**MASTER EN SCIENCES INFORMATIQUES**

## Cartorithmique

## Un Concept Inventory Augmenté Pour Identifier Les Mauvaises Representations Des Novices

Magis, Tom

*Award date:*
2020

*Awarding institution:*
Universite de Namur

UNIVERSITÉ DE NAMUR

FACULTÉ D'INFORMATIQUE

# Cartorithmique - Un Concept Inventory Augmenté Pour Identifier Les Mauvaises Représentations Des Novices

Tom MAGIS

RUE GRANDGAGNAGE, 21 ● B-5000 NAMUR(BELGIUM)

# Remerciements

Je tiens à remercier toutes les personnes qui ont aidé à la réalisation de ce projet de développement d'un dispositif pédagogique d'aide à l'apprentissage de la programmation et à la concrétisation de ce mémoire.

Je voudrais tout d'abord remercier mes promoteurs, Bruno Dumas, Julie Henry et Benoit Vanderose qui m'ont encadré, guidé et aidé tout au long de cette aventure. Je voudrais tout particulièrement remercier Julie Henry pour le temps qu'elle a consacré autant lors des séances de tests que lors de la relecture du mémoire.

Merci à Renaud Gaspard pour son soutien technique, à Ines Sanchez Cienfuegos pour son soutien général et à Max Magis pour son soutien mathématique. Je voudrais également remercier ma famille et mes amis qui m'ont soutenu tout au long de cette période.

# Résumé

Les cours d'introduction à la programmation sont un défi pour les enseignants et sont considérés comme difficiles par les étudiants novices. Pour détecter rapidement ce qui pose problème et pouvoir y remédier, un « concept inventory » tangible a été développé, suivant une approche orientée-conception faisant notamment participer les futurs utilisateurs et menant à des collaborations entre experts, enseignants et apprenants. Deux cycles de développement ont pu être menés durant ce mémoire et ont permis d'aboutir à un prototype mature, bien qu'encore améliorable. L'outil ainsi développé vise à aider les enseignants et les apprenants à prendre conscience des représentations erronées qui sont construites autour des concepts de base en programmation. Il vise également à aider l'apprenant à corriger ses propres représentations erronées de façon autonome. Le prototype a été validé auprès de huit enseignants et de neuf étudiants. Les résultats obtenus sont prometteurs et laissent apparaître une certaine efficacité de l'outil dans la détection et la correction des représentations erronées.

***Mots-clés***— Concept Inventory, Interface tangible, Design-oriented research, Mauvaises représentations

# Table des matières

# Chapitre 1

# Introduction

La difficulté d'apprentissage des concepts de programmation n'est pas un fait nouveau [87]. Ces concepts théoriques, parfois très abstraits, représentent à la fois une difficulté pour l'enseignant, dont la préoccupation principale est de transmettre un discours clair et cohérent, et pour l'étudiant qui doit les comprendre et les assimiler à ses connaissances préalablement acquises. Malgré les efforts fournis par les enseignants, les représentations que construit l'étudiant au cours de son apprentissage, appelées aussi « modèles mentaux », sont parfois erronées. Par exemple, il est courant chez les novices en programmation d'inverser le sens d'une assignation ou de l'interpréter comme une égalité mathématique. Les représentations erronées sont la conséquence de plusieurs éléments. Ainsi, la syntaxe de langages de programmation est parfois similaire à celle d'autres concepts déjà vus au préalable : dans le cas de l'assignation, le « = » similaire à l'égalité mathématique. Qui plus est, plusieurs représentations erronées pour un même concept peuvent coexister chez un étudiant. Si elles ne sont pas décelées et corrigées suffisamment tôt, le risque est alors que l'étudiant bloque dans son apprentissage, soit découragé par ses incompréhensions et qu'il se retrouve en situation décrochage scolaire. De nombreuses recherches traitent de cette thématique. Certaines ont identifié les différentes représentations erronées associées à chaque concept de base en programmation. Pour cela, des outils ont été développés pour les détecter chez les apprenants. Ces outils prennent parfois la forme de solutions logicielles spécifiques qui demandent par exemple de simuler le comportement de l'ordinateur étape par étape pour un programme donné. Plus couramment, il s'agit de questionnaires à choix multiples qui permettent d'évaluer les connaissances d'un étudiant et d'identifier, à travers ses (mauvaises) réponses, les représentations erronées qu'il possède. On parle de *concept inventory (CI)* [122].

Il n'existe à priori aucune solution matérielle, dite « tangible », pour détecter les

représentations erronées. D'ailleurs, de manière plus générale, les outils tangibles d'aide à l'apprentissage de la programmation ciblent principalement les jeunes enfants [30]. Pourtant, les bienfaits des interfaces tangibles ne sont pourtant plus à démontrer [98]. En plus d'offrir un aspect plus ludique, il apparaît que la gestuelle améliore l'apprentissage [21]. De plus, la manipulation d'éléments prédéterminés permet de faire abstraction de certaines complexités propres à l'ordinateur (pour autant que l'interface tangible soit conçue de manière intuitive et ergonomique). Ces interfaces permettent à l'étudiant de se concentrer sur l'élaboration d'une solution (par exemple, un programme), sans se heurter à un échec récurrent causé par des éléments externes (problèmes de syntaxe, de configuration, etc.). Elles pourraient même faciliter l'expérimentation de différentes implémentations pour un programme demandé, via des retours complets sur l'exécution de la solution formée et faciliteraient alors le débogage.

Ce mémoire a pour objectif de pallier l'absence d'outil tangible d'aide à la détection des représentations erronées. Il vise principalement un public de jeunes adultes, des étudiants en première année de bachelier suivant un cours d'introduction à la programmation. Compte tenu de ce contexte particulier, quelles seraient les fonctionnalités- clé d'un tel outil ? Quelle interface, adaptée au public cible, permettrait de manipuler les concepts de base en programmation sans se heurter aux problèmes de syntaxe ? Comment rendre un tel outil efficace à détecter les représentations erronées des étudiants et comment illustrer de façon tangible ces représentations erronées ? Autant de questions auxquelles ce travail propose d'apporter des réponses.

Dans un premier temps, un état de l'art (Chapitre 2) a été réalisé sur l'enseignement/apprentissage de la programmation et sur les représentations erronées liées à cet apprentissage. Une classification des outils tangibles d'aide à la programmation a également été effectuée.

Le contexte de la recherche dans lequel a eu lieu ce mémoire lui-même basé sur un projet de thèse et est décrit dans le chapitre 3.

La problématique de la recherche, décomposée en plusieurs hypothèses de recherche est décrite dans le chapitre 4.

Les méthodologies de développement de l'outil tangible et de sa validation sont décrites dans le chapitre 5.

Le chapitre 6 décrit les contributions majeures de ce mémoire, à savoir la définition des besoins et les différents composants de l'outil tangible développé.

Deux phases de validation de l'outil ont été réalisées. Les données collectées durant ces évaluations sont présentées et analysées dans le chapitre 7, ainsi que les adaptations qui en ont découlé.

Le chapitre 8 discute chaque hypothèse de recherche, en vue d'apporter des éléments de réponse à la question de recherche au centre de ce mémoire. Des pistes

sont lancées pour une possible suite à ce travail.

Enfin, un bref résumé de la recherche est proposé dans le dernier chapitre, des conclusions sont tirées quant à la possibilité de détecter les représentations erronées présentes chez des programmeurs novices au moyen d'un outil tangible.

# Chapitre 2

# État de l'art

Le monde actuel manque d'informaticiens. Face à cette pénurie, les études visant notamment à former des programmeurs sont en forte demande. Pourtant, l'apprentissage de la programmation n'est pas facile [23, 87], et ce quel que soit l'âge de l'apprenant. De nombreuses recherches ont vu le jour pour identifier et tenter d'expliquer les difficultés éprouvées par les novices, espérant ainsi y apporter des solutions.

Les difficultés, et *a contrario* les facteurs facilitants, identifiés par ces recherches portent à la fois sur les caractéristiques d'un premier langage de programmation, sur les représentations erronées que possèdent les apprenants vis-à-vis des concepts de base en programmation et sur les particularités de certains outils d'aide à l'apprentissage.

## 2.1 Un premier langage de programmation

Le choix d'un langage de programmation est une étape importante dans la conception d'un cours d'initiation à la programmation. Prokop, Trofimenko, Severin et Bukata [83] ont établi des critères de sélection aidant au choix de ce premier langage (Fig. 2.1). Ainsi, un premier langage doit être simple à apprendre, doit répondre à la demande sur le marché, doit posséder tous les concepts majeurs en programmation et doit faciliter le passage à un autre langage.

En ce qui concerne la simplicité d'apprentissage, celle-ci est mesurée par la possibilité d'assimiler à travers le langage et par la pratique de ce dernier, sans trop de difficultés, les concepts de base. Tous les langages n'offrent pas une même simplicité. En effet, un langage à la syntaxe plus complexe risque de provoquer plus d'erreurs de la part des apprenants, que ce soit des erreurs de syntaxe ou de compilation [61, 66]. Ces erreurs impactent notamment la motivation extrinsèque

de l'étudiant. Cependant, Johnson, McQuistin et O'Donnell [40] ont montré que des langages possédant une syntaxe simple introduisent malgré eux une complexité sémantique. Ainsi, si l'apprenant peut écrire un programme fonctionnel plus rapidement, il tire des conclusions trop hâtives sur ce qui se passe réellement au niveau de l'ordinateur. Ce comportement impacte alors sa bonne compréhension du fonctionnement de la machine. Par exemple, le langage Python possède une syntaxe peu complexe, permettant aux apprenants de réaliser plus rapidement des projets plus importants (et donc, plus motivants du point de vue de l'apprenant [5]). Le risque est cependant grand de créer des confusions dans l'esprit des apprenants, notamment concernant l'impact qu'a l'opérateur d'addition sur des entiers, des listes, des chaînes de caractères, etc. Cette complexité sémantique, acceptable pour les concepts de base, peut s'avérer déstabilisante lorsque les concepts plus avancés du langage sont abordés.

L'apprentissage d'un langage doit être valorisé sur le marché de l'emploi [86]. Tout apprenant doit être « fonctionnel », prêt à travailler le plus rapidement possible. Apprendre un langage fortement demandé par les employeurs aura plus de succès et de sens pour les apprenants [75] que d'apprendre des langages spécifiquement développés pour l'éducation, telle que le langage de programmation en blocs Scratch [1]. Bien que ce dernier permette d'aborder facilement les concepts de base [73] tels que la variable ou les structures conditionnelles, son apprentissage exclusif n'a aucune valeur sur le marché [106].

Le langage choisi doit répondre aux exigences du cours dans lequel il va s'intégrer. Parce qu'il s'agit de cours d'introductions à la programmation, dans le cas d'un premier contact avec ce domaine, les concepts que l'on doit retrouver dans le langage sont les concepts de base : la variable, la structure conditionnelle, la boucle et la fonction/procédure. Si ces concepts sont présents dans certains langages de programmation en blocs destinés aux plus jeunes, tels que Scratch, Blockly [2] ou le langage micro :bit [3], ça n'est pas une généralité. Certains langages, dépendant du public cible, omettent certains concepts pourtant essentiels. Ce constat sera rediscuté ultérieurement, dans le cadre des outils d'apprentissage.

L'apprentissage d'un langage doit donner les clés pour faciliter l'apprentissage d'autres langages, bien souvent de manière autonome. Ce transfert de connaissances d'un langage à l'autre peut être facilité par des syntaxes similaires (Java, C++, Kotlin), ou encore des principes et concepts communs (langages orientés-objet VS langages procéduraux, le principe de « garbage collection » VS la gestion manuelle de la mémoire, etc.) [28, 83]. Un langage qui partage des similitudes avec plusieurs autres facilite une autonomie d'apprentissage, rendant possible une future

1. https://scratch.mit.edu/
2. https://blockly.games/
3. https://microbit.org/

reconversion professionnelle ou une meilleure adaptation au marché de l'emploi. Le travail de l'apprenant se centre alors sur la compréhension des différences entre les deux langages et sur les moyens de compenser ces différences. Par exemple, les spécificités du langage Python ne facilitent pas le passage à un autre langage. Ainsi, certains concepts ne sont pas ou peu abordés, comme le typage à la déclaration, la compilation, le passage de paramètres, les pointeurs, etc. De même, les langages de programmation en blocs comme Scratch ne préparent pas suffisamment à l'apprentissage autonome d'une syntaxe plus complexe.

L'équilibre entre facilité d'apprentissage, intérêt, complétude et transférabilité d'un langage semble difficile à trouver. Dans la plupart des textes consultés dans le cadre de ce travail de recherche, le langage Python fait office de premier langage de programmation pour les apprenants [42, 85, 70, 101, 124, 59, 60, 52]. Ce langage semble proposer un bon compromis entre facilité syntaxique, couverture des concepts de base, facilité à mettre en oeuvre et utilisation sur le marché professionnel. Cependant, si le langage peut apporter certaines facilités lors de son apprentissage [52], celles-ci sont loin d'être suffisantes pour écarter les difficultés que pose l'enseignement de la programmation.

## 2.2 Les représentations erronées

### 2.2.1 Les modèles mentaux et conceptuels

Selon Johnson-Laird [41], les modèles mentaux sont des modèles incomplets, dynamiques et fonctionnels que les humains utilisent pour comprendre le monde. Dans un contexte éducatif, ce sont des modèles que les apprenants créent à partir d'un phénomène physique, dont le but est d'être utile pour leur permettre de comprendre des phénomènes [25]. Ils sont construits à partir de la perception du monde par les élèves, ou de la compréhension du discours de l'enseignant.

Ben-Ari révèle que « a (beginning) computer science student has no effective (mental) model of a computer » [4]. « Since computer science deals with artifacts—programming languages and software, the creator of the artifact employed a very detailed model and the learner must construct a similar, though not necessarily identical, model » [4]. Norman [69] suggère qu'un utilisateur construit les modèles mentaux des systèmes informatiques (*système cible*) en interagissant avec eux et affine constamment les modèles tout au long de ces interactions. Selon Ben-Ari [4], les modèles d'artefacts informatiques doivent être explicitement enseignés. Les apprenants suivent souvent un cours de programmation sans disposer des modèles mentaux efficaces dont ils ont besoin pour développer une compréhension appropriée du matériel d'apprentissage. Les apprenants ne sont pas capables de construire des modèles mentaux en se

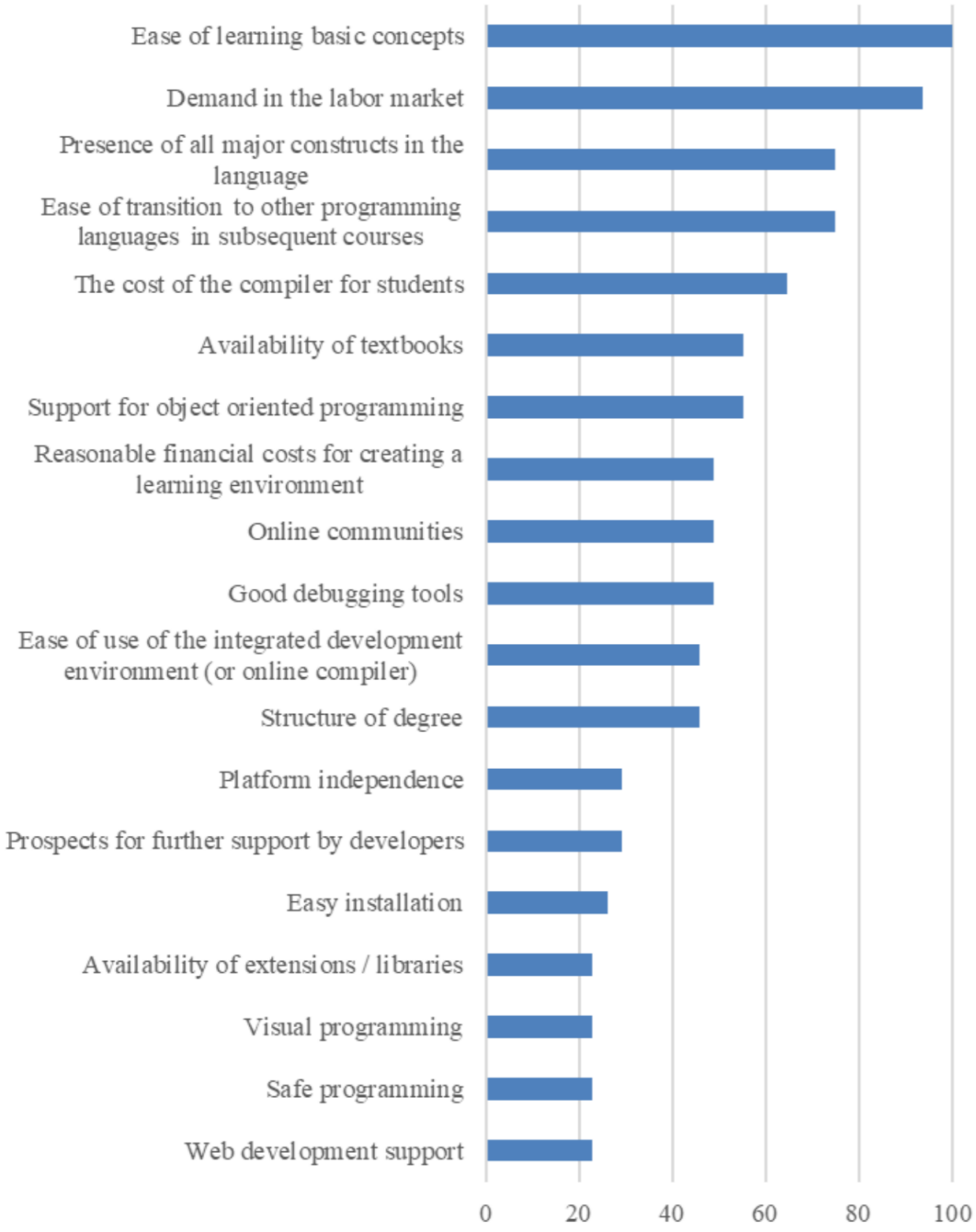



FIGURE 2.1 – Évaluation des critères pour choisir le premier langage de programmation issu de l'article de Prokop, Trofimenko, Severin et Bukata [83]. L'axe X représente la moyenne entre l'occurrence de chaque critère dans la littérature et l'évaluation qualitative de ces critères par les enseignants en fonction de leur expérience.

contentant d'écouter des cours ou de lire des manuels. S'ils le font, ils risquent de construire des modèles mentaux inappropriés basés sur une mauvaise utilisation de leurs connaissances préalables et de leurs modèles intuitifs. Par ailleurs, Henry [42] montre que plus les apprenants sont en difficulté, plus ils possèdent de modèles mentaux différents qui coexistent.

Gentner [20] a expliqué que la compréhension des modèles mentaux intuitifs des apprenants est cruciale pour la préparation de matériel d'apprentissage approprié :« If typical incorrect models are understood, then instructors and designers can create materials that minimize the chances of triggering errors ». Les enseignants devraient explicitement aider les apprenants à construire des modèles mentaux viables à un stade précoce avant qu'ils ne construisent des modèles mentaux incorrects. Pour ce faire, les enseignants tentent généralement de transmettre leurs propres modèles mentaux. Ils utilisent alors des modèles conceptuels, à savoir la représentation de leurs modèles mentaux. C'est à partir de ces modèles conceptuels que les apprenants vont construire leurs propres modèles mentaux. Pour constituer une base de travail efficace, les modèles conceptuels doivent être « assimilables », « suffisamment complets » et « utilisables ».

### 2.2.2 La machine notionnelle

Sorva [107] décrit la machine notionnelle comme « A notional machine is a characterization of the computer in its role as executor of programs in a particular language or a set of related languages. A notional machine encompasses capabilities and behaviors of hardware and software that are abstract but sufficiently detailed, for a certain context, to explain how programs get executed and what the relationship of programming language commands is to such executions. ». En d'autres termes, il s'agit d'une représentation simplifiée du fonctionnement de l'ordinateur permettant d'expliquer le fonctionnement d'un programme. Elle est généralement dépendante du paradigme de programmation au centre de l'apprentissage. C'est principalement dans la construction du modèle mental de cette machine notionnelle que se créent les représentations erronées des apprenants. Cela traduit une mauvaise compréhension du fonctionnement de la machine exécutant les instructions qui lui ont été soumises, ou une mauvaise compréhension des instructions elles-mêmes.

Le modèle mental de la machine notionnelle est construit étape par étape, concept par concept, sachant que certains concepts sont plus difficiles à appréhender que d'autres (par exemple, les pointeurs ou la récursion) [114, 94, 22]. Outre la construction d'un modèle mental valide, il est important que les apprenants identifient les limites de la portée de ce modèle mental, qu'ils fassent la distinction entre ce qui ressort du fonctionnement de la machine et ce qui est de leur propre

responsabilité. Sans la création du modèle mental de la machine notionnelle, le cours de programmation se contenterait d'être un apprentissage « par coeur » de concepts théoriques, rendant presque impossible la mise en pratique de ces concepts. De plus, l'apprentissage de concepts plus avancés serait rendu difficile par la mauvaise compréhension des concepts de base.

C'est pour faciliter le développement du modèle mental de la machine notionnelle chez les apprenants qu'il est conseillé de représenter, en parallèle à la lecture d'instructions dans un langage, la gestion de ces instructions par la mémoire [24]. Cette représentation constitue alors un modèle conceptuel utilisable par les apprenants. Elle leur permet notamment de visualiser l'état de la mémoire, comment sont représentés les variables et les objets, ainsi que l'évolution de leurs valeurs en fonction des instructions exécutées. Cela permet typiquement de montrer pourquoi un objet n'est pas « copié » lors d'une assignation et facilite l'introduction, par exemple, des notions de pointeur, de passage par référence et de passage par valeur.

### 2.2.3 Les représentations existantes et erronées

Il n'est pas facile pour les apprenants d'abandonner leurs représentations existantes et d'en adopter de nouvelles [14]. Souvent, les apprenants ne peuvent pas se rendre compte que leurs représentations existantes sont en conflit avec les représentations enseignées (modèles conceptuels) [34]. Il est donc important que l'enseignement soit en mesure d'aider les apprenants à prendre conscience de ce problème et de les aider ensuite à construire des représentations cohérentes.

Les représentations existantes sont parfois des représentations erronées. Dans le contexte de l'enseignement de la programmation, Sorva [101] a défini les représentations erronées comme « understandings that are deficient or inadequate for many practical programming contexts ». Cette définition comprend, entre autres, une mauvaise compréhension des concepts de programmation. Quian [84] a défini les représentations erronées comme des erreurs de compréhension conceptuelle, des malentendus sur les constructions de programmation tels que les variables et énoncés d'affectation, les expressions conditionnelles ou les boucles. Ces idées fausses doivent être identifiées pour permettre aux enseignants d'intervenir de manière appropriée à l'avenir [116]. En plus d'évaluer la compréhension des concepts fondamentaux de la programmation par les apprenants, il est recommandé d'explorer l'évolution des représentations erronées par rapport aux stratégies et aux outils utilisés dans les cours, mais aussi par rapport aux différences individuelles [84]. Les facteurs généralement liés aux représentations erronées comprennent des modèles mentaux incomplets ou non viables [84]. De nombreuses recherches étudient la relation modèles mentaux-représentations erronées [101, 42,

40, 44, 16, 84, 11, 8, 64, 10].

Selon Maier [55], la manière de résoudre ou de prévenir les représentations erronées est de confronter directement l'apprenant à une expérience qui provoque un déséquilibre. Remettre en question les représentations existantes des apprenants les encourage à détecter les problèmes de compréhension et les motive à construire des compréhensions appropriées [96]. En général, une stratégie d'enseignement des conflits cognitifs comporte trois étapes : enquêter sur les connaissances préalables des élèves et les représentations existantes ; mettre les élèves au défi avec des informations contradictoires ; évaluer le changement conceptuel entre les représentations préalables des apprenants et les représentations actuelles [53].

### 2.2.4 Détecter les représentations erronées

#### Le « concept inventory »

Des questionnaires spécifiques, appelés *concept inventories* (CI), semblent être l'option la plus utilisée pour identifier des représentations erronées.

Selon Wittie *et al.*, un *concept inventory* (CI) est « a research-based multiple-choice test that seeks to measure a student's knowledge of a set of concepts while also capturing conceptions and misconceptions they may have about the topic under consideration » [122]. La mise au point d'un CI évaluant la compréhension des concepts fondamentaux par les apprenants dans le cadre d'un cours d'introduction à la programmation permet aux enseignants d'identifier les représentations erronées des apprenants et de fournir ensuite une intervention appropriée à l'avenir [116]. Cet instrument d'évaluation peut être utilisé comme test de diagnostic, pour identifier les activités d'enseignement et d'apprentissage appropriées, pour évaluer l'impact d'un changement de méthodes d'enseignement sur la compréhension des élèves, pour donner un retour d'information aux élèves, pour comparer les méthodes d'enseignement ou pour évaluer l'apprentissage global et les effets de l'enseignement.

Dans l'enseignement de l'informatique, des CI ont été proposés [116], entre autres, pour les arbres de recherche binaires [13, 45], la logique numérique [32], les tables de hachage [45], les systèmes d'exploitation [121], la séquence et l'affectation [42, 16, 101, 108], et les bases de programmation [27, 9, 54, 117]. Les CI ne sont cependant pas exclusifs à l'enseignement de l'informatique. On les retrouve ainsi dans d'autres disciplines telles que la physique [33] ou encore l'astronomie [89].

Les travaux de Dehnadi [16, 15], et particulièrement la définition de 11 modèles mentaux utilisés par les apprenants pour résoudre des problèmes d'affectation de variable (Fig. 2.2), ont servi de base à l'élaboration d'un CI consacré à la variable. Ce CI a notamment été utilisé par Plass [82], mais également par Henry qui en a

| Mental Model | Description |
|---|---|
| M1 | The value of the variable on the right is assigned to the variable on the left; the variable on the right is initialized to 0 |
| M2 | The value of the variable on the right is assigned to the variable on the left; the variable on the right retains its original value |
| M3 | The value of the variable on the left is assigned to the variable on the right; the variable on the left is initialized to 0 |
| M4 | The value of the variable on the left is assigned to the variable on the right; the variable on the left retains its original value |
| M5 | The value of the variable on the right is added to the value of the variable on the left; the variable on the right is initialized to 0 |
| M6 | The value of the variable on the right is added to the value of the variable on the left; the variable on the right retains its original value |
| M7 | The value of the variable on the left is added to the value of the variable on the right; the variable on the left is initialized to 0 |
| M8 | The value of the variable on the left is added to the value of the variable on the right; the variable on the left retains its original value |
| M9 | The values assigned to the two variables are exchanged |
| M10 | Variables retain their original values |
| M11 | Mathematical equality is applied |

FIGURE 2.2 – Les modèles mentaux identifiés par Dehnadi pour l'affectation de type a=b [6]

| | |
|---|---|
| G1 | Une boucle effectue les mêmes opérations à chaque itération. |
| G2 | L'opérateur *OR* renvoie faux quand les deux conditions sont vraies. |

TABLE 2.1 – Les représentations erronées identifiées par Gover et Basu [26].

proposé une version enrichie [42, 29, 43].

Parmi les 162 représentations erronées répertoriées par Sorva [108] dans la littérature, 30 concernent les concepts de structure conditionnelle et de boucle. Parmi celles-ci, Swidan [113] a repris les plus fréquentes (Fig. 2.3). Ces représentations erronées sont à la base de la construction de nombreux CI [64, 97]. Par ailleurs, Grover et Basu[26] ont démontré que la liste répertoriée par Sorva n'était pas exhaustive en mettant en évidence de nouvelles représentations erronées 2.1.

### La simulation visuelle de programmes

Outre les questionnaires, d'autres outils permettant de détecter les représentations erronées ont été développés. Ainsi, Sorva et Sirkiä [101, 100] ont conçu un outil (UUhistle) testant les concepts plus techniques de la machine notionnelle et mettant dès lors à l'épreuve les modèles mentaux de l'apprenant. Cet outil permet de simuler visuellement l'exécution d'un programme [109] (VPS, pour Visual Program Simulation).

La VPS fait participer l'apprenant à des simulations interactives dans lesquelles il joue le rôle de l'ordinateur en tant qu'exécuteur d'un programme. L'apprenant utilise une visualisation donnée d'une machine notionnelle pour illustrer ce qui se passe en mémoire lorsque l'ordinateur traite le programme. Si la VPS est

| | | | | |
|---|---|---|---|---|
| M26 | A false condition ends program if no `else` branch exists | IF ELSE | IF touching the color [black] then { Say *"Auw!!"*; }; Say *"I am moving"*; | Gobo does not say anything |
| M30 | Adjacent code executes within loop | Variables & Loops | Set [counter] to 0; Repeat 5 { Change [counter] by 1;}; Say [counter]; | Gobo says 1 till 5 |
| M31 | Control goes back to start when condition is `false` | IF ELSE | Say *"I am moving"*; IF touching the color [black] then { Say *"Ouch!!"*; }; Say *"Done!!"*; | Gobo says *"I am moving"* |
| M33 | Loops terminate as soon as condition changes to `false` | Variables & Loops | Set [number] to 1; Repeat until [number]=3 { Change [number] by 1; Say [number]; }; | Gobo says 2 |

FIGURE 2.3 – Les différentes mauvaises représentations de Sorva répertoriées par plusieurs articles et sélectionnés par [113] pour la boucle et la structure conditionnelle.

considérée comme un outil à privilégier avec des novices (en particulier, chez les plus jeunes) [68], elle ne permet pas d'arborer des concepts avancés de programmation, par exemple les concepts issus du paradigme orienté-objet. Cependant, son utilisation pour mettre en évidence certaines représentations erronées au niveau des concepts de base en programmation [101] et pour identifier les modèles mentaux incorrects de la machine notionnelle en fait un outil d'apprentissage puissant.

### 2.2.5 Éviter les représentations erronées

Qian et Lehman [84] répertorient différents moyens pédagogiques à mettre en oeuvre pour éviter la création de représentations erronées par les apprenants.

- Donner de bons exemples de programmes pour faire comprendre les concepts de base. Un bon exemple est un exemple qui illustre correctement le concept abordé, en minimisant la charge cognitive nécessaire à sa compréhension.
- Choisir un environnement de développement intégré (en anglais IDE pour Integrated Development Environment) notifiant l'apprenant de ses erreurs de syntaxe de manière compréhensible pour un novice et afin de lui éviter les erreurs de compilation.
- Utiliser un outil permettant d'éviter les erreurs de syntaxe, comme Scratch. Ils avancent également que l'aspect visuel des outils de programmation en blocs facilite la compréhension de certains concepts, par exemple la boucle.
- Choisir un langage proche du langage naturel pour faciliter la lecture, la compréhension et l'écriture du code, tel que MOOSE [7], un langage développé pour les enfants.

— Utiliser un outil de visualisation, tel que Online Python Tutor, Greenfoot ou UUhistle, permettant à l'apprenant de comprendre l'impact de chaque instruction d'un programme. Cependant, l'utilisation d'un tel outil augmente la charge cognitive de l'apprenant, ce qui n'est pas recommandé.
— Déboguer (de manière simplifiée) un programme.
— Disposer d'un outil d'aide à la compréhension des messages d'erreur, afin de les rendre plus explicites pour l'apprenant de manière à ce qu'il comprenne leur origine. Deux outils sont mentionnés : Decaf [3] qui permet d'expliquer les messages d'erreur et Whyline [48] qui offre à l'apprenant la possibilité d'obtenir une explication sur l'échec d'exécution d'un programme.

Tous ces moyens pédagogiques viennent apporter de l'aide aux apprenants en complément de l'enseignement dispensé par l'enseignant, sans s'y substituer.

## 2.3 Les outils tangibles d'aide à l'enseignement de la programmation

L'interaction tangible est une approche privilégiée pour aider les enfants à développer des compétences en résolution de problèmes [98]. Des travaux pionniers concernant l'apprentissage de la programmation à l'aide de robots ou de dispositifs embarqués peuvent être attribués à Papert [74] et Perlman [80]. Si ces recherches datent d'il y a plus e 40 ans, la réponse apportée par le tangible au défi d'enseigner la programmation n'est à ce jour que partielle.

Si l'approche du tangible pour enseigner les concepts de base en programmation connaît un assez grand succès auprès d'un jeune public, très peu d'outils tangibles d'aide à l'apprentissage ont été développés pour un public de jeunes adultes (16 ans et plus) [30]. De plus, il semble qu'aucun outil tangible n'a été considéré pour détecter les représentations erronées présentes chez les apprenants.

Un état de l'art a été réalisé, basé sur le travail effectué par Henry et al. [30]. Cet état de l'art a pour objectif d'identifier des outils tangibles destinés à un public de jeunes adultes.

### 2.3.1 Méthodologie

Une revue systématique de la littérature a été menée [18, 81] (Tab. 2.2).

Sur le portail Google Scholar, la recherche s'est limitée aux 100 premiers articles. Parmi ceux-ci, 12 n'étaient pas disponibles. Sur les 88 articles disponibles, 24 seulement présentaient un outil tangible abouti, disposant d'un prototype. Cinq articles supplémentaires ont été ajoutés par une recherche « boule de neige », à savoir en analysant les références citées et les références citant les 24 articles

| Portail | Résultats (pertinents) de la requête « novice AND tangible AND programming » |
|---|---|
| Google Scholar [4] | 100+ (24) |
| ResearchGate [5] | 21 (5) |
| IEEE Xplore [6] | 9 (2) |
| ACM DL [7] | 100+ (9) |

TABLE 2.2 – Critères de la revue systématique menée

sélectionnés. L'article de Henry [30] faisait partie des 88 articles disponibles. Sa lecture a ajouté huit articles, pour un total de 37 articles pertinents (24 en recherche directe et 13 en recherche indirecte). Les portails de recherche ResearchGate et ACM DL, n'ont pas apporté de nouveaux articles, tous ayant déjà été identifiés lors de la recherche sur Google Scholar. Le portail IEEE Xplore a permis de rajouter un dernier article, portant le total à 38.

Les articles pertinents ont été sélectionnés manuellement, après lecture de l'« abstract » voir, lorsque cela s'avérait nécessaire, de l'article complet.

Les 38 articles collectés ont été analysés selon une grille de critères définie préalablement (Tableau 2.3). Une fiche de lecture (avec illustration) a été réalisée pour chacun des articles.

- **Âge du public cible**. Il est question d'identifier les outils utilisables auprès d'un public de jeunes adultes, âgés de 16 ans et plus.
- **Concepts abordés**. Il s'agit d'identifier les concepts de programmation qui sont mis en place à travers l'outil : variable, structure conditionnelle, boucle ou fonction.
- **Retour à l'utilisateur**. Le type de retour présenté à l'utilisateur peut être simple (par exemple, le déplacement d'un robot dans un parcours ou l'émission d'un son) ou complet, comme le serait un VPS (par exemple, la présentation des valeurs contenues dans les compteurs à chaque itération d'une boucle).
- **Langage manipulé**. L'outil permet à l'apprenant de manipuler un langage de programmation possédant une syntaxe plus complexe que de simples mots-clés, tels que « Move », « Right » ou encore « Defrost ».

La liste des outils tangibles collectés via cette revue de la littérature n'est évidemment pas exhaustive, compte tenu des choix méthodologiques portés. Elle fournit cependant une base de travail intéressante et inspirante.

### 2.3.2 Analyse et Discussion

La revue de la littérature a permis de recenser et d'analyser les 38 outils suivants : T_Butterfly [90], T_ProRob [91], GameBlocks [102], RockBlocks [104], Dialando [103], Tern [38], Bloctopus [88], Quetzal [37], BOTS &(MAIN)Frames [63], StoryBlocks [50], T-Maze [119], tactusLogic [105], TanPro-Kit [118], McNerney's TCB [62], ARcadia [47], E-Block [120], Algo.Rhythm [78], CHERP [110], ToRTis [79], MakerWear [46], Torino [65], X Card Game [51], Electronic Blocks [123], roBlocks [95], FlowBlocks [126], SystemBlocks [125], Topobo [76], AlgoBlock [112], BeeBot [49], Dr. Wagon [12], Kibo [111], Robo-Blocks [99], StarLoop [57], Tangicons [92], Strawbies [39], Toque [115], TurTan [19], Thymio [67].

Les représentations de ces différents outils ont été extraites des articles et sont consultables à l'annexe H.

#### Catégorisation

La majorité des outils a pour objectif de faire se déplacer un élément (24/38 - 63%). Cet élément peut être physique (robot) ou virtuel, tel un personnage dans un labyrinthe ou point traçant une figure, à l'image de la tortue Logo. Il s'agit de la catégorie *A* du tableau.

Viennent ensuite les outils qui consistent à faire intervenir des senseurs pour obtenir une réaction conditionnelle (5/38 - 16%). L'objectif à atteindre par l'utilisateur est en général plus libre : l'utilisateur veut réaliser un projet personnel à l'aide des éléments qui lui sont proposés, par exemple une casquette dotée d'une led qui s'allume quand il fait sombre. L'interaction entre l'outil et l'utilisateur se limite à assembler des modules (3/5 - 60%) pour parvenir à un objectif fixé. Dans d'autres outils par contre, l'utilisateur peut ou doit programmer le comportement que les modules auront entre eux via une application (2/5 - 40%). La partie programmation n'est alors pas faite de manière tangible, mais via l'aide d'un ordinateur. Il s'agit de la catégorie *B* du tableau.

La dernière catégorie concerne les outils qui visent un objectif narratif (3/38 - 8%). Par objectif narratif, il est entendu de réaliser un programme permettant de raconter une histoire ou d'énoncer une procédure à suivre, comme une recette. Il s'agit de la catégorie *C* du tableau.

Six derniers ne rentrent pas à proprement parler dans une des catégories définies. Deux outils facilitent une vision des flux et de leur manipulation. Un autre permet l'enregistrement d'une séquence de mouvements qu'un pantin reproduira. Un outil demande de manipuler des sons à l'aide d'éléments pensés pour les personnes malvoyantes. Un autre est composé de blocs (Fig. H.2) capables de reproduire via un petit élément percutant une séquence de percussions perçues.

L'élément percutant peut alors être orienté vers un second bloc, créant une chaîne. Le dernier outil permet quant à lui de réaliser un programme à l'aide d'éléments similaires à ceux d'un langage de programmation, mais via une interface tangible. L'outil propose une visualisation imagée : un feu de signalisation est utilisé pour définir le concept de boucle (Fig. 2.4). Il s'agit de la catégorie $D$ du tableau.

### Public cible

L'introduction de la programmation par moyens tangibles a déjà fait ses preuves auprès des enfants [98]. Cette tendance se retrouve dans les articles sélectionnés où la plupart des outils sont développés pour des jeunes. Ce constat avait été fait par Henry, Dumas et Bodart [30]. Seuls quatre outils ont été pensés pour les apprenants plus âgés : Bloctopus [88], BOTS &(MAIN)Frames [63], roBlocks [95] et Thymio [67].

**Concepts abordés** Le concept le plus présent dans les différents outils analysés est la boucle (21/38 - 55%). Cela peut s'expliquer par le jeune âge général du public cible, la boucle étant alors vue comme la répétition d'une action. La boucle n'est cependant pas toujours tangibilisée de la même manière (Fig. 2.4). Dans des outils comme Strawbies [39] ou Quetzal [37], le concept de boucle est paramétrable : l'utilisateur choisit le nombre d'itérations. Il en va de même dans Robo-Blocks [99], même si la boucle est implicite au design de l'outil. D'autres outils proposent plutôt une boucle « conditionnelle » à utiliser pour continuer une opération jusqu'à la validation d'une condition spécifiée, comme par exemple Dr. Wagon[12]. Enfin, l'outil tactusLogic [105] va plus loin que les autres dans la boucle conditionnelle, celle-ci permettant de réaliser ses propres comparaisons comme condition d'itération : les opérateurs proposés sont « $<$ », « $>$ », « $\leq$ », « $\geq$ » et « $=$ ».

La structure conditionnelle est mise en place dans moins de la moitié des outils (18/38 - 47%). Tout comme la boucle, elle est parfois non paramétrable (et est donc prédéterminée), comme c'est le cas par exemple dans Tern [38] (Fig 2.5). Cet outil propose une structure conditionnelle intitulée « si bloqué » où l'utilisateur choisit le comportement normal et le comportement en cas de blocage du robot sur son parcours. Elle est parfois aussi configurable grâce à des éléments prédéterminés, comme dans Quetzal [37]. Enfin, la structure conditionnelle des outils fait parfois intervenir une algèbre booléenne plus complète. C'est le cas pour tactusLogic [105] où les conditions sont alors établies par l'apprenant avec les opérateurs de comparaison déjà cités pour le concept de boucle.

La variable est un concept nettement moins représenté dans les différents outils (moins de 30% en disposent). Une fois encore, l'approche de la variable se

| Nom | + | 1 | 2 | 3 | 4 | C | L | G |
|---|---|---|---|---|---|---|---|---|
| ARcadia | | X | X | X | X | | | B |
| Algo.Rhythm | | | | | | | | D |
| AlgoBlock | | X | X | X | | | | A |
| BOTS & (MAIN)Frames | X | | X | X | X | | | A |
| BeeBot | | | | | | | | A |
| Bloctopus | X | X | X | X | | | X | B |
| CHERP | | X | X | X | X | | | A |
| Dialando | | | | | | | | A |
| Dr. Wagon | | | X | X | | | | A |
| E-Block | | | | | | | | A |
| Electronic Blocks | | | X | | | | | B |
| FlowBlocks | | | X | X | | X | | D |
| GameBlocks | | | | | | | | A |
| Kibo | | X | X | X | | | | A |
| MakerWear | | | X | | | | | B |
| McNerney's TCB | | X | X | X | X | | | C |
| Quetzal | | | X | X | | | | A |
| roBlocks | X | | X | | | | | B |
| Robo-Blocks | | | | X | | | | A |
| RockBlocks | | | | | | | | A |
| StarLoop | | X | | X | | | | A |
| StoryBlocks | | | X | X | | | | C |
| Strawbies | | | | X | | | | A |
| SystemBlocks | | | | | | | | D |
| T-Maze | | | | X | | | | A |
| T_Butterfly | | | | | | | | A |
| T_ProRob | | X | X | X | X | | | A |
| tactusLogic | | X | X | X | | X | X | D |
| TanPro-Kit | | | | | | | | A |
| Tangicons | | | | | | | | A |
| Tern | | | X | X | X | | | A |
| Thymio | X | | X | | | | | A |
| ToRTis | | X | | | X | | | A |
| Topobo | | | | | | | | D |
| Toque | | X | | X | | | | C |
| Torino | | | | X | | | | D |
| TurTan | | | | X | | | | A |
| X Card Game | | | | | | | | A |
| TOTAL | 4 | 11 | 18 | 21 | 7 | 2 | 2 | |

TABLE 2.3 – Classification des outils tangibles sélectionnés selon les critères définis.

Légende

| | | | |
|---|---|---|---|
| 1 | variable | C | retour complet |
| 2 | structure conditionnelle | + | 16 ans et plus |
| 3 | boucle | L | vrai langage manipulé |
| 4 | fonction ou procédure | G | Catégorisation de l'outil |
| A | Déplacement d'un élément | C | Objectif narratif |
| B | Assemblage de capteurs | D | Autres |

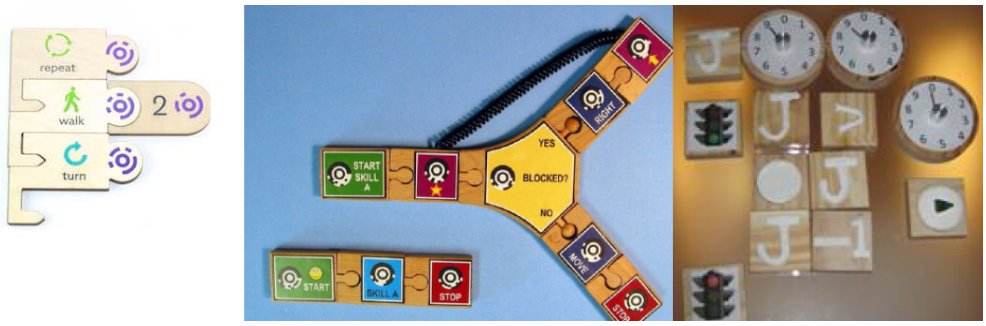



FIGURE 2.4 – Les différents types de boucles. À gauche, une boucle paramétrable de Strawbies [39] ; au centre, une boucle conditionnelle de Tern [36] (représentée par le fil torsadé qui attache les deux extrémités) ; à droite, la boucle de tactusLogic [105].

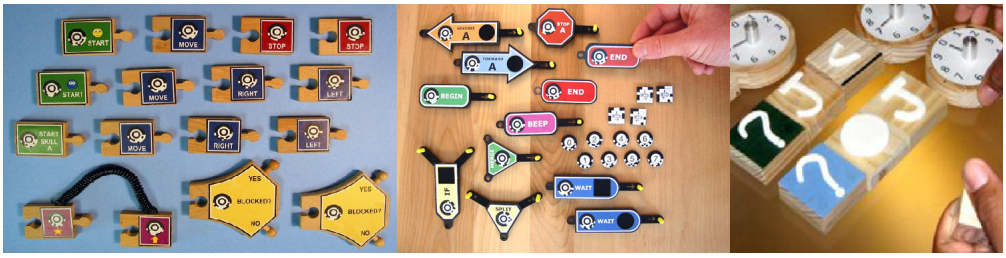



FIGURE 2.5 – Les différents types de structures conditionnelles. À gauche, une condition prédéfinie de Tern [36] ; au centre, une condition paramétrable de Quetzal [37] (la condition vient s'insérer dans le rectangle noir) ; à droite, la condition de tactusLogic [105].

limite généralement à l'affectation d'une valeur. La modification d'une variable n'est jamais envisagée, excepté dans tactusLogic [105] qui permet additions, soustractions, et autres opérations sur les variables.

La définition de fonction est rendue possible dans certains langages. La fonction est cependant le concept le moins présent (7/38 - 18%). Lorsque celle-ci est présente, elle ne permet pas l'introduction de paramètres. Il s'agit simplement d'un ensemble d'instructions que l'utilisateur définit et qu'il n'a qu'à invoquer. Cela permet notamment d'éviter la répétition de séquences d'instructions, via la définition d'un comportement réutilisable (par exemple, une stratégie d'évitement si un obstacle est rencontré). L'utilisation des fonctions est parfois obligatoire si l'utilisateur veut résoudre un exercice, car celui-ci dispose d'un nombre limité de blocs (c'est le cas pour BOTS &(MAIN)Frames [63]). Si l'outil ToRTis [79] propose la mise en place du concept de fonction et non pas celui de boucle ni de structure conditionnelle, c'est parce que la fonction y est utilisée pour enregistrer une suite d'actions. Cette suite, stockée en mémoire, est ensuite disponible et peut être appelée via un bouton.

**Retour à l'utilisateur** Classer le niveau de détails qu'offre le retour de l'outil à l'étudiant n'a pas été aisé. En effet, pour qu'un outil soit considéré comme disposant d'un niveau de détails « complet », il faut que celui-ci offre la possibilité à l'apprenant de voir l'état de son programme à chaque itération de la boucle ou après chaque instruction s'il le souhaite. Deux outils offrent ce niveau de détails. Le premier, tactusLogic [105], permet à l'apprenant d'instaurer dans son programme la fonction « print » s'il souhaite voir la valeur d'une variable lors de l'exécution d'une boucle par exemple. Le second, FlowBlocks [126], permet à l'apprenant de placer un « debugger » à plusieurs endroits de son programme. L'apprenant peut alors, lors de l'exécution d'une boucle, voir le compteur s'incrémenter en fonction de chaque passage d'un « flux » dans la branche de la boucle. Cela permet de visualiser en temps réel les itérations du programme. Si Robo-Blocks [99] permet de ralentir l'exécution du programme qui pilote le robot défini par l'utilisateur, il ne permet cependant pas de savoir à quelle itération précise le robot aurait dû s'arrêter de répéter son action pour effectuer le parcours correct. C'est donc de l'essai-erreur.

### Langage manipulé

Seul l'outil Bloctopus [88] peut introduire un vrai langage de programmation (en l'occurrence, Javascript) pour le manipuler. La partie tangible se fait en parallèle avec une partie logicielle qui définit le comportement de la partie tangible (Fig. 2.6). Cependant, l'outil tactusLogic [105] offre aussi une expérience très proche de l'utilisation d'un langage, avec une très grande souplesse pour l'utilisateur. L'inconvénient est alors que l'utilisateur peut plus facilement faire des erreurs de

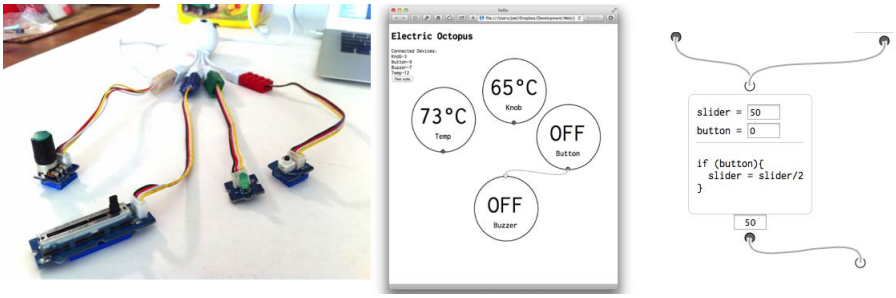



FIGURE 2.6 – L'outil Bloctopus : à gauche, différents capteurs branchés ; au centre, l'interface permettant de relier deux éléments ; à droite, l'écriture de code pour rendre les liaisons plus complètes.

syntaxe lors de l'élaboration de son programme.

### Des outils adaptables ?

Certains outils pourraient être modifiés pour mieux correspondre à un public de jeunes adultes. En effet, si un système considéré comme « fermé » tel que BeeBot [49] peut difficilement être détourné de la fonction pour laquelle il a été créé, d'autres s'avèrent plus aisément détournables. Par exemple, les outils Tern [38] et Quetzal [37] offrent une interface assez souple dont on pourrait imaginer remplacer les labels pour obtenir une syntaxe plus proche d'un vrai langage de programmation. D'ailleurs, Tangicons [92] est une adaptation, pour un public plus jeune, de l'outil Quetzal. Pour sa part, tactusLogic [105] offre une couverture des concepts de programmation assez large, ainsi qu'une granularité dans la définition du contenu de ses blocs. Un bloc ne représente qu'une variable ou un opérateur. Cette finesse de découpe peut cependant être considérée comme une faiblesse, celle-ci pouvant mener l'utilisateur à des erreurs de syntaxe.

# Chapitre 3

# Contexte de recherche

Cette recherche s'intègre dans un projet de thèse mené à l'Université de Namur (UNamur) et portant sur l'évolution des modèles mentaux que possèdent les étudiants suivant un cours d'introduction à la programmation [29, 43, 42].

## 3.1 Le projet de thèse

Dans son projet de thèse, Henry utilise, enrichit et développe notamment des CI pour chacun des concepts suivants : variable, structure conditionnelle, fonction et boucle. L'approche qu'elle propose permet non seulement d'identifier les modèles mentaux des étudiants, mais surtout de visualiser l'évolution de ces modèles mentaux sur une période d'enseignement donnée (Fig. 3.1).

Selon Henry [29, 43], les travaux existants concernant les modèles mentaux développés dans le cadre de cours d'informatique se limitent généralement à une mesure ponctuelle, à la fin de l'apprentissage. Pourtant, si les étudiants doivent construire des modèles mentaux viables à un stade précoce, l'enseignant doit les aider lors des premiers cours, notamment parce que ces cours sont souvent les plus suivis.

Concernant les représentations erronées, Henry [29, 43] souligne que les travaux existants dans ce domaine ne font généralement pas référence aux modèles mentaux. Pourtant, les deux notions sont clairement liées : les étudiants utilisent les représentations erronées entre autres, pour construire leurs modèles mentaux erronés. Puisqu'il faut confronter les étudiants à leurs représentations erronées, celles-ci constituent une ressource pédagogique de premier ordre pour les enseignants. La construction de CI permet non seulement d'identifier les représentations erronées, mais également d'identifier les modèles mentaux des élèves.

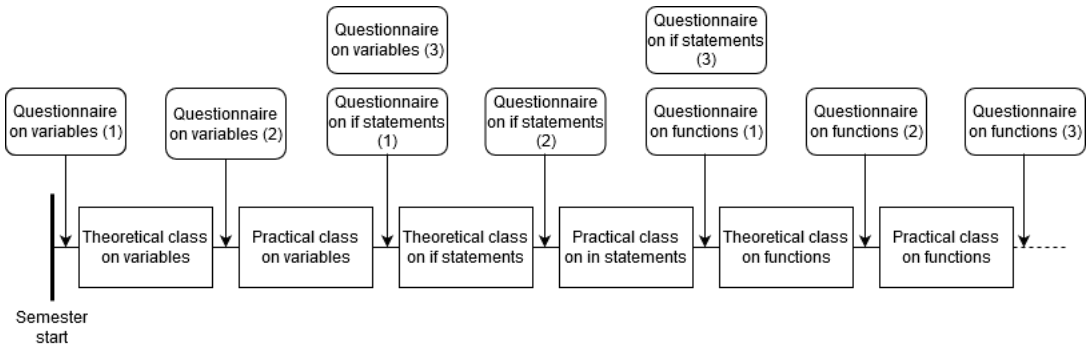



FIGURE 3.1 – Organisation, selon un calendrier, de la passation des CI pendant un semestre.

Dans des contextes autres que l'enseignement de l'informatique, un CI est administré avant (prétest) et après (post-test) une période d'enseignement. S'il a été avancé que les étudiants en informatique n'auraient pas de représentation préconçue sur le domaine et que, par conséquent, l'administration d'un CI en tant que prétest ne serait pas utile (« It is likely that a majority of student misconceptions are a result of instruction in computer science rather than based upon a set of common, naive understandings [that students] bring to the topic from their experience in the world » [117]), Henry [29, 43] se positionne en faveur d'une évaluation sur le long terme. En effet, les sujets informatiques que sont la programmation et l'algorithmique feront bientôt partie du programme d'enseignement en Belgique francophone. Les étudiants auront dès lors de plus en plus l'occasion de découvrir les concepts de base de la programmation avant l'université (et le cours d'introduction à la programmation).

La recherche présentée ici tire son inspiration des CI développés par Henry et des résultats obtenus à ce jour (publiés ou non) dans le cadre de son projet de thèse [29, 43, 42].

## 3.2 Le cours d'introduction à la programmation

Un cours d'introduction à la programmation organisé en première année de bachelier à l'UNamur (INFOB131) constitue le terrain de mesure de cette recherche. Ce cours est organisé sur un semestre selon un schéma spécifique : chaque concept de base de programmation fait l'objet, sur une semaine, d'un cours magistral de quatre heures, suivi de trois heures de travaux pratiques (TP). Trois semaines

sont consacrées à des projets intégratifs. Les concepts de base abordés sont, dans l'ordre chronologique, les variables et leurs valeurs, les structures conditionnelles, les fonctions et leurs spécifications, les boucles, les structures de données simples, les fichiers, des bases d'algorithmique (recherche de minimum, tri, complexité, etc.) et la récursion. Une introduction à la qualité logicielle et au test logiciel est également au programme. Les différents concepts sont illustrés en Python.

Les cours magistraux sont dispensés en auditoire, à l'aide d'une présentation sous forme de diaporama et d'un IDE (Python Notebook). Les TP sont composés de deux parties distinctes. Une activité d'une heure se déroule sans ordinateur et propose de résoudre sur papier de courts problèmes liés au concept abordé durant la semaine. Elle est suivie de deux heures d'exercices effectués sur ordinateur. Il n'est pas obligatoire de suivre ni les cours magistraux ni les TP.

À travers les trois projets proposés durant le semestre, les étudiants sont appelés à mettre en oeuvre les compétences et connaissances du cours pour réaliser, en équipe, un programme de plus grande envergure. Le premier projet implique les concepts de variables, de structures conditionnelles et de fonction. Le deuxième projet y ajoute le concept de boucle. Le troisième projet inclut les concepts de structures de données et de fichiers. Parce qu'ils constituent une évaluation continue, la participation à ces projets est obligatoire.

Le cours est suivi par les étudiants inscrits dans un master en informatique (Info) et ceux inscrits dans un master en ingénierie de gestion (option « technologies et management de l'information » - IngMI). Il en résulte une diversité particulièrement dans le parcours des étudiants et dans leurs attentes par rapport au cours. Par ailleurs, le cours d'introduction à la programmation est le seul cours d'informatique prévu en première année du cursus d'ingénierie de gestion.

Le cours est un prérequis au cours de programmation orientée-objet organisé en deuxième bachelier (INFOB234).

# Chapitre 4

# Problématique de recherche

L'apprentissage de la programmation est considéré comme difficile par les novices et les étudiants du cours INFOB131 ne font pas exception à la règle. Si le taux d'échec global est acceptable sur trois sessions, un trop grand nombre d'étudiants ne semblent pas maîtriser les concepts de base à leur entrée en deuxième année de bachelier. Cette situation s'avère problématique pour le bon déroulement du cours INFOB234 [31]. Si l'équipe enseignante (enseignant et assistants en charge du cours) n'en a pas réellement conscience [43], c'est notamment parce qu'aucun outil pédagogique en place à l'heure d'écrire ce mémoire ne permet de souligner concrètement les difficultés éprouvées par chaque étudiant.

Ce travail de recherche a pour objectif de tirer parti des résultats de Henry pour proposer un outil d'aide à l'apprentissage de la programmation permettant à l'équipe enseignante de détecter rapidement les difficultés rencontrées par les étudiants.

Compte tenu du contexte de la recherche et de l'intérêt pédagogique des outils tangibles dans l'apprentissage de la programmation chez des novices, la question de recherche posée est la suivante : « Dans quelle mesure un outil tangible d'aide à l'apprentissage de la programmation peut-il influencer les représentations erronées des étudiants suivant un cours d'introduction à la programmation ? »

Pour apporter des éléments de réponse à cette question, quatre hypothèses sont formulées :

- Un outil tangible peut aider à l'apprentissage de la programmation dans un contexte d'enseignement supérieur (H1)
- Un outil tangible (d'aide à l'apprentissage de la programmation) est utilisable avec un public de jeunes adultes (16 ans et +) (H2)
- Un outil tangible peut aider à détecter les représentations erronées des étudiants (H3)

— Un outil tangible peut aider à corriger les représentations erronées des étudiants (H4)

# Chapitre 5

# Méthodologie

Cette recherche se veut interdisciplinaire, située entre les domaines que sont l'informatique et l'éducation. Les réflexions qui l'animent et les méthodologies choisies pour valider les hypothèses sont donc logiquement issues de ces deux disciplines.

## 5.1 Développement de l'outil tangible

La méthodologie guidant le développement de l'outil est inspirée de la *design-oriented research* [2]. Cette approche consiste à mener un processus itératif qui articule des phases de conception, de mise en place de cet outil à différents niveaux et d'analyse des résultats obtenus (Fig. 5.1). Le développement n'est donc pas uniquement orchestré par le chercheur et les experts, mais fait aussi intervenir les enseignants qui pourraient utiliser l'outil dans le cadre de leurs cours et les étudiants qui pourraient à terme l'utiliser.

### 5.1.1 La phase de conception

L'approche utilisée pour concevoir l'outil tangible destiné à l'apprentissage de la programmation a comporté quatre étapes : s'approprier le sujet, prendre du recul par rapport aux connaissances des experts (littérature scientifique), définir les caractéristiques attendues de l'outil et créer l'outil (Fig. 5.2). Les trois dernières étapes étaient itératives.

L'objectif de l'étape d'**appropriation** est l'acquisition de connaissances liées à l'enseignement de la programmation et aux outils tangibles. Cette étape repose notamment sur l'état de l'art, mais également sur les connaissances des promoteurs

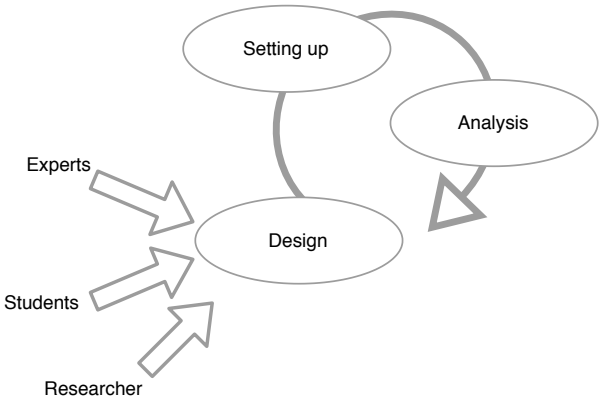



FIGURE 5.1 – Représentation visuelle de la méthodologie *design-oriented research*.

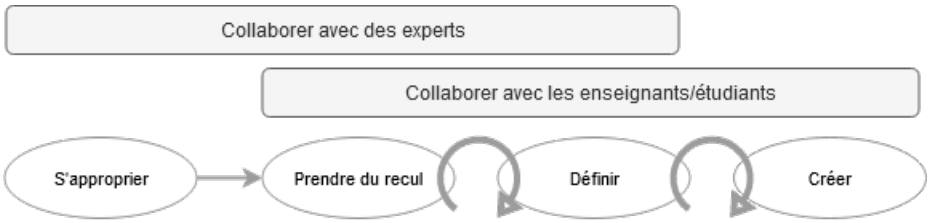



FIGURE 5.2 – L'approche de conception.

de cette recherche issus des domaines de l'ingénierie logicielle, des interactions homme-machine et de la didactique de l'informatique.

Puis, suit la **prise de recul**. Il s'agit d'extraire des discours scientifiques les concepts et principes clés qui permettront d'apporter des éléments de réponse à la question de recherche. Considérant les difficultés rencontrées par les novices dans le cadre d'un cours d'introduction à la programmation, niveau universitaire, le choix s'est porté sur trois concepts de base en programmation : la variable, la structure conditionnelle et la boucle. Ces concepts, abordés en début d'apprentissage, engendrent d'après la littérature de nombreuses représentations erronées chez les étudiants. Une sélection de ces représentations erronées a été faite. En ce qui concerne la variable, cette sélection est basée sur les travaux respectifs de Dehnadi[16], Sorva[101, 108] et Henry [43, 29, 42]. Les représentations erronées des concepts de structure conditionnelle et de boucle se basent, elles, sur le travail de recensement de Sorva [108]. Ainsi, seules les représentations erronées recensées par au moins deux études ont été considérées, à l'instar de ce qui avait été fait dans le travail de Swidan, Hermans et Smit [113].

La **définition** des caractéristiques de l'outil tangible a été inspirée par les recommandations de Qian et Lehman [84], mais également les discussions avec les promoteurs-experts.

Enfin, l'étape de **création** de l'outil a consisté en plusieurs cycles et a bénéficié des retours à la fois des promoteurs-experts, mais également des futurs utilisateurs (enseignants et étudiants).

### 5.1.2 La phase de mise en place

La deuxième phase de la méthodologie requiert la mise en place de l'outil à différents niveaux (Fig. 5.1).

Deux séances d'évaluation ont été menées afin de collecter des données d'utilisation sur l'outil développé. La première séance s'est déroulée avec des membres du personnel enseignant responsables de cours magistraux ou de TP en programmation, et donc susceptibles d'utiliser l'outil dans ce cadre.

Des observations sur les comportements et manipulations effectuées avec l'outil ont été menées et retranscrites dans un cahier. Lorsqu'un comportement interpellait l'observateur, la personne était interrogée sur la raison de ce comportement. Des retours, remarques et pistes d'amélioration spontanés ont également été capturés. Enfin, un questionnaire standard d'utilisabilité (UEQ [93]) a été passé auprès de chaque participant à l'évaluation après qu'il ait utilisé l'outil.

La deuxième séance d'évaluation a eu lieu avec des étudiants inscrits au cours INFOB131. Les tests mis en place visaient non seulement à évaluer l'utilisabilité de l'outil, mais aussi à comparer l'impact, sur la compréhension des étudiants, de

FIGURE 5.3 – La disposition des éléments durant la deuxième séance d'évaluation. Le téléviseur à droite affiche le contenu de l'écran de la tablette utilisée par les étudiants.

certaines de ses caractéristiques.

Du point de vue des données collectées, cette deuxième séance d'évaluation a été entièrement filmée selon trois points de vue : un premier filmant la manipulation par les étudiants de la partie tangible de l'outil, un second offrant une vue d'ensemble et un troisième capturant la partie logicielle de l'outil (application) (Fig. 5.3). D'autre part, des notes manuscrites ont à nouveau été prises sur les comportements des étudiants vis-à-vis de l'outil. Enfin, les différentes solutions proposées par les étudiants ont été enregistrées sur un serveur distant créé spécifiquement dans ce but [1].

Le protocole suivi durant ces évaluations avait été établi au préalable avec les promoteurs-experts. Il a subi des adaptations entre les deux séances d'évaluation. Il est disponible dans l'annexe F. Ce protocole comprend le matériel nécessaire au

1. Le développement de ce serveur est détaillé dans la section 6.3.3.

déroulement du test, les conditions nécessaires au bon déroulé de l'évaluation et la séquence à suivre.

### 5.1.3 La phase d'analyse

Tout d'abord, en ce qui concerne la séance avec les membres du personnel enseignant, les résultats des questionnaires UEQ ont été encodés dans l'outil UEQ_Data_Analysis_Tool_Version7 [2] afin d'en dégager les tendances générales. Les différentes notes sur les comportements et remarques prises au cours de la séance ont été retranscrites sur ordinateur et ont été analysées pour mener à un plan d'action. Les résultats de ces analyses sont disponibles dans la section 7.1.

En ce qui concerne la séance avec les étudiants, les enregistrements vidéo ont été visionnés et manuellement retranscrits après la séance (échanges verbaux et interactions avec l'outil) afin de pouvoir mettre en évidence différents types de comportements. Par souci d'écologie, les 52 pages de retranscription n'ont pas été jointes dans les annexes. Elles sont cependant consultables sur demande. Les différentes notes prises ont également été retranscrites et ont permis de compléter la retranscription des vidéos d'enregistrement. Les différentes statistiques collectées par l'application ont été analysées pour déterminer le nombre d'occurrences de représentations erronées, ainsi que d'autres métriques comme le nombre moyen d'essais par étudiant.

De par le caractère cyclique de la méthodologie, l'analyse des résultats obtenus après chaque séance a permis d'améliorer l'outil.

## 5.2 Évaluation de l'outil tangible : public cible

Les évaluations ont permis de collecter des données auprès de deux publics cibles différents : les membres du personnel enseignant en charge d'un cours de programmation et les étudiants novices en programmation.

### 5.2.1 L'équipe enseignante

Huit membres de l'équipe enseignante, des assistants en charge de séances de TP, ont participé à la première séance d'évaluation de l'outil tangible d'aide à la programmation. Parmi ceux-ci, deux étaient dans leur première année de doctorat (master+1), cinq dans leur deuxième année (master+2) et un dans sa cinquième année (master+5) (Tab. 5.2.1).

2. Le questionnaire UEQ et l'outil UEQ_Data_Analysis_Tool_Version7 sont disponibles au lien suivant : https://www.ueq-online.org/

| Background | Nombre de participants |
|---|---|
| Master+1 | 2 |
| Master+2 | 5 |
| Master+5 | 1 |
| Total | 8 |

TABLE 5.1 – Répartition des participants à l'évaluation 1.

| Section | Hommes | Femmes | Total |
|---|---|---|---|
| Info | 3 | 1 | 4 |
| IngMI | 2 | 3 | 5 |
| Total | 5 | 4 | 9 |

TABLE 5.2 – Répartition des participants à l'évaluation 2.

### 5.2.2 Les étudiants

Lors de la seconde évaluation, neuf étudiants ont eu l'occasion de tester le prototype d'outil tangible et donner leurs retours. Ceux-ci ont été sélectionnés de manière aléatoire dans deux sections différentes, à savoir en section Info (quatre étudiants, dont trois hommes et une femme) et en section IngMI (cinq étudiants, dont trois femmes et deux hommes) (Tab. 5.2.2).

# Chapitre 6

# Contributions personnelles

## 6.1 La définition des besoins

### 6.1.1 Exigences explicites

#### Viser le bon public cible

L'outil tangible d'aide à l'apprentissage doit s'adresser principalement aux étudiants qui suivent un cours d'introduction à la programmation en enseignement supérieur. Il s'agit donc d'un public de jeunes adultes, d'un âge estimé entre 16 et 21 ans. L'outil peut ne pas se limiter à ce public. Ces étudiants, qui découvrent pour la plupart la programmation pour la toute première fois, peuvent suivre différents cursus scolaires. En effet, les cours de programmation sont de moins en moins l'exclusivité des cursus d'informatique et sont proposés dans d'autres filières tels que les mathématiques, les sciences chimiques et l'ingénierie de gestion.

#### Susciter l'intérêt

Dans le contexte spécifique de cette recherche, il est reconnu qu'une partie des étudiants éprouvent difficilement de l'intérêt pour le cours de programmation, si ce n'est l'obligation de le réussir pour avancer dans leur cursus. Ces étudiants ont du mal à visualiser les concepts abstraits s'ils se contentent de les "manipuler" à travers un IDE. Il s'agit généralement des étudiants issus de la section IngMI. Pour susciter l'intérêt de ces étudiants et les motiver dans leur apprentissage, outre l'aspect tangible permettant la manipulation concrète des concepts, l'outil pourrait présenter un aspect "défi à résoudre", plongeant de façon ludique l'étudiant dans un apprentissage actif.

#### Privilégier l'expérimentation

Durant son apprentissage des concepts de programmation, l'étudiant doit pouvoir expérimenter, tester facilement différentes alternatives pour pouvoir atteindre un même objectif. L'outil doit donc favoriser ce comportement au moyen d'une grande modularité. La phase durant laquelle l'étudiant analyse sa solution (pour en évaluer la validité) doit être simple et rapide, facilement reproduite.

#### Détecter les représentations erronées

La détection des représentations erronées chez les étudiants est une solution aux difficultés qu'ils rencontrent lors de leur apprentissage. L'outil doit donc permettre à son utilisateur (étudiant), ou à l'enseignant de ce dernier de se rendre compte des représentations erronées qu'il possède. Cette mise en évidence doit être explicite, facilement compréhensible par l'étudiant lui-même et/ou facilement détectable par l'enseignant. Au terme de l'apprentissage au moyen de l'outil, les représentations erronées doivent être corrigées.

#### Aider à l'apprentissage

L'analyse de la solution produite par l'étudiant ne doit pas se limiter à lui indiquer s'il a réussi l'exercice ou non. Elle doit également lui donner des informations l'aidant à construire des modèles mentaux corrects de la machine notionnelle (contenu des variables utilisées, nombre d'itérations, etc.) et discuter si la solution proposée est optimale ou non. Si une erreur est produite, outre les représentations erronées, celle-ci doit être indiquée de la manière la plus explicite possible.

### 6.1.2 Exigences implicites

#### Utilisabilité

L'outil s'inscrit dans le cadre d'une séance de TP, sans ordinateur. L'idée est de promouvoir la manipulation des concepts de base et la construction de petits programmes sans se heurter à la complexité des environnements de développement. Son utilisation doit être la plus simple et la plus intuitive possible, afin de ne pas confronter l'étudiant à des difficultés et une charge cognitive inutile.

#### Robustesse

La robustesse d'un système est primordiale, surtout dans un contexte pédagogique. En effet, l'étudiant pourrait être découragé si son utilisation de l'outil

est parsemée de problèmes. Il faut aussi s'attendre à ce que l'étudiant, qui n'a jamais manipulé l'outil et possède une expérience très limitée en programmation, fasse des opérations non prévues initialement (volontairement ou non). De plus, l'approche essai-erreur de l'outil demande à ce que la rapidité d'analyse permette une expérimentation répétée dans un court laps de temps.

## 6.2 Un CI tangible et augmenté

Compte tenu des exigences précédemment définies, le choix de l'outil tangible s'est porté sur un CI tangible et augmenté. Pour rappel, les CI sont des questionnaires à choix multiples, traitant d'un concept spécifique et dont chaque option fausse est créée pour correspondre à une ou plusieurs représentations erronées de ce concept. À travers la détection des représentations erronées, les CI permettent d'évaluer la compréhension qu'à l'étudiant du concept en question. L'utilisation du tangible dans cette détection est un domaine encore inexploré.

Le CI est ici inspiré des travaux de Henry [29, 43, 42], ce qui garantit son adéquation au public visé.

Par rapport à un CI "classique", le CI tangible met l'étudiant en action : ce dernier ne doit plus seulement comprendre l'exécution du code qu'il a sous les yeux, il doit former une solution qu'il pense valide pour arriver à un résultat demandé. L'étudiant peut expérimenter une ou plusieurs solutions jusqu'à trouver la solution correcte. Le nombre de tentatives n'est pas limité.

Pour ce faire, l'outil se veut modulaire et prend la forme de blocs magnétiques que l'on assemble comme un puzzle. L'aspect "jeu" vise à stimuler l'intérêt de l'étudiant et sa motivation à expérimenter. L'étudiant dispose de blocs prédéfinis pour résoudre des exercices-défis, adaptables en termes de difficulté.

Une application va de pair avec les blocs et augmente les fonctionnalités de l'outil tangible. Les blocs contiennent chacune des instructions, correctes ou incorrectes, illustrant de mauvaises représentations. Une combinaison de blocs produit un code. L'application permet une analyse rapide des combinaisons formées par l'étudiant et identifie les représentations erronées présentes. Elle peut également évaluer si la solution proposée est optimale ou non. Enfin, elle affiche pour l'étudiant le résultat de l'exécution de la solution proposée.

Au-delà de la détection des représentations erronées de l'étudiant, l'application offre à ce dernier l'opportunité d'en déterminer l'origine, ce qui peut l'aider à les corriger. Il a été privilégié de ne discuter que les représentations erronées de l'étudiant et non l'ensemble des représentations erronées existantes, et ce afin d'éviter une certaine confusion [29, 43]. Lorsqu'une solution est incorrecte, l'étudiant peut lire les retours qui tentent de l'aider à comprendre la cause de cet

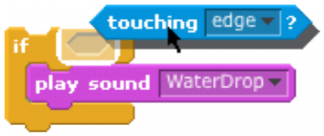



FIGURE 6.1 – Une structure conditionnelle dans Scratch [56]

échec. Après cela, il peut tenter une nouvelle approche.

## 6.3 La Conception

### 6.3.1 Les blocs

Les blocs trouvent leur inspiration dans le langage de programmation visuelle Scratch[56] [1] et dans de nombreux outils tangibles existants [58, 36, 35, 39].

Scratch permet à son utilisateur d'expérimenter la programmation via un assemblage, par glisser-déposer, de blocs graphiques. Les blocs ont une forme telle que des éléments incompatibles ne peuvent pas être combinés. Par exemple (Fig. 6.1), les conditions à insérer dans les structures conditionnelles ont les côtés pointus. Cette contrainte d'assemblage impose à l'utilisateur une certaine syntaxe et lui permet de comprendre de réaliser ce qu'il peut et ne peut pas faire. La lecture des programmes écrits via Scratch se fait de haut en bas, à la manière d'un langage conventionnel. L'outil TaBGO [58] offre une représentation tangible de Scratch, orientée pour les déficients visuels. L'outil Strawbies [39] permet d'assembler différents éléments représentant des instructions séquentielles permettant de faire se déplacer un personnage virtuel (Fig. 6.2). La syntaxe est complètement abstraite et permet à l'utilisateur de se concentrer sur le jeu. Tern [36] et Quetzal [35] offrent une approche similaire, mais avec des blocs spécifiques représentant la structure conditionnelle et la boucle. L'objectif est également de faire se déplacer un élément. Ces deux outils proposent une lecture du programme qui suit l'assemblage des blocs. De façon générale, ces outils mettent en jeu des éléments tangibles passifs possédant un marqueur détectable par un système externe.

Dans le cas de cette recherche, les blocs devraient idéalement :

— respecter une granularité (en termes de contenu visuel : un bloc affiche plusieurs instructions, une instruction, une portion d'instruction, etc.) suffisamment réfléchie ;

1. https://scratch.mit.edu/

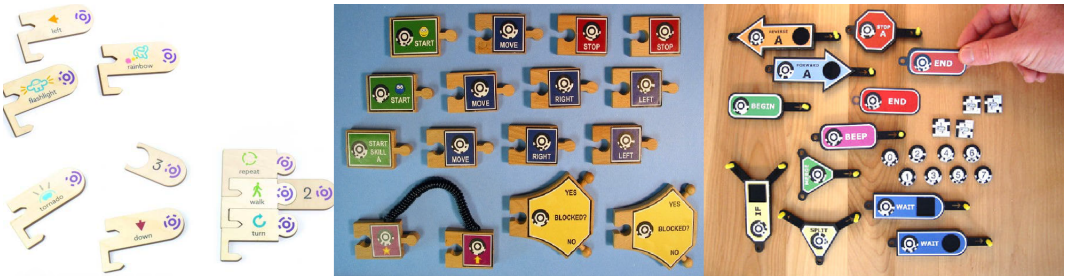



FIGURE 6.2 – De gauche à droite, des blocs dans Strawbies [39], Tern [36] et Quetzal [35].

— proposer une syntaxe cohérente au niveau des formes et couleurs, notamment rendre possible l'indentation exigée par le langage Python ;
— avoir une taille physique permettant une manipulation aisée ;
— disposer d'un moyen d'assemblage facile à faire et défaire ;
— être facilement identifiable par un système externe ;
— demander un coût de production raisonnable.

### Les choix techniques

Le prototype final présenté dans ce travail est un prototype de troisième génération. Tous les choix techniques posés sont le résultat d'un équilibre entre facilité de manipulation et de production des blocs.

Du point de vue de la granularité, il a été décidé qu'un bloc contiendrait une instruction complète et un marqueur d'identification. Aucune règle de syntaxe n'a été exprimée par les blocs eux-mêmes. De ce fait, tous les blocs sont identiques et polyvalents : leur contenu est imprimé sur une étiquette de papier interchangeable (Fig. 6.3). Un exercice proposé à l'utilisateur est un ensemble prédéfini d'étiquettes à poser sur des blocs.

La taille d'un bloc a été fixée à 7,4 cm sur 16 cm, assurant une bonne lisibilité du contenu.

Deux formes d'assemblage ont été évaluées avec les experts : l'assemblage puzzle et l'assemblage magnétique (Fig. 6.4). C'est ce dernier qui a été choisi pour la facilité du "faire et défaire". Chaque bloc possède six aimants permettant de s'attacher aux autres et rendant possible l'indentation.

Les formes des bords gauches et droits sont arrondies (Fig. 6.5), afin de rendre les blocs plus agréables à manipuler.

Le matériau qui compose les blocs a été sélectionné après plusieurs essais (Annexe A). Il permet une production plus aisée et efficace : un bloc est imprimé

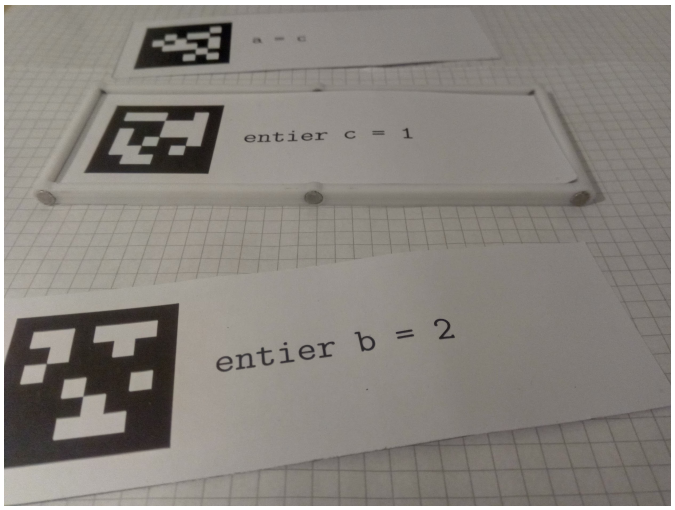



FIGURE 6.3 – Un bloc avec deux cartons supplémentaires.

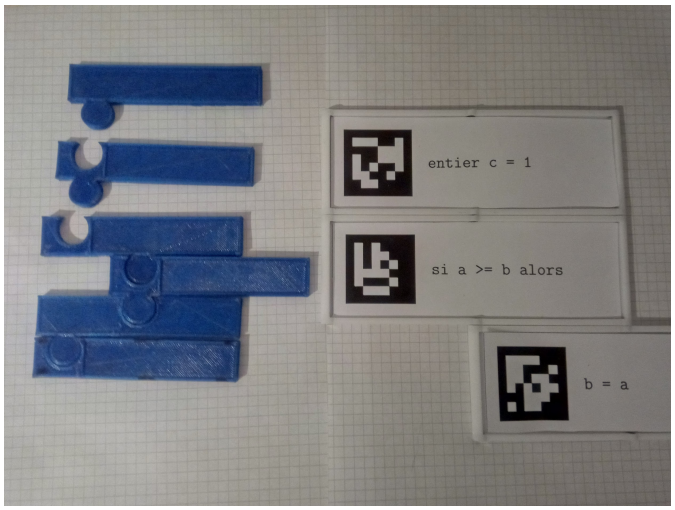



FIGURE 6.4 – Les deux types d'assemblages. À droite, l'assemblage magnétique et à gauche l'assemblage puzzle.

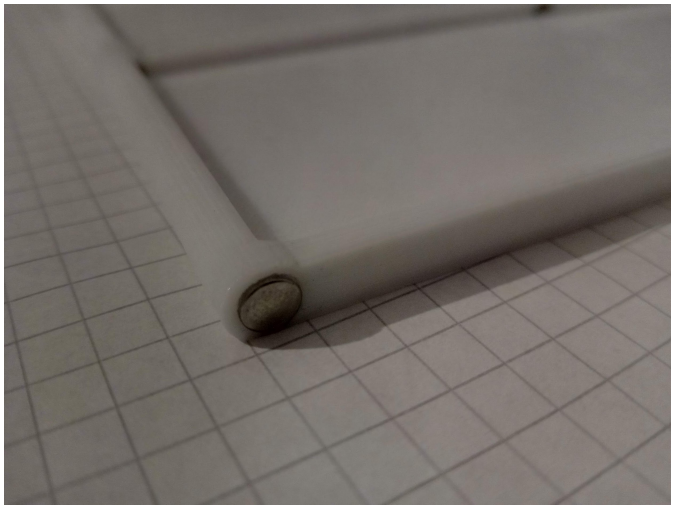


FIGURE 6.5 – Les bords arrondis des blocs recouvrant l'aimant.

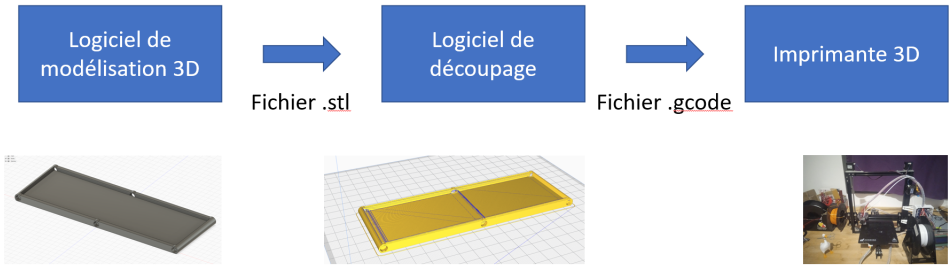



FIGURE 6.6 – De la modélisation à la production d'un bloc.

en 4h. Au total, quarante blocs seront produits. Les étiquettes sont en papier haute densité (160g/m$^2$)pour une meilleure résistance à la manipulation.

### Le cycle de création

La réalisation d'un prototype de matériel tangible via impression 3D passe par plusieurs étapes (Fig. 6.6). Tout d'abord, le bloc est dessiné dans un logiciel de modélisation tel que FreeCad [2] ou Fusion360 [3]. Une fois finalisé, ce modèle est exporté dans un fichier de stéréolithographie, au format STL. Ce fichier contient alors la représentation de la surface du modèle que l'on souhaite concrétiser. Il

2. https://www.freecadweb.org/
3. https://www.autodesk.com/products/fusion-360/overview

est ensuite importé dans un logiciel de découpe (un slicer, en anglais) comme Cura [4] pour transformer l'objet désiré en G-code, langage d'instruction-machine utilisé notamment par les imprimantes 3D. Le fichier G-code produit par le slicer contient toutes les opérations à effectuer pour construire le modèle par dépôt de couches successives de matière. À noter que le G-code généré est dépendant de la machine et du matériel utilisé : il contient tous les paramètres techniques relatifs à l'impression comme la taille de la buse, le taux de remplissage du modèle, la taille du plateau d'impression, la température de la buse et du lit chauffant, etc. Une fois le G-code généré, celui-ci est importé dans l'imprimante 3D qui l'exécute. Quand l'impression est terminée, le bloc est détaché de la surface d'impression et nettoyé de ses éventuelles imperfections.

### 6.3.2 L'application

Les blocs présentés précédemment sont des blocs sans aucun élément électronique. Dès lors, l'analyse et l'exécution du code reposent sur un système externe qui devrait idéalement :

- être capable de reconnaître les blocs, leur position dans l'espace et l'ordre de leur placement avec suffisamment de précision que pour éviter de mauvaises détections ;
- être en mesure d'exécuter le code présent sur chaque bloc individuellement et sur une combinaison de blocs ;
- fournir un retour à l'étudiant après exécution d'un code. Si le code n'est pas exécutable à cause d'un mauvais agencement (indentation) ou d'un assemblage qui n'a pas de sens (variable utilisée non définie), le système doit être en mesure d'exprimer clairement le problème.
- être capable de reconnaître certaines combinaisons de blocs comme des représentations erronées ;
- offrir une explication à l'étudiant sur l'origine de sa représentation erronée, pour l'aider à la surmonter ;
- présenter une aide contextuelle à l'étudiant afin de l'aider à comprendre ce qui a posé problème dans sa solution ;
- disposer d'une interface intuitive et robuste ;
- offrir une robustesse logicielle ;
- offrir une flexibilité d'édition permettant notamment l'ajout ultérieur d'exercices.
- enregistrer les statistiques des soumissions des étudiants ;
- être un système mobile (tablette ou smartphone) pour pouvoir être utilisé par les étudiants en séance de TP sans ordinateur ;

4. https://ultimaker.com/fr/software/ultimaker-cura

### Les choix techniques

Ces différents points ont porté le choix sur une application conçue pour fonctionner sur un appareil Android (7.0 ou supérieur).

Outre l'aspect mobile et « sans câble » qui facilite son utilisation lors d'une séance TP sans ordinateur, tablette et smartphone disposent généralement d'une caméra intégrée, utile pour les fonctionnalités de reconnaissance et de la capacité de calcul nécessaire. L'appareil doit disposer d'un espace de stockage de 30 Mo au minimum. Il est important que l'éclairage soit suffisant pour distinguer les marqueurs présents sur les blocs, et que les blocs non utilisés soient éloignés pour ne pas interférer avec la détection des blocs désirés.

L'application intègre une base de données SQLite (dont le schéma est détaillé dans l'annexe B) gérée avec ORMLite [5]. Les statistiques des différentes analyses effectuées par les étudiants y sont stockées.

Au niveau des interfaces, l'application, dans sa dernière version, comporte trois écrans. Le premier est l'écran d'accueil, qui explique le fonctionnement de l'application à l'étudiant. Le second est l'écran d'analyse, dans lequel l'image captée par la caméra ainsi que le nombre de marqueurs détectés sont affichés. Le dernier écran comprend le résultat de l'exercice généré en fonction du programme soumis par l'étudiant. Au préalable, l'écran d'accueil faisait place à l'écran de sélection des exercices, mais après les premiers essais sur personne (voir le chapitre 7), il arrivait que le mauvais exercice soit sélectionné par l'utilisateur de l'application. La détection de l'exercice est depuis lors faite de manière automatique sur base des marqueurs détectés et une erreur est renvoyée si les marqueurs de plusieurs exercices sont détectés.

L'application est en mesure d'interpréter le code présent sur les étiquettes pour donner le résultat de son exécution. En effet, dans la base de données de l'application, les exercices sont représentés comme des ensembles de marqueurs. Un marqueur contient un identifiant, un label (le code écrit sur l'étiquette papier) et du code interne. Ce code interne est l'équivalent du code représenté sur l'étiquette, mais écrit en Javascript. Un interpréteur va alors exécuter dans l'ordre de détection des blocs le code Javascript assigné à chaque marqueur.

Si l'exécution mène à une erreur, l'interprétation du code est arrêtée et l'erreur est communiquée à l'utilisateur. Celle-ci est reformulée pour en faciliter la compréhension. Si l'exécution du code réussit, les objectifs de l'exercice sont vérifiés un par un. Ces objectifs sont traduits en code Javascript. Par exemple, si un objectif est de placer une certaine valeur dans une variable, la valeur de cette variable va être extraite de l'état final du programme et être comparée à la valeur attendue. Si les deux valeurs sont les mêmes, l'objectif est atteint. Sinon, un

5. https://ormlite.com/

message informe l'utilisateur de son échec.

Cette méthode d'interprétation permet de traduire les exercices dans plusieurs langages, mais également d'avoir une exécution réelle du programme effectué. Cette exécution subit cependant les limitations du Javascript. La réalisation d'un langage spécifique à l'application pour l'exécution du code a été envisagée, mais ANTLR [77] n'est pas compatible avec la machine virtuelle Java exécutée par Android. Les boucles et structures conditionnelles sont un cas particulier, car elles doivent être exécutées d'un seul bloc. Elles ne sont donc exécutées que lorsque tous les blocs de sa structure ont été reconnus.

L'application est capable de détecter les représentations erronées à travers les assemblages incorrects de blocs. Cette détection a demandé la mise en place d'un mécanisme particulier : une fois le code interprété et les objectifs vérifiés (qu'ils soient atteints ou non), les représentations erronées le sont également. Une représentation erronée est définie ici comme un pattern spécifique dans la disposition de blocs, traduisant la mauvaise compréhension d'un concept. Quinze patterns différents ont été créés dans le cadre de ce travail. L'utilisation d'un bloc contenant une instruction incorrecte est un de ces patterns. Dans ce cas, la représentation erronée est déclenchée lorsque le marqueur est détecté . Un autre exemple de pattern est l'absence d'une instruction qui doit se trouver impérativement avant une autre. Par exemple, si un utilisateur veut accéder au contenu d'une variable et que celle-ci n'a été ni déclarée, ni initialisée. La représentation erronée associée est alors déclenchée parce le marqueur attendu n'est pas détecté. Les patterns sont donc fortement liés aux exercices. Ces patterns ne sont pas uniquement utilisés pour les représentations erronées, ils peuvent aussi prévenir l'utilisateur qu'il n'est pas sur la bonne voie. Ainsi, si un énoncé demande l'utilisation d'une structure conditionnelle et qu'aucune n'est présente dans le programme proposé, un message s'affiche pour souligner le problème.

**AprilTag** Le framework AprilTag [6][71] a été choisi pour la détection des blocs dans l'espace. Prévu à la base pour la réalité augmentée et la robotique, ce framework garantit plus la souplesse dans la manipulation de la caméra et offre davantage de marqueurs que reacTIVision (587 pour la famille 36h11). De plus, ces marqueurs ressemblent à des QR-Codes (Fig. 6.7) dont l'aspect visuel ne perturbe plus personne. Parce que le support Java n'est pas disponible dans AprilTag, les appels doivent être réalisés via JNI (Java Native Interface). AprilTag offre les coordonnées des marqueurs détectés, ce qui permet de déterminer la position relative des différents blocs de code et de calculer une éventuelle indentation. Le calcul de l'indentation entre deux marqueurs se base d'ailleurs sur les coordonnées

6. https://april.eecs.umich.edu/software/apriltag

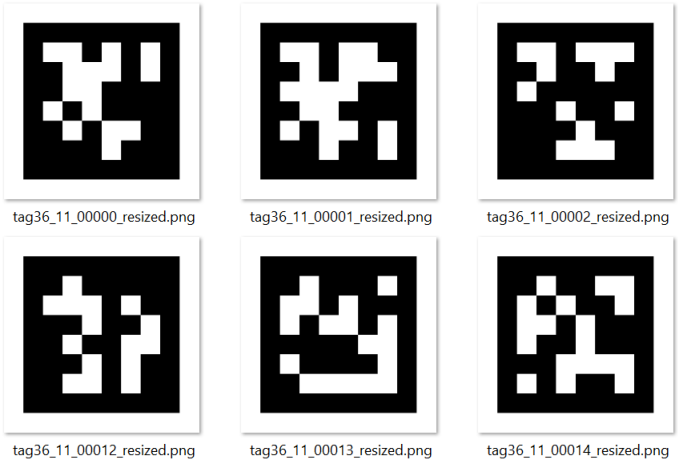



FIGURE 6.7 – Six marqueurs fiduciaires de AprilTag, parmi les 587 de la famille 36h11. À noter que le contour extérieur blanc fait partie du marqueur et doit être présent.

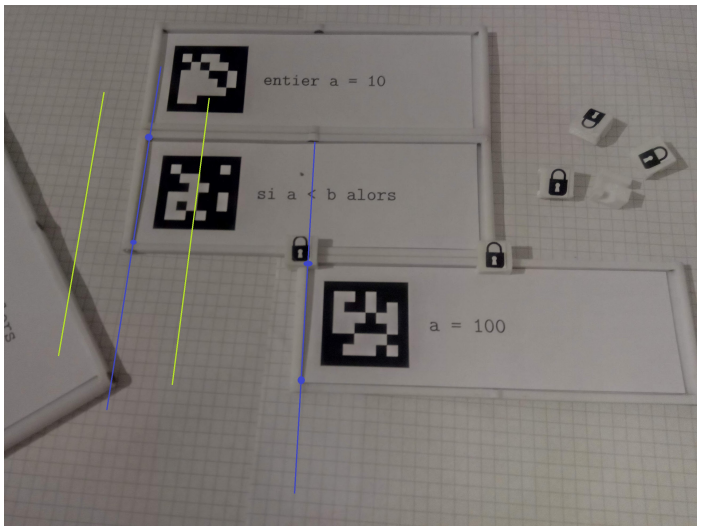



FIGURE 6.8 – Calcul de l'indentation. La zone entre les deux droites vertes est la zone dans laquelle la droite bleue du second marqueur doit se situer pour que deux marqueurs soient considérés comme correctement alignés.

de deux points de chaque marqueur (Fig. 6.8).

Dans le but de collecter des données durant les évaluations permettant une analyse statistique, un mécanisme de sauvegarde des assemblages (de blocs) formés et des messages affichés a été mis en place. Inspiré de la solution mise en place par Sorva [101], le mécanisme développé ici se doit de fonctionner en l'absence d'une connexion à Internet. Parce qu'un stockage en local comporte un risque de perte de données en cas de problème avec l'appareil Android, le choix s'est porté sur une solution hybride. Ainsi, si l'appareil est connecté à Internet, les données statistiques récoltées sont envoyées à un serveur. Si la connexion n'est pas disponible, les données sont conservées directement sur l'appareil. Elles seront envoyées ultérieurement, au prochain lancement de l'application ou à la prochaine soumission d'un assemblage de blocs. En cas de réussite de l'envoi, la base de données locale est vidée.

Les données statistiques récoltées sont :

— la liste des marqueurs détectés, avec les coordonnées du centre de la détection ;
— tous les messages affichés sur l'écran des résultats ;
— la date et l'heure de l'analyse, ainsi que la date et l'heure de l'enregistrement de la donnée sur le serveur ;
— la version de la base de données interne à l'application.

### 6.3.3 Le serveur

Comme expliqué précédemment, l'application envoie les données statistiques d'utilisation à un serveur. Ce serveur est accessible depuis Internet via une connexion HTTPS et expose les API nécessaires pour que l'application puisse envoyer ses données (en JSON). L'application est alors identifiée au moyen d'un token d'api, qui l'autorise à publier ses résultats. Le serveur d'API fonctionne en PHP et est basé sur le framework Laravel [72]. Il est lié à une base de données MySQL dans laquelle les données statistiques sont stockées. Ces données peuvent alors être accédées soit via l'API (moyennant une authentification), soit via des requêtes SQL directement en base de données. Le schéma de la base de données dans lequel les données statistiques sont stockées est décrit dans l'annexe C.

### 6.3.4 Les exercices

Quatre exercices ont été développés dans le cadre de ce mémoire pour expérimenter plusieurs concepts de base en programmation. Chacun des exercices est traduit dans deux langages : en pseudo-code et en langage naturel. Le pseudo-code est un langage typé en français qui a une syntaxe très proche d'un

vrai langage de programmation. Le langage naturel est un langage explicite où chaque opération effectuée est décrite textuellement. Les labels en langage naturel contiennent donc plus de texte que les labels équivalents en pseudo-code. Les étiquettes des exercices dans les deux langages sont disponibles dans les annexes D et E.

**1** Le premier exercice est le cas d'école classique de la permutation du contenu de deux variables. Il illustre la représentation erronée de l'inversion du sens de l'assignation. Il introduit le concept de variable et l'assignation, abordés par le cours INFOB131. Cet exercice est composé de douze blocs, dont six sont nécessaires pour le résoudre. Il existe plusieurs solutions valides. La résolution de l'exercice nécessite l'utilisation d'une variable intermédiaire. Aucune indentation ne doit être effectuée dans cet exercice. Trois étiquettes sont incorrectes et présentent des déclarations de variable écrites dans le mauvais sens.

**2** Le second exercice évalue les opérateurs de comparaison. L'étudiant est amené à sélectionner le bon opérateur, la bonne condition dans la structure conditionnelle pour réussir l'exercice, en plus d'assembler un programme correct. L'énoncé demande explicitement l'utilisation d'un opérateur spécifique, que l'étudiant doit reconnaître et utiliser. Il permet de confirmer que l'étudiant a bien acquis les opérateurs de comparaison avant d'effectuer l'exercice de la boucle. Cet exercice demande l'utilisation de l'indentation des blocs et introduit le concept de structure conditionnelle. Il est composé de huit blocs, dont quatre sont nécessaires pour le résoudre. Il n'a qu'une seule solution valide. Quatre étiquettes contiennent un opérateur de comparaison non adapté, mais syntaxiquement correct.

**3** Le troisième exercice illustre la difficulté qu'ont les étudiants à gérer le cas « else » lorsque la condition de la structure conditionnelle est fausse. Dans cet exercice, l'étudiant est obligé, en raison de blocs verrouillés entre eux, de sélectionner une condition fausse et de prévoir le cas « else » pour arriver à résoudre le problème. Cet exercice exige l'indentation du code et introduit le concept de variable booléenne. Il est composé de dix blocs, dont cinq sont nécessaires pour le résoudre. Il existe plusieurs solutions valides. Trois structures conditionnelles sont proposées, dont deux dont l'évaluation sera toujours fausse et une dont l'évaluation est vraie, mais ne donne pas le bon résultat. Une structure « else » est aussi disponible et offre le bon résultat, mais elle doit être combinée à une des conditions fausses.

**4** Le quatrième exercice aborde le concept de la boucle. L'étudiant doit alors maîtriser la structure de la boucle et comprendre quel opérateur il doit choisir pour

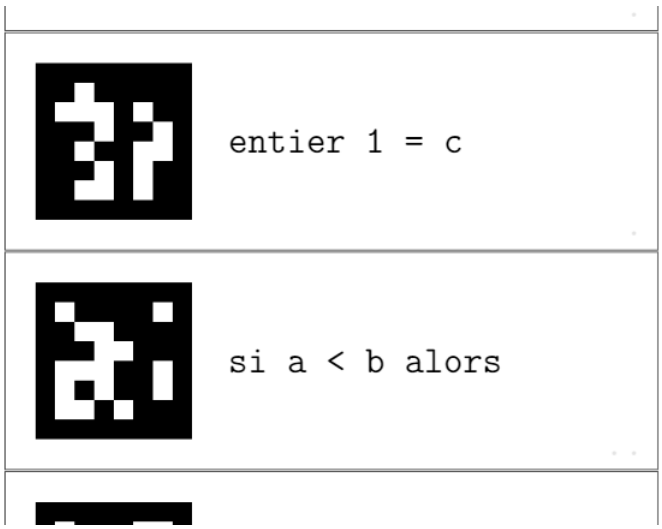



FIGURE 6.9 – Deux étiquettes, de deux exercices différents, générés par LaTeX. On peut apercevoir l'identifiant visuel de l'exercice en bas à droite dans un gris très clair.

que le bon nombre d'itérations soit exécuté. Il évalue la représentation erronée de l'arrêt de la boucle lorsque la condition devient fausse. Cet exercice impose une indentation du code. Cet exercice est composé de sept blocs, dont trois sont nécessaires pour le résoudre. Il n'a qu'une seule solution valide. Quatre blocs contiennent une condition de boucle non adaptée, mais syntaxiquement correcte.

### 6.3.5 Les étiquettes

Pour faciliter la génération d'étiquettes, un canevas LaTeX a été créé dans Overleaf [7], imposant la taille des étiquettes et le nombre de lignes de code (entre une et quatre). Pour des raisons de cohérence visuelle, toutes les étiquettes d'un exercice sont générées avec une même taille de police de caractères. Le canevas LaTeX prend en entrée un fichier .csv contenant l'identifiant du marqueur fiduciaire à assigner au code, le texte du label et l'identifiant visuel de l'exercice (Fig. 6.9). Cet identifiant permet de regrouper les étiquettes par exercice. La discrétion de cet identifiant évite que l'étudiant ne le remarque.

7. https://www.overleaf.com/read/drwntznhwbjk

## 6.4 Les limitations

Le prototype final (Fig. 6.10) propose une solution fonctionnelle flexible qui permet d'ajouter facilement des exercices et des représentations erronées. Les exigences explicites sont respectées dans les grandes lignes.Certains aspects pourraient toutefois être enrichis.

Ainsi l'interface de l'application pourrait encore être améliorée, notamment via l'ajout d'un nouvel écran, et une meilleure ergonomie des trois écrans actuels. En effet, les retours à l'étudiant ne sont pas aussi complets que souhaité. L'étudiant peut voir la valeur finale des variables concernées par un objectif, mais ne peut pas voir l'évolution de la valeur de ces variables au cours de l'exécution du programme. Un écran supplémentaire pourrait être ajouté à cette fin dans une prochaine version. Celui-ci reprendra une vision du code exécuté et la valeur de chaque variable après exécution des blocs, afin que l'étudiant comprenne mieux l'impact du code qu'il exécute.

L'application n'est pas (encore) disponible sur la bibliothèque officielle des applications Android et demande donc un appareil configuré en mode développeur pour pouvoir installer le distribuable. L'installation des mises à jour demande également une installation manuelle.

Le serveur est actuellement hébergé sur une infrastructure non redondante et peut cesser de fonctionner en cas de panne d'électricité ou de connexion à Internet. Une destruction du serveur entraînerait alors une perte irrécupérable des données statistiques collectées.

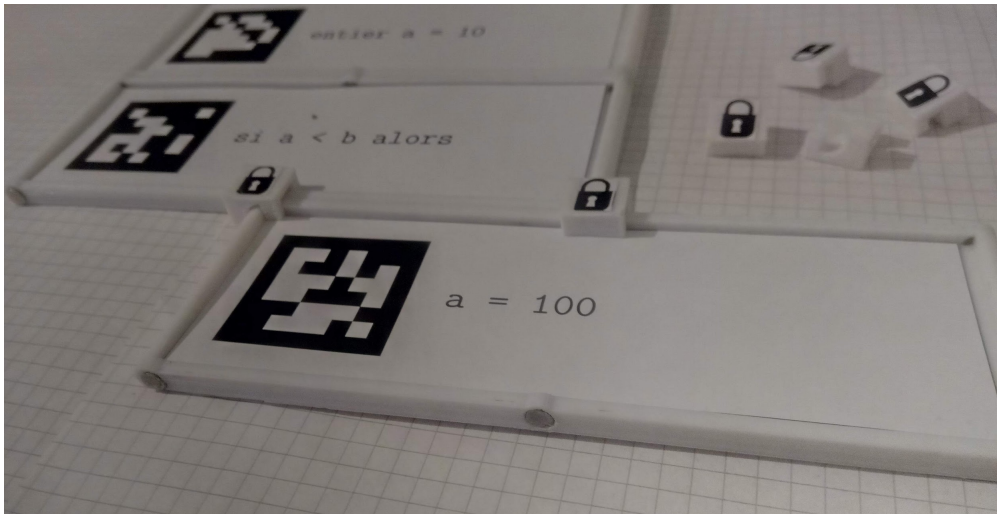



FIGURE 6.10 – Les blocs du prototype final, reproduisant une indentation. Le langage utilisé est le pseudo-code. Des verrous (avec le pictogramme de cadenas) sont placés sur la condition.

# Chapitre 7

# Évaluations

Deux types d'évaluations ont été réalisées : avec des enseignants-assistants susceptibles d'utiliser l'outil tangible dans leurs contextes de travail et avec des étudiants novices en programmation.

## 7.1 Évaluations auprès des enseignants

Huit enseignants ont pris part à cette phase d'évaluation. Tous n'ont pas testé l'ensemble des exercices disponibles. Ainsi, le premier et le quatrième exercice ont été testés par l'ensemble du groupe. Seules six personnes ont testé le troisième exercice. Le deuxième exercice n'a pas été testé. Les descriptions complètes des exercices sont disponibles au point 6.3.4. Lors de ces évaluations, plusieurs remarques et observations ont pu être faites.Pour rappel, ces évaluations ont donné lieu à des modifications au niveau du prototype.

Concernant les modalités d'évaluation, il n'avait pas été prévu une impression des énoncés des différents exercices, ceux-ci étant disponibles via l'interface. Ces énoncés ont été réclamés par les participants. Plusieurs participants ont exprimé une difficulté à remplir le questionnaire UEQ, trouvant certaines questions ambiguës.

### 7.1.1 Retours sur le CI tangible

Concernant le matériel tangible, deux personnes (soit 25% des participants) ont éprouvé des difficultés à gérer l'ensemble des blocs d'un exercice. Ils ont signalé avoir difficilement une bonne vision des éléments mis à leur disposition et ont globalement mis plus de temps pour identifier les blocs dont ils avaient besoin pour résoudre un exercice.

Le pseudo-code a été considéré comme déstabilisant par deux enseignants (25%), habitués à manipuler Python. Deux autres ont été étonnés du typage explicite des variables ( par exemple, « *entier* a = 3 »). Six personnes (75%) ont oublié d'inclure le bloc « fin si » lors de l'assemblage d'une structure conditionnelle. L'erreur n'a cependant jamais été répétée dans un second exercice.

Un expert a essayé de détacher deux blocs verrouillés afin de récupérer l'élément des deux blocs contraints qui l'intéressait.

Enfin, l'exercice numéro trois a été considéré comme inadapté aux novices par la totalité des évaluateurs de cet exercice.

Concernant l'interface de l'application (exécutée sur une tablette Android), un certain nombre de remarques touchaient les retours aux étudiants.

Plusieurs enseignants ont fait remarquer que les problèmes d'indentation étaient signalés par un avertissement et non par une erreur, comme cela devait être le cas.

La plupart des illustrations dans les retours aux utilisateurs utilisaient la couleur rouge. Celle-ci était jugée trop agressive pour informer un étudiant se trouvant déjà en situation d'échec pour l'exercice. La disposition des éléments textuels et de l'image explicative n'était pas optimale. Le regard s'attardait, selon les participants, uniquement sur l'image explicative (en bas de page) et évitait le texte, écrit trop petit (Fig. 7.1).

Outre les problèmes liés aux retours, des observations ont pu mettre en évidence des lacunes ergonomiques. Ainsi, un expert a sélectionné par inadvertance un mauvais exercice, engendrant une incompréhension totale devant les erreurs qui lui étaient présentées.

La manipulation de la tablette était rendue difficile par le mauvais positionnement du bouton permettant de lancer le scan.

La majorité des participants n'a pas vu le compteur de blocs détectés (présent alors sur le bouton Analyse). Lorsqu'il a s'agit de relancer une analyse, trois experts ont été bloqués, ne pensant pas à utiliser le bouton retour sur la tablette pour retourner sur l'écran précédent.

Après une certaine durée d'utilisation, l'application devenait lente et devait même être redémarrée.

### 7.1.2 Résultats du questionnaire UEQ

Les résultats collectés au moyen du questionnaire UEQ (disponible à l'annexe G) sont globalement positifs. Les valeurs pour chaque aspect mesuré (attraction, compréhensibilité, efficacité, contrôlabilité, stimulation et originalité) sont représentées sur une échelle de -1 à 2,5 (Fig. 7.2) et comparées aux valeurs dont dispose l'outil d'analyse fourni avec le questionnaire « UEQ_Data_Analysis_Tool_Version7 ». Cet outil dispose des valeurs récoltées


FIGURE 7.1 – Les proportions des éléments sur tablette. On remarque l'image trop grande, le texte trop petit et la couleur rouge du texte. Le bouton retour est celui natif à Android, à savoir le triangle à gauche en bas.

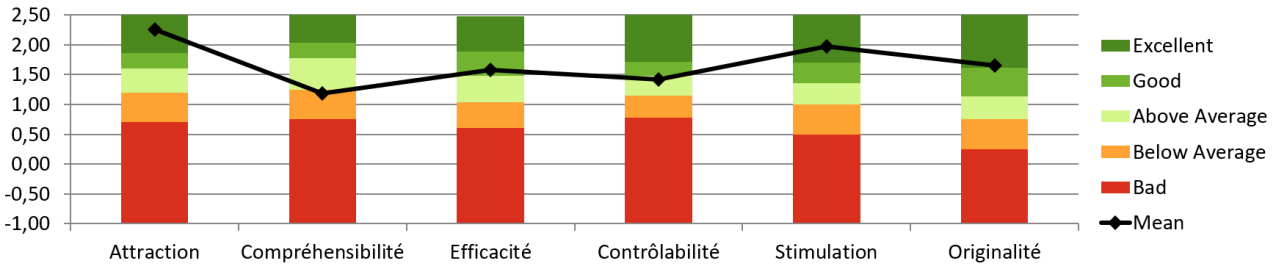



FIGURE 7.2 – Représentation visuelle des résultats de l'UEQ.

via 452 études évaluant une large gamme de produits, fournissant ainsi des moyennes pour chacun des aspects mesurés par le questionnaire[93]. Il est dès lors possible de comparer ces moyennes aux valeurs obtenues durant les évaluations auprès des enseignants, permettant de situer le CI tangible par rapport à chacun des aspects du questionnaire.

Ainsi, l'attraction obtient un excellent score de 2,25 (sur 2,5) et se place bien au-dessus de la moyenne (1,19). La compréhensibilité obtient pour sa part un score de 1,19, très légèrement inférieur à la moyenne (1,25). L'efficacité affiche un bon score de 1,58, supérieur à la moyenne (1,04). La contrôlabilité pointe au score de 1,42, à comparer à la moyenne de 1,15. La stimulation obtient, elle, un score de 1,97, ce qui la place bien au-dessus de la moyenne (1,0). Enfin, l'originalité mesure un score de 1,66, également supérieur à la moyenne (0,75).

Certaines questions du test UEQ ont été jugées ambiguës pour plusieurs testeurs. Ainsi, les questions 11, 20 et 23 n'ont été remplies que par sept des huit enseignants. La question 8 qui demande si l'outil est prévisible ou imprévisible présente cinq réponses neutres.

Pour la plupart des rubriques, les valeurs obtenues sont au-dessus de la moyenne, affichant des scores bons voir excellents. Seule la compréhensibilité de l'outil obtient une note plus basse. Compte tenu des retours faits durant les évaluations auprès des enseignants, cette compréhensibilité pourrait être améliorée par une disposition plus claire des éléments affichés à l'écran et une amélioration du pseudo-code. Le nombre d'éléments constituant le CI tangible et la difficulté ressentie par certains enseignants d'en avoir un aperçu global peut également avoir impacté cette métrique.

## 7.2 Amélioration du CI tangible

Les différents retours ont eu pour conséquence diverses modifications détaillées ci-dessous, que ce soit au niveau de l'interface de l'application ou au niveau du fonctionnement de l'outil. Le côté déstabilisant du pseudo-code a inspiré la création d'un nouveau langage. Il a été voulu le plus proche possible de la langue française et le plus explicite possible. Il est appelé « langage naturel » dans la suite de ce mémoire. L'ensemble des étiquettes produites jusqu'alors ont été traduites dans ce langage (voir annexe E) afin de le proposer aux étudiants, dans le cadre d'une nouvelle phase d'évaluations.

La tangibilisation des structures conditionnelles et des boucles jusqu'alors imposant d'être terminées avec un bloc spécifique (« fin si » ou « fin tant que »), a été revue. C'est désormais à travers l'indentation que le système détermine le début et la fin du code impacté par une structure conditionnelle ou une boucle, à l'image du langage python.

Au départ, les verrous de blocs ne contenaient pas de pictogramme de cadenas, ce qui pouvait porter à confusion et laisser entendre qu'il était autorisé de les enlever. Un message d'erreur informait cependant l'utilisateur qu'il avait enfreint un verrou si les blocs contraints n'étaient pas détectés consécutivement. Le pictogramme de cadenas a donc été ajouté sur les verrous, et le message d'erreur est resté dans l'hypothèse ou un utilisateur tenterait à nouveau de les enlever.

Au niveau ergonomique, le bouton permettant d'analyser un code produit a été supprimé. L'analyse se fait simplement via une pression sur l'écran. Le compteur de blocs détectés a été rendu plus visible, s'affichant sur l'entièreté de l'écran. Celui-ci est translucide afin que l'utilisateur puisse toujours voir les éléments qui se trouvent derrière lui (Fig. 7.3).

Les erreurs d'indentation affichées comme des avertissements ont été modifiées pour qu'elles apparaissent comme des erreurs, d'autant que l'indentation a été rendue obligatoire par les changements opérés concernant les structures conditionnelles et les boucles.

La couleur rouge a été évitée dans l'ensemble des messages informatifs à destination des utilisateurs.

Les écrans étaient, à la base, optimisés pour une utilisation avec smartphone, ce qui provoquait des incohérences de taille entre les éléments dans l'affichage sur tablette. Tout a été repensé pour obtenir un affichage dynamique et cohérent, indépendamment de la taille ou du format de l'écran.

Le retour à l'écran précédent était, dans la version de l'application testée, le bouton de retour standard de Android. Il était cependant probablement inconnu des utilisateurs d'un autre OS mobile. L'ajout d'un bouton retour et d'une explication de son utilisation dans l'écran d'accueil a été effectué (Fig. 7.3).

L'application détermine d'elle-même, à partir des marqueurs fiduciaires (voir annexe B), l'exercice en cours. Cette modification facilite en outre l'utilisation de l'outil, permettant de passer d'un exercice à l'autre sans revenir sur un écran de sélection d'exercices.

Le problème de lenteur de l'application lors d'une utilisation prolongée a été corrigé. Il s'agissait en fait d'un « memory leak ». Celui-ci a été détecté à l'aide de l'outil LeakCanary [1] et de la visualisation de la mémoire proposée par AndroidStudio [2].

L'exercice 3 considéré comme trop difficile sera réservé à une validation des représentations erronées plus tardive dans le cursus de l'étudiant. Il n'a donc été ni modifié ni supprimé de l'application. Pour rappel, cet exercice demande l'utilisation d'une condition fausse et d'un « else » pour parvenir à l'objectif attendu.

## 7.3 Évaluations auprès des étudiants

### 7.3.1 Retours sur les langages

Les cinq étudiants d'ingénierie de gestion (IngMI) et un étudiant d'informatique (Info) ont expérimenté le langage naturel en premier. Parmi ces six étudiants (66% des testeurs), deux ne sont pas parvenus à résoudre le problème en langage naturel (un étudiant en IngMI et l'étudiant en Info). Parmi les deux étudiants ayant éprouvé des difficultés avec le langage naturel, l'un d'entre eux en a également rencontré pour résoudre l'exercice en pseudo-code. Les étudiants ayant commencé avec le langage naturel ont, pour la moitié d'entre eux, confondu le sens de l'assignation exprimée par les textes, inversant celui-ci.

La majorité des étudiants (89%) a trouvé que le langage naturel était plus difficile que le langage pseudo-code. Selon eux, la difficulté provient en partie causée de la longueur des textes qui complexifie la reconnaissance d'un bloc dans une vue d'ensemble. Un étudiant d'Info a précisé qu'il craignait de voir des pièges dans les textes du langage naturel, où un mot pourrait avoir été inversé ou remplacé par un autre. Ce même étudiant a également fait remarquer que le mot « variable » était indiqué sur les blocs en langage naturel, et qu'il pouvait poser problème aux personnes qui ne le connaissent pas. Cette difficulté plus élevée avec le langage naturel peut aussi être expliquée par le fait que le pseudo-code est assez proche du langage Python qu'ils étudient en cours. Cependant, un étudiant IngMI a mentionné que le langage naturel offrait une meilleure compréhension des impacts d'une instruction, car celui-ci est plus explicite.

1. https://square.github.io/leakcanary/
2. https://developer.android.com/studio/profile/memory-profiler

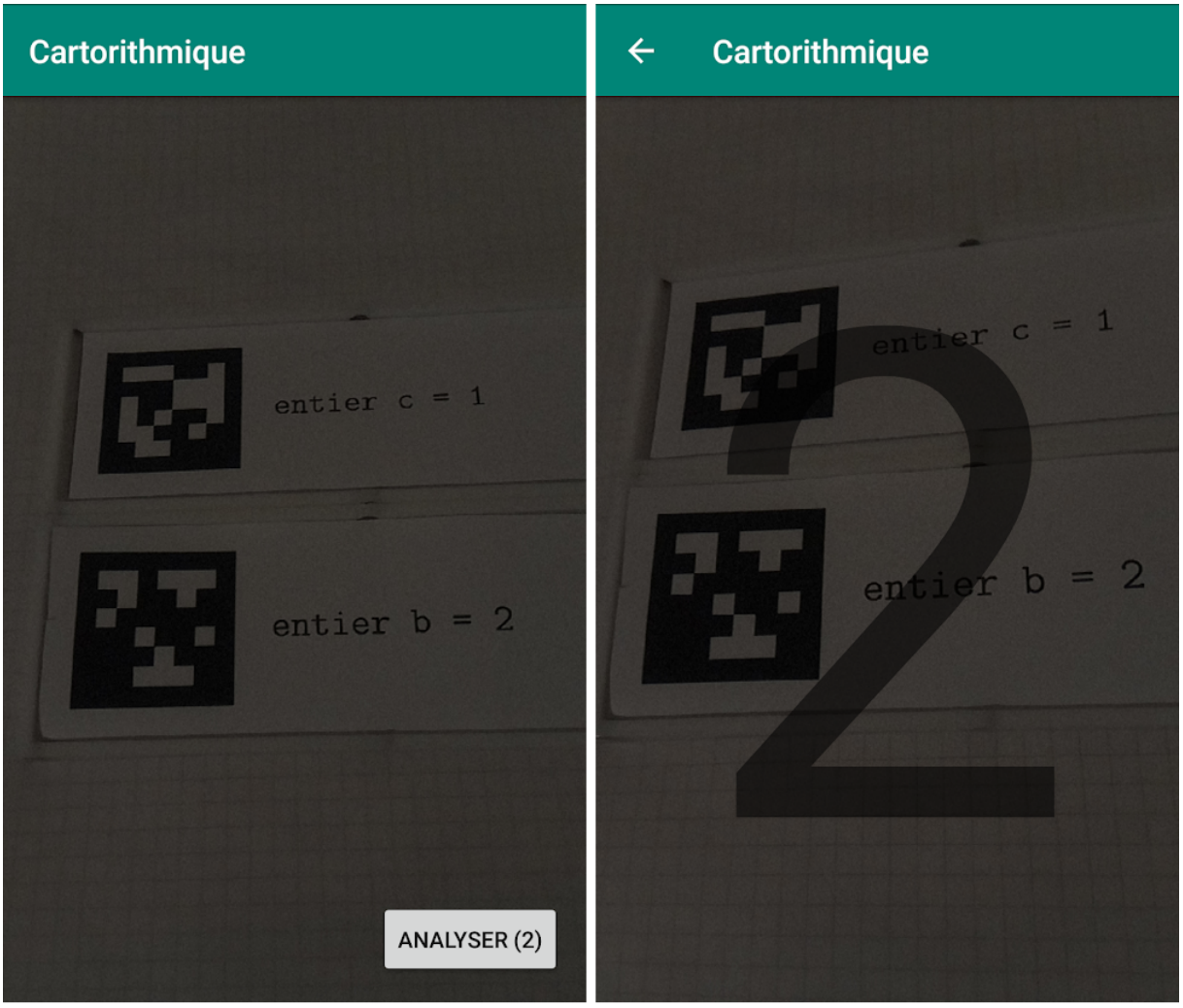



FIGURE 7.3 – La mise à jour de l'écran de scan des blocs. À gauche, l'ancienne version avec le bouton permettant de lancer l'analyse et le compteur entre parenthèses. À droite, la nouvelle version activée par une pression sur l'écran. La flèche en haut à gauche de l'écran de droite permet le retour à l'écran précédent.

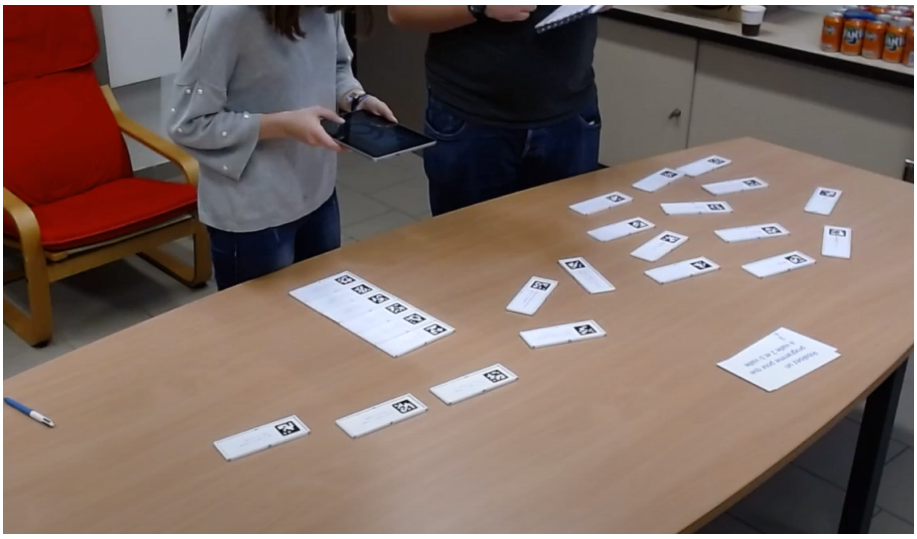


FIGURE 7.4 – À gauche, une étudiante en train de scanner à l'aide de la tablette sa solution construite en langage naturel. À droite, le chercheur en train de prendre des notes. L'exercice situé à la droite de l'image (pseudo-code) n'est pas encore résolu. L'énoncé, commun aux deux exercices, est présent en bas à la droite de l'image. Le dessus de l'image a été découpé par souci d'anonymat.

Plusieurs étudiants ont effectué un tri parmi les blocs, en les regroupant par catégories avant de résoudre l'exercice. Un étudiant a d'ailleurs proposé une coloration des blocs en fonction du type d'instruction exécuté par celui-ci. Cela pourrait ainsi améliorer la vision dans l'ensemble des différents types de blocs (assignation, déclaration, etc.).

Trois étudiants, tous en Informatique, n'ont pas compris et ont donc ignoré le typage présent sur les blocs de pseudo-code. Ils se sont alors contentés de les ignorer pour se concentrer sur les éléments des blocs qu'ils avaient déjà appris en cours. Ils ont cependant précisé que s'ils avaient manipulé l'outil plus tôt dans le cursus, ils auraient probablement eu une réaction différente.

Lorsqu'un étudiant commençait l'exercice de l'autre langage, celui-ci pouvait s'aider de sa solution précédemment établie. Seuls deux étudiants ont remarqué que les QR codes présents sur les blocs étaient différents d'un langage à l'autre, pour un même exercice. Ceux-ci tentaient de se servir de cet élément pour s'aider dans la résolution d'un exercice.

Un seul étudiant (IngMI) a réussi à résoudre l'exercice dans les deux langages lors de son premier scan avec la tablette. À l'inverse, un seul étudiant (Info) n'est parvenu à résoudre aucun des deux exercices.

### 7.3.2 Retours sur l'utilisabilité

Plusieurs étudiants ont découvert que les blocs étaient magnétiques par accident. Deux blocs se sont alors attachés entre eux lorsqu'ils en ont déplacé un suffisamment proche d'un autre, créant un effet de surprise. Malgré cette découverte, plusieurs étudiants ont préféré garder un espace entre les blocs de leurs solutions en cours de construction. Ils les assemblaient alors au dernier moment, juste avant d'analyser leur solution.

Deux étudiants (22% - un IngeMi et un Info) ont éprouvé des difficultés pour revenir à l'écran précédent en manipulant l'application. Cette difficulté a pu être surmontée rapidement via une explication rapide de l'emplacement du bouton retour.

Trois étudiants (33%), tous en Info n'ont pas compris le terme « bloc » utilisé dans les retours de l'application. En effet, lorsqu'une erreur survient, l'application indique dans quel bloc elle se trouve (par exemple, « variable non déclarée dans le troisième bloc »). Lorsque la question « qu'est-ce qu'un bloc selon toi » a été posée à un étudiant, il a répondu que pour lui un bloc était un groupe de lignes de code, par exemple indentées.

La lecture des messages de retours a souvent été écourtée par les étudiants. En effet, quatre étudiants (44%) sur les neuf, dont deux Info et deux InGMI n'ont pas lu l'entièreté des retours qui leur était présentée. Ils se sont contentés

de lire l'emplacement de l'erreur puis ont réfléchi d'eux-mêmes. Aussi, lorsqu'un étudiant semblait ne pas comprendre son erreur, il était interrogé sur le message de l'application. L'étudiant lisait alors (à voix haute) le message et comprenait alors l'origine de son erreur. Une nouvelle présentation des retours devra être conçue avec les experts pour encourager davantage l'étudiant à les lire complètement.

De façon anecdotique, deux étudiants ont laissé tomber un bloc, perdant ainsi l'étiquette associée. Celle-ci a été correctement replacée par l'étudiant, probablement par chance.

### 7.3.3 Retours sur l'utilité

Les étudiants ont répondu par l'affirmative lorsqu'il leur a été demandé si ce genre d'outil pouvait les aider lors d'une première séance d'introduction à la programmation. Un seul étudiant (11%) en Info a formulé une préoccupation exprimant que ce genre d'activité risquait de créer une dissipation dans la classe, car il la considérait comme trop ludique et pas assez sérieuse.

### 7.3.4 Données statistiques

Cette partie reprend les données extraites des enregistrements vidéos et des soumissions des étudiants effectués pendant les essais.

Tous les étudiants en Info qui ont effectué le test en langage naturel après le pseudo-code (3/4 - 75% des Info) ont réussi du premier coup. Sur 21 solutions vérifiées par les Info, 7 étaient valides (33%). Les 14 solutions non valides étaient dues à une variable non déclarée ou non assignée (9/14 - 64%), à l'utilisation d'une assignation inversée incorrecte (2/14 - 14%, dont une lors d'une expérimentation ne comprenant que ce bloc), à une double permutation (2/14 - 14%) ou à un écrasement de valeur (1/14 - 7%). Cependant, sur les 9 erreurs de variable non déclarée, 3 auraient mené à une solution valide si la déclaration des variables était faite à la manière de python.

Sur 25 solutions vérifiées par les étudiants en IngMI, 9 étaient valides (9/25 - 36%). Deux étudiants qui ont terminé l'exercice en langage naturel n'ont pas réussi l'exercice en pseudo-code du premier coup. Le premier a fait une erreur d'écrasement de valeur, le second a fait une erreur de variable non déclarée puis deux erreurs d'écrasement de valeur. Les erreurs sur le langage naturel (12) étaient causées par une variable non déclarée (5/12 - 42%), un écrasement de valeur (4/12 - 33%) l'utilisation d'une assignation inversée incorrecte (2/12 - 17%) ou une double permutation (1/12 - 8%). Cependant, sur les six erreurs de variable non déclarée dans les deux langages, deux (2/6 - 33%) auraient mené à une solution valide si la déclaration des variables était faite à la manière de python.

Ces différentes statistiques sont reprises dans un tableau (Tab. 7.1) qui détaille les différentes erreurs faites par les étudiants en fonction du groupe et du langage.

## 7.4 Améliorations envisagées du CI tangible

Les retours des étudiants ont permis de dégager plusieurs pistes d'amélioration pour l'application, essentiellement. Cette future nouvelle version n'est, à l'heure de la rédaction, pas encore terminée et devra être testée à nouveau auprès des utilisateurs.

Tout d'abord, l'interface de l'application devra être repensée afin de permettre un retour plus facile à l'écran précédent. Un bouton spécifique plus visible devra être mis en place, sans gêner la vue des autres éléments. Le vocabulaire propre à l'application devra également être revu. Le terme « bloc » qui désigne les éléments physiques de l'application devra être remplacé par un terme moins ambigu. En effet, la confusion uniquement présente chez les étudiants Info provient probablement du terme « bloc d'instructions ».Un nouveau terme pourrait alors être envisagé, par exemple le terme « brique ». L'utilisation d'un adjectif qualificatif pourrait également lever l'ambiguïté, comme « bloc blanc » ou encore « bloc magnétique ». Le message explicatif renvoyé à l'étudiant devra aussi être revu, afin de motiver l'étudiant à le lire jusqu'au bout. Les retours à l'étudiant pourraient par exemple être divisés sur plusieurs écrans. L'application demanderait alors à l'étudiant s'il veut des détails supplémentaires. Plusieurs pistes d'amélioration ont déjà été évoquées par l'expert IHM, comme la possibilité de masquage de certains éléments et la réduction de la quantité de texte à l'écran.

En ce qui concerne les blocs, la manipulation a été simple pour la plupart des étudiants. Des blocs sont tombés et les étiquettes en sont sorties. Un système de fixation (par exemple, avec une colle non permanente de type pâte réutilisable, ou avec des attaches auto-agrippantes) est envisagé pour éviter ce type de problème. Le magnétisme des blocs a été perçu parfois par accident chez les étudiants, lors d'une manipulation. Une coloration des extrémités magnétiques ou un pictogramme plus explicite est également envisagé. De plus, certains étudiants ont préféré garder les blocs espacés pour les manipuler plus facilement, sans le système de magnétisme. Une baisse de la puissance des aimants pourrait être expérimentée sur un prochain prototype, afin de confirmer si cet élément est déterminant.

Le langage naturel a été considéré comme trop difficile par la majorité des étudiants. Il devra donc être revu pour faciliter sa lecture, jugée trop difficile (par exemple avec des colorations en fonction des types d'instructions). Un autre langage plus imagé pourrait également être conçu, afin de faire découvrir les concepts de programmation aux plus jeunes. Le langage naturel (modifié ou non) devra

| | TOTAL | Info | IngMI |
|---|---|---|---|
| Nombre d'étudiants | 9 | 4 | 5 |
| Solutions vérifiées | 46 | 21 | 25 |
| Solutions valides | 16 | 7 | 9 |
| Solutions valides pseudo-code | 9 | 4 | 5 |
| Solutions valides langage naturel | 7 | 3 | 4 |
| Solutions invalides | 30 | 14 | 16 |
| Solutions invalides pseudo-code | 18 | 14 | 4 |
| Solutions invalides langage naturel | 12 | 0 | 12 |
| Variable non déclarée | 15 | 9 | 6 |
| Variable non déclarée pseudo-code | 10 | 9 | 1 |
| Variable non déclarée langage naturel | 5 | 0 | 5 |
| Variable non déclarée et invalide | 11 | 6 | 5 |
| Variable non déclarée et invalide pseudo-code | 7 | 6 | 1 |
| Variable non déclarée et invalide langage naturel | 4 | 0 | 4 |
| Double permutation | 3 | 2 | 1 |
| Double permutation pseudo-code | 2 | 2 | 0 |
| Double permutation langage naturel | 1 | 0 | 1 |
| Écrasement de valeur | 8 | 1 | 7 |
| Écrasement de valeur pseudo-code | 4 | 1 | 3 |
| Écrasement de valeur langage naturel | 4 | 0 | 4 |
| Mauvaise assignation | 4 | 2 | 2 |
| Mauvaise assignation pseudo-code | 2 | 2 | 0 |
| Mauvaise assignation langage naturel | 2 | 0 | 2 |
| Essais par étudiant | 5,11 | 5,25 | 5 |
| Taux de succès | 35% | 33% | 36% |

TABLE 7.1 – Les différentes statistiques des solutions vérifiées par les étudiants. Pour les erreurs de variables non déclarées, Python offre certaines souplesses que le pseudo-code et le langage naturel n'offrent pas. Cependant, la ligne « Variable non déclarée et invalide » indique le nombre de solutions erronées si la variable était gérée comme en python dans ces deux langages.

```
+--------------+
| entier b = 2 |
|        a = b |
| entier a = 3 |
|        b = a |
+--------------+
```

FIGURE 7.5 – La première solution proposée par trois étudiants Info.


FIGURE 7.6 – À gauche, l'assemblage fait par l'étudiant. À droite, la solution qu'il a formée.

cependant faire l'objet d'une expérimentation supplémentaire sur de vrais novices afin de vérifier si malgré sa longueur, il permet une meilleure compréhension de la sémantique des instructions.

Au niveau du serveur des statistiques, plusieurs améliorations devront être effectuées. En effet, collecter les statistiques en l'état est une étape trop chronophage, car leur extraction demande l'utilisation de requêtes SQL. De plus, les statistiques ne sont pas directement reliées à un utilisateur de l'application et ne sauvegardent qu'une seule mauvaise représentation (celle affichée à l'étudiant).

### 7.4.1 Représentations erronées détectées

Certaines mauvaises représentations commentées/identifiées dans la littérature scientifique semblent présentes parmi les étudiants du groupe testé.

— Un étudiant (Info) a formulé une réponse en deux parties séparées (Fig 7.6).

Lorsqu'il a scanné sa solution, il s'est d'abord étonné que sa réponse ne soit pas correcte, puis a compris que la lecture du code se faisait séquentiellement. Il peut alors s'agir de la mauvaise représentation MIS8 « Magical parallelism : several lines of a (simple non concurrent) program can be simulaneously active or known ».

— Trois étudiants (2 Info, 1 IngMI) ont fait une double permutation. La double permutation est le fait de faire une assignation de type $a = b$ et de directement ensuite faire l'opération inverse $b = a$ en pensant que les valeurs sont inversées. Cette erreur a été étudiée par Dehnadi [16], Sorva [101] et est présente dans le CI de Henry [42].
— Trois étudiants Info ont composé la même première solution erronée (Fig. 7.5). Elle est similaire à la précédente si l'on considère les déclarations de variables comme en python. Cependant, la double permutation est séparée par une assignation. Dans le langage pseudo-code, cette séquence est invalide, car la variable n'a pas été déclarée.
— Un étudiant (IngMI) a, après avoir réussi la solution en langage naturel, inversé le sens des trois assignations variable vers variable, ce qui signifie que si on inverse le sens de l'assignation, sa solution est correcte. Cette mauvaise représentation est également documentée par Dehnadi.

### 7.4.2 Pistes d'améliorations discutées avec l'expert IHM

Après échanges avec l'expert IHM, plusieurs pistes d'amélioration ont été proposées pour améliorer la lisibilité de l'interface. Pour l'écran d'accueil, le fait de devoir retenir l'emplacement et l'icône du bouton retour est à améliorer. Un vrai bouton « retour » devra donc être intégré aux deux autres écrans. Au niveau de l'écran permettant l'analyse du code, le chiffre indiquant le nombre de blocs détectés doit être rendu plus visible. Un liseré blanc en surimpression pourrait rendre l'information plus visible, indépendamment de la noirceur de l'image analysée. Le troisième écran contenant les retours à l'étudiant sur son code est celui qui devra faire l'objet du plus grand nombre de modifications. Le message affichant le résultat de l'exercice « Essaie encore » devra être changé en « Dommage, essaie encore » et être affiché plus en évidence avec une police différente et centrée. Les explications devront être affichées dans un cadre qu'il sera possible de masquer pour ne pas encombrer l'écran. La quantité d'éléments textuels présents à l'écran devra donc être très fortement revue à la baisse pour offrir une meilleure lisibilité.

## 7.5 Limites des évaluations

Certaines questions du questionnaire UEQ ont été considérées comme ambiguës par plusieurs testeurs. Les résultats de ce questionnaire peuvent donc avoir été influencés par ce facteur.

La taille des échantillons (enseignants et étudiants) était limitée. Des tests à plus grande échelle doivent être envisagés pour les prochaines versions de l'application afin de confirmer les tendances des données statistiques.

Seuls trois des quatre exercices ont été testés avec les enseignants, et un seul des quatre exercices a été testé avec les étudiants. Davantage de tests devront être effectués sur chaque exercice existant et futur avant de pouvoir utiliser l'application en contexte réel. De plus, les étudiants qui ont expérimenté l'exercice numéro 1 n'étaient plus, au moment du test, des novices. Il faudra donc planifier des essais sur les novices en début d'année scolaire afin de confirmer les retours obtenus et de valider, entre autres, que l'outil est bien adapté à ce public.

# Chapitre 8

# Discussion

Pour rappel, ce travail de recherche tente de déterminer « dans quelle mesure un outil tangible d'aide à l'apprentissage de la programmation peut influencer les représentations erronées des étudiants suivant un cours d'introduction à la programmation ».

Pour apporter des éléments de réponse à cette problématique, quatre hypothèses ont été formulées :

- Un outil tangible peut aider à l'apprentissage de la programmation dans un contexte d'enseignement supérieur (H1)
- Un outil tangible (d'aide à l'apprentissage de la programmation) est utilisable avec un public de jeunes adultes (16 ans et +) (H2)
- Un outil tangible peut aider à détecter les représentations erronées des étudiants (H3)
- Un outil tangible peut aider à corriger les représentations erronées des étudiants (H4)

Il s'agit avant tout de vérifier ces différentes hypothèses, ce qui est discuté ci-dessous. À noter que la faible taille de l'échantillon (17) ne permet pas de vérification rigoureuse des hypothèses. Au mieux, il s'agit de savoir si l'hypothèse peut ou ne peut pas être rejetée. Rien de définitif ne peut être avancé au-delà de ce travail de recherche, de l'échantillon et des données récoltées durant les évaluations en question quant au statut de chaque hypothèse.

Un CI tangible a été développé de façon itérative et en collaboration avec des experts en informatique et en éducation. Deux phases d'évaluations ont été organisées avec des publics différents : d'abord, des enseignants et assistants susceptibles d'utiliser le CI tangible dans le cadre de leurs cours et/ou TP ; ensuite, des étudiants suivant un cours d'introduction à la programmation, public cible de la recherche.

De manière générale, les retours sur le CI tangible sont positifs, quel que soit le public. Si les évaluations avec les membres du personnel éducatif et les étudiants ont permis de mettre en avant plusieurs pistes pour améliorer l'outil, elles ont surtout montré la prise en main rapide de l'outil par des adultes (jeunes et moins jeunes) et l'enthousiasme de ces derniers lors de la manipulation des blocs. L'hypothèse H2 ne peut donc être rejetée.

La création d'exercices illustrant certaines représentations erronées avait pour objectif de rendre facilement détectable la présence de ces dernières chez les étudiants. Durant les évaluations, certains étudiants ont assemblé les blocs en reproduisant intuitivement ces représentations erronées. Par cette manipulation, les représentations sont non seulement rendues directement détectables pour l'équipe éducative (qui peut vérifier rapidement l'assemblage de blocs), mais également au niveau des données statistiques collectées. De plus, un message d'aide contextuel est affiché par l'application, informant l'étudiant de l'existence de cette représentation erronée dans sa solution. Bien que les évaluations n'aient porté que sur quelques exercices et représentations erronées, l'hypothèse H3 ne peut également pas être rejetée, l'outil tangible développé aidant par différents moyens à la détection des représentations erronées présentes chez les étudiants. La création d'exercices supplémentaires illustrant les représentations erronées les plus courantes selon la littérature est envisagée comme travaux futurs.

Les messages d'aide affichés à la détection d'une représentation erronée, entre autres, informent l'utilisateur de son problème de compréhension et lui permettent de le corriger. En aucun cas ces messages ne fournissent une réponse toute faite. Il s'agit de souligner théoriquement de façon à en faire prendre conscience. Durant les évaluations auprès des étudiants, les étudiants, une fois informés, ont corrigé leurs assemblages de blocs en reconsidérant l'ensemble des blocs et en trouvant parmi eux le(s) bloc(s) répondant à la théorie dont ils venaient de prendre connaissance. Dès lors, l'hypothèse H4 ne peut être rejetée. En effet, l'outil tangible développé aide les étudiants à corriger leurs propres représentations erronées. Cependant, il serait nécessaire, lors de travaux futurs, de mesurer la rétention sur le long terme des apprentissages et prises de conscience réalisés grâce à l'outil. Sur le court terme, il a été observé qu'une erreur réalisée sur un premier exercice n'était plus reproduite sur les suivants.

Il est plus délicat de vérifier l'hypothèse H1. Sur base des discours informels tenus avec les enseignants, il semble que le CI tangible ait du potentiel pour aider des novices en programmation à comprendre les concepts de base en programmation. Les études menées par Henry dans le cadre de sa thèse appuient cette affirmation. De plus, l'utilisation du cet outil durant les séances de TP, en complément du cours magistral, semble parfaitement envisageable. Cette information est d'ailleurs confirmée par les étudiants eux-mêmes. Toutefois, il

aurait fallu effectuer des évaluations auprès de véritables novices et en contexte réel d'apprentissage pour pouvoir rigoureusement vérifier l'hypothèse H1. Dans le cadre de cette recherche, ces évaluations n'ont pas été possibles, le cours INFOB131 étant programmé au quadrimestre 1.

La vérification de chacune de ces quatre hypothèses, bien qu'incomplète, apporte quelques éléments de réponse à la question « dans quelle mesure un outil tangible d'aide à l'apprentissage de la programmation peut influencer les représentations erronées des étudiants suivant un cours d'introduction à la programmation ». Ainsi, dans le contexte particulier de cette recherche, le CI tangible s'avère un outil pédagogique faisant sens pour les enseignants, notamment comme aide à la détection des problèmes présents chez leurs étudiants. Les étudiants pourraient vraisemblablement envisager d'utiliser cet outil en complément de ce qui est mis actuellement en place dans le cadre du cours INFOB131 et, mieux encore, semble prendre conscience de leurs représentations erronées suite à la manipulation de l'outil.

La question de recherche doit cependant être creusée davantage. Un plus grand échantillon d'étudiants, notamment, doit être touché pour conforter certains résultats. Une utilisation en contexte réel doit être envisagée. Une collaboration avec un expert en éducation est à prévoir, de manière plus approfondie, concernant la rédaction des messages d'aide à destination des étudiants. Une réflexion reste également à faire sur le « langage » à utiliser sur les étiquettes. Enfin, si certains problèmes d'utilisabilité ont déjà été mis en évidence, des améliorations restent à apporter à l'application.

# Chapitre 9

# Conclusion

Ce mémoire décrit le développement d'un outil tangible d'aide à l'apprentissage de la programmation chez les novices. Parce qu'un *Concept Inventory* (CI) permet de détecter les mauvaises représentations, l'outil se veut être un CI tangible. Là où un CI propose des courts codes à lire et à comprendre, le CI tangible invite l'utilisateur à écrire de courts codes au moyen de blocs tangibles et à exécuter ces codes au moyen d'une application.

Développé dans le cadre d'un cours d'introduction à la programmation dispensé à l'Université de Namur, le CI tangible est destiné aux étudiants en première année de bachelier. La population de ces étudiants est issue de deux filières : informatique et ingénierie de gestion. De précédentes études montrent les difficultés qu'ont ces étudiants à comprendre les concepts de base en programmation. L'idée est de mettre à leur disposition (et à disposition des enseignants en charge du cours) un moyen supplémentaire de prendre conscience des représentations erronées qu'ils possèdent et de les aider à les corriger.

La méthodologie guidant le développement de l'outil est inspirée de la *design-oriented research*. Cette approche consiste à mener un processus itératif qui articule des phases de conception, d'évaluation de cet outil à différents niveaux et d'analyse des résultats obtenus.

Un premier cycle de développement a été mené sur base d'une collaboration entre le chercheur, auteur de ce mémoire, et des experts en informatique et éducation. Des évaluations ont été mises en place auprès avec d'enseignants et assistants susceptibles d'utiliser le CI tangible dans le cadre de leur travail. L'analyse des données collectées a abouti à un second cycle de développement, visant une évaluation auprès des étudiants, public cible de l'outil.

Le dernier prototype développé dans le cadre de ce mémoire consiste en des blocs tangibles, imprimés en 3D, sur lesquels viennent s'intégrer des étiquettes

en papier. Chaque étiquette reprend une instruction et un marqueur visuel. Deux langages ont été envisagés pour décrire les instructions : un pseudo-code et le langage naturel (français). Les blocs s'assemblent entre eux, créant un code. Les marqueurs permettent une reconnaissance de chaque bloc et une exécution du code formé via une application.

Quatre exercices ont été développés, illustrant des représentations erronées courantes en ce qui concerne les concepts de base en programmation que sont la variable, la structure conditionnelle et la boucle. Un code proposé par un étudiant est analysé par l'application, s'ensuit un retour informant des erreurs produites et une explication théorique des représentations erronées détectées. Par un jeu d'essai-erreur, l'étudiant va prendre conscience de ses problèmes de compréhension et tenter d'y remédier.

Des évaluations ont eu lieu en deux phases, d'abord avec huit enseignants et ensuite, 9 étudiants en première année de bachelier. Les résultats obtenus au questionnaire UEQ lors de la première phase ont validé l'utilisabilité du CI tangible. Cependant, observations et retours informels ont mis en évidence des points à améliorer : quelques petits problèmes ergonomiques au niveau des interfaces, un exercice mal adapté au public cible, un pseudo langage déroutant, entre autres. La seconde phase d'évaluations a révélé le potentiel du CI tangible à détecter les représentations erronées présentes chez les étudiants. En ce qui concerne l'aide apportée par l'outil dans la correction de ces représentations, si les observations effectuées sont prometteuses, il apparaît que du travail reste à faire sur la forme et le fond des messages affichés par l'application en vue d'aider l'étudiant.

À ce stade,les évaluations du CI tangible sont insuffisantes pour tirer des conclusions sur son efficacité en termes d'aide à l'apprentissage. Des tests supplémentaires devront être menés sur le long terme, avec un échantillon plus large et, avant cela, une nouvelle version de prototype devra voir le jour. En effet, de nombreuses pistes d'améliorations ont été évoquées en réponse aux deux phases d'évaluation.

Le CI tangible, baptisé Cartorithmique - *contraction des mots cartes et algorithmique* - n'en est qu'à ses premiers pas. L'enseignement de la programmation est en passe d'être introduit dans de nombreux cursus universitaires, mais également chez les plus jeunes. Cet enseignement étant réputé difficile, il est important que des recherches soient menées sur des outils aidant son apprentissage. En espérant que ceci ne soit que le commencement d'un nouveau cycle de développement...

# Bibliographie


[1] Mohammad S Alsoufi and AE Elsayed. Warping deformation of desktop 3d printed parts manufactured by open source fused deposition modeling (fdm) system. *International Journal of Mechanical and Mechatronics Engineering*, 17(4) :7–16, 2017.

[2] Terry Anderson and Julie Shattuck. Design-based research : A decade of progress in education research ? *Educational researcher*, 41(1) :16–25, 2012.

[3] Brett A Becker. An effective approach to enhancing compiler error messages. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 126–131, 2016.

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

[5] Susan Bergin and Ronan Reilly. The influence of motivation and comfort-level on learning to program. 2005.

[6] Richard Bornat, Saeed Dehnadi, et al. Mental models, consistency and programming aptitude. In *Proceedings of the tenth conference on Australasian computing education-Volume 78*, pages 53–61. Australian Computer Society, Inc., 2008.

[7] Amy Susan Bruckman. *MOOSE crossing : Construction, community and learning in a networked virtual world for kids*. PhD thesis, Massachusetts Institute of Technology, 1997.

[8] Ricardo Caceffo, Pablo Frank-Bolton, Renan Souza, and Rodolfo Azevedo. Identifying and validating java misconceptions toward a cs1 concept inventory. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pages 23–29, 2019.

[9] Ricardo Caceffo, Guilherme Gama, Raysa Benatti, Tales Aparecida, Tania Caldas, and Rodolfo Azevedo. A concept inventory for cs1 introductory programming courses in c. Technical report, Tech. rep., University of Campinas, SP, Brasil, 2018.

[10] Ricardo Caceffo, Steve Wolfman, Kellogg S Booth, and Rodolfo Azevedo. Developing a computer science concept inventory for introductory programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 364–369, 2016.

[11] Michael E Caspersen, Kasper Dalgaard Larsen, and Jens Bennedsen. Mental models and programming aptitude. In *Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 206–210, 2007.

[12] Kunal Chawla, Megan Chiou, Alfredo Sandes, and Paulo Blikstein. Dr. wagon : a'stretchable'toolkit for tangible computer programming. In *Proceedings of the 12th international conference on interaction design and children*, pages 561–564. ACM, 2013.

[13] Holger Danielsiek, Wolfgang Paul, and Jan Vahrenhold. Detecting and understanding students' misconceptions related to algorithms and data structures. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 21–26, 2012.

[14] Joan Davis. Conceptual change. *Emerging perspectives on learning, teaching, and technology*, 19, 2001.

[15] Saeed Dehnadi. Testing programming aptitude. In *PPIG*, page 9, 2006.

[16] Saeed Dehnadi. *A cognitive study of learning to program in introductory programming courses.* PhD thesis, Middlesex University, 2009.

[17] Rebecca B Dupaix and Mary C Boyce. Finite strain behavior of poly (ethylene terephthalate)(pet) and poly (ethylene terephthalate)-glycol (petg). *Polymer*, 46(13) :4827–4838, 2005.

[18] Frank Elberzhager, Jürgen Münch, and Vi Tran Ngoc Nha. A systematic mapping study on the combination of static and dynamic quality assurance techniques. *Information and Software Technology*, 54(1) :1–15, 2012.

[19] Daniel Gallardo, Carles F Julia, and Sergi Jorda. Turtan : A tangible programming language for creative exploration. In *Horizontal Interactive Human Computer Systems, 2008. TABLETOP 2008. 3rd IEEE International Workshop on*, pages 89–92. IEEE, 2008.

[20] Dedre Gentner and Albert L Stevens. *Mental models.* Psychology Press, 2014.

[21] Susan Goldin-Meadow. *Hearing gesture : How our hands help us think.* Harvard University Press, 2005.

[22] Ken Goldman, Paul Gross, Cinda Heeren, Geoffrey Herman, Lisa Kaczmarczyk, Michael C Loui, and Craig Zilles. Identifying important and

difficult concepts in introductory computing courses using a delphi process. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 256–260, 2008.

[23] Anabela Gomes and António José Mendes. Learning to program-difficulties and solutions. In *International Conference on Engineering Education–ICEE*, volume 2007, 2007.

[24] Leonard Goodwin and Mohammad Sanati. Learning computer programming through dynamic representation of computer functioning : evaluation of a new learning package for pascal. *International journal of man-machine studies*, 25(3) :327–341, 1986.

[25] Ileana Maria Greca and Marco Antonio Moreira. Mental models, conceptual models, and modelling. *International journal of science education*, 22(1) :1–11, 2000.

[26] Shuchi Grover and Satabdi Basu. Measuring student learning in introductory block-based programming : Examining misconceptions of loops, variables, and boolean logic. In *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education*, pages 267–272, 2017.

[27] Luke Gusukuma, Austin Cory Bart, Dennis Kafura, and Jeremy Ernst. Misconception-driven feedback : Results from an experimental study. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, pages 160–168, 2018.

[28] Said Hadjerrouit. Java as first programming language : a critical evaluation. *ACM SIGCSE Bulletin*, 30(2) :43–47, 1998.

[29] Julie Henry and Bruno Dumas. Approach to develop a concept inventory informing teachers of novice programmers' mental models. In *2020 IEEE Frontiers in Education Conference (FIE)*. IEEE, 2020.

[30] Julie Henry, Bruno Dumas, and Antoine Bodart. Programmation tangible pour les enfants : analyse de l'existant, classification et opportunités. 2018.

[31] Julie Henry, Bruno Dumas, Patrick Heymans, and Tony Leclercq. Object-oriented programming : Identifying novices' representations to make a diagnosis. In *2020 IEEE Frontiers in Education Conference (FIE)*. IEEE, 2020.

[32] Geoffrey L Herman, Michael C Loui, and Craig Zilles. Creating the digital logic concept inventory. In *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 102–106. ACM, 2010.

[33] David Hestenes, Malcolm Wells, and Gregg Swackhamer. Force concept inventory. *The physics teacher*, 30(3) :141–158, 1992.

[34] Peter W Hewson and Mariana GA'Beckett Hewson. The role of conceptual conflict in conceptual change and the design of science instruction. *Instructional Science*, 13(1) :1–13, 1984.

[35] Michael S. Horn and Robert J. K. Jacob. Designing tangible programming languages for classroom use. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*, TEI '07, page 159–162, New York, NY, USA, 2007. Association for Computing Machinery.

[36] Michael S. Horn and Robert J. K. Jacob. Tangible programming in the classroom with tern. In *CHI '07 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '07, page 1965–1970, New York, NY, USA, 2007. Association for Computing Machinery.

[37] Michael S Horn and Robert JK Jacob. Tangible programming in the classroom : a practical approach. In *CHI'06 extended abstracts on Human factors in computing systems*, pages 869–874. ACM, 2006.

[38] Michael S Horn and Robert JK Jacob. Tangible programming in the classroom with tern. In *CHI'07 extended abstracts on Human factors in computing systems*, pages 1965–1970. ACM, 2007.

[39] Felix Hu, Ariel Zekelman, Michael Horn, and Frances Judd. Strawbies : explorations in tangible programming. In *Proceedings of the 14th International Conference on Interaction Design and Children*, pages 410–413. ACM, 2015.

[40] Fionnuala Johnson, Stephen McQuistin, and John O'Donnell. Analysis of student misconceptions using python as an introductory programming language. In *Proceedings of the 4th Conference on Computing Education Practice 2020*, pages 1–4, 2020.

[41] Philip Nicholas Johnson-Laird. *Mental models : Towards a cognitive science of language, inference, and consciousness*. Number 6. Harvard University Press, 1983.

[42] Henry Julie and Dumas Bruno. Towards the identification of profiles based on the understanding of programming concepts : the case of the variable. In *2019 IEEE FIE Conference*, pages 1–8. IEEE, 2019.

[43] Henry Julie and Dumas Bruno. Developing an assessment to profile students based on their understanding of the variable programming concept. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '20)*. ACM, 2020.

[44] Lisa C Kaczmarczyk, Elizabeth R Petrick, J Philip East, and Geoffrey L Herman. Identifying student misconceptions of programming. In *Proceedings*

*of the 41st ACM technical symposium on Computer science education*, pages 107–111, 2010.

[45] Kuba Karpierz and Steven A Wolfman. Misconceptions and concept inventory questions for binary search trees and hash tables. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 109–114, 2014.

[46] Majeed Kazemitabaar, Jason McPeak, Alexander Jiao, Liang He, Thomas Outing, and Jon E Froehlich. Makerwear : A tangible approach to interactive wearable creation for children. In *Proceedings of the 2017 chi conference on human factors in computing systems*, pages 133–145, 2017.

[47] Annie Kelly, R Benjamin Shapiro, Jonathan de Halleux, and Thomas Ball. Arcadia : A rapid prototyping platform for real-time tangible interfaces. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–8, 2018.

[48] Andrew J Ko and Brad A Myers. Designing the whyline : a debugging interface for asking questions about program behavior. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 151–158, 2004.

[49] Vassilis Komis and Anastasia Misirli. Robotique pédagogique et concepts préliminaires de la programmation à l'école maternelle : une étude de cas basée sur le jouet programmable bee-bot. In *Sciences et technologies de l'information et de la communication en milieu éducatif : Analyse de pratiques et enjeux didactiques.*, pages 271–281. Athènes : New Technologies Editions, 2011.

[50] Varsha Koushik, Darren Guinness, and Shaun K Kane. Storyblocks : A tangible programming game to create accessible audio stories. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2019.

[51] Divna Krpan, Saša Mladenović, and Biserka Ujević. Tangible programming with augmented reality. In *12th International Technology, Education and Development Conference*, 2018.

[52] Vambola Leping, Marina Lepp, Margus Niitsoo, Eno Tõnisson, Varmo Vene, and Anne Villems. Python prevails. In *Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*, pages 1–5, 2009.

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

[54] Andrew Luxton-Reilly, Brett A Becker, Yingjun Cao, Roger McDermott, Claudio Mirolo, Andreas Mühling, Andrew Petersen, Kate Sanders, and Jacqueline Whalley. Developing assessments to determine mastery of programming fundamentals. In *Proceedings of the 2017 ITiCSE Conference on Working Group Reports*, pages 47–69, 2018.

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

[56] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4) :1–15, 2010.

[57] Javier Marco, Clara Bonillo, and Eva Cerezo. A tangible interactive space odyssey to support children learning of computer programming. In *Proceedings of the 2017 ACM International Conference on Interactive Surfaces and Spaces*, pages 300–305. ACM, 2017.

[58] Jean-Baptiste Marco, Nadine Baptiste-Jessel, and Philippe Truillet. Tabgo : programming with tangibles blocks. In *Proceedings of the 30th Conference on l'Interaction Homme-Machine*, pages 179–185, 2018.

[59] Raina Mason and Graham Cooper. Introductory programming courses in australia and new zealand in 2013-trends and reasons. In *Proceedings of the Sixteenth Australasian Computing Education Conference-Volume 148*, pages 139–147, 2014.

[60] Raina Mason, Tom Crick, James H Davenport, and Ellen Murphy. Language choice in introductory programming courses at australasian and uk universities. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 852–857, 2018.

[61] Linda McIver. The effect of programming language on error rates of novice programmers. In *PPIG*, page 15, 2000.

[62] Timothy S McNerney. From turtles to tangible programming bricks : explorations in physical language design. *Personal and Ubiquitous Computing*, 8(5) :326–337, 2004.

[63] Edward F Melcer and Katherine Isbister. Bots & (main) frames : exploring the impact of tangible blocks and collaborative play in an educational programming game. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2018.

[64] Monika Mladenović, Ivica Boljat, and Žana Žanko. Comparing loops misconceptions in block-based and text-based programming languages at the k-12 level. *Education and Information Technologies*, 23(4) :1483–1500, 2018.

[65] Cecily Morrison, Nicolas Villar, Anja Thieme, Zahra Ashktorab, Eloise Taysom, Oscar Salandin, Daniel Cletheroe, Greg Saul, Alan F Blackwell, Darren Edge, et al. Torino : A tangible programming language inclusive of children with visual disabilities. *Human–Computer Interaction*, pages 1–49, 2018.

[66] Ioana Tuugalei Chan Mow. Analyses of student programming errors in java programming courses. *Journal of Emerging Trends in Computing and Information Sciences*, 3(5) :739–749, 2012.

[67] Andrea Mussati, Christian Giang, Alberto Piatti, and Francesco Mondada. A tangible programming language for the educational robot thymio. In *2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pages 1–4. IEEE, 2019.

[68] Mark Noone and Aidan Mooney. First programming language : Visual or textual ? *arXiv preprint arXiv :1710.11557*, 2017.

[69] Donald A Norman. Some observations on mental models. mental models. *Lawrence Erlbaum :*, pages 99–citation_lastpage, 1983.

[70] Joseph D Oldham. What happens after python in cs1 ? *Journal of computing sciences in colleges*, 20(6) :7–13, 2005.

[71] Edwin Olson. Apriltag : A robust and flexible visual fiducial system. In *2011 IEEE International Conference on Robotics and Automation*, pages 3400–3407. IEEE, 2011.

[72] Taylor Otwell. Laravel. *The PHP Framework For Web Artisans*, 2016.

[73] Dincer Ozoran, N Cagiltay, and Damla Topalli. Using scratch in introduction to programming course for engineering students. In *2nd International Engineering Education Conference (IEEC2012)*, volume 2, pages 125–132, 2012.

[74] Seymour Papert. *Mindstorms : Children, computers, and powerful ideas*. Basic Books, Inc., 1980.

[75] Kevin R Parker, Joseph T Chao, Thomas A Ottaway, and Jane Chang. A formal language selection process for introductory programming courses. *Journal of Information Technology Education : Research*, 5(1) :133–151, 2006.

[76] Amanda J Parkes, Hayes Solos Raffle, and Hiroshi Ishii. Topobo in the wild : longitudinal evaluations of educators appropriating a tangible interface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1129–1138, 2008.

[77] Terence J. Parr and Russell W. Quong. Antlr : A predicated-ll (k) parser generator. *Software : Practice and Experience*, 25(7) :789–810, 1995.

[78] Huaishu Peng. Algo. rhythm : computational thinking through tangible music device. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction*, pages 401–402, 2012.

[79] Radia Perlman. Tortis (toddler's own recursive turtle interpreter system). 1974.

[80] Radia Perlman. Using computer technology to provide a creative learning environment for preschool children. 1976.

[81] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12*, pages 1–10, 2008.

[82] Danny Plass-Oude Bos. Identifying and addressing common programming misconceptions with variables (part 1). Master's thesis, University of Twente, 2015.

[83] Yuliia Prokop, Elena Trofimenko, Nikolay Severin, and Liudmila Bukata. An analysis of criteria for choosing a first programming language in universities. *Archived Volume*, page 420, 2019.

[84] Yizhou Qian and James Lehman. Students' misconceptions and other difficulties in introductory programming : A literature review. *ACM Transactions on Computing Education (TOCE)*, 18(1) :1–24, 2017.

[85] Atanas Radenski. " python first" a lab-based digital introduction to computer science. *ACM SIGCSE Bulletin*, 38(3) :197–201, 2006.

[86] Masura Rahmat, Kamsuriah Ahmad, Sufian Idris, and Noor Faridatul Ainun Zainal. Relationship between employability and graduates' skill. *Procedia-Social and Behavioral Sciences*, 59 :591–597, 2012.

[87] Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming : A review and discussion. *Computer science education*, 13(2) :137–172, 2003.

[88] Joel Sadler, Kevin Durfee, Lauren Shluzas, and Paulo Blikstein. Bloctopus : A novice modular sensor system for playful prototyping. In *Proceedings of the ninth international conference on tangible, embedded, and embodied interaction*, pages 347–354, 2015.

[89] Philip M Sadler, Harold Coyle, Jaimie L Miller, Nancy Cook-Smith, Mary Dussault, and Roy R Gould. The astronomy and space science concept inventory : development and validation of assessment instruments aligned with the k–12 national science standards. *Astronomy Education Review*, 8(1) :010111, 2010.

[90] Theodosios Sapounidis and Stavros Demetriadis. Touch your program with hands : qualities in tangible programming tools for novice. In *Informatics (PCI), 2011 15th Panhellenic Conference on*, pages 363–367. IEEE, 2011.

[91] Theodosios Sapounidis and Stavros N Demetriadis. Exploring children preferences regarding tangible and graphical tools for introductory programming : evaluating the proteas kit. In *2012 IEEE 12th International Conference on Advanced Learning Technologies*, pages 316–320. IEEE, 2012.

[92] Florian Scharf, Thomas Winkler, and Michael Herczeg. Tangicons : algorithmic reasoning in a collaborative game for children in kindergarten and first class. In *Proceedings of the 7th international conference on Interaction design and children*, pages 242–249. ACM, 2008.

[93] Martin Schrepp, Andreas Hinderks, and Jörg Thomaschewski. Design and evaluation of a short version of the user experience questionnaire (ueq-s). *International Journal of Interactive Multimedia and Artificial Intelligence*, 4 :103, 01 2017.

[94] Carsten Schulte and Jens Bennedsen. What do teachers teach in introductory programming ? In *Proceedings of the second international workshop on Computing education research*, pages 17–28, 2006.

[95] Eric Schweikardt and Mark D Gross. roblocks : a robotic construction kit for mathematics and science education. In *Proceedings of the 8th international conference on Multimodal interfaces*, pages 72–75, 2006.

[96] PH Scott, HM Asoko, and RH Driver. Teaching for conceptual change : A review of strategies. *Research in physics learning : Theoretical issues and empirical studies*, pages 310–329, 1992.

[97] Takayuki Sekiya and Kazunori Yamaguchi. Tracing quiz set to identify novices' programming misconceptions. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, pages 87–95, 2013.

[98] Orit Shaer, Eva Hornecker, et al. Tangible user interfaces : past, present, and future directions. *Foundations and Trends® in Human–Computer Interaction*, 3(1–2) :4–137, 2010.

[99] Arnan Sipitakiat and Nusarin Nusen. Robo-blocks : designing debugging abilities in a tangible programming system for early primary school children. In *Proceedings of the 11th International Conference on Interaction Design and Children*, pages 98–105. ACM, 2012.

[100] Teemu Sirkiä. Recognizing programming misconceptions. *Aalto University, Espoo*, 2012.

[101] Teemu Sirkiä and Juha Sorva. Exploring Programming Misconceptions. *Koli Calling*, 12 :19–28, November 2012.

[102] Andrew C Smith. Gameblocks : an entry point to ict for pre-school children. 2007.

[103] Andrew C Smith. Dialando : tangible programming for the novice with scratch, processing and arduino. 2010.

[104] Andrew Cyrus Smith. Visual perception skills testing : Preliminary results. In *Proceedings of the 3rd international conference on tangible and embedded Interaction*, pages 207–208, 2009.

[105] Andrew Cyrus Smith, Heinrich Springhorn, Steven Bruce Mulligan, Ireyan Weber, and Jackie Norris. tactuslogic : Programming using physical objects. In *2011 IST-Africa Conference Proceedings*, pages 1–9. IEEE, 2011.

[106] David Smith and Azad Ali. Assessing market demand for web programming languages/technologies. *Issues in Information Systems*, 15(2), 2014.

[107] Juha Sorva. Notional machines and introductory programming education. *ACM Transactions on Computing Education*, 13 :8 :1–8 :31, 06 2013.

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

[109] Juha Sorva, Ville Karavirta, and Lauri Malmi. A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education (TOCE)*, 13(4) :1–64, 2013.

[110] Amanda Strawhacker and Marina U Bers. "i want my robot to look for food" : Comparing kindergartner's programming comprehension using tangible, graphic, and hybrid user interfaces. *International Journal of Technology and Design Education*, 25(3) :293–319, 2015.

[111] Amanda Sullivan, Mollie Elkin, and Marina Umaschi Bers. Kibo robot demo : engaging young children in programming and engineering. In *Proceedings of the 14th international conference on interaction design and children*, pages 418–421. ACM, 2015.

[112] Hideyuki Suzuki and Hiroshi Kato. Interaction-level support for collaborative learning : Algoblock—an open programming language. In *The first international conference on Computer support for collaborative learning*, pages 349–355. L. Erlbaum Associates Inc., 1995.

[113] Alaaeddin Swidan, Felienne Hermans, and Marileen Smit. Programming misconceptions for school students. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, pages 151–159, 2018.

[114] Phit-Huan Tan, Choo-Yee Ting, and Siew-Woei Ling. Learning difficulties in programming courses : undergraduates' perspective and perception. In *2009 International Conference on Computer Technology and Development*, volume 1, pages 42–46. IEEE, 2009.

[115] Sureyya Tarkan, Vibha Sazawal, Allison Druin, Evan Golub, Elizabeth M Bonsignore, Greg Walsh, and Zeina Atrash. Toque : designing a cooking-based programming language for and with children. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2417–2426. ACM, 2010.

[116] Cynthia Taylor, Daniel Zingaro, Leo Porter, Kevin C Webb, Cynthia Bailey Lee, and Mike Clancy. Computer science concept inventories : past and future. *Computer Science Education*, 24(4) :253–276, 2014.

[117] Allison Elliott Tew. *Assessing fundamental introductory computing concept knowledge in a language independent manner*. PhD thesis, Georgia Institute of Technology, 2010.

[118] Danli Wang, Yunfeng Qi, Yang Zhang, and Tingting Wang. Tanpro-kit : a tangible programming tool for children. In *Proceedings of the 12th International Conference on Interaction Design and Children*, pages 344–347. ACM, 2013.

[119] Danli Wang, Cheng Zhang, and Hongan Wang. T-maze : a tangible programming tool for children. In *Proceedings of the 10th international conference on interaction design and children*, pages 127–135. ACM, 2011.

[120] Danli Wang, Yang Zhang, and Shengyong Chen. E-block : A tangible programming tool with graphical blocks. *Mathematical Problems in Engineering*, 2013, 2013.

[121] Kevin C Webb and Cynthia Taylor. Developing a pre-and post-course concept inventory to gauge operating systems learning. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 103–108, 2014.

[122] Lea Wittie, Anastasia Kurdia, and Meriel Huggard. Developing a concept inventory for computer science 2. In *FIE Conference*, pages 1–4. IEEE, 2017.

[123] Peta Wyeth and Gordon F Wyeth. Electronic blocks : Tangible programming elements for preschoolers. In *IFIP TC. 13 International Conference on Human-Computer Interaction*, volume 1, pages 496–503. IOC Press, 2001.

[124] Aharon Yadin. Reducing the dropout rate in an introductory programming course. *ACM inroads*, 2(4) :71–76, 2011.

[125] Oren Zuckerman, Saeed Arida, and Mitchel Resnick. Extending tangible interfaces for education : digital montessori-inspired manipulatives. In

*Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 859–868, 2005.

[126] Oren Zuckerman, Tina Grotzer, and Kelly Leahy. Flow blocks as a conceptual bridge between understanding the structure and behavior of a complex causal system. In *Proceedings of the 7th international conference on Learning sciences*, pages 880–886. International Society of the Learning Sciences, 2006.

# Annexe A

# Prototypes préliminaires

## A.1 Premier prototype

Le premier prototype devait, à l'image de Scratch, offrir un assemblage similaire à celui d'un puzzle. De plus, des emplacements sur ces blocs devaient être prévus pour placer les labels, contenant le code. Dans un premier temps, seuls trois blocs ont été dessinés, à savoir le bloc de début de programme, le bloc d'instruction et le bloc de fin de programme. Pour permettre de gérer les structures conditionnelles et les boucles, trois blocs supplémentaires ont été dessinés, à savoir le bloc de début de structure, le bloc de fin de structure et le bloc « else » (Fig. A.1). Les différents éléments ont été dessinés dans un logiciel de modélisation open source, à savoir FreeCAD. Le choix de FreeCAD a été motivé par l'absence de connaissances dans le domaine de la modélisation et par la gratuité de celui-ci. Pour ce qui est du matériau d'impression du premier prototype, un PET-G bleu translucide a été utilisé. Le PET est le matériau utilisé pour fabriquer les bouteilles en plastique. Il s'agit de PET-G lorsque le PET est combiné à du glycol [17]. Le PLA, aussi appelé acide polyactique est un polymère courant dans le domaine de l'impression 3D, fabriqué à partir d'amidon de maïs. Sa température d'impression est plus basse que celle du PET-G (seulement 210°C pour la tête d'impression et 60°C pour le lit chauffant), et son prix également moins élevé. Celui-ci nécessite une température de 235°C pour la tête d'impression et de 70°C pour le lit d'impression. Un bloc prend alors 1h30 à imprimer. La zone utile du bloc est, pour le bloc d'instruction normal, de 7,4 cm sur 1,6 cm.

FIGURE A.1 – Les blocs du premier prototype, de haut en bas le bloc « else » , le bloc « début de programme » , le bloc « if » , le bloc « instruction normale » , le bloc « fi » (fin si) et le bloc de « fin de programme ». Les marques noires sur le dernier bloc indiquent les emplacements des aimants pour le second prototype.

## A.2 Analyse

Une fois le premier set de blocs créé, le prototype de matériel tangible a été présenté et essayé avec les trois experts promoteurs du travail. Après analyse, plusieurs points nécessitaient encore des améliorations.

- Il faut disposer d'un grand nombre de blocs différents (six), en plusieurs exemplaires, ce qui ne facilite pas la production des blocs ni leur interchangeabilité.
- La partie utile du bloc, permettant l'insertion du label, est trop petite et ne laisse pas la place à plusieurs lignes de code. De plus, pour permettre la détection des blocs, il aurait fallu qu'une partie de cet espace soit consacré à un tag, réduisant encore la place.
- Un tag fiduciaire rétréci à l'échelle du bloc ne serait pas détectable par une caméra conventionnelle.
- Le texte écrit sur les labels des blocs est trop petit et ne facilite pas la vue des blocs dans leur ensemble.
- La manipulation des blocs, avec leur forme de puzzle, est difficile. La construction et le désassemblage du programme avec des éléments puzzle prennent plus de temps que souhaité, et demandent trop d'efforts à l'utilisateur.

Ces différents éléments ont débouché sur la création d'un second prototype, créé dans le but de corriger les points énumérés ci-dessus. Certains points positifs sont cependant également ressortis de ce premier prototype. Notamment, le concept de remplacer les étiquettes des blocs pour pouvoir réutiliser les blocs dans plusieurs contextes a reçu des retours positifs. De plus, le matériel utilisé offre une bonne prise en main et est, malgré les difficultés d'assemblage et de désassemblage, agréable à manipuler. Pour ce qui est du matériau d'impression, le PET-G bleu translucide n'était plus disponible. Un PET-G d'une autre marque, orange et opaque cette fois, a été utilisé. Celui-ci nécessite une température de 260°C pour la tête d'impression, et de 70°C pour le lit d'impression. Un bloc prend alors 3h15 à imprimer.

## A.3 Second prototype

Le second prototype tente une nouvelle approche au niveau du mécanisme d'assemblage, de la taille utile et du matériel utilisé. Un bloc (Fig. A.2) est alors une dalle dont l'espace utile est de 5 cm sur 15 cm. Il est donc beaucoup plus grand que le modèle précédent (plus de six fois la surface utile), et laisse la place à la fois au tag fiduciaire, mais aussi à plusieurs lignes de code de programmation. Pour faciliter l'assemblage et le désassemblage, le mécanisme puzzle a été abandonné au profit d'aimants au Néodyme cylindriques de 5 mm x3 mm. La puissance de

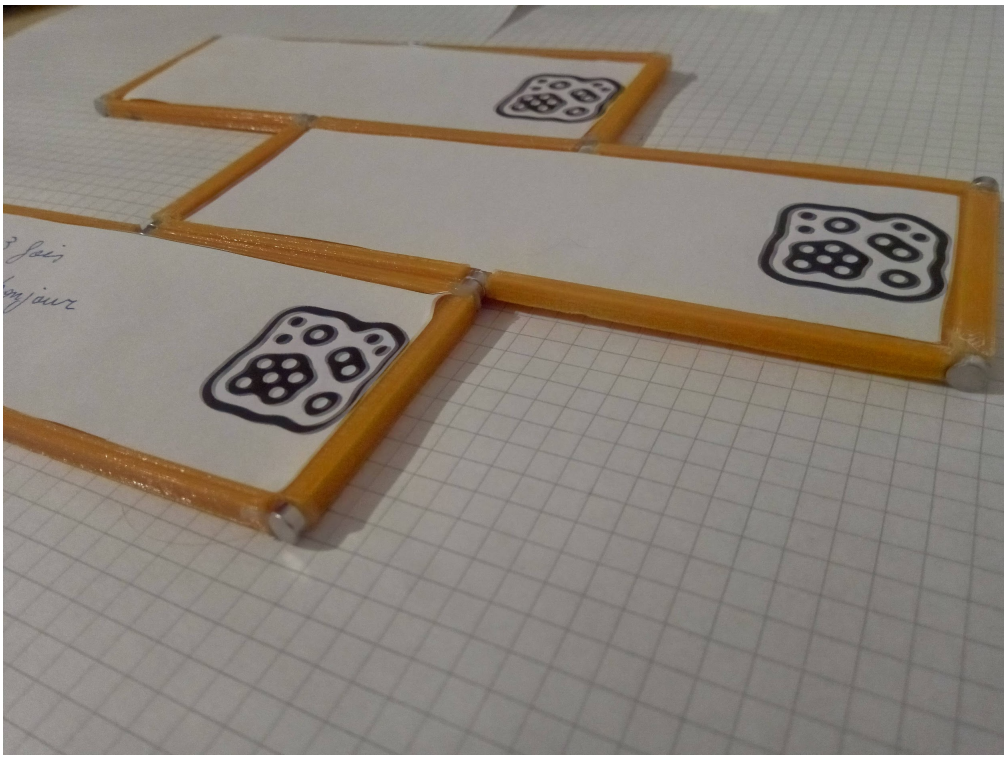


FIGURE A.2 – Les blocs du second prototype, reproduisant une indentation. Le code n'est pas présent sur l'étiquette et le même tag reacTIVision est employé sur les trois.

ces aimants doit permettre d'assembler les blocs, mais doit aussi permettre de les détacher facilement. C'est pour cette raison que la puissance magnétique la plus faible (N35) a été sélectionnée. Un bloc contient alors six aimants (trois en haut et trois en bas), collés à la colle chaude dans des encoches prévues à cet effet. Ces six aimants doivent permettre l'indentation du code. Pour ce faire, seuls deux aimants sur les trois sont utilisés pour effectuer l'assemblage. La polarité des six aimants d'un bloc est orientée dans la même direction. Cela permet à n'importe quel bloc de s'attacher avec un autre, et empêche aussi qu'un bloc soit attaché à l'envers. Il faut donc également que les blocs soient dans la même direction avant d'y apposer les étiquettes de code dessus, car sinon ils seraient impossibles à assembler.

## A.4 Analyse

Lorsque le second jeu de blocs a été produit, un essai de ce nouveau matériel a été effectué avec les experts. À nouveau, une liste de points forts et de points faibles a été produite. Les points forts sont les suivants :

- La prise en main est meilleure que les précédents et la manipulation est facile. On les assemble et désassemble facilement.
- L'utilisation d'un seul modèle de bloc rend la production plus facile, un bloc peut plus facilement être remplacé et devient davantage polyvalent.
- L'indentation magnétique est originale et un point fort de cette version.
- L'espace utile consacré au code est suffisamment grand, même pour un code sur plusieurs lignes.
- Le fait de pouvoir replacer les étiquettes limite le nombre de blocs nécessaires, et rend l'outil plus adaptable.

Certains points négatifs sont cependant ressortis, à savoir que les aimants ne tiennent pas bien en place (la colle chaude n'est pas suffisante pour les maintenir lorsque deux blocs sont désassemblés plusieurs fois) et que les étiquettes ont tendance à se plier et à sortir de leurs blocs. Ces points négatifs ne justifiaient cependant pas la réalisation d'une nouvelle version du design des blocs, car ils pouvaient être réglés via un remplacement de la colle et un changement de la densité du papier. Une idée supplémentaire fait aussi son apparition, à savoir qu'il faudrait pouvoir contraindre l'étudiant à manipuler des assemblages de blocs. En effet, étant donné que deux blocs sont nécessaires pour effectuer une indentation, il est actuellement impossible d'obliger l'étudiant à prendre un assemblage de condition/code prédéfini ce qui limite les possibilités d'exercices.

Après discussion avec un expert, si le matériau d'impression devait être remplacé, celui-ci devait être similaire au niveau de ses propriétés physiques, mais surtout au niveau de sa couleur. En effet, mélanger des blocs de deux

couleurs différentes aurait été déstabilisant pour les étudiants, car ils pourraient y voir une information qu'on tente de lui faire parvenir (type d'instruction différent par exemple). Or, tous les blocs doivent être identiques, puisqu'ils sont censés être interchangeables. Étant donné qu'il était impossible de continuer d'utiliser le matériau actuel pour les raisons précédemment expliquées, et parce qu'il était impossible d'obtenir un filament de même couleur, les blocs ont dû être complètement réimprimés. Cette nouvelle production a été aussi l'occasion d'améliorer la surface d'attache pour les aimants, moyennant de petites modifications du modèle qui sera alors le modèle final.

## A.5 Problèmes d'impression

Bien que le design du second modèle de blocs soit fonctionnel et prêt pour une production à plus grande échelle, plusieurs problèmes de fabrication des blocs rendent la production difficile. La haute température d'impression du matériau utilisé (260°C) crée, par différence de température, un effet de warping (Fig. A.3). Le warping est, tel qu'expliqué par Alsoufi et Elsayed [1], une déformation d'un élément imprimé en 3D causé par une trop grande différence de température lors de l'impression. Le warping est influencé par plusieurs paramètres, dont la différence de température avec l'environnement ambiant et la surface de la pièce imprimée. Les nouveaux blocs sont beaucoup plus larges que les précédents, et la température plus élevée. Cela a pour conséquence de faire échouer plusieurs impressions qui se décollent du plateau d'impression à cause de la déformation. Une tentative d'amélioration de l'adhésion au lit d'impression via réduction de l'espace entre la tête d'impression et le lit est alors réalisée. L'impression est bonne, mais le bloc a fusionné avec le lit et l'endommage. Enfin, la trop haute température d'impression du matériau utilisé fait fondre le tube interne en Téflon de l'imprimante 3D. Celle-ci est alors hors d'usage, et de nouvelles pièces doivent être commandées. Un nouveau matériau d'impression doit également être envisagé étant donné que l'actuel est trop difficile à utiliser.

Plusieurs colles sont testées pour faire tenir les aimants. En effet, les deux matériaux ne sont pas faciles à assembler. L'encoche, de la forme d'un parallélépipède rectangle, épouse mal la forme de l'aimant. Les coins des blocs, endroits où les aimants ont le plus tendance à se détacher, n'offrent que deux points d'attache entre le bloc et l'aimant (Fig. A.3), ce qui est insuffisant pour une manipulation prolongée.

Deux logiciels de modélisation ont été utilisés pour réaliser les prototypes. En effet, le logiciel open source FreeCAD [1] utilisé au départ connaît des problèmes de

1. https://www.freecadweb.org/

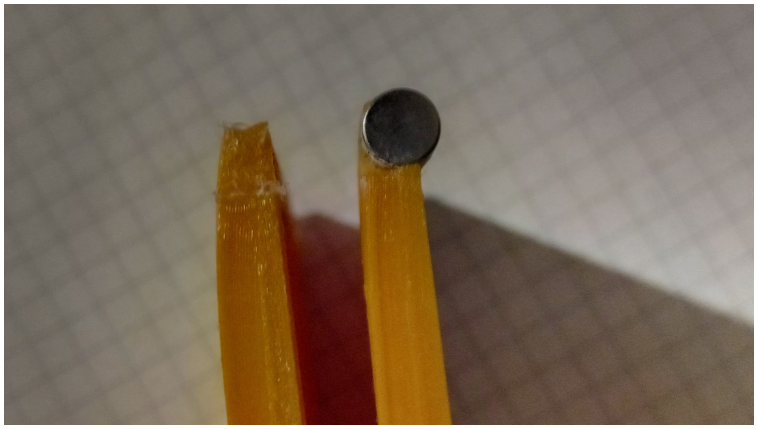


FIGURE A.3 – À gauche, un bloc présentant un léger warping (on remarque que l'extrémité du bloc est trop fine). À droite, la faible surface de contact entre le bloc et l'aimant.

stabilité qui interrompent son fonctionnement de manière aléatoire. Le programme Autodesk Fusion360 [2] a été utilisé. Il ne s'agit cependant pas d'un logiciel open source : son utilisation commerciale est payante, mais une période d'un an d'utilisation est accordée aux particuliers et aux étudiants.

2. https ://www.autodesk.com/products/fusion-360/overview

# Annexe B

# Représentation du schéma DB de l'application

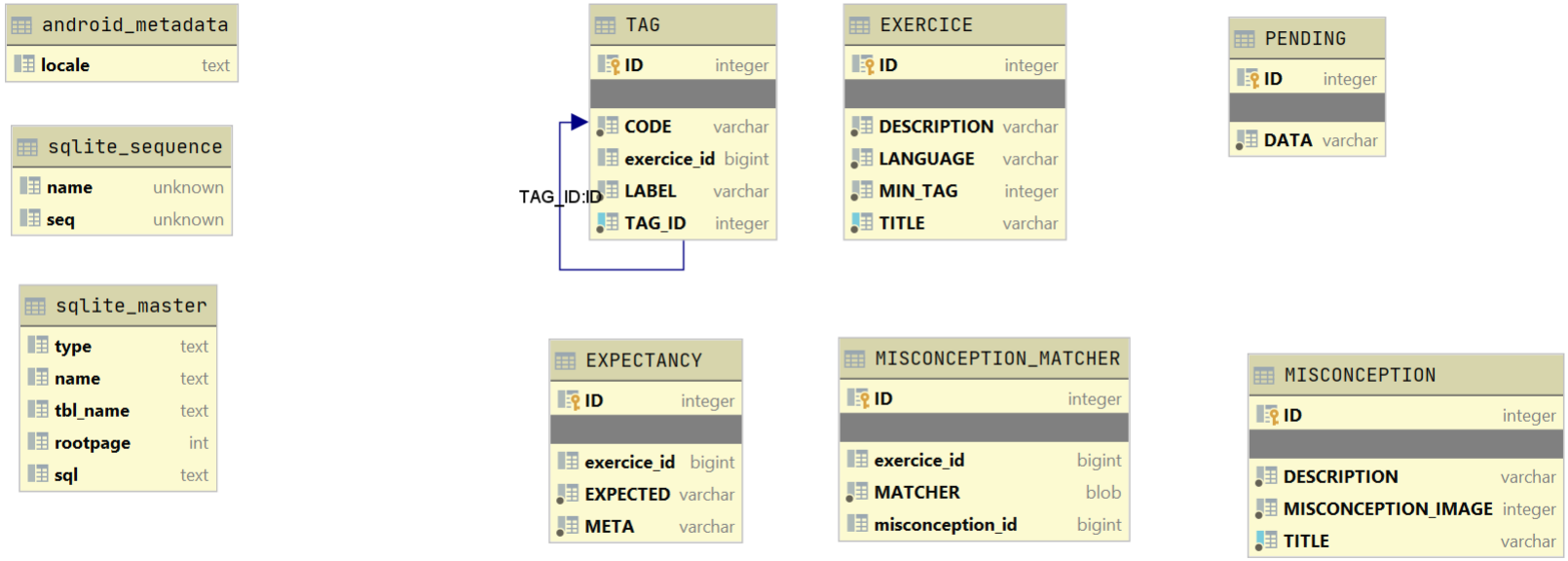



FIGURE B.1 – Schéma de la base de données de l'application, généré avec DataGrip. S'agissant d'une base de données SQLite, les références ne sont pas représentées. Les trois tables à gauche sont des tables internes à Android.

# Annexe C

# Représentation du schéma DB du serveur

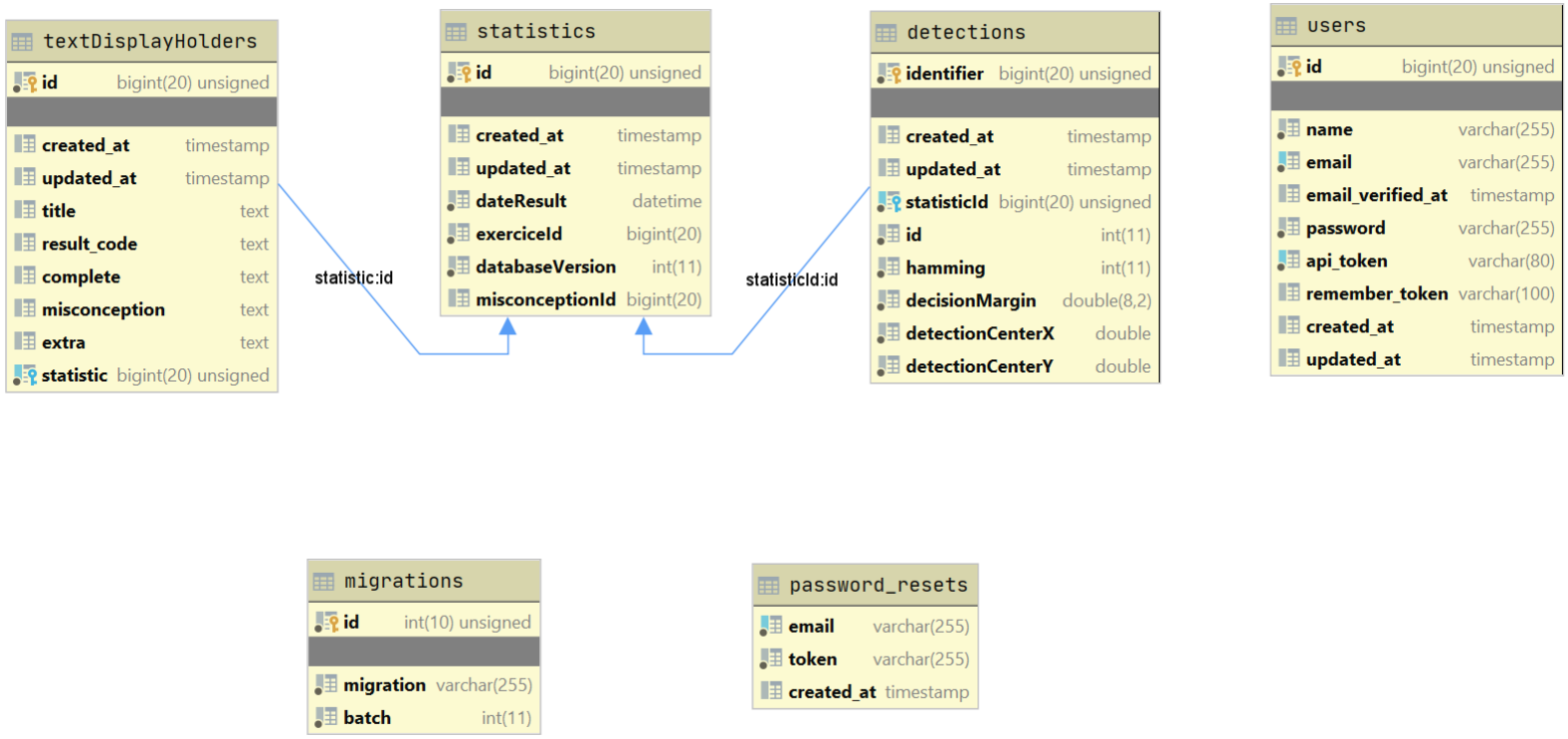


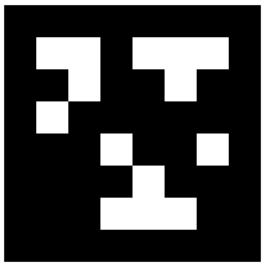

FIGURE C.1 – Schéma de la base de données du serveur, généré avec DataGrip. Les statistiques sont stockées dans les tables textDisplayHolders, statistics et detections. La gestion des accès est faite dans la table users (token) et password_resets. La table migration est une table interne à Larabel, utilisée pour effectuer des mises à jour du schéma.

# Annexe D

# Étiquettes des exercices en pseudo-code

# Exercice 1

```
entier a = 3
```

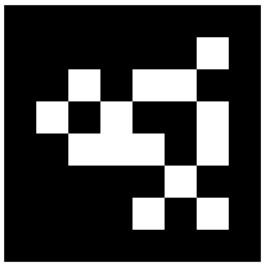


```
entier b = 2
```

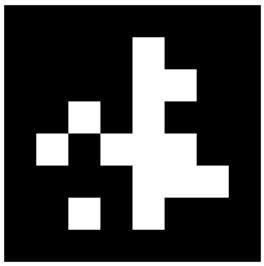
```
entier c = 1
```

```
a = b
```

b = a



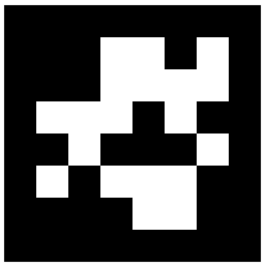
a = c



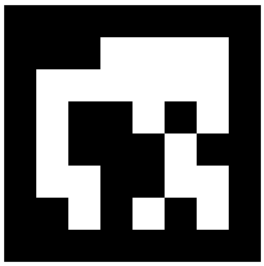
c = a

b = c

c = b



entier 3 = a



entier 2 = b

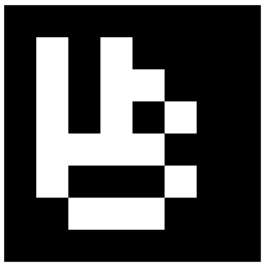
entier 1 = c

# Exercice 2

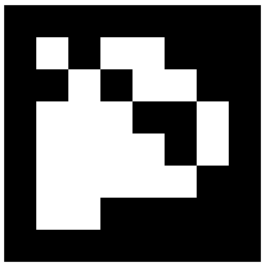
```
si a < b alors
```

```
si a > b alors
```

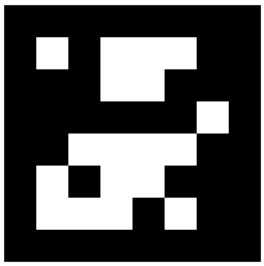
```
si a <= b alors
```



```
si a >= b alors
```



```
entier a = 10
```

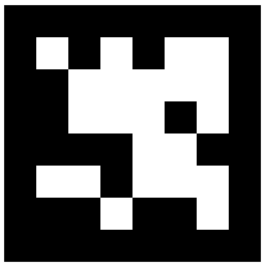


```
entier b = 20
```

```
a = 100
```

```
si a == b alors
```

# Exercice 3



```
booléen y = vrai

booléen n = faux
```

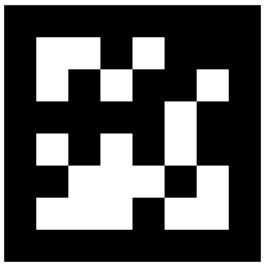
. . .

```
si n ET y alors
```

. . .

```
a = 100
```

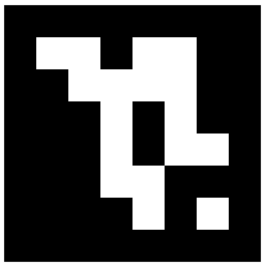
. . .



```
si n OU y alors
```

. . .

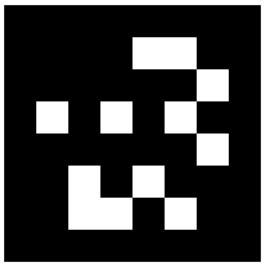
```
a = 42
```



```
entier a
```

```
sinon
```

```
a = 100
```



```
si n alors
```

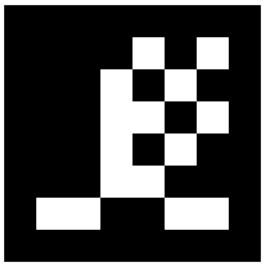
```
a = 100
```

# Exercice 4

```
tant que a < 10 alors
```

```
a = a + 1
```

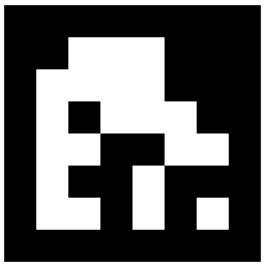
```
entier a = 0
```



```
tant que a <= 10 alors
```

tant que a == 10 alors

tant que a > 10 alors



tant que a >= 10 alors

# Annexe E

# Étiquettes des exercices en langage naturel

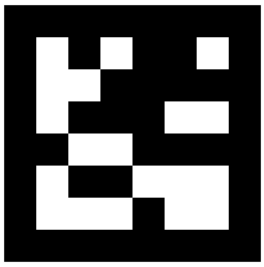
# Exercice 1

je déclare une variable de type entier,

la nomme ’a’ et l’initialise à 3

je déclare une variable de type entier,

la nomme ’b’ et l’initialise à 2

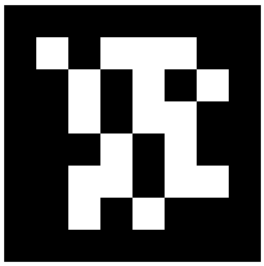


je déclare une variable de type entier,

la nomme ’c’ et l’initialise à 1

je fixe la valeur de la variable ’a’

à la valeur de la variable ’b’

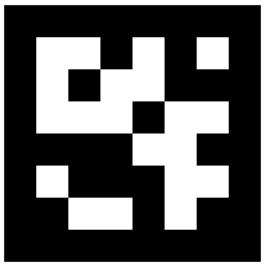


je fixe la valeur de la variable ’b’

à la valeur de la variable ’a’

je fixe la valeur de la variable ’a’

à la valeur de la variable ’c’



je fixe la valeur de la variable ’c’

à la valeur de la variable ’a’

je fixe la valeur de la variable ’b’

à la valeur de la variable ’c’

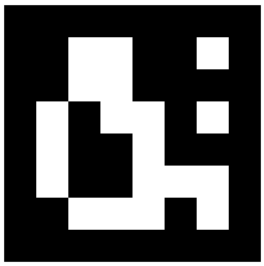
je fixe la valeur de la variable ’c’

à la valeur de la variable ’b’

je déclare 3 de type entier

et l'initialise à la valeur de la

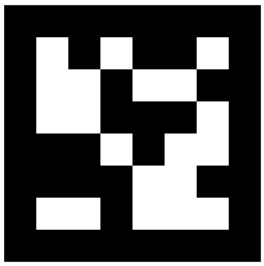
variable a



je déclare 2 de type entier

et l'initialise à la valeur de la

variable b

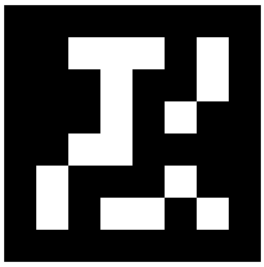


je déclare 1 de type entier

et l'initialise à la valeur de la

variable c

# Exercice 2

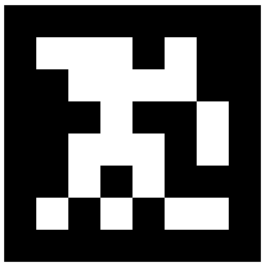


je vérifie si la valeur de la variable a est strictement inférieure à la valeur de la variable b. Si oui, j'exécute la séquence d'instructions suivante :

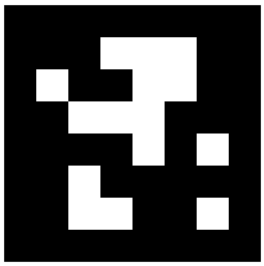
je vérifie si la valeur de la variable a est strictement supérieure à la valeur de la variable b. Si oui, j'exécute la séquence d'instructions suivante :

je vérifie si la valeur de la variable a est inférieure ou égale à la valeur de la variable b. Si oui, j'exécute la séquence d'instructions suivante :

je vérifie si la valeur de la variable a est supérieure ou égale à la valeur de la variable b. Si oui, j'exécute la séquence d'instructions suivante :

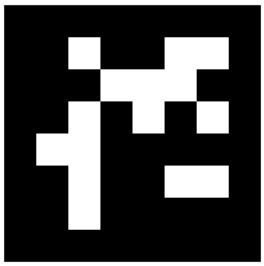


je déclare une variable de type entier,
la nomme ’a’ et l’initialise à 10



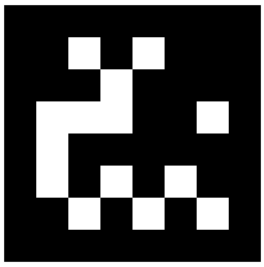
je déclare une variable de type entier,
la nomme ’b’ et l’initialise à 20

je fixe la valeur de la variable ’a’ à 100



je vérifie si la valeur de la variable a est
égale à la valeur de la variable b. Si oui,
j’exécute la séquence d’instructions
suivante :

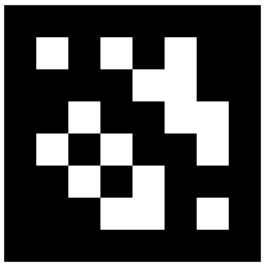
# Exercice 3



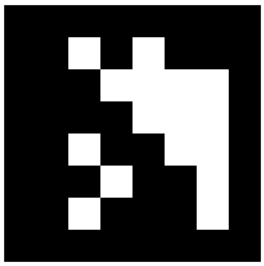
je déclare une variable de type booléen, la nomme ’y’ et l’initialise à vrai et je déclare une variable de type booléen, la nomme ’n’ et l’initialise à faux

je vérifie que la variable n contient la valeur vrai et que la variable y contient la valeur vrai. Si oui, j’exécute la séquence d’instructions suivante :

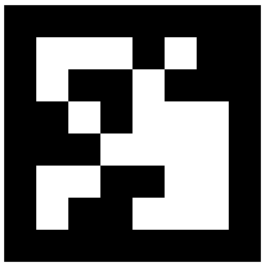
je fixe la valeur de la variable ’a’ à 100



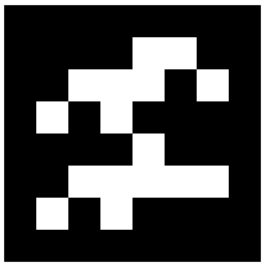
je vérifie que la variable n contient la valeur vrai ou que la variable y contient la valeur vrai. Si oui, j’exécute la séquence d’instructions suivante :



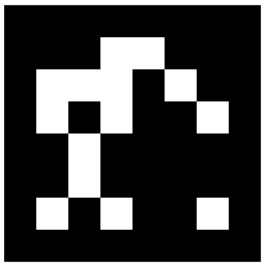
je fixe la valeur de la variable ’a’ à 42



je déclare une variable de type entier
et la nomme ’a’



sinon, j’exécute la séquence d’instructions
suivante :



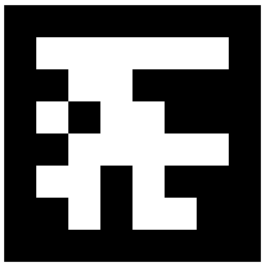
je fixe la valeur de la variable ’a’ à 100

je vérifie que la variable n contient la
valeur vrai. Si oui, j’exécute la séquence
d’instructions suivante :

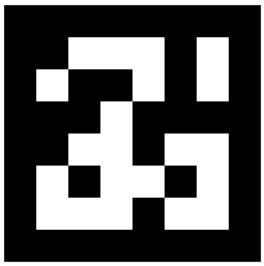
je fixe la valeur de la variable ’a’ à 100

# Exercice 4

tant que la valeur de la variable ’a’ est strictement inférieure à 10, j’exécute la séquence d’instructions suivante :

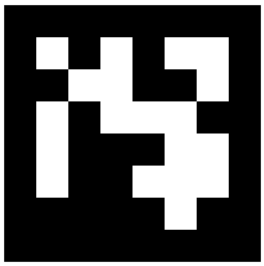


j’ajoute 1 à la valeur de la variable ’a’ et je fixe la valeur de la variable ’a’ à cette nouvelle valeur



je déclare une variable de type entier, la nomme ’a’ et l’initialise à 0

tant que la valeur de la variable ’a’ est inférieure ou égale à 10, j’exécute la séquence d’instructions suivante :



tant que la valeur de la variable ’a’ est égale à 10, j’exécute la séquence d’instructions suivante :

tant que la valeur de la variable ’a’ est strictement supérieure à 10, j’exécute la séquence d’instructions suivante :

tant que la valeur de la variable ’a’ est supérieure ou égale à 10, j’exécute la séquence d’instructions suivante :

# Annexe F

# Protocole de recherche


# Protocole de test de l'outil "Cartorithmique"

21/02/2020

Tom Magis

---

## 1. Pré-requis

### *1.1. Public cible*

Le public visé par le test de cet outil est un public de jeunes étudiants débutants en informatique. Ceux-ci auront eu une explication préalable des concepts de base de la programmation, que l'outil permettra de tester. Le test de cette compréhension des concepts de base leur permettra de prendre conscience des mauvaises représentations qu'ils auraient sur certains concepts.

### *1.2. Temps requis*

En comptant l'accueil, l'explication, le temps de réflexion, le retour utilisateur et la remise en place des blocs, un délai de 20 à 40 minutes sera consacré par test.
Pour permettre une meilleur optimisation du temps, un utilisateur sera placé par exercice.

### *1.3. Environnement*

L'outil est tangible et les blocs mesurent 160mm de long sur 75mm de large. L'exercice ayant le plus de blocs en contient 12, l'exercice en contenant

le moins en contient 6.
Un espace (idéalement une table) permettant de disposer l'entièreté des blocs, de les manipuler et d'en assembler une sélection tout en laissant les autres à l'écart (pour ne pas interférer avec les blocs sélectionnés et assemblés par l'utilisateur) doit être mis à disposition.
Par mesure de gain de temps et pour éviter de mélanger les blocs, un espace indépendant sera consacré à chaque exercice.
Dans la mesure du possible, la séparation des tables ne permettra pas à un utilisateur de regarder la résolution d'un autre exercice qu'un autre utilisateur est en train de résoudre.
Les dimensions optimales sont une table de 90cm sur 140cm. La table ne doit pas être métallique ni magnétique pour éviter d'interférer avec les aimants des blocs, et ne doit pas trop refléter la lumière pour ne pas parasiter la détection des blocs par l'application. La luminosité doit être suffisante pour permettre la lecture par l'utilisateur mais aussi par l'appareil effectuant la lecture des blocs.
Pour éviter toute déconcentration des utilisateur, l'endroit doit idéalement être isolé du passage et calme (par exemple un local dédié).

### *1.4. Matériel*

Les éléments suivants doivent être préparés au préalable :

- Ce protocole.
- Une table comme décrite au point précédent par exercice.
- Les blocs de chaque exercice avec un jeu de cartons-code de rechange.
- Un carton "énoncé" par énoncé, afin que l'utilisateur puisse le relire si nécessaire.
- Un appareil Android (version 7.0 ou supérieur) avec écran tactile, une caméra arrière décente, 1Go d'espace libre et la dernière version de l'application Cartorithmique installée.
- Du matériel d'écriture permettant la prise de note des retours sur l'utilisation de l'application.
- Un dispositif d'enregistrement des sessions d'exercices permettant de déceler après l'utilisation les éventuels problèmes de manipulation de l'outil.
- Une connexion à Internet n'est pas nécessaire pendant les tests, mais doit être réalisée sur l'appareil Android après les tests afin de sauver les statistiques anonymes sur les solutions proposées.

*1.5. Disposition du matériel*

Les tables accueillant chacune un exercice seront séparées afin d'éviter de faire glisser un bloc d'un exercice d'une table à l'autre.
Les blocs seront tous séparés et éparpillés sur la moitié de l'espace de la table. Les cartons-code seront déposés sur les blocs, en prenant garde de respecter la même orientation pour le texte et pour les pôles magnétiques des blocs, afin que ceux-ci puissent s'assembler.

## 2. Étapes du test

- Accueil de la personne
- Explication du contexte du test : Le test porte sur l'essai d'un outil d'aide à l'apprentissage de la programmation par manipulation de blocs prédéfinis pour résoudre un problème.
- Réconfort de la personne : Il est expliqué à l'utilisateur qu'ici, ce ne sont pas ses compétences qui sont testées mais bien l'outil présenté.
- Explication des règles : Il s'agit d'un test dans lequel il faut manipuler des blocs contenant des morceaux de programme en pseudo code. Le programme construit doit respecter une logique de programmation, être syntaxiquement correct et correctement indenté. Les blocs sélectionnés doivent être assemblés (via assemblage magnétique). Les blocs non sélectionnés doivent être mis à l'écart pour ne pas interférer lors de l'analyse. Il est demandé de ne pas prendre les blocs d'un autre exercice, et de ne pas abîmer les blocs ni les cartons-code. Le temps n'est pas limité, ni le nombre de tentatives. Lorsqu'une solution est prête, elle peut être vérifiée via l'appareil Android mis à disposition.
- Vérification de la solution : L'appareil Android sera mis à la disposition de l'utilisateur. Pour valider une solution, il faut avoir dans la vue de la caméra l'ensemble des blocs assemblés (en particulier, le tag à sa gauche). Pour s'assurer de bien scanner tous les blocs, l'écran affiche un nombre qui est le nombre de blocs actuellement détectés. Toucher l'écran mène à l'étape suivante. Il est primordial de tenir l'appareil Android dans le bon sens pour effectuer l'analyse. Cette orientation peut être vérifiée via les éléments textuels présents à l'écran.
- Lecture des résultats : En cas de réussite, un message avertira l'utilisateur qu'il a réussi l'exercice. Il est possible de réussir un exercice sans avoir le "nombre de blocs minimal", l'application avertira alors l'utilisateur qu'il a réussi bien que sans avoir la solution optimale. Si

l'utilisateur n'a pas réussi, il aura alors un message avec le concept de programmation qu'il n'a pas bien acquis ainsi qu'une image explicative du concept en question.
- Nouvel essai : Si l'utilisateur a échoué, il peut après avoir lu le texte d'explication appuyer sur la touche retour de l'appareil pour revenir en mode caméra.
- Retour : Après utilisation de l'outil, quelques questions orales seront posées à l'utilisateur.
- Remerciement de la personne.

## 3. Exercices disponibles

### *3.1. Exercice 1 - Permutation de variable*

- Nombre de blocs : 12
- Énoncé : Réalisez un programme pour que a vaut 2 et b vaut 3.
- Explication : L'utilisateur doit penser à utiliser une variable tampon.

### *3.2. Exercice 2 - Gestion des conditions*

- Nombre de blocs : 8
- Énoncé : Réalisez un programme tel que : si a est plus petit que b, a vaut 100.
- Explication : L'utilisateur doit comprendre la condition "si" et les opérateurs de comparaison.

### *3.3. Exercice 3 - Gestion des conditions 2*

- Nombre de blocs : 10
- Nombre de verrous : 8
- Énoncé : Réalisez un programme pour que a vaut 100.
- Explication : L'utilisateur doit comprendre qu'aucune des conditions vraies ne peut l'aider, et qu'il doit utiliser le "sinon".

### *3.4. Exercice 4 - La boucle*

- Nombre de blocs : 7
- Énoncé : Réalisez un programme pour que a vaut 10.
- Explication : L'utilisateur doit choisir la bonne condition de boucle, et comprendre le fonctionnement de la boucle elle même.

# Annexe G

# Questionnaire UEQ

**S'il vous plaît donnez-nous votre avis**

Afin d'évaluer le produit, veuillez remplir le questionnaire ci-dessous. Il se compose de paires opposées de propriétés que peuvent avoir le produit. La gradation entre les contraires est représentée par des cercles. En cochant l'un de ces cercles, vous pouvez exprimer votre approbation d'un concept.

Exemple :

| | | |
|---|---|---|
| Attractif | ○ ⊗ ○ ○ ○ ○ ○ | Rébarbatif |

Cette réponse indique que vous jugez le site plus attractif que rébarbatif.

Répondez le plus spontanément possible.

Veuillez répondre à toutes les questions, même si vous n'êtes pas sûr de l'évaluation.

Il n’y a pas de «bonne» ou de «mauvaise» réponse, seul votre avis compte !

Dites-nous maintenant ce que vous pensez de notre produit (une seule réponse possible par qualité/défaut).

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | |
|---|---|---|---|---|---|---|---|---|---|
| Agaçant | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Agréable | 1 |
| Incompréhensible | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Compréhensible | 2 |
| Moderne | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Sans fantaisie | 3 |
| Appropriation simple | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Appropriation compliquée | 4 |
| Apporte de la valeur | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Peu de valeur ajoutée | 5 |
| Ennuyeux | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Captivant | 6 |
| Inintéressant | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Intéressant | 7 |
| Imprévisible | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Prévisible | 8 |
| Rapide | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Lent | 9 |
| Original | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Conventionnel | 10 |
| Rigide | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Facilitant | 11 |
| Bien | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Médiocre | 12 |
| Compliqué | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Simple | 13 |
| Repoussant | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Attractif | 14 |
| Habituel | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Avant-gardiste | 15 |
| Désagréable | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Agréable | 16 |
| Sécurisant | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Insécurisant | 17 |
| Stimulant | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Soporifique | 18 |
| Répond aux attentes | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Ne répond pas aux attentes | 19 |
| Inefficace | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Efficace | 20 |
| Clair | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Déroutant | 21 |
| Non pragmatique | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Pragmatique | 22 |
| Sobre | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Surchargé | 23 |
| Attrayant | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Rébarbatif | 24 |
| Sympathique | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Inamical | 25 |
| Conservateur | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Innovant | 26 |

# Annexe H

# Représentation visuelle des différents outils tangibles extraite des articles analysés

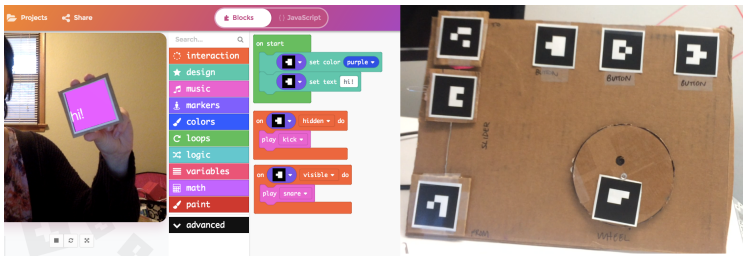



FIGURE H.1 – ARcadia

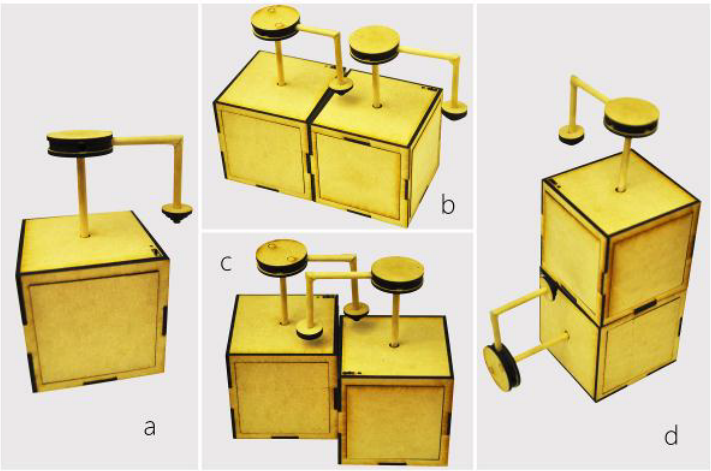



FIGURE H.2 – Algo.Rhythm

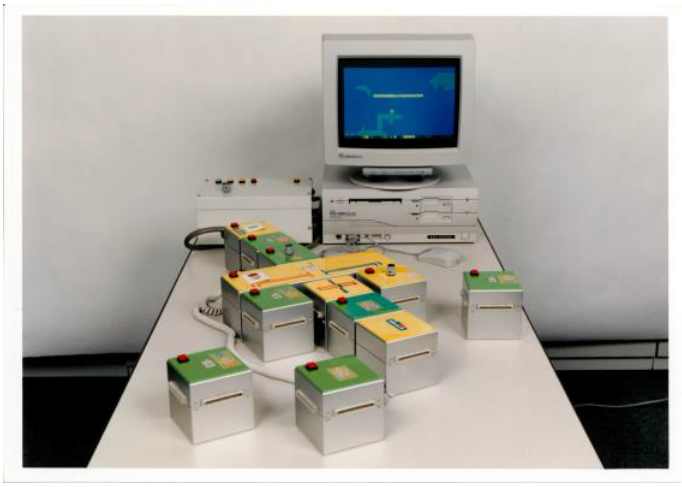


FIGURE H.3 – AlgoBlock

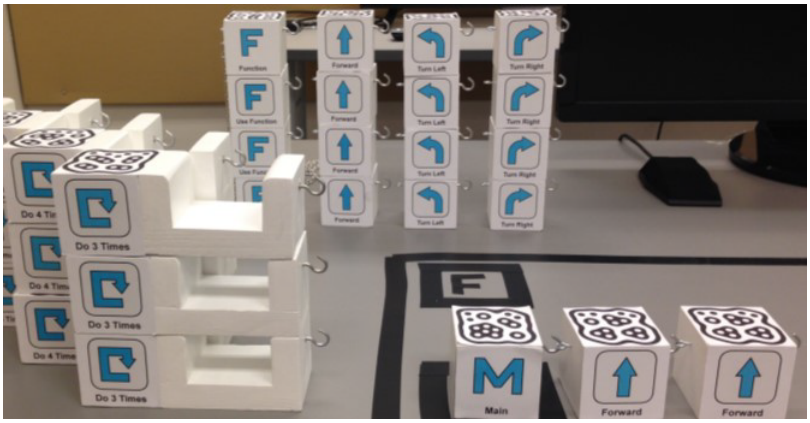



FIGURE H.4 – BOTS & (MAIN)Frames

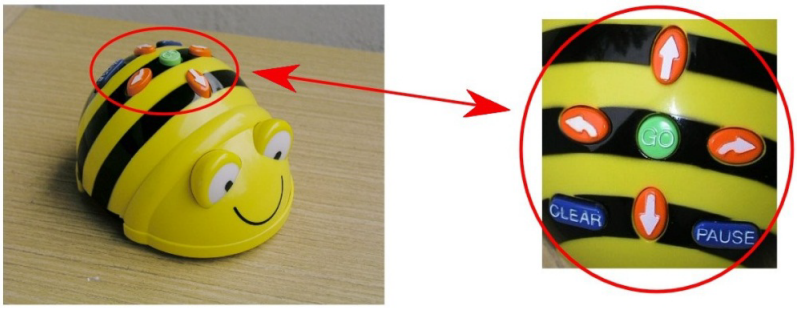



FIGURE H.5 – BeeBot

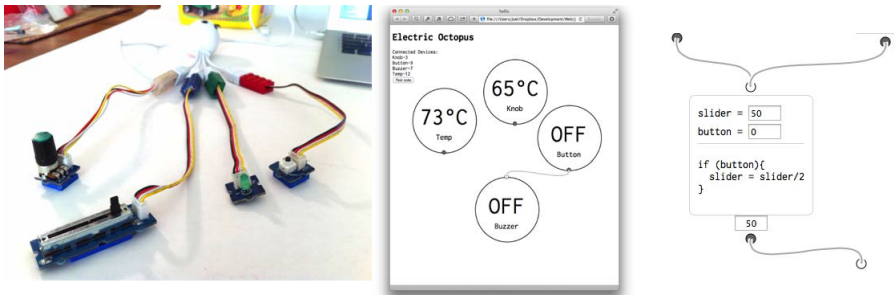



FIGURE H.6 – Bloctopus


FIGURE H.7 – CHERP

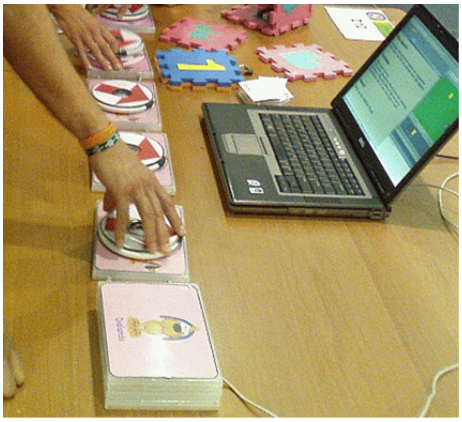


FIGURE H.8 – Dialando

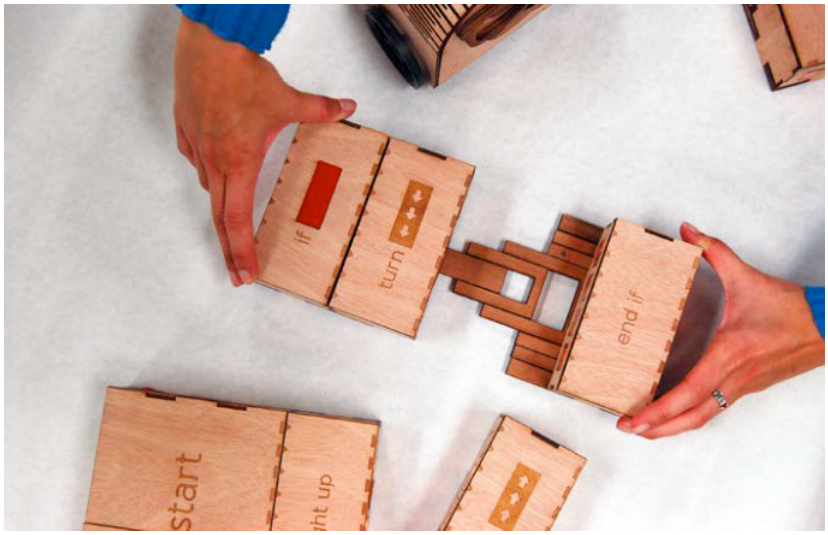



FIGURE H.9 – Dr. Wagon

FIGURE H.10 – E-Block

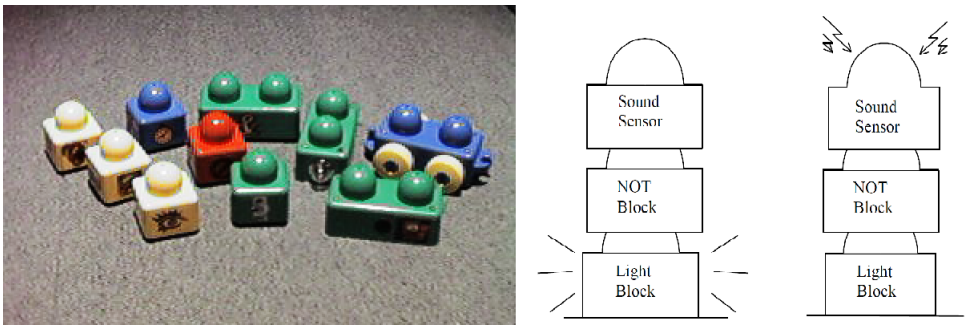



FIGURE H.11 – Electronic Blocks

FIGURE H.12 – FlowBlocks

FIGURE H.13 – GameBlocks


FIGURE H.14 – Kibo

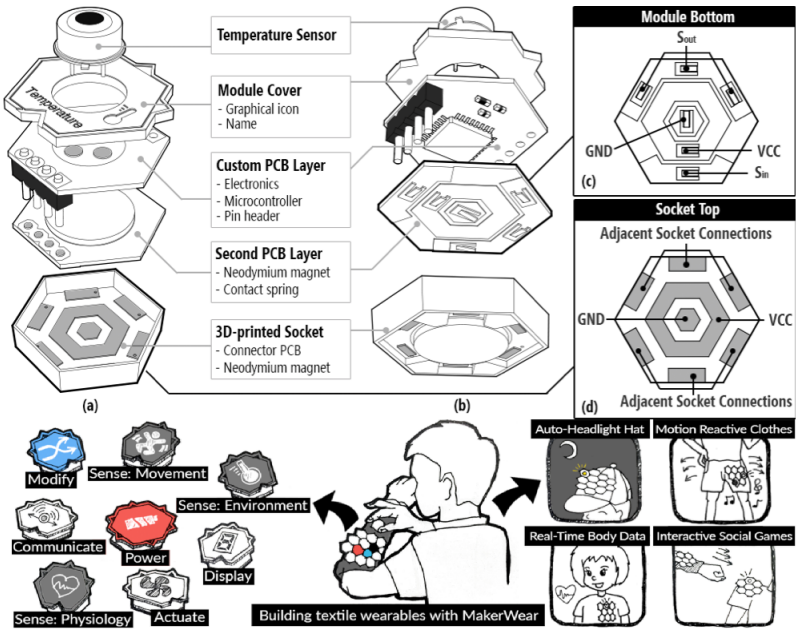



FIGURE H.15 – MakerWear

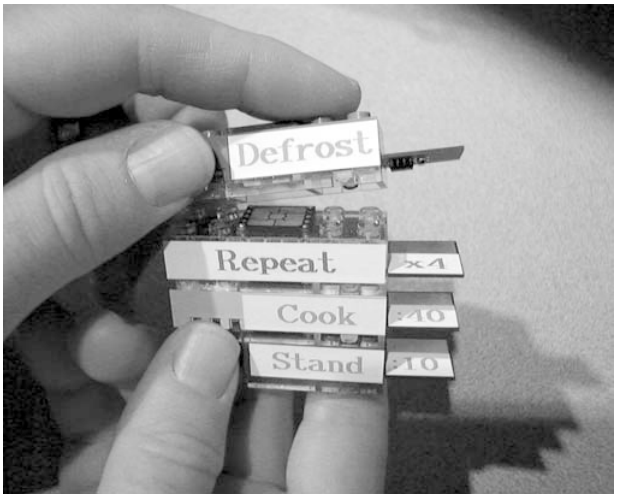



FIGURE H.16 – McNerney's TCB


FIGURE H.17 – Quetzal

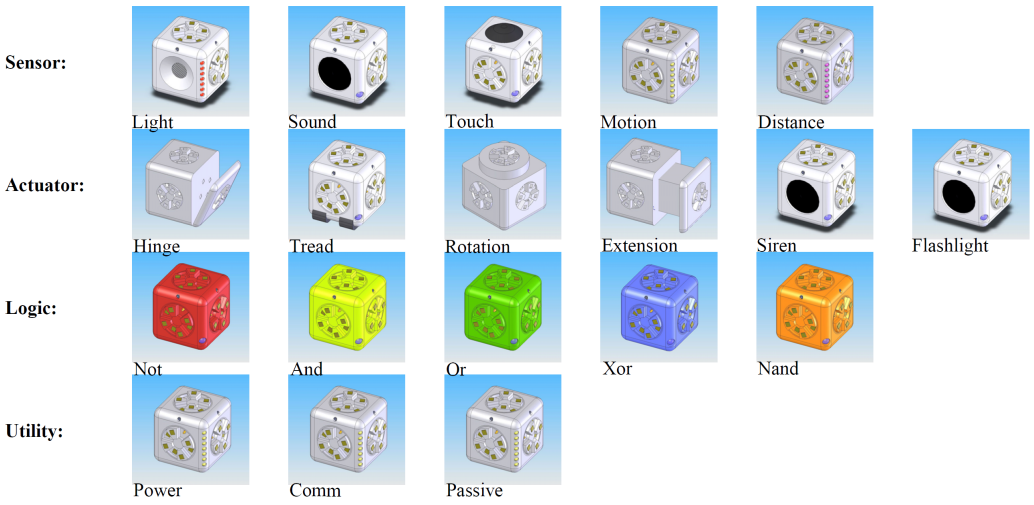



FIGURE H.18 – roBlocks

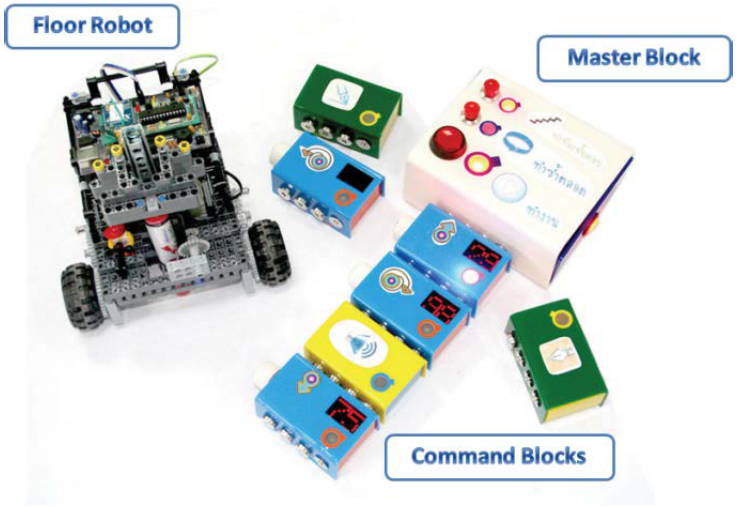



FIGURE H.19 – Robo-Blocks

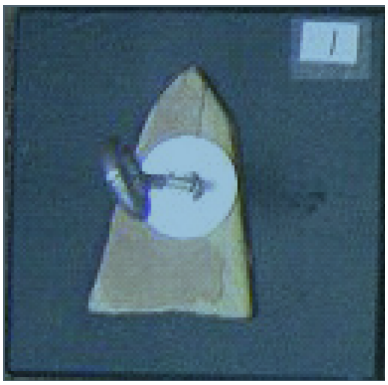


FIGURE H.20 – RockBlocks

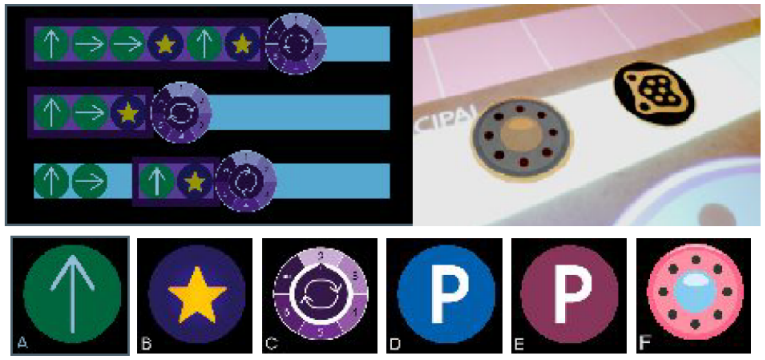



FIGURE H.21 – StarLoop

FIGURE H.22 – StoryBlocks


FIGURE H.23 – Strawbies

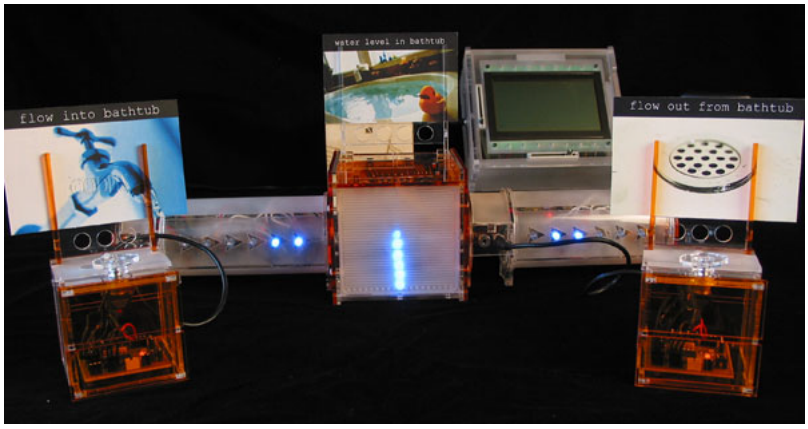



FIGURE H.24 – SystemBlocks

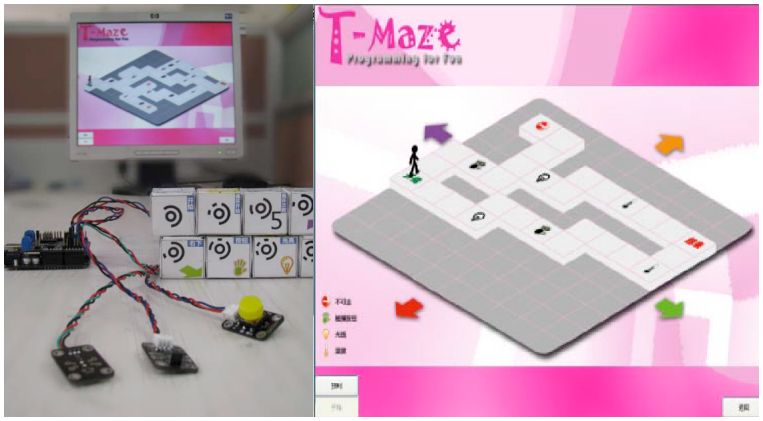



FIGURE H.25 – T-Maze

FIGURE H.26 – T_Butterfly


FIGURE H.27 – T_ProRob

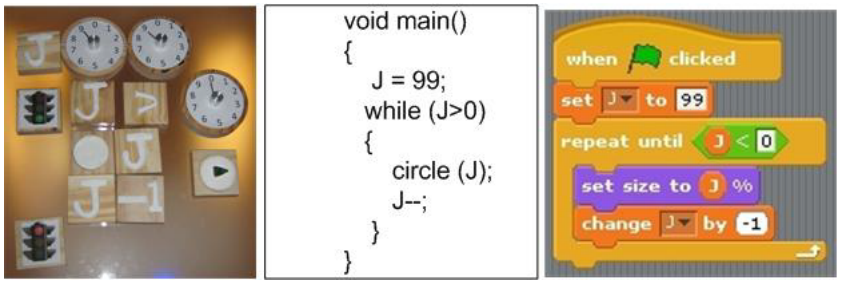



FIGURE H.28 – tactusLogic


FIGURE H.29 – TanPro-Kit

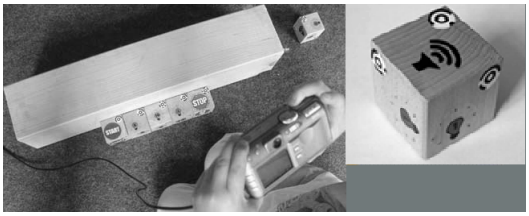


FIGURE H.30 – Tangicons


FIGURE H.31 – Tern

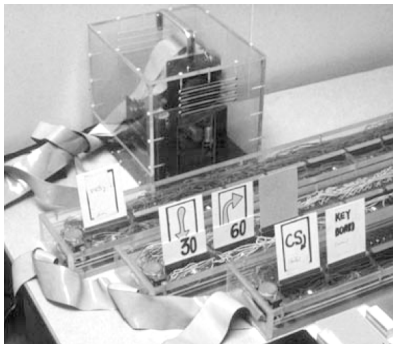



FIGURE H.32 – ToRTis

FIGURE H.33 – Topobo

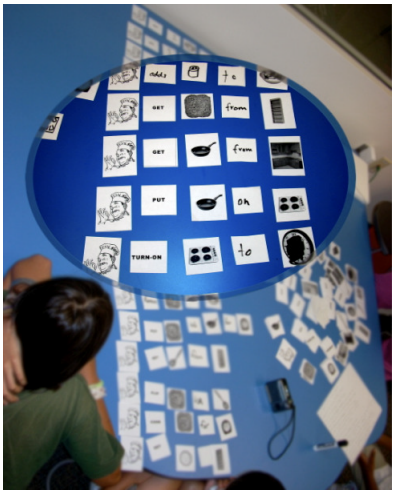



FIGURE H.34 – Toque

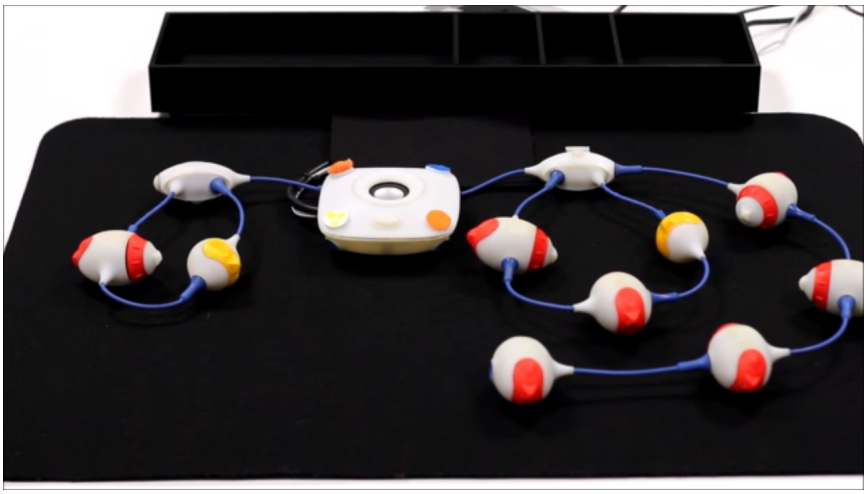


FIGURE H.35 – Torino

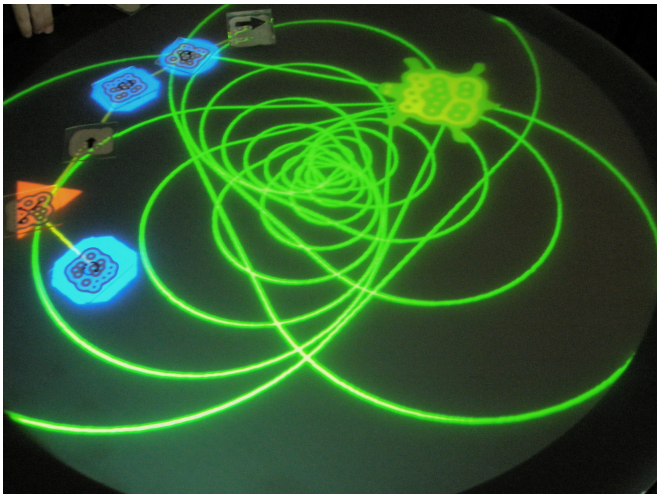


FIGURE H.36 – TurTan

FIGURE H.37 – X Card Game