la valeur **False**) les blocs dans faire.. seront ignorés. On exécute uniquement les blocs après le bloc if.



Il est possible de rajouter des sinon.. comme dans le bloc ci-dessous. Les blocs contenus dans le sinon.. seront exécutés si la condition retourne la valeur False.

Savoir faire !

- Savoir afficher un texte.
- Savoir mettre en majuscule ou un minuscule un texte (changer la casse).
- Savoir utiliser un bloc qui compte le nombre de caractères.
- Savoir utiliser un bloc qui retrouve une sous-chaîne dans une plus grande chaîne.
- Savoir afficher le contenu d'une variable.
- Savoir faire la différence entre le nombre 123 et le texte "123"

Bibliographie

[1] T. A. Kochan and L. Dyer, *Shaping the future of work: a handbook for building a new social contract*. Routledge, 2020.

[2] N. Joris and J. Henry, "L'enseignement de l'informatique en belgique francophone: état des lieux," *1024: Bulletin de la Société Informatique de France*, no. 2, pp. 107–116, 2014.

[3] J. Henry and N. Joris, "Informatics at secondary schools in the french-speaking region of belgium: myth or reality," *ISSEP 2016*, vol. 13, 2016.

[4] J. Goode, G. Chapman, and J. Margolis, "Beyond curriculum: The exploring computer science program," *ACM Inroads*, vol. 3, no. 2, pp. 47–53, 2012.

[5] O. Astrachan and A. Briggs, "The cs principles project," *ACM Inroads*, vol. 3, no. 2, pp. 38–42, 2012.

[6] I. Lee, F. Martin, J. Denner, B. Coulter, W. Allan, J. Erickson, J. Malyn-Smith, and L. Werner, "Computational thinking for youth in practice," *Acm Inroads*, vol. 2, no. 1, pp. 32–37, 2011.

[7] D. Parsons and P. Haden, "Parson's programming puzzles: a fun and effective learning tool for first programming courses," in *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*, pp. 157–163, 2006.

[8] C. Schulte, "Block model: an educational model of program comprehension as a tool for a scholarly approach to teaching," in *Proceedings of the fourth international workshop on computing education research*, pp. 149–160, 2008.

[9] S. Sentance, J. Waite, and M. Kallia, "Teaching computer programming with primm: a sociocultural perspective," *Computer Science Education*, vol. 29, no. 2-3, pp. 136–176, 2019.

[10] G. Derval, A. Gego, P. Reinbold, B. Frantzen, and P. Van Roy, "Automatic grading of programming exercises in a MOOC using the INGInious platform," *European Stakeholder Summit on experiences and best practices in and around MOOCs (EMOOCS'15)*, pp. 86–91, 2015.

[11] S. Sentance, "What is PRIMM." `https://primmportal.com/`, 2018.

[12] S. Sentance, "The i in PRIMM." https://helloworld.raspberrypi.org/articles/hw14-the-i-in-primm, 2021.

[13] S. Sentance and J. Waite, "PRIMM: Exploring pedagogical approaches for teaching text-based programming in school," in *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, pp. 113–114, 2017.

[14] RaspberriPiAssociation, "Quick read: Using PRIMM to structure programming lessons." https://blog.teachcomputing.org/using-primm-to-structure-programming-lessons/, 2021.

[15] S. Sentance, "The i in PRIMM." https://helloworld.raspberrypi.org/articles/hw14-the-i-in-primm, 2021.

[16] S. Sentance, "Exploring pedagogies for teaching programming in school." https://web.archive.org/web/20210227235433/https://blogs.kcl.ac.uk/cser/2017/02/20/exploring-pedagogies-for-teaching-programming-in-school/, 2017.

[17] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, *et al.*, "A multi-national study of reading and tracing skills in novice programmers," *ACM SIGCSE Bulletin*, vol. 36, no. 4, pp. 119–150, 2004.

[18] A. Venables, G. Tan, and R. Lister, "A closer look at tracing, explaining and code writing skills in the novice programmer," in *Proceedings of the fifth international workshop on Computing education research workshop*, pp. 117–128, 2009.

[19] R. Lister, C. Fidge, and D. Teague, "Further evidence of a relationship between explaining, tracing and writing skills in introductory programming," *Acm sigcse bulletin*, vol. 41, no. 3, pp. 161–165, 2009.

[20] S. Sentance and A. Csizmadia, "Computing in the curriculum: Challenges and strategies from a teacher's perspective," *Education and Information Technologies*, vol. 22, no. 2, pp. 469–495, 2017.

[21] A. Walqui, "Scaffolding instruction for english language learners: A conceptual framework," *International journal of bilingual education and bilingualism*, vol. 9, no. 2, pp. 159–180, 2006.

[22] P. Bagge, "Using the PRIMM approach at primary level." https://helloworld.raspberrypi.org/articles/hw15-using-the-primm-approach-at-primary-level, 2018.

[23] S. Sentance, "PRIMM exercices 1." https://primmportal.com/2018/08/21/set-of-primm-worksheets/primmportal.com/2018/08/21, 2018.

[24] S. Sentance, "PRIMM exercices 2." https://primmportal.com/2018/08/23/primm-materials-2018/primmportal.com/2018/08/23, 2018.

[25] Docker, "Site officiel de docker." `https://www.docker.com/`, 2022.

[26] D. Weintrop and U. Wilensky, "Comparing block-based and text-based programming in high school computer science classrooms," *ACM Transactions on Computing Education (TOCE)*, vol. 18, no. 1, pp. 1–25, 2017.

[27] D. Weintrop and U. Wilensky, "Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs," in *Proceedings of the eleventh annual international conference on international computing education research*, pp. 101–110, 2015.

[28] T. W. Price and T. Barnes, "Comparing textual and block interfaces in a novice programming environment," in *Proceedings of the eleventh annual international conference on international computing education research*, pp. 91–99, 2015.

[29] O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari, "Learning computer science concepts with scratch," *Computer Science Education*, vol. 23, no. 3, pp. 239–264, 2013.

[30] K. Johnsgard and J. McDonald, "Using alice in overview courses to improve success rates in programming i," in *2008 21st Conference on Software Engineering Education and Training*, pp. 129–136, IEEE, 2008.

[31] B. Moskal, D. Lurie, and S. Cooper, "Evaluating the effectiveness of a new instructional approach," in *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pp. 75–79, 2004.

[32] P. Mullins, D. Whitfield, and M. Conlon, "Using alice 2.0 as a first language," *Journal of Computing Sciences in Colleges*, vol. 24, no. 3, pp. 136–143, 2009.

[33] O. Bonaventure, "Initiation à la bioinformatique." `https://inginious.org/course/blockly_bioinfo`, 2022.

[34] C. Debongnie, "Séquence exercice mémoire debongnie cyrille." `https://inginious.org/course/Memoire_Debognie_blockly`, 2022.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**