MASTER EN SCIENCES INFORMATIQUES

Quelles caractéristiques d'interfaces utilisateur tangibles permettent d'apprendre la programmation à des novices ?

Rajaona, Andry

Award date:
2020

Awarding institution:
Universite de Namur

Université de Namur
Faculté d'informatique
Année académique 2019–2020

Quelles caractéristiques d'interfaces utilisateur tangibles permettent d'apprendre la programmation à des novices ?

Andry Rajaona

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)
Bruno Dumas

Co-promotrice : Julie Henry

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Remerciements

Je voudrais remercier mon Promoteur de mémoire, Monsieur Bruno Dumas, pour sa disponibilité et ses précieux conseils afin de me guider dans la direction de mon travail et dans ma réflexion.

Je remercie également ma Co-promotrice, Madame Julie Henry, pour le temps qu'elle a consacré pendant les différentes réunions afin de discuter sur l'avancement de mes recherches, de mon travail et les différents points à approfondir.

Je remercie particulièrement ma grande sœur pour tout ce qu'elle a fait afin que j'aie pu faire des études et pour son temps passé à la relecture de ce travail.

Résumé

La compréhension de la programmation n'est pas un exercice facile pour les novices. C'est pour cette raison que plusieurs chercheurs s'intéressent à son apprentissage. Une étude a été faite sur l'apprentissage de la programmation avec des interfaces utilisateur tangibles destinées aux moins de douze ans. Ce travail s'en est suivie afin de répondre à la question : « Quelles caractéristiques d'interfaces utilisateur tangibles permettent d'apprendre la programmation à des novices ? ». Ainsi, vingt interfaces tangibles de programmation ont été sélectionnées et ont ensuite été réparties dans des différentes classifications : classification de Fishkin, classification de Bodart, classification de Palaude et une classification sur les contraintes physiques. Sur la base de ces classifications, des analyses ont été réalisées pour mettre en évidence les tendances. Les tendances constatées concernent la présentation des concepts de base de la programmation, la forme et l'utilisation du dispositif physique. A partir de ces analyses, une nouvelle interface de programmation est proposée avec une validation qui démontre son utilisabilité et son utilité.

Table des matières

1. Introduction

Depuis des décennies, l'enseignement de la programmation à des novices fait l'objet de nombreuses recherches. Les difficultés éprouvées par les étudiants dans l'apprentissage de la programmation sont soulignées unanimement par la littérature dans le domaine et constituent encore aujourd'hui une problématique majeure. En effet, l'enseignement de la programmation est en passe d'être réintroduit dans les cursus scolaires du fondamental et du secondaire en Belgique francophone, comme cela se fait ailleurs dans le monde. Il convient dès lors de se pencher sur les méthodes d'enseignement de cette matière et, plus spécifiquement, sur les outils d'aide à son apprentissage.

Actuellement, la communauté française de Belgique a opté pour une introduction timide, motivée par l'équipement des écoles à travers notamment les projets École Numérique et leurs packages "découverte des principes de la programmation" (Thymio et Makeblock). Si l'enseignement de la programmation au moyen de micro-ordinateurs et de robots est souvent mis en avant, il existe une pléthore d'outils d'aide à l'apprentissage de la programmation pour les novices. Parmi ces outils, nous pouvons citer les interfaces tangibles de programmation, interfaces utilisateur tangibles (IUT) destinés à l'apprentissage de la programmation.

Si de nombreuses interfaces tangibles de programmation existent pour les jeunes enfants de moins de douze ans, peu d'exemples sont trouvés dans la littérature pour les jeunes plus âgés. Pourtant, elles pourraient constituer un moyen efficace de manipuler et de comprendre les concepts de base de la programmation. En effet, une IUT est constistuée d'objets physiques, représentant des informations, pouvant être manipulés par les mains. Dans le contexte de la programmation, les objets physiques représentent les instructions et, dans certains cas, les concepts de base en programmation constituant les instructions.

Sur ce sujet, une étude a été déjà faite sur les interfaces tangibles de programmation pour les jeunes enfants de moins de douze ans par conséquent, faire une étude sur le même sujet pour les jeunes de plus de douze ans était logique. Par rapport à cela, une question se pose « Quelles caractéristiques d'interfaces utilisateur tangibles permettent d'apprendre la programmation à des novices ? » Pour être plus précis, cette question est partagée en quatre sous-questions :

- Quelle autre taxonomie en plus des taxonomies existantes permet de décrire et d'analyser les interfaces tangibles de programmation ?
- La programmation tangible est-elle destinée seulement aux jeunes enfants de moins de douze ans ?
- Quelles sont les tendances que nous pouvons observer dans le domaine des interfaces tangibles de programmation ?

— Sur la base des taxonomies et des tendances, quel pourrait être le prototype d'interface tangible de programmation pour les plus âgés ?

Chaque sous-question sera discutée en détail dans la partie qui explicite la méthodologie et la question de recherche.

Afin de répondre aux différentes sous-questions et à la problématique elle-même, nous allons dans la section suivante, discuter l'existant et établir un état de l'art du domaine. En suite, la méthodologie de recherche sera décrite, suivie de la présentation des interfaces tangibles sélectionnées. Celles-ci seront alors organisées selon différentes classifications. Des analyses seront menées afin d'apporter des réponses. Sur base des résultats, une nouvelle interface tangible de programmation sera proposée et évaluée. Enfin, nous allons tirer une conclusion du travail et définir des travaux pouvant être effectués dans le futur.

2. Etat de l'art

Les difficultés rencontrées par les novices dans le cadre d'un cours de programmation peuvent être dues à plusieurs facteurs [1]. D'abord la matière elle-même, la programmation est reconnue comme difficile [2] [3] [4]. Ensuite, le choix d'une méthode d'enseignement par l'enseignant [5] [6] et enfin, la motivation des apprenants [7].

De nombreux chercheurs se sont intéressés à la problématique de l'enseignement de la programmation [8] et se sont posés différentes questions : Quoi enseigner ? Comment l'enseigner ? Avec quels outils ? Selon quelles méthodes ?

Tout d'abord, nous allons voir en quoi l'apprentissage de la programmation est difficile. Le choix d'une méthode d'enseignement et la motivation des apprenants ne sont pas discutés parce que les deux facteurs sont liés respectivement à l'enseignant et aux apprenants.

2.1 Difficulté liée à la programmation

Des études et Des enquêtes ont été menées afin de déterminer les raisons sur les difficultés des novices à l'initiation de la programmation.

Les études [9] [10] montrent que les raisons de la difficulté d'apprendre la programmation peuvent être : (a) orientation générale, à quoi servent les programmes et que pouvons-nous en faire, (b) la machine fictive, le modèle de l'ordinateur en ce qui concerne l'exécution des programmes, (c) la notation, la syntaxe et la sémantique d'un langage de programmation particulier, (d) des structures, l'assemblage des éléments du programme. Ces difficultés sont montrées par les erreurs de programmation que les novices font au cours des différents exercices [11]. Ces erreurs peuvent être causées par la mauvaise compréhension des concepts et s'ajoutent aux erreurs liées à la création d'un programme.

Les enquêtes [7] [9] menées sur l'apprentissage de la programmation montrent les ressentis des novices sur leurs expériences au cours d'initiation à la programmation. Le résultat de ces enquêtes a confirmé ce qu'est déjà mentionné auparavant sur les difficultés d'apprendre la programmation : (a) des problèmes lors de la conception d'un programme pour résoudre certaines tâches, (b) l'apprentissage de la syntaxe de programmation, (c) la recherche de bug dans le programme, (d) la difficulté à comprendre le concept de base des structures de programmation.

L'ensemble de ces études et de ces enquêtes font que la programmation est difficile.

Un moyen qui peut aider les novices à réduire ces difficultés est la programmation par blocs [12]. L'utilisation des objets physiques permet de faciliter la compréhension

d'un domaine à un novice. Par exemple, l'utilisation des bâtonnets permet aux enfants d'effectuer les opérations mathématiques plus aisément [13]. Les difficultés liées à la machine, à la notation ou aux structures peuvent être plus faciles à comprendre à travers des objets physiques.

2.2 Interface utilisateur tangible

A la place d'une interface utilisateur graphique (Figure 2.1), une interface utilisateur tangible (IUT) (Figure 2.2) est utilisée pour créer un programme. L'IUT est un terme créé par Ishii et Ullmer [14]. Il s'agit d'une interface homme machine qui permet de manipuler un système informatique au moyen d'objets physiques. Ils représentent et contrôlent en même temps le système informatique. Les IUT visent à résoudre le manque d'interaction physique en tirant parti de la capacité humaine à manipuler des objets et des matériaux physiques [15] [14].

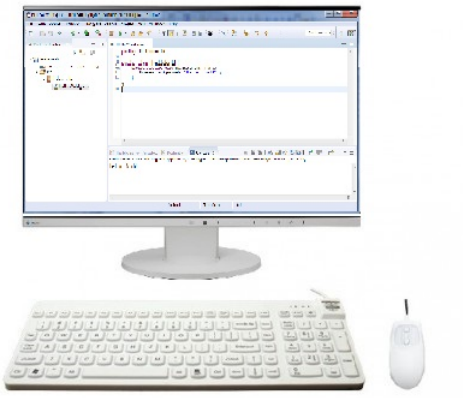


FIGURE 2.1 – Interface utilisateur graphique

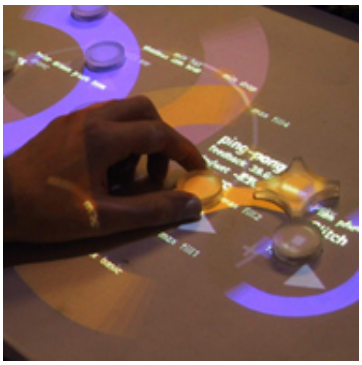


FIGURE 2.2 – Interface utilisateur tangible

Il existe trois types de manipulation possible dans l'IUT : la manipulation par assemblage, la manipulation spatiale et la manipulation basée sur l'appui, la traction, le pincement et autres actions de déformation.

Topobo [16] (figure 2.3) illustre une manipulation par assemblage. Ce type de manipulation permet de construire un système en assemblant différents objets et de rendre ce système programmable.

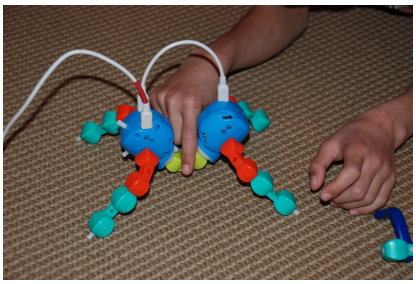


FIGURE 2.3 – Topobo

Querying European Data [16] (figure 2.4) est un exemple de manipulation spatiale. Il est possible d'interagir avec le système en déplaçant des objets sur une table, l'information renvoyée étant fonction du type d'objet et de sa position.

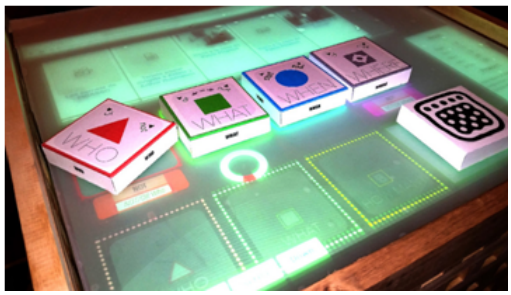


FIGURE 2.4 – Querying European Data

Sandscape [17] (figure 2.5) illustre une manipulation basée sur l'appui, la traction, le pincement et autres actions de déformation. Ce type de manipulation permet de former des données en continu, dans plusieurs dimensions, avec une expérience haptique.

FIGURE 2.5 – Sandscape

La connaissance de ces trois types de manipulation nous permet de distinguer les interfaces tangibles de programmation dans notre recherche. Puis ces interfaces tangibles de programmation vont être classées dans les taxonomies existantes.

2.3 Classifications d'interfaces tangibles de programmation

Les taxonomies existantes sont la classification de Fishkin [18], la classification de Bodart [19], la classification de Palaude [20] et la classification d'Yu [21].

Classification de Fishkin

La classification de Fishkin [18] présente une interface tangible de programmation sur deux aspects : l'incarnation et la métaphore.

L'incarnation décrit la forme de l'interface tangible, elle est caractérisée par la relation entre l'entrée (input) et la sortie (output). Complète, l'input et l'output sont sur un objet par exemple un écran tactile. Proche, l'input et l'output sont étroitement liés par exemple une tablette tactile avec un stylet. Environnementale, l'input et l'output peuvent être sur une seule interface ou séparés, mais l'output est quelque chose que nous sentons par exemples un son, une lumière ou une chaleur. Distante, l'input et l'output sont séparés. Ils se trouvent chacun sur deux objets différents par exemple l'input, le clavier d'un ordinateur et l'output son écran.

La métaphore se porte sur l'action et la relation faites par l'utilisateur à partir de l'input et le résultat que donne l'output. « Est-ce que l'effet d'une action faite par un utilisateur a une analogie avec une action similaire dans la vie réelle ? » Nom, l'exemple est les fichiers sur le bureau d'un système d'exploitation sont similaires aux papiers qui se trouvent sur un meuble bureau. Par contre, certaines analogies ne sont pas représentées, telles que le froissement du papier. Verbe, l'exemple est quand nous travaillons sur un éditeur de texte, nous pouvons choisir la taille des caractères et c'est pareil quand nous écrivons sur un papier. Nom et Verbe, l' exemple est dans la fenêtre d'un système d'exploitation, nous pouvons « glisser et déposer » un document dans la corbeille, comme dans le monde réel où nous jetons un papier dans un bac. Complète, il n'y a pas d'analogie lorsque nous manipulons l'interface, cela produit le même effet dans le monde réel.

	Incarnation				Métaphore			
	Complète	Proche	Environnementale	Distante	Nom	Verbe	Nom & Verbe	Complète
T Maze				■		■		
E Block			■	■		■		
AR Maze			■	■		■		
Robot Block				■		■		
Dr Wagon				■		■		
StarLoop		■		■		■		
TanProStory				■		■		
Queztal				■		■		
Butterfly				■		■		
Turtan		■					■	
P Cube				■		■		
AlgoBlock				■		■		
Cube Sifteo	■		■			■		
Turtable		■					■	
T ProRob				■		■		
Story Block			■			■		
Note Code			■	■		■		
Tern				■		■		
TaBGo				■		■		
Sheets				■		■		

FIGURE 2.6 – Classification d'interfaces tangibles de programmation selon Fishkin telles que décrite par Bodart [19]

Classification de Bodart

La classification proposée par Bodart [19] pour les dispositifs tangibles de programmation permet de les catégoriser en fonction de l'âge du public cible, des concepts de programmation mis en œuvre et de la description du dispositif physique.

La partie âge du public-cible est divisée en trois catégories : (1) de deux à six ans, (2) de sept à douze ans, (0) plus de douze ans.

Dans la partie concept de programmation, il y a (1) algorithme, (2) séquence, (3) structure conditionnelle, (4) boucle, (5) variable, (6) la fonction.

La partie dispositif physique est partagée en quatre. (1) Tout en un, le dispositif est en un seul élément, (2) Composé. L'input et l'output sont séparés, input : (a) continu : l'utilisation de l'interface tangible de programmation est en continue, (b) discret : l'utilisation de l'interface tangible de programmation est en plusieurs étapes, (c) capteur embarqués : l'interface tangible contient un capteur, (d) caméra : le dispositif physique est composé d'un caméra, (e) microphone : le dispositif physique est composé d'un microphone, (f) autre capteurs, output : (a) led, écran, robot : la sortie est visuelle, (b) haut-parleur : la sortie est audible, (c) dispositifs haptiques.

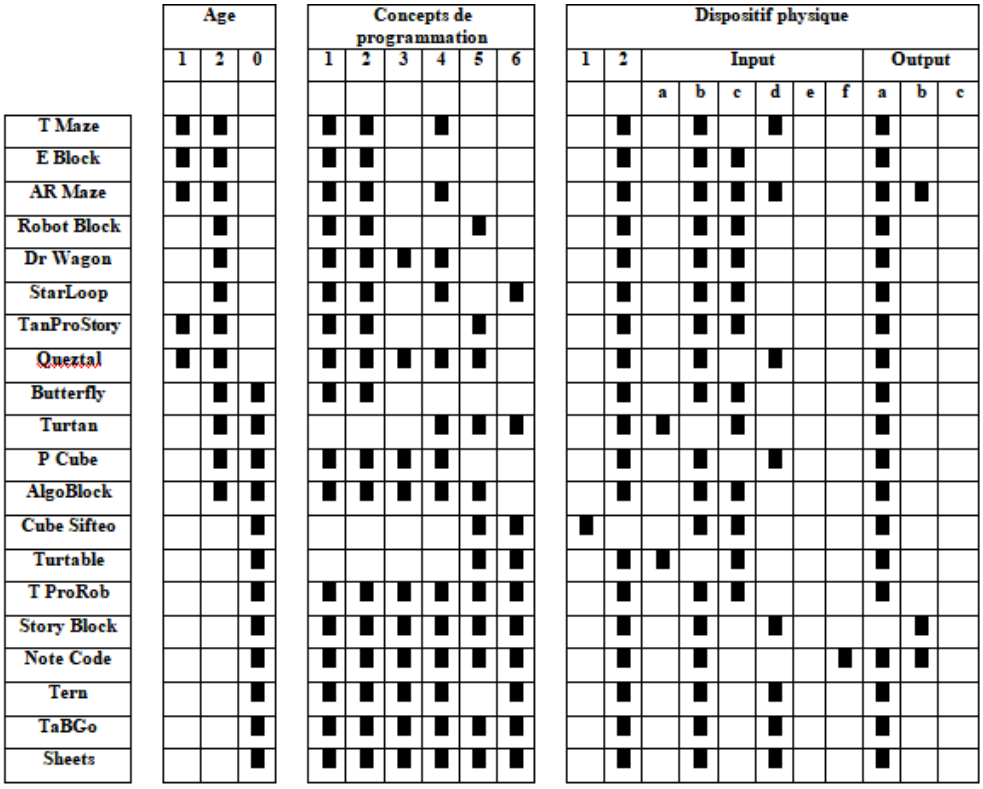


FIGURE 2.7 – Exemple de classification d'interfaces tangibles de programmation selon Bodart [19]

Classification de Palaude

La classification de Palude [20] se base sur l'analyse des symboles présents sur les objets physiques. Elle a été créée pour déterminer l'efficacité, l'utilisabilité et la compréhension des symboles d'une interface tangible de programmation. Deux critères utilisés pour catégoriser les symboles : la signification des symboles (le symbole est significatif s'il donne une indication sur le fonctionnement d'un concept) et la spécification des symboles (le symbole est spécifique s'il n'est compréhensible que dans le contexte de l'interface tangible de programmation).

	Significatif	Non Significatif	Spécifique	Non spécifique
T Maze	■			■
E Block	■			■
AR Maze	■			■
Robot Block	■			■
Dr Wagon	■			■
StarLoop	■	■	■	■
TanProStory		■	■	
Queztal		■	■	
Butterfly		■	■	
Turtan	■			■
P Cube	■			■
AlgoBlock		■	■	
Cube Sifteo		■	■	
Turtable		■	■	
T ProRob	■			■
Story Block		■	■	
Note Code		■	■	
Tern		■	■	
TaBGo		■	■	
Sheets	■	■	■	■

FIGURE 2.8 – Exemple de classification d'interfaces tangibles de programmation selon Palaude [20]

Classification d'Yu

La classification de Yu [21] catégorise les IUT selon leurs caractéristiques physiques en cinq parties : les kits physiques sans électronique (quand l'interface tangible ne contient pas de l'électronique), les kits physique avec de l'électronique (quand l'interface tangible contient de l'électronique), les kits virtuels ou une interface utilisateur graphique, les kits hybrides avec blocs de programmation virtuels et les kits hybrides avec blocs de programmation tangibles.

Les kits hybrides sont constitués de parties physiques et virtuelles.

	kits physiques sans électronique	Kits physique avec de l'électronique	Kits virtuels	Kits hybrides avec blocs de programmation virtuels	Kits hybrides avec blocs de programmation tangibles
T Maze	▮				▮
E Block		▮			▮
AR Maze		▮			▮
Robot Block		▮			
Dr Wagon		▮			
StarLoop		▮			▮
TanProStory		▮			▮
Queztal	▮				
Butterfly		▮			▮
Turtan		▮			▮
P Cube	▮				
AlgoBlock		▮			▮
Cube Sifteo		▮			▮
Turtable		▮			▮
T ProRob		▮			
Story Block	▮				
Note Code		▮			▮
Tern	▮				▮
TaBGo	▮				▮
Sheets	▮				▮

FIGURE 2.9 – Classification d'interfaces tangibles de programmation selon Yu telles que décrite par Bodart [19]

Ces taxonomies vont nous permettre d'analyser les interfaces tangibles de programmation en fonction de plusieurs paramètres. Cependant aucunes d'elles ne parle des contraintes sur leurs utilisations.

2.4 Contraintes des interfaces utilisateur tangibles

Suite à ce manque dans la taxonomie, une étude sur ces contraintes va être réalisée.

Nous pouvons diviser ces contrainte en deux catégories : la contrainte liée à l'utilisation de l'interface utilisateur tangible [16] et la contrainte physique de l'interface utilisateur tangible [22] [23] [24] [25]. Les deux contraintes sont complémentaires et garantissent une meilleure utilisation de l'interface utilisateur tangible.

La contrainte liée à l'utilisation consiste à fournir un manuel d'utilisation de l'interface utilisateur tangible et une explication pour les utilisateurs. L'existence de cette contrainte évite une mauvaise utilisation.

La contrainte physique de l'interface utilisateur tangible est liée à l'input de l'interface utilisateur tangible et permet également de réduire sa mauvaise utilisation. La

contrainte peut être catégorisée en fonction de la position de l'input, à l'orientation de l'input et à sa forme.

— La contrainte liée à la position aide à assembler les dispositifs physiques dans le bon ordre.
— La contrainte liée à l'orientation aide à axer les dispositifs dans le bon sens.
— La contrainte liée à la forme aide à trouver les dispositifs physiques qui s'assemblent.

Avec ces trois catégories de contraintes, nous pouvons proposer une taxonomie en plus.

3. Méthodologie et question de recherche

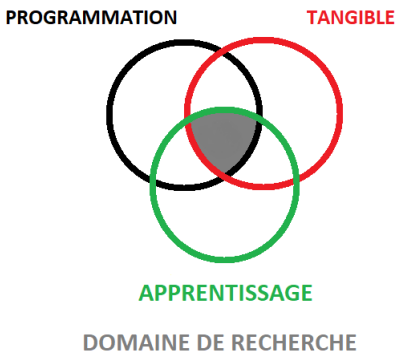


FIGURE 3.1 – Domaine de recherche

Cette recherche constitue une suite au travail effectué par Antoine Bodart [19]. Bien que ce dernier ait déjà sélectionné un ensemble d'interfaces tangibles de programmation, une recherche a été malgré tout effectuée sur les plateformes Google scholar, ACM DL et IEEEXplore. Cette recherche avait comme objectif d'augmenter le nombre d'interfaces à étudier, notamment en ce qui concerne les interfaces destinées aux enfants de plus de douze ans. Une recherche par « effet boule de neige » a également été menée.

L'identification et la sélection d'interfaces tangibles de programmation ont été effectuées en trois étapes en utilisant les mots clés suivants : « programmation tangible », « tangible programming » et « tangible programming tool ».

Dans un premier temps, nous avons recherché des interfaces tangibles de programmation destinées aux jeunes de plus de dix-huit ans et nous en avons trouvée une seule. Ensuite, la recherche a été complétée par des interfaces tangibles de programmation adressées aux jeunes de plus de douze ans qui nous a permis d'en trouver onze. Enfin, nous avons recherché des interfaces tangibles pour les jeunes enfants de moins de douze ans. Pour cette dernière partie, nous en avons trouvé une trentaine.

Afin de limiter le nombre d'interfaces tangibles de programmation utilisées dans ce travail, une sélection a été réalisée selon deux critères. Le premier critère impose aux interfaces tangibles de programmation de présenter au moins deux concepts de programmation. Pour le deuxième critère, la création d'un programme doit être un assemblage de bloc. Au final, vingt interfaces tangibles de programmation ont été retenues.

Le nombre d'interfaces tangibles de programmation pour les plus de douze ans étant peu élevé, nous avons étendu la sélection aux moins de douze ans afin d'obtenir plus d'éléments à discuter.

Sur la base des interfaces tangibles de programmation sélectionnées, nous avons répondu à la problématique « Quelles caractéristiques d'interfaces utilisateur tangibles permettent d'apprendre la programmation à des novices ? ». Pour être plus précis, nous avons répondu aux quatre sous-questions.

Quelle autre taxonomie en plus des taxonomies existantes permet de décrire et d'analyser les interfaces tangibles de programmation ?

Avec les taxonomies existantes, qui comprennent la classification de Fishkin [18], la classification de Bodart [19], la classification de Palaude [20] et la classification d'Yu [21], nous pourrons analyser la relation entre l'utilisateur et le dispositif physique, la relation entre l'input et l'output d'un dispositif physique, l'âge du public cible, les concepts de programmation, le dispositif physique et les symboles d'une interface tangible de programmation. En plus de cela, nous proposerons une autre classification en rapport avec les contraintes des dispositifs physiques.

La programmation tangible est-elle destinée seulement aux jeunes enfants de moins de douze ans ?

Pour répondre à cette question, nous partirons du travail de Bodart [19] « L'utilisation des interfaces tangibles pour l'apprentissage des concepts de programmation chez les jeunes ». Ce travail a été réalisé pour les interfaces tangibles de programmation destinées aux moins de douze ans. Nous focaliserons donc les analyses sur les interfaces tangibles de programmation destinées aux plus de douze ans.

Quelles sont les tendances que nous pouvons observer dans le domaine des interfaces tangibles de programmation ?

Sur la base des interfaces tangibles de programmation sélectionnées et les taxonomies utilisées, nous allons observer et croiser les caractéristiques, qui se généralisent, afin de mette en évidence les tendances privilégiées par les concepteurs.

Sur la base des taxonomies et des tendances, quel pourrait être le prototype d'interface tangible de programmation pour les plus âgés ?

En partant de la tendance et le manque d'interface tangible de programmationpour les jeunes de plus de dix huit ans, nous allons proposer une autre interface tangible de programmation.

4. Présentation des interfaces tangibles de programmation

Après les recherches, vingt interfaces tangibles de programmation sont sélectionnées : AlgoBlock [26], AR Maze [27], Butterfly [28], Cube Sifteo [29] [30], Dr Wagon [31], E Block [32], Note Code [33], P Cube [34], Quetzal [35], Robot Block [36], Sheets [37], StarLoop [38], StoryBlock [39], T Maze [40], T Prorob [28], TaBGo [41], TanProStory [42], Tern [43], Turtan [44], Turtable [45].

Sur base de ces informations, nous allons compléter les différentes classifications et répondre à la problématique. Mais avant cela, nous allons d'abord présenter les vingt interfaces tangibles de programmation sur quatre aspects. Le premier est la tranche d'âge cible de l'interface tangible de programmation, suivi de son dispositif physique : il présente les différents éléments input et output de l'interface tangible de programmation. Après l'utilisation de l'interface tangible de programmation : il montre sa manipulation. Et le concept de programmation, il présente les différents concepts de programmation qu'elle contient.

AlgoBlock [26] (1993)

FIGURE 4.1 – AlgoBlock

Age

L'interface tangible de programmation est destinée pour les jeunes dans les écoles primaires et secondaires.

Dispositif physique

L'interface tangible de programmation est composée de blocs tangibles et d'un ordinateur. Les blocs tangibles sont de type Logos qui contiennent chacun un micro-contrôleur. Ils jouent le rôle de l'input. L'écran de l'ordinateur est l'output qui affiche le résultat.

Utilisation

Les utilisateurs créent un programme en connectant les blocs les uns aux autres. Ensuite, les instructions sont envoyées vers l'ordinateur pour guider le sous-marin sur l'écran.

Les concepts de programmation

L'AlgoBlock montre cinq concepts de programmation. La séquence et l'algorithme sont représentés, vu que l'ensemble des blocs forme l'algorithme pour guider le sous-marin. Chaque instruction est exécutée étape par étape. La conditionnelle et la boucle sont évidentes parce que chaque concept est présenté par un bloc distinct. L'utilisateur peut voir directement leur effet. La variable est mentionnée dans l'article mais le concept n'est pas explicite, même s'il y a des blocs qui peuvent changer de paramètre.

AR Maze [27] (2018)

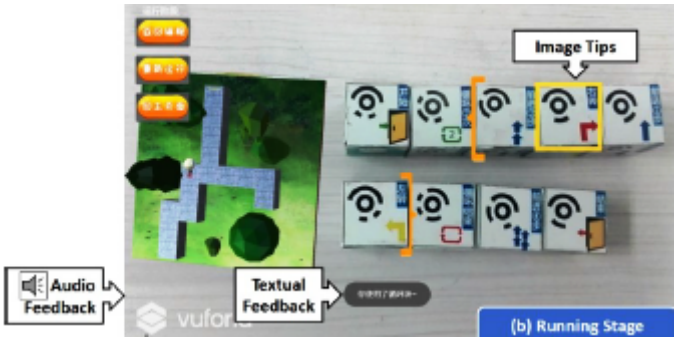


FIGURE 4.2 – AR Maze

Age

L'interface tangible de programmation est destinée aux jeunes enfants.

Dispositif physique

L'interface tangible de programmation est composée de blocs et d'un appareil mobile. Les blocs jouent le rôle de l'input et ils contiennent des capteurs. L'output est un appareil mobile qui affiche le personnage sur un chemin, donnant un retour textuel et émettant un son.

Utilisation

L'utilisateur crée un programme en assemblant les blocs les uns aux autres pour guider le personnage d'un point A au point B. L'exécution du programme se fait sur un appareil mobile. A partir du résultat de l'exécution, l'utilisateur peut débugger son programme s'il rencontre des erreurs.

Les concepts de programmation

L'AR Maze montre quatre concepts de programmation (la boucle, la fonction, la séquence, la gestion d'erreur). Les concepts de programmation ne sont pas difficiles à comprendre. Chaque bloc représente les instructions qu'il peut réaliser.

Butterfly [28] (2011)

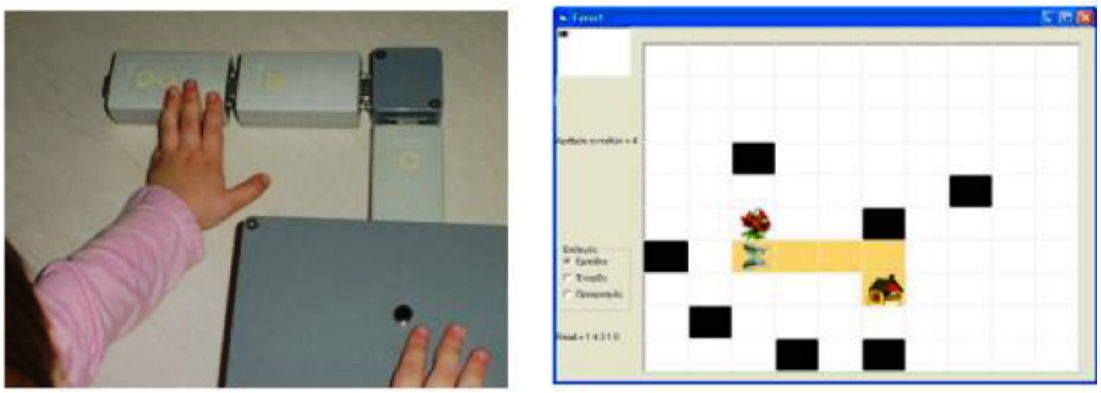


FIGURE 4.3 – Butterfly

Age

L'interface tangible de programmation est destinée aux enfants de onze à quatorze ans.

Dispositif physique

L'interface tangible de programmation est composée de blocs et d'un ordinateur. Les blocs sont les inputs. Ils ont la forme d'un boitier cubique qui contient des microcontrôleurs. Le boîtier est connecté à l'ordinateur via un câble RS232. L'écran de l'ordinateur joue le rôle de l'output.

Utilisation

L'utilisateur crée un programme en connectant les blocs les uns aux autres. L'ensemble des instructions sont traduites par l'ordinateur et exécutées afin de guider le papillon. Chaque instruction est exécutée séquentiellement.

Les concepts de programmation

Butterfly montre deux concepts de programmation : l'algorithme et la séquence. Les concepts de programmation enseignés par l'interface tangible de programmation sont faciles à comprendre car ils ne sont que deux. L'ensemble des blocs forme un algorithme. Chaque bloc est exécuté étape par étape pour guider le papillon.

Cube Sifteo [29] [30] (2012)

FIGURE 4.4 – Cube Sifteo

Age

L'interface tangible de programmation est destinée aux élèves qui ont une connaissance en programmation procédurale.

Dispositif physique

L'interface tangible de programmation est composée de plusieurs périphéries physiques : des cubes et d'un appareil pour placer les cubes. Les cubes jouent le rôle de l'input et de l'output.

Utilisation

L'utilisateur crée une classe à partir des cubes. Exemple, une classe couleur est créée avec différents attributs et différentes méthodes. Pendant l'exécution, le cube est initialisé par un couleur et il peut en changer en fonction des instructions de l'utilisateur.

Les concepts de programmation

Cube Sifteo montre trois concepts de programmation : l'attribut, la fonction et la notion d'objet. Cube Sifteo montre la manière de créer une classe à partir des différents attributs.

Dr Wagon [31] (2013)

FIGURE 4.5 – Dr Wagon

Age

L'interface tangible de programmation est destinée aux enfants de six à douze ans.

Dispositif physique

L'interface tangible de programmation est composée de blocs et d'un robot. Les blocs jouent le rôle de l'input. Ils contiennent des microcontrôleurs. Le robot est l'output.

Utilisation

L'utilisateur crée un programme en connectant les blocs les uns aux autres pour diriger le robot. A l'aide d'un bouton Start, le robot commence à exécuter les commandes. La communication entre le robot et les blocs est une communication sans fil.

Les concepts de programmation

Dr Wagon montre quatre concepts de programmation. Les blocs représentent chacun un concept. Les concepts sont donc faciles à comprendre car l'utilisateur peut voir l'effet des instructions réalisées. Les concepts de programmation sont le concept d'action, la structure conditionnelle et la boucle. Chaque bloc représente une instruction qui est exécutée une par une.

E block [32] (2013)

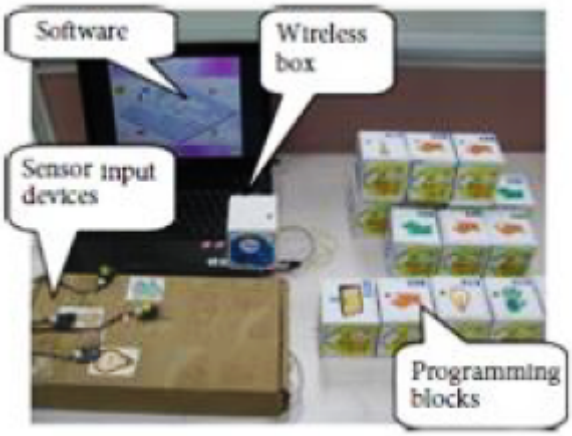


FIGURE 4.6 – E block

Age

L'interface tangible de programmation est destinée aux enfants de cinq à neuf ans.

Dispositif physique

L'interface tangible de programmation est composée de blocs avec un microprocesseur, d'un capteur, d'une communication sans fil et d'un ordinateur avec un logiciel pour E block. Les blocs jouent le rôle d'input. L'output est l'écran d'ordinateur et un système LED.

Utilisation

L'utilisateur crée un programme en connectant les blocs les uns aux autres. La suite d'instruction est envoyée à l'ordinateur pour qu'il la compile et dirige le personnage.

Les concepts de programmation

E block montre quatre concepts de programmation. Elles sont facile à assimiler car les blocs représentent chacun un concept comme la conditionnelle et la boucle. L'algorithme est présenté par l'ensemble des blocs assemblés et qui sont exécutés séquentiellement.

Note Code [33] (2015)

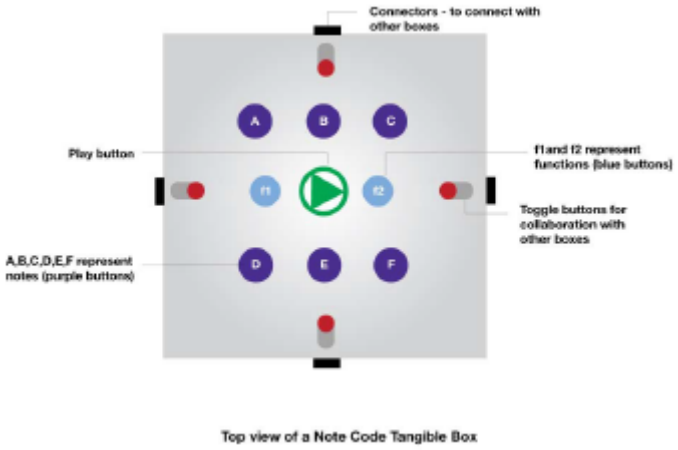


FIGURE 4.7 – Note Code

Age

L'âge n'est pas défini dans l'article mais l'interface tangible de programmation est destinée pour les novices en programmation.

Dispositif physique

L'interface tangible de programmation est composée d'une table interactive contenant des boutons et un écran. Les boutons sont les inputs. L'output est représenté par la mélodie et l'écran. L'écran affiche le déroulement du programme en montrant l'instruction exécutée.

Utilisation

L'utilisateur crée un programme en tapotant sur les boutons. Ainsi, les instructions sont jouées et affichées sur l'écran.

Les concepts de programmation

Les concepts de programmation présentés par l'interface tangible de programmation sont la fonction, la structure conditionnelle, la boucle et la variable. La variable est montrée par les boites qui contiennent une note de musique. La séquence est montrée par l'exécution des instructions étape par étape. L'algorithme est une mélodie créée à partir des boites. Il existe un bouton qui joue la fonction d'enregistrement.

P Cube [34] (2014)

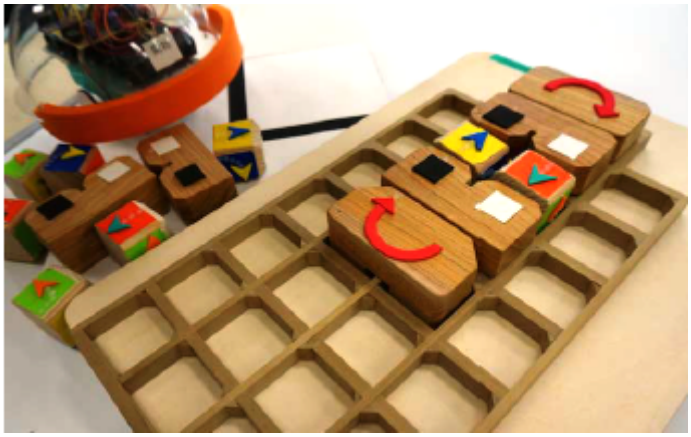


FIGURE 4.8 – P Cube

Age

L'âge n'est pas mentionné dans l'article mais l'interface tangible de programmation est destinée aux enfants.

Dispositif physique

L'interface tangible de programmation est composée d'un robot, d'un tapis, de blocs et d'un ordinateur. Les blocs jouent le rôle d'input, ils sont fabriqués en bois et caractérisés par des motifs. Le tapis permet d'assembler les blocs pour créer un programme. L'output est un robot qui se déplace.

Utilisation

L'utilisateur crée un programme à partir des blocs en les plaçant sur le tapis afin de guider le robot. L'ensemble des blocs sur le tapis est capturé par une caméra puis la photo est compilée par un ordinateur avant d'être exécutée par le robot.

Les concepts de programmation

P Cube montre quatre concepts de programmation. La conditionnelle et la boucle sont faciles à comprendre grâce aux différents blocs et les motifs qui y sont gravés. L'algorithme est l'ensemble des blocs assemblés et qui est exécuté de manière séquentielle.

Quetzal [35] (2007)

FIGURE 4.9 – Quetzal

Age

L'interface tangible de programmation est destinée aux enfants de cinq à neuf ans.

Dispositif physique

Le dispositif est composé de puzzles en plastique, d'une caméra, d'un ordinateur et d'un robot. L'input est représenté par les puzzles. Le robot joue le rôle d'output.

Utilisation

L'utilisateur crée un programme à partir des pièces de puzzles qui sont attachées les unes aux autres. L'ensemble des pièces est ensuite pris en photo et compilé par un ordinateur avant d'être envoyé vers le robot pour être exécuté.

Les concepts de programmation

Quetzal montre six concepts de programmation. Chaque concept est présenté par une pièce du puzzle (la conditionnelle, la boucle). L'utilisateur peut voir à quoi sert chaque pièce du puzzle. Le concept de variable est bien présenté en comptant le nombre de jeton utilisé. L'algorithme est l'ensemble du puzzle qui est exécuté séquentiellement.

Robot block [36] (2012)

FIGURE 4.10 – Robot block

Age

L'interface tangible de programmation est destinée aux enfants de huit et neuf ans.

Dispositif physique

L'interface tangible de programmation est composée de blocs et d'un robot. Les blocs contiennent des microcontrôleurs. Ils se connectent entre eux à l'aide d'une connectique RS-232. La communication entre les blocs et le robot se fait par une communication sans fil. L'input est l'ensemble des blocs et l'output est le robot.

Utilisation

L'utilisateur crée un programme en connectant les blocs les uns aux autres. Puis le programme est envoyé vers le robot afin que le robot l'exécute.

Les concepts de programmation

Robot block montre trois concepts de programmation. La notion de variable est bien montrée par la distance que le robot doit parcourir. L'algorithme est présenté par l'ensemble des blocs qui sont exécutés un par un (séquentiellement).

Sheets [37] (2015)

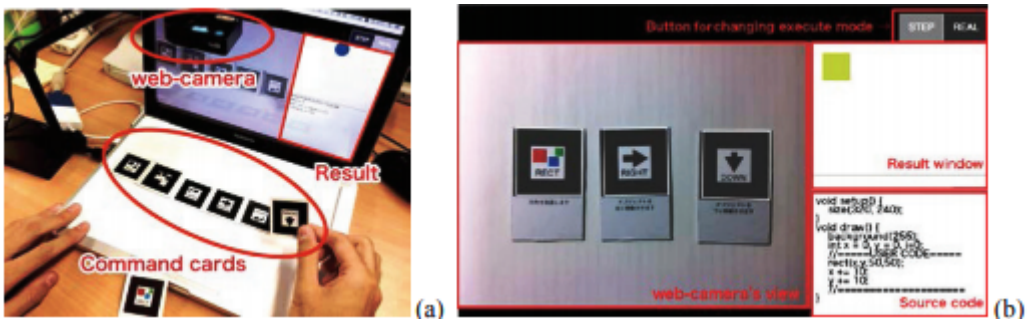


FIGURE 4.11 – Sheets

Age

Pour effectuer l'évaluation de l'interface tangible de programmation, les concepteurs de Sheet ont sélectionnés huit personnes entre 18 et 22 ans.

Dispositif physique

L'interface tangible de programmation est composée de cartes en papier, d'une caméra et d'un ordinateur. Les cartes sont marquées par des motifs et elles jouent le rôle d'input. L'output est l'écran de l'ordinateur.

Utilisation

Un utilisateur crée un programme en alignant les cartes. Les cartes sont prises en photo. La photo est ensuite compilée par un ordinateur afin de la traduire en ligne de code.

Les concepts de programmation

Sheets montre sept concepts de programmation qui sont les variables, les boucles et les branches. Ils montrent aussi le nombre d'exécutions d'une boucle et le changement d'une ou des variables. La fonction est représentée par des cartes personnalisées conçues par l'utilisateur. L'alignement des cartes permet de créer un programme avec une exécution séquentielle.

StarLoop [38] (2017)

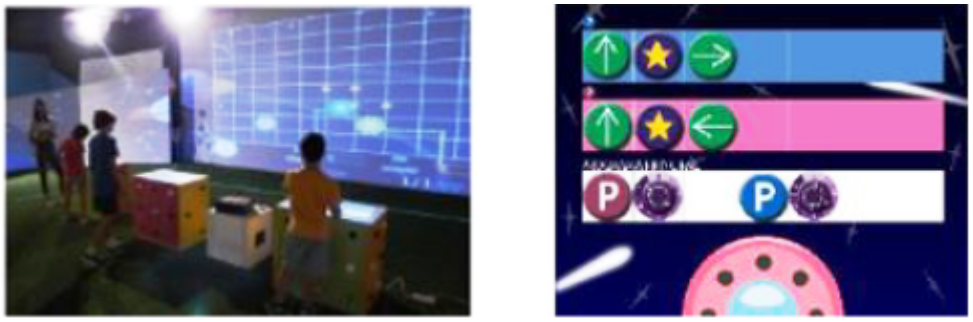


FIGURE 4.12 – StarLoop

Age

L'interface tangible de programmation est destinée pour les élèves âgés de huit à onze ans.

Dispositif physique

L'interface tangible de programmation est composée d'une table interactive et des objets tangibles qui représentent les différentes instructions. L'input est l'ensemble de la table et des objets tangibles. L'output est affiché sur la table elle-même ou sur une projection. Le résultat affiché est le mouvement d'un vaisseau spatial.

Utilisation

L'utilisateur crée un programme en alignant les objets tangibles les uns à la suite des autres sur la table interactive. La table enregistre chaque objet placé pour l'exécuter par la suite.

Les concepts de programmation

StarLoop montre quatre concepts de programmation : l'algorithme, la séquence, la boucle et la fonction. Ils sont faciles à comprendre car chaque objet représente un concept. L'ensemble des objets placés sur la table forme l'algorithme qui est exécuté un par un (séquentiellement).

StoryBlock [39] (2017)

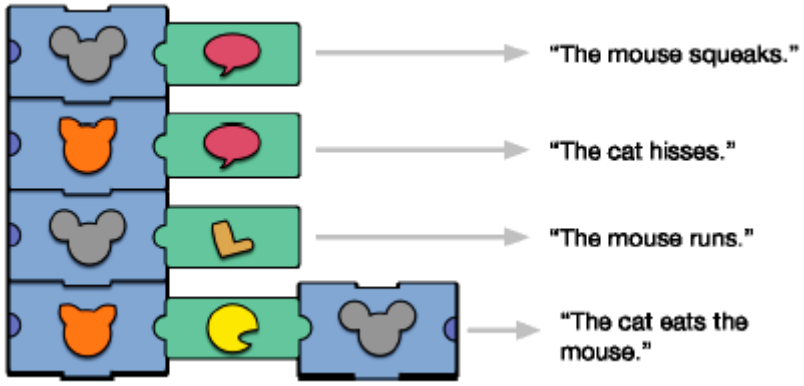


FIGURE 4.13 – StoryBlock

Age

L'interface tangible de programmation est destinée à toute personne qui débute dans la programmation, les enfants inclus.

Dispositif physique

L'interface tangible de programmation est composée de puzzles, d'un appareil mobile qui peut prendre des photos et d'un logiciel pour traduire la photo en audio. L'input est représenté par des puzzles et l'output en forme audio qui raconte une histoire. Les puzzles peuvent être construits soit en carton, soit en bois ou soit en plastique.

Utilisation

L'utilisateur crée une histoire en assemblant les puzzles. Ensuite, l'utilisateur compile le programme en prenant en photo l'ensemble des blocs afin d'obtenir une histoire, en retour.

Les concepts de programmation

StoryBlock montre six concepts de programmation. L'exécution de l'algorithme se fait étape par étape (séquentiellement) suivant l'ordre de placement des pièces du puzzle. Les deux concepts d'algorithme et de séquence sont faciles à appréhender. La variable est montrée par les différents caractères ; les fonctions sont des fonctions prédéfinies. La conditionnelle et la boucle sont citées sans explication.

T Maze [40] (2011)

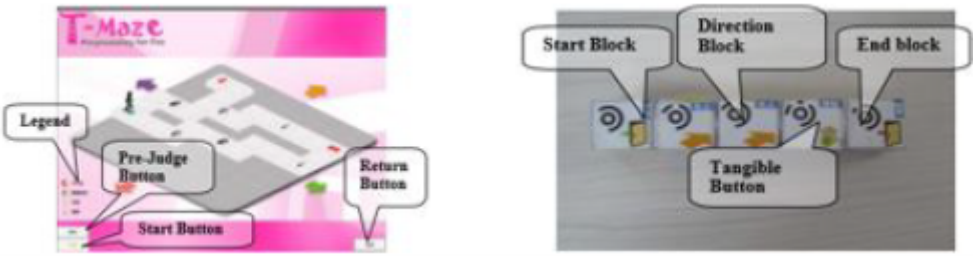


FIGURE 4.14 – T Maze

Age

L'interface tangible de programmation est destinée aux enfants de cinq à neuf ans.

Dispositif physique

L'interface tangible de programmation est composée de blocs, d'une caméra et d'un ordinateur. Les blocs sont une forme cubique fabriquée en bois. Ils jouent le rôle d'input. L'output est l'écran de l'ordinateur.

Utilisation

Un utilisateur crée un programme en assemblant les blocs les uns après les autres. Les blocs sont pris en photo, la photo est compilée par l'ordinateur puis il l'exécute afin de guider le personnage.

Les concepts de programmation

T Maze montre trois concepts de programmation. L'exécution de l'algorithme se fait un par un (séquentiellement). Le concept de la boucle est bien expliqué à l'aide du motif imprimé sur le bloc et l'effet qu'il réalise sur le personnage.

T ProRob [28] (2011)

FIGURE 4.15 – T ProRob

Age

L'interface tangible de programmation est destinée aux enfants de six à douze ans mais il est accessible aux débutants de tout âge.

Dispositif physique

L'interface tangible de programmation est composée de blocs et d'un robot. Les blocs contiennent chacun un microcontrôleur et un connecteur. Chaque concept de programmation est différencié par leur connecteur. La communication entre le bloc et le robot se fait par Bluetooth. Les blocs jouent le rôle d'input et le robot joue le rôle d'output, avec un led. En plus du robot, l'output est caractérisé par des leds qui donnent un feedback.

Utilisation

L'utilisateur crée un programme pour diriger le robot en connectant les blocs. Pour transmettre les instructions au robot, l'utilisateur appuie sur un bouton.

Les concepts de programmation

T ProRob montre six concepts de programmation. L'algorithme se crée en assemblant et en interconnectant les blocs. L'exécution de l'algorithme se fait bloc par bloc en respectant le rang d'assemblage (séquentiel). La structure conditionnelle et la boucle sont montrées par des blocs avec motifs et l'effet de ces concepts sur le robot. L'interface tangible de programmation permet de gérer les erreurs avec sa communication en double sens.

TaBGo [41] (2018)

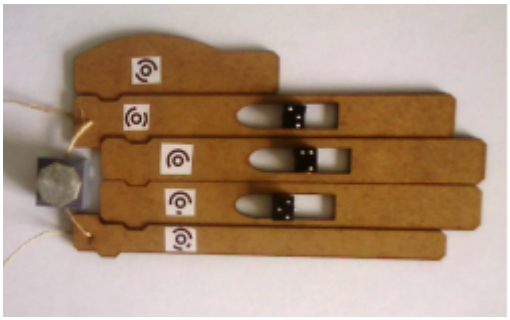


FIGURE 4.16 – TaBGo

Age

L'âge n'est pas défini dans l'article.

Dispositif physique

L'interface tangible de programmation est composée d'un objet de la même forme que scratch avec un appareil photo, d'un logiciel et d'un ordinateur. L'objet tangible est fabriqué en bois. Il joue le rôle de l'input. L'output est l'écran de l'ordinateur.

Utilisation

Un utilisateur crée un programme comme dans Scratch mais à l'aide des objets en bois. Il les prend en photo. Le logiciel compile la photo puis le résultat est affiché à l'écran sous forme de code.

Les concepts de programmation

TaBGo montre cinq concepts de programmation. Tous les concepts enseignés par le programme scratch sont enseignés par TaBGo (la variable, la structure conditionnelle, la boucle, la séquence et le programme).

TanProStory [42] (2015)

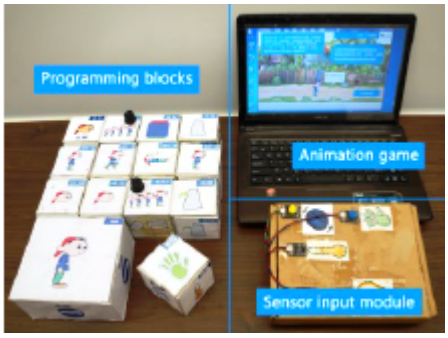


FIGURE 4.17 – TanProStory

Age

L'interface tangible de programmation est destinée aux enfants de six à neuve ans.

Dispositif physique

L'interface tangible de programmation est composée de blocs et d'un ordinateur. Les blocs jouent le rôle d'input. Ils peuvent être un bloc de nom, un bloc de vêtement ou un bloc d'action, pour réaliser des animations. La communication se fait par infrarouge. Le jeu d'animation est exécuté sur un ordinateur et le résultat du programme est affiché sur un écran. Les blocs contiennent des capteurs.

Utilisation

Un utilisateur crée un personnage dans lequel il initialise les attributs et construit une séquence avec les actions.

Les concepts de programmation

TanProStory montre cinq concepts de programmation. Les motifs signifient l'action que le bloc peut effectuer. L'interface tangible de programmation enseigne la notion d'objet en créant un personnage avec les blocs qui le caractérisent. Le concept de variable est indiqué par les attributs de l'objet, ils peuvent changer au cours de l'exécution d'un programme. L'utilisateur crée un algorithme en assemblant les blocs avec un rang défini pour être exécuté séquentiellement.

Tern [43] (2007)

FIGURE 4.18 –

Age

L'interface tangible de programmation est destinée aux collégiens et aux lycéens.

Dispositif physique

L'interface tangible de programmation est composée de blocs en bois en forme de puzzle, d'un appareil photo et d'un ordinateur. Le puzzle est l'input. L'output est un robot affiché sur l'écran d'un ordinateur.

Utilisation

L'utilisateur crée un programme en assemblant les pièces du puzzle. Il doit toujours commencer par un bloc START et terminer par un bloc STOP. Quand l'utilisateur finit d'assembler les pièces du puzzle, il le prend en photo, la photo est compilée par un ordinateur afin de guider le robot.

Les concepts de programmation

Tern montre six concepts de programmation. Ils sont faciles à apprendre parce que chaque concept est bien identifié avec les pièces du puzzle. L'algorithme est bien défini entre le bloc de début et le bloc de fin. L'exécution se fait une par une, commençant toujours par START et se terminant par STOP. L'interface tangible de programmation permet la gestion des erreurs.

Turtan [44] (2008)

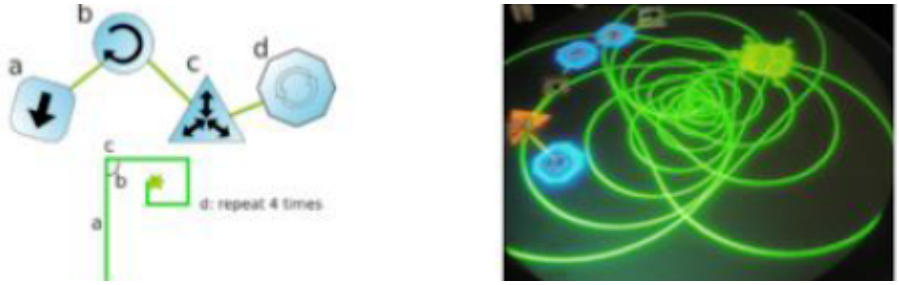


FIGURE 4.19 – Turtan

Age

L'interface tangible de programmation est destinée aux enfants. L'âge n'est pas précisé.

Dispositif physique

L'interface tangible de programmation est composée d'une table interactive qui contient des caméras et des objets tangibles. La table interactive joue le rôle de l'input et de l'output. Le déplacement de la tortue est affiché sur la table avec des traces. La caméra permet de détecter chaque bloc placé sur la table. Les objets sont en forme de cube. Ces objets jouent le rôle de l'input.

Utilisation

Le but du jeu est de manipuler une tortue en temps réel. Pour ce faire, l'utilisateur place les objets sur la table.

Les concepts de programmation

Turtan montre trois concepts de programmation. Ils sont interprétés par les mouvements provoqués par l'utilisateur. Les concepts de programmation présentés sont la boucle, la variable couleur et les fonctions qui permettent de changer de couleurs.

TurtleTable [45] (2017)

FIGURE 4.20 – TurtleTable

Age

L'interface tangible de programmation est destinée aux collégiens et aux lycéens.

Dispositif physique

L'interface tangible de programmation est composée d'une table interactive et des objets tangibles. Les objets jouent le rôle d'input. La table joue à la fois le rôle d'input et d'output.

Utilisation

L'utilisateur doit déplacer les objets sur la table pour créer une instruction et pour passer à l'étape suivante.

Les concepts de programmation

Turtletable montre trois concepts de programmation : la variable, la boucle et la fonction. Les concepts sont introduits progressivement sur les différents niveaux de jeux. Le dispositif permet de voir, en temps réel, le déroulement de l'exécution.

5. Classification

Dans ce chapitre, nous allons discuter la classification de Fishkin [18], de Palaude [20], de Bodart [19] et d'Yu [21]. Celles-ci vont nous servir de base pour mener une réflexion sensée apporter des éléments de réponse à la question de recherche. Non exhaustives, ces classifications (et les enseignements qu'on peut tirer) seront complétées par le développement d'une classification originale basée sur les contraintes des interfaces utilisateur tangibles.

Sur base de la classification de Fishkin [18] (figure 5.1), nous allons analyser deux choses : l'analogie entre le virtuel et l'objet physique de l'interface tangible de programmation et la connexion entre son input et son output.

La classification de Palaude [20] (figure 5.2) permettra de distinguer les interfaces tangibles de programmation selon leur symbole de voir si les interfaces tangibles de programmation pour les plus de douze ans utilise le même symbole que pour les moins de douze ans.

La classification Bodart [19] permettra d'analyser les interfaces tangibles de programmation en fonction de plusieurs groupements : l'âge du public cible, les concepts de programmation mis en œuvre et le dispositif physique. Cependant, avec les nouveaux interfaces tangibles de programmation, les éléments de chaque groupement de la classification ne sont pas assez à les classifiés. Par conséquent, nous allons ajouter des paramètres et modifier la classification.

La classification de Yu [21] ne va pas être prise en compte dans ce travail pour deux raisons. D'une part dans la classification, la partie kits virtuels et la partie kits hybrides avec blocs de programmation virtuels ne vont jamais être considérées parce que l'interface tangible de programmation n'est pas quelque chose de virtuelle. D'autre part les autres parties : kits physiques sans électronique, kits physique avec de l'électronique, Kits hybrides avec blocs de programmation tangibles vont être représentées dans la partie dispositif physique de la classification de Bodart [19] modifiée.

	Incarnation				Métaphore			
	Complète	Proche	Environnementale	Distante	Nom	Verbe	Nom & Verbe	Complète
T Maze				■		■		
E Block			■	■		■		
AR Maze			■	■		■		
Robot Block				■		■		
Dr Wagon				■		■		
StarLoop		■		■		■		
TanProStory				■		■		
Queztal				■		■		
Butterfly				■		■		
Turtan		■					■	
P Cube				■		■		
AlgoBlock				■		■		
Cube Sifteo	■		■			■		
Turtable		■					■	
T ProRob				■		■		
Story Block			■			■		
Note Code			■	■		■		
Tern				■		■		
TaBGo				■		■		
Sheets				■		■		

FIGURE 5.1 – Classification de Fishkin [18]

	Significatif	Non Significatif	Spécifique	Non spécifique
T Maze	■			■
E Block	■			■
AR Maze	■			■
Robot Block	■			■
Dr Wagon	■			■
StarLoop	■	■	■	■
TanProStory		■	■	
Queztal		■	■	
Butterfly		■	■	
Turtan	■			■
P Cube	■			■
AlgoBlock		■	■	
Cube Sifteo		■	■	
Turtable		■	■	
T ProRob	■			■
Story Block		■	■	
Note Code		■	■	
Tern		■	■	
TaBGo		■	■	
Sheets	■	■	■	■

FIGURE 5.2 – Classification de Palaude [20]

Classification de Bodart modifiée

Dans la partie âge cible, nous partageons l'âge en cinq partie en se basant du système éducatif en Belgique : (A) moins de douze ans : l'école primaire, (B) entre douze et quinze ans : cycle inférieur de l'école secondaire, (C) entre quinze et dix-huit ans : cycle supérieur de l'école secondaire, (D) dix-huit ans ou plus : enseignement supérieur et (E) pour les âges qui ne sont pas mentionné dans les articles.

Dans la partie concept de programmation, en plus de ce qui existe déjà (A) variable, (B) structure conditionnelle, (C) boucle, (D) fonction, (E) algorithme, (F) séquence. Les rôles des variables sont plus détaillés et deux concepts sont ajoutés. Une variable peut avoir plusieurs rôles [46] selon son utilisation (1) variable conteneur (la variable permet de stocker une valeur qui peut changer au cours de l'exécution), (2) variable d'itération (la variable traverse une succession de valeur). Le premier concept qui s'ajoute est (G) notion d'objet ou la notion d'une classe, elle est composée des attributs et des méthodes qui permettent d'accéder, de modifier ou de supprimer des données. Le deuxième concept qui s'ajoute est (H) gestion d'erreur, l'erreur peut causer le mauvais fonctionnement d'un programme ou le non-exécution du programme.

Dans la partie dispositif physique, la modification est la suivante. Le paramètre est partagé en deux : l'input et l'output.

L'input est constitué par (1 - a) manipulation continue, l'utilisation de l'interface tangible de programmation est en continue et (1 - b) la manipulation discrète, l'utilisation de l'interface tangible de programmation est en plusieurs étape (2) système actif [47] ou kits avec de l'électronique , l'input est composé par des circuits électroniques intégrés et impliqués, (3) système passif [47] ou kits sans l'électronique, l'input n'a pas de système électronique et il est basé sur la reconnaissance d'images, (4) programmation par bloc, l'input est un cube.

L'output est constitué par (1) résultat affiché à l'écran : (a) ligne de code et (b) animation. (2) quelque chose d'audible, (3) led, (4) robot qui exécute le programme.

La colonne tout en un et la colonne composée sont enlevées parce qu'elles sont déjà représentées par l'incarnation de la classification de Fishkin [18]. Les différentes colonnes : capteur (mouvement et environnement), caméra vidéo sont regroupées dans deux colonnes. Les inputs avec des capteurs ou des circuits embarqués sont assemblés dans le système actif, les inputs sans électronique, mais qui utilisent une caméra pour être traduite pas l'ordinateur, sont mis dans le système passif.

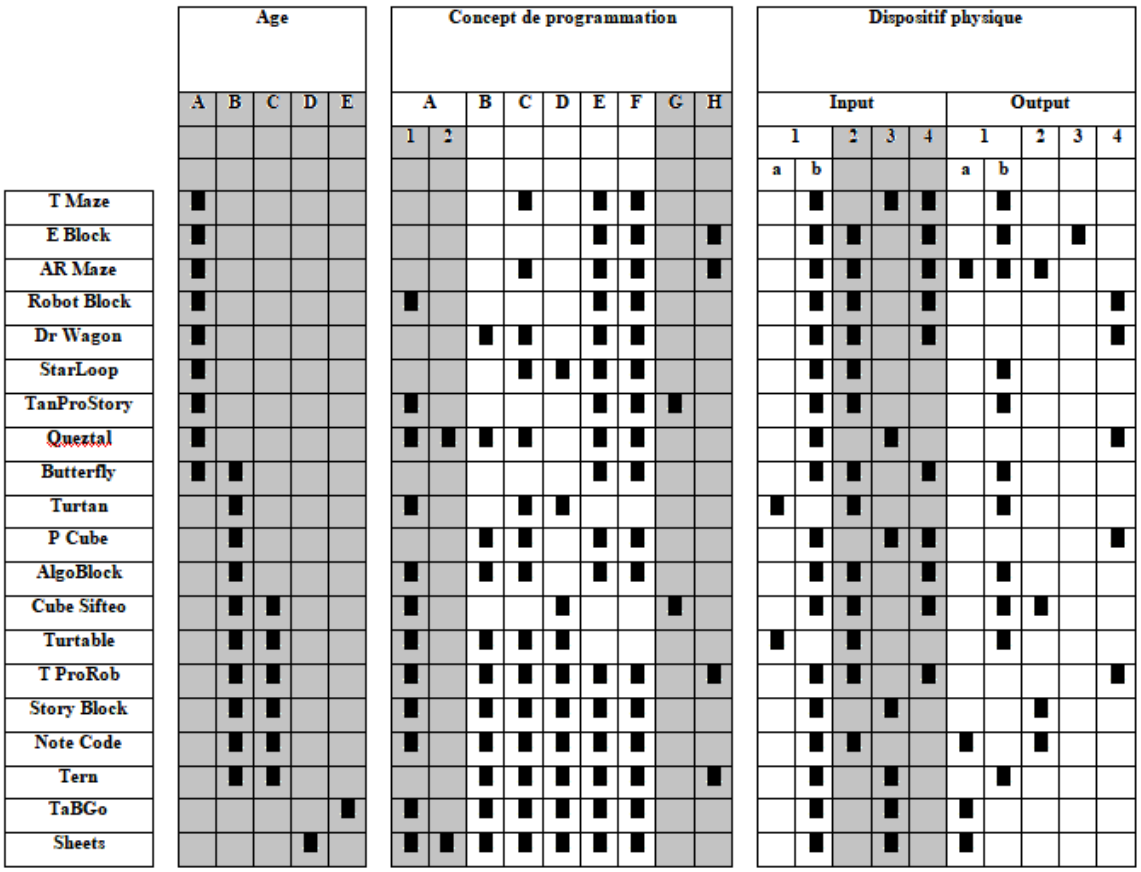


FIGURE 5.3 – Classification de Bodart avec des modifications en fond gris

Classification des contraintes

Les contraintes aident les utilisateurs à créer une structure spécifique. Elles fournissent des limites physiques afin de contraindre les utilisateurs sur la création d'un programme et d'empêcher des assemblages non valides. Ces contraintes sont mises en place afin d'éviter les erreurs de syntaxe ou d'assemblage.

Il existe deux contraintes pour l'interface tangible, il y a la contrainte d'utilisation [16] destinée à informer l'utilisateur sur son bon usage, par exemple l'assemblage des blocs doit commencer par un bloc de commencement et interpréter de gauche à droite. Elle décrit comment l'interface tangible va être utilisée. Et la contrainte qui nous intéresse est la contrainte physique des interfaces tangibles de programmation [22] [23] [24] [25]. La contrainte physique des interfaces utilisateur tangibles est partagée en trois parties. La contrainte par rapport au positionnement, elle facilite la manière de placer les différents blocs. La contrainte par rapport à l'orientation, elle aide à bien orienter un bloc par rapport à un axe. Et la contrainte par rapport à la forme, elle permet de bien associer les blocs qui vont ensemble.

Après une analyse des vingt interfaces tangibles de programmation, nous avons distingué six facteurs qui peuvent contraindre ou aider les utilisateurs à bien les utiliser : le connecteur, le capteur, le symbole d'identification, le puzzle, le tapis et la table interactive.

Le connecteur joue deux rôles dans un assemblage de blocs. Il permet de bien les positionner les uns après les autres et de les orienter sinon les blocs ne se connectent pas. AlgoBlock [26] est un exemple.

Le capteur permet de positionner les blocs, il détecte si les blocs ne sont pas à la bonne place en signalant une erreur. E Block [32] est un exemple.

Le symbole d'identification est une contrainte faible, il permet de guider l'utilisateur à la bonne orientation du bloc par rapport à un axe. AR Maze [27] est un exemple.

Le puzzle permet d'associer les blocs facilement. Avec la particularité de la forme des puzzles, l'utilisateur peut savoir quelle forme de bloc correspond à un autre bloc. Tern [43] est un exemple.

Le tapis permet de placer les blocs facilement selon sa forme. Il aide les utilisateurs à bien choisir les blocs qu'il faut mettre. Chaque emplacement correspond à un bloc bien déterminé. P Cube [34] est un exemple.

La table interactive permet de bien arranger les blocs, elle indique les emplacements des blocs aux utilisateurs. StarLoop [38] est un exemple.

A partir de ces informations, nous avons établi une classification des contraintes des interfaces tangibles de programmation.

	Positionnement	Orientation	Forme
T Maze		▮	
E Block	▮	▮	
AR Maze	▮	▮	
Robot Block	▮	▮	
Dr Wagon	▮	▮	
StarLoop	▮		
TanProStory	▮	▮	
Quetzal	▮		
Butterfly	▮		
Turtan	▮		
P Cube	▮		▮
AlgoBlock	▮	▮	
Cube Sifteo	▮		
Turtable	▮		
T ProRob	▮		
Story Block	▮		▮
Note Code			
Tern	▮		▮
TaBGo	▮		▮
Sheets		▮	

FIGURE 5.4 – Classification des contraintes

Note code [33] n'est pas classé parce que son interface tangible est un ensemble de boutons déjà placé sur une surface plate. Pour créer un programme, l'utilisateur tapote les boutons mais il n'assemble pas de blocs.

6. Analyse et discussion

Classification de Bodart

La classification de Bodart [19] donne un résumé global sur ce qu'une interface tangible de programmation peut faire. Elle permet de distinguer les concepts de programmation présentés et la composition de son dispositif physique. Le paramètre âge permet de référencer l'outil à un groupe d'utilisateurs.

Age

La majorité des interfaces tangibles de programmation est destinée aux moins de onze ans. Pour les plus de onze ans, il n'en existe que douze. Parmi ces douze interfaces tangibles de programmation, un seul est destiné aux plus de dix-huit ans.

En nous basant sur ces chiffres, nous pourrions dire que la programmation tangible est créée pour les enfants et les plus jeunes afin de les familiariser aux concepts de base de la programmation. Les jeunes n'ont que la méthode d'apprentissage utilisée dans les écoles supérieures ou les universités à travers les cours d'initiation à la programmation et les travaux pratiques sur les différents langages de programmation pour apprendre les concepts de base de programmation. Néanmoins, il est intéressant d'utiliser une autre manière d'apprentissage comme la programmation tangible pour initier les étudiants à la programmation. Cette autre méthode pourra faciliter la compréhension des concepts de base de la programmation pour les étudiants qui auront du mal avec la méthode d'apprentissage standard. Ainsi le taux d'échec en première année en option informatique pourrait baisser.

Concept de programmation

Les concepts de programmation présentés par les interfaces tangibles de programmation sont les concepts de base de programmation : la variable, la structure conditionnelle, la boucle, la fonction, la séquence, l'algorithme et la notion d'objet, plus la gestion d'erreur.

Douze interfaces tangibles de programmation sur vingt ont le concept de variable. Comme le montre la classification, il y a deux types de variables présentés comme suit :

- La variable conteneur est présentée par un bloc ou un puzzle qui peut contenir une valeur. Exemple : Le Quetzal [35] où un bloc peut contenir un jeton qui représente la distance à effectuer par un robot.
- La variable d'itération est mise en avant en comptant chaque itération. Exemple : le Quetzal [35], il permet de compter les jetons utilisés dans un assemblage de programme.

Dix-huit des interfaces tangibles de programmation qui ont le concept de variable n'ont que la variable normale. Il y a que Quetzal [35] et Sheets [37] qui ont la variable d'itération. La variable normale est facile à mettre en place et les blocs de variable montre vraiment le concept de variable : un bloc qui contient une valeur.

Onze interfaces tangibles de programmation sur vingt montrent la structure conditionnelle, il y a plusieurs manières de la présenter dans une interface tangible de programmation tangible

- Un bloc avec une entrée et deux sorties. En fonction de la condition du bloc, le programme prend une des deux sorties. Exemple : AlgoBlock [26].
- Un bloc qui indique le début de la structure conditionnelle et un bloc qui marque la fin de la condition. Exemple : Dr Wagon [31].
- Un bloc avec un symbole point d'interrogation (figure 6.1) qui signifie qu'il y a un choix à faire en fonction de l'état d'exécution. Exemple : T ProRob [28].

La structure conditionnelle est montrée par les interfaces tangibles de programmation par le fait qu'il y a une partie de l'assemblage qui sera exécutée en fonction de l'état d'exécution. Il y a un choix à faire par les programmeurs en fonction de l'environnement d'exécution.

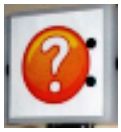


FIGURE 6.1 – T ProRob
[28]

Quatorze sur vingt interfaces tangibles de programmation montrent la boucle qui se présente de la manière suivante :

- Un bloc qui a le symbole qui marque le début de la boucle (figure 6.2) Exemple : P Cube [34], AR Maze [27].
- Un assemblage qui fait un tour et revient sur un des blocs d'avant. Exemple : Tern [43].

Sur ces deux techniques, la répétition est bien présentée et elle est facile à comprendre. Le symbole utilisé pour la boucle est intuitif en signifiant une répétition mais la condition de sortie n'est pas bien expliquée. L'assemblage montre qu'après l'exécution des blocs, l'exécution revient au bloc précédent et elle continue. Pour cette partie, la boucle est associée à une structure conditionnelle pour sortir de la boucle.

FIGURE 6.2 – AR Maze
[27]

Dix interfaces tangibles de programmation sur vingt montrent le concept de fonction. Les interfaces tangibles de programmation ne permettent pas de créer des fonctions avec leurs inputs. Les fonctions sont prédéfinies et elles sont créées et définies à l'avance. La fonction est considérée comme une action dans une interface tangible de programmation. Par exemple avec Tern [43], une fonction représente un ensemble d'actions.

Dix-sept interfaces tangibles de programmation montrent le concept d'algorithme et le concept de séquence. Les deux concepts sont liés dans l'interface tangible de programmation car s'il n'y a pas d'algorithme créé, l'exécution n'est pas séquentielle. Le concept d'algorithme se présente par l'assemblage des inputs. L'assemblage forme un programme qui s'exécute séquentiellement. L'assemblage des blocs permet à l'utilisateur de mettre en avant leur compréhension des différents rôles des blocs afin de résoudre un problème. Cette technique ressemble à une écriture de code. L'assemblage des inputs peut être considéré comme une écriture de code.

Deux interfaces tangibles de programmation sur vingt montrent la notion d'objet. Le concept d'objet est un cours avancé dans l'apprentissage de programmation. Ce concept est encore plus abstrait pour les personnes qui s'initient à la programmation. Même s'il n'est pas très utilisé par les outils de programmation tangible, cela n'affectera pas l'apport de la programmation tangible dans l'apprentissage de la programmation.

Quatre interfaces tangibles de programmation sur vingt ont une gestion d'erreur. La gestion d'erreur n'est pas un concept de base de la programmation. Sa présence permet à l'utilisateur d'identifier facilement leur erreur lors d'une utilisation de l'interface tangible de programmation.

Input / Output

Dans le dispositif physique en input, la manipulation discrète est la plus utilisée. Cette manipulation permet à un utilisateur d'assembler les inputs (programme) avant de l'exécuter. Par contre la manipulation continue exécute l'interaction de l'utilisateur avec l'interface tangible en temps réel. Les deux outils qui ont une manipulation continue sont des tables interactives. En output, l'animation interprétée par le guidage d'un personnage sur un labyrinthe ou par le guidage d'un robot est la plus utilisée car un environnement semblable à un jeu permet d'attirer facilement les enfants ou les plus jeunes à utiliser les outils de programmation tangible. Par ailleurs, à travers l'interface tangible de programmation, ils peuvent se familiariser aux concepts de base de programmation qu'elle présente.

Relation entre l'âge et le concept de programmation

En croisant la colonne de l'âge et celle colonne des concepts de programmation, nous verrons que le nombre de concepts de programmation présenté par l'interface tangible de programmation augmente en fonction de l'âge des personnes pour qui l'interface tangible est destinée. Et il y a également des concepts qui sont présentés par les outils destinés aux plus âgés et qui ne sont pas présentés aux enfants. C'est le cas de la fonction et de la structure conditionnelle.

Pour toutes les interfaces tangibles de programmation avec un assemblage d'input (hors Turtan [44], Turtable [45], Cube Sifteo [29] [30]), elles présentent le concept de séquence et d'algorithme. Ces abstractions sont présentées dès le plus jeune âge parce que leurs fonctionnements consistent simplement à bien assembler les inputs.

Le concept de boucle est présenté par quelques interfaces tangibles de programmation destinées aux enfants comme E Block [32]. Dans une programmation tangible, ce concept n'est pas difficile à comprendre parce qu'il indique simplement une répétition d'action, par exemple avancer n fois.

Le concept de structure conditionnelle n'est introduit dans les interfaces tangibles

de programmation que pour les enfants âgés de onze ans et plus. Ce concept n'est pas présenté au moins de onze ans parce que son utilisation demande une réflexion sur les cas possibles d'exécution. Ce problème peut être complexe pour les jeunes enfants.

Le concept de fonction est introduit à des jeunes aux alentours de quinze ans ou plus. Le fait d'avoir un bloc d'input qui a un plus (exécuter plusieurs actions) peut complexifier l'utilisation de l'outil pour les moins de quinze ans.

En nous basant sur ces informations, les concepts plus faciles à comprendre à travers les interfaces tangibles de programmation sont introduits pour les plus jeunes. Ces concepts sont complétés par d'autres en fonction de l'âge auquel elle est destinée.

A noter que les concepts de boucle sont faciles à comprendre dans les interfaces tangibles de programmation parce qu'elles n'utilisent pas de test de condition pour sortir de la boucle. Le bloc indique simplement qu'il y a une répétition à exécuter.

Relation entre l'âge et le dispositif physique

Toutes les interfaces tangibles de programmation destinées aux moins de onze ans ont un input passif, ce qui veut dire que l'input est composé d'électronique (microcontrôleur, capteur). Et aussi parce que le fait d'avoir un système électronique dans les interfaces tangibles permet à l'utilisateur de les utiliser aisément. T Maze [40] en est une exception mais il est remplacé par AR Maze [27] quelques temps après.

Toutes les interfaces tangibles de programmation destinées aux moins de quinze ans ont pour output un guidage, soit par un personnage dans un labyrinthe sur écran, soit par un robot. Ce système rend l'apprentissage plus amusant pour les jeunes parce qu'ils ont l'impression de jouer en apprenant les concepts de programmation.

A partir de quinze ans, les outils de programmation tangible ont comme output autre chose que le guidage. L'output peut être affiché en ligne de code. La différence sur les outputs se voit à travers les différents âges. Pour les enfants ou pour les plus jeunes, le dispositif physique des interfaces tangibles de programmation est conçu pour les attirer à jouer avec ou à l'utiliser.

Relation entre le concept de programmation et le dispositif physique

L'analyse qui se distingue des deux paramètres de classification est que les interfaces tangibles de programmation avec une manipulation en continu ne présentent pas le concept d'algorithme et de séquence parce que la manipulation en continu a un effet en temps réel sur l'interaction entre l'utilisateur et l'interface tangible. Or, pour créer un algorithme, un temps de réflexion est demandé plus un assemblage, que ce soit un assemblage de mot dans un langage ou un assemblage de bloc.

Classification de Fishkin

La classification de Fishkin [18] caractérise l'interaction d'un utilisateur à une interface tangible qui est composée de deux types dont l'incarnation et la métaphore

Incarnation

L'incarnation la plus utilisée est l'incarnation distante. Seize interfaces tangibles de programmation sur vingt la présentent. L'avantage d'utiliser cette incarnation pour les

utilisateurs est que l'input et l'output de l'interface tangible apparaissent de manière bien distincte. Ainsi, ils sont bien identifiés, les parties qui ordonnent l'action et celles qui l'exécutent et l'interprètent. Avec ce système, chaque input (ou chaque bloc) aura un rôle défini avec le concept de programmation qu'il montre ; par lequel l'output affiche ou interprète le résultat de l'input. Cela évite aux utilisateurs de mélanger les actions qu'ils font et les résultats qui en sortent.

Certaines interfaces tangibles de programmation présentent une autre incarnation en plus de l'incarnation distante. C'est le cas de l'AR Maze [27], E Block [32], StarLoop [38], et Note Code [33]. L'existence de deux incarnations sur un outil rend l'interface tangible plus attractive pour les enfants car cela donne un effet réalité augmenté. Les enfants ont le sentiment de jouer en apprenant les concepts de base de programmation.

Les quatre interfaces tangibles de programmation qui n'ont pas l'incarnation distante sont Turtan [44], Cube Sifteo [29] [30], Turtable [45] et Story Block [39]. Turtan [44]et Turtable [45] sont des tables interactives. Leurs utilisations se font en temps réel. Par conséquent, ils n'ont pas l'esprit d'assemblage ou de création de programme. Les concepts de programmation sont montrés en interagissant avec la table, en temps réel. Pour le Cube Sifteo [29] [30], l'input et l'output s'affichent sur la même interface tangible. Le Story Block [39] pourrait être regroupé avec les seize interfaces tangibles de programmation dans l'incarnation distante mais le fait que son output est quelque chose que nous sentons (audible), il se retrouve dans l'incarnation environnementale. Toutefois, cela ne veut pas dire que l'input et l'output sont associés sur un même objet.

Métaphore

La métaphore est unanime pour toutes les interfaces tangibles de programmation car elles ont une métaphore verbe (pour les tables interactive nom et verbe). Sur les vingt interfaces tangibles de programmation, aucune ne présente la métaphore nom ou complète.

La métaphore verbe signifie ici que la création d'un programme avec un langage de programmation sur un environnement de développement est pareille avec l'utilisation d'une interface tangible de programmation.

Relation entre la classification de Fishkin et la classification de Bodart

Nous n'avons pas trouvé de relation en croisant la classification Fishkin [18] avec l'âge ou le concept de programmation pour deux raisons. D'abord, l'incarnation et la métaphore des interfaces tangibles se présentent sur les mêmes colonnes. Puis la classification de Fishkin est associée à l'élément physique d'un outil mais pas à l'enseignement qu'une interface tangible peut apporter.

Dans le croisement de la classification de Fishkin[18] et de Bodart [19], nous pouvons remarquer qu'une interface tangible de programmation qui a une incarnation proche, une métaphore nom et verbe plus le dispositif physique de l'interface tangible avec un input : manipulation continue, composé de circuit électrique et l'output animation. Ces croisements peuvent définir une table interactive qui s'utilise en temps réel.

Classification de Palaude

La classification de Palaude [20] sur les interfaces tangibles de programmation est une classification qui n'a pas encore été publiée mais qui a fait l'objet d'un stage.

Le symbole permet d'identifier le rôle d'un input tangible, il décrit ce que l'input peut faire ou à quoi ce dernier représente. Mais si nous regardons les vingt interfaces tangibles de programmation sélectionnées, il n'y a pas que le symbole qui permet d'identifier le rôle d'un input. La forme (Tern [43]), l'assemblage des inputs (Quetzal [35]), les textes marqués dessus (Dr Wagon [31]) le permettent aussi.

La validation des symboles a été menée qu'avec des enfants ou des utilisateurs plus jeunes. Et leur jugement sur la signification et la spécificité des symboles sont subjectifs. Cela dépend de leur expérience personnelle (l'âge, ce qu'ils ont déjà vue, etc).

Néanmoins si nous nous basons sur l'étude qui a été faite sur la classification de Palaude [20], nous avons considéré qu'un symbole significatif n'est pas spécifique et un symbole spécifique n'est pas significatif. Et les symboles considérés significatifs sont les symboles mis en évidence par l'étude des symboles : les flèches de direction, des flèches qui tournent sur elles-mêmes, le point d'interrogation. Nous avons donc complété la classification sur cette base.

Relation entre la classification de Palaude et l'âge

75 % des interfaces tangibles de programmation sélectionnées pour les moins de onze ans ont un symbole significatif et non spécifique. Ce pourcentage confirme l'étude de Palaude [20], les enfants utilisent facilement les interfaces tangibles avec des symboles marqués des flèches de direction, des flèches qui tournent sur elles-mêmes, le point d'interrogation, etc. Pour les plus de onze ans, les symboles sont plus spécifiques en fonction des concepteurs qui adaptent leur symboles en fonction de l'évolution cognitive des utilisateurs.

Relation entre la classification de Palaude et les concepts de programmation

Sur les interfaces tangibles de programmation prises dans ce travail, il n'y pas de symbole associé aux concepts de variable, d'algorithme et de séquence. Le fait que ces concepts n'ont pas de symbole est logique. Pour la variable, elle contient une valeur. La meilleure façon de le montrer est avec l'interface tangible elle-même. La même chose pour l'algorithme et la séquence.

Le concept de fonction est représenté par des symboles mais les symboles ne sont pas standards. Chaque concepteur personnalise le symbole par rapport aux actions que la fonction peut effectuer.

Pour la structure conditionnelle, le point d'interrogation (figure 6.1) est le symbole le plus utilisé. Pour la boucle, le symbole dans la figure 6.2 est le plus utilisé. Comme dit dans la partie concept de programmation, ces symboles facilitent la compréhension des rôles de l'interface tangible, soit c'est un choix que l'utilisateur fait, soit c'est une répétition d'action.

Relation entre la classification de Palaude et le dispositif physique

Notons que même si l'interface tangible de programmation destinée aux jeunes de onze à dix-huit ans n'a pas le symbole avec des flèches, elle prend la forme du symbole ou le symbole est traduit en texte lorsque le système est un guidage.

La relation entre la classification de Palaude [20] et le dispositif physique rejoinnent l'analyse sur l'âge et le dispositif physique. Pour les interfaces tangibles de programmation avec des symboles de flèche (forme, texte), elles permettent de guider un personnage sur un labyrinthe ou guider un robot.

La classification de Palaude [20] est une classification qui n'est pas encore validée par les pairs. Néanmoins elle peut apporter quelque précision sur les symboles utilisés par les concepteurs.

Relation entre la classification de Palaude et la classification de Fishkin

Nous n'avons pas trouvé de tendance entre ces deux classifications pour les mêmes raisons que la classification Fishkin avec l'âge et les concepts de programmation. D'abord l'incarnation et la métaphore des interfaces tangibles se présentent sur les mêmes colonnes. Puis la classification de Fishkin [18] est associée à l'élément physique d'une interface tangible mais pas au symbole qu'elle porte.

Classification des contraintes

La classification des contraintes montre les différentes contraintes mises en place par les concepteurs d'interface tangible pour bien utiliser l'input et dans le but de réduire l'erreur d'assemblage ou l'erreur syntaxique. Toutefois ces contraintes ne sont pas bien utilisées sans un support d'utilisation pour bien les expliqués. L'avantage de ces contraintes est de permettre à l'utilisateur de se concentrer sur les concepts de programmation que l'interface tangible de programmation met en avant.

Nous pouvons diviser ces contraintes en trois parties : la contrainte faible (le symbole d'identification), la contrainte moyenne (le connecteur, le capteur, la table alternative) et la contrainte forte (le tapis, et la table interactive). Les contraintes et le support d'utilisation ont une relation : plus la contrainte est faible, le support d'utilisation doit être complet et plus la contrainte est forte, le support d'utilisation peut être léger.

La contrainte de positionnement (mettre les inputs à la bonne place) est utilisée par la majorité des outils de programmation tangible. La moitié des outils ont une contrainte d'orientation (placer les inputs sur l'axe x et y). Quatre outils ont une contrainte par rapport à leur forme.

Relation entre la classification des contraintes et l'âge.

Idem dans le croisement de la classification des symboles et âges, 75 % des interfaces tangibles de programmation destinées au moins de onze ans ont la contrainte d'orientation et toutes les interfaces tangibles de programmation ont la contrainte liée au positionnement. Ces deux contraintes sont utilisées afin de faciliter l'utilisation de l'interface tangible de programmation. L'existence des deux contraintes sur une interface tangible permet de diminuer l'erreur syntaxique ou l'erreur d'assemblage. Ceci est prouver par deux outils T Maze [40] et AR Maze [27]. AR Maze [27] est une version

améliorée de T Maze [40], la différence entre les deux interfaces tangibles de programmation est qu'AR Maze [27] contient un capteur et un circuit électronique qui signalent aux utilisateurs en cas d'erreur d'assemblage. Entre onze et dix-huit ans, la contrainte de positionnement est toujours utilisée, certaines interfaces tangibles de programmation ajoutent d'autres contraintes. Pour celle destinée au plus de dix-huit ans, elle n'a que la contrainte d'orientation. Ceci montre, plus l'interface tangible est utilisée par les personnes plus âgée, plus son utilisation est compliquée.

Relation entre la classification des contraintes et le concept de programmation

La contrainte et le concept de programmation n'ont pas de relation. Sur certaines interfaces tangibles, leurs formes permettent d'identifier le concept de programmation qu'elles présentent. Cette forme n'a rien à voir avec la contrainte liée à la forme. Par exemple l'AlgoBlock [26], la structure conditionnelle a une forme particulière et pour autant il n'a pas la contraire forme. En StoryBlock [39], il n'a pas de concept associé à la forme de l'interface tangible mais il a une contrainte liée à la forme.

Relation entre la classification des contraintes et le dispositif physique

Vu que la classification des contraintes et le paramètre âge ont révélé une tendance aux interfaces tangibles de programmation destinées au moins de onze ans, cela implique aussi la même analyse. Les interfaces tangibles de programmations destinées aux moins de onze ans peuvent être considérées comme un jeu afin d'attirer les enfants à l'utiliser, en plus les concepteurs ont fait en sorte que l'utilisation de l'interface tangible est la plus facile possible avec les différentes contraintes.

Relation entre la classification des contraintes et la classification de Fishkin

Nous n'avons pas trouvé de relation ou de croisement entre les deux classifications même si les classifications sont liées à l'aspect physique de l'interface tangible. La classification des contraintes se concentre sur l'assemblage d'un programme en fonction de l'aspect physique et la classification de Fishkin [18] se concentre sur la manière d'utiliser l'input et la relation entre l'input et l'output.

Relation entre la classification des contraintes et la classification de Palaude

Nous n'avons pas trouvé de relation entre les deux classifications. La classification des contraintes est liée à l'aspect physique de l'interface tangible et la classification de Palaude [20] est liée au rôle de l'input.

Conclusion

L'interface tangible de programmation est différente selon l'âge à qui elle est destinée. Nous avons constaté que plusieurs paramètres (concept de programmation, nombre de concept de programmation, le dispositif physique, les symboles, les contraintes) en dépendent. Chaque tranche d'âge présente une tendance.

Sur base de cette tendance, nous pouvons innover en proposant une nouvelle interface tangible de programmation.

7. Nouvelle interface tangible de programmation

L'idée de cette nouvelle interface tangible de programmation est venue après avoir constaté le manque d'interface tangible de programmation pour les jeunes de plus de onze ans. Dans notre recherche, nous n'avons trouvé que douze interfaces tangibles de programmation, les restes sont destinées aux enfants de moins de onze ans. Et vu que les interfaces tangibles peuvent aider les jeunes à comprendre les concepts de base de programmation, une nouvelle interface tangible de programmation sera proposée.

L'input du dispositif physique est inspiré du Sheets [37] parce que c'est la seule interface tangible destinée au jeune de plus de dix-huit ans et qu'elle est facile à produire. Les concepts de programmation pris sont les concepts de base, le but est d'introduire la programmation aux novices.

7.1 Design

L'idée est de créer une interface tangible plus proche des syntaxes de langage de programmation. Une envie existait aussi de mettre en avant les différents types primitifs. Aucune interface tangible de programmation existante ne parle de ce concept. Les symboles de la boucle et de la structure conditionnelle s'inspirent des résultats de Palaude [20].

Au final, l'interface tangible sera composée de blocs physiques comme illustrés cidessous. Le prototype présenté ici en est la représentation 2D, sur papier.

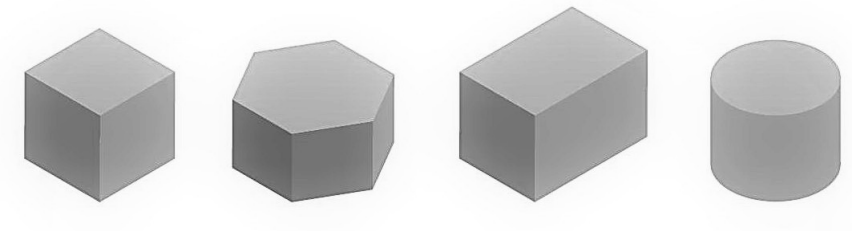


FIGURE 7.1 – Blocs physiques

Variable

Nous allons introduire trois types de variables : la chaine de caractères, le nombre et le booléen. Chaque type de variable est associé à une forme distincte.

Tout caractère ou toute chaine de caractères sont à écrire sur le dispositif physique.

FIGURE 7.2 – Bloc physique d'une chaine de caractère

Idem pour les nombres.

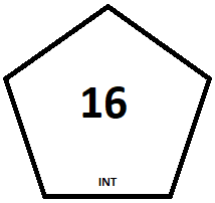


FIGURE 7.3 – Bloc physique d'un nombre

La valeur du booléen est V pour vraie ou F pour faux.

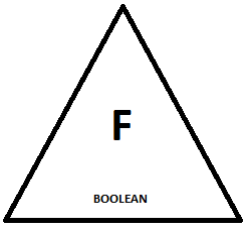


FIGURE 7.4 – Bloc physique d'un booléen

L'idée de la forme est d'identifier le type de la variable en la touchant, en complément de l'identification visuelle sur la valeur qu'elle présente.

Opérateur

Les opérateurs sont présentés sur une autre forme comme le montre la figure 7.5.

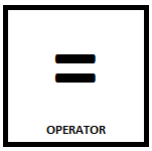


FIGURE 7.5 – Bloc physique d'un opérateur

Les différents opérateurs qui pourraient être utilisés sont : l'opérateur d'affectation (=), l'opérateur arithmétique (+, -, *, /) et l'opérateur de comparaison (<, >, <=, >=, ==, !=).

Fonction

Les fonctions peuvent être associées à un ou plusieurs paramètres. Elles sont créées au niveau du système. Pour chaque fonction créée, son utilisation et son fonctionnement seront détaillés dans une guide d'utilisation. Il n'est pas encore possible de créer une fonction à partir des interfaces tangibles. Dans cette interface tangible de programmation, il y a juste les fonctions créées dans le système qui peuvent être utilisées. Elles se présentent sous la forme suivante.

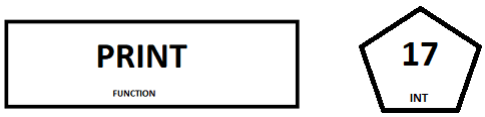


FIGURE 7.6 – Bloc physique d'une fonction

Elle a le rôle d'afficher 17.

Boucle et structure conditionnelle

La création de ces deux concepts de programmation est inspirée du résultat du travail de Palaude [20]. Son travail a mis en évidence les symboles qui facilitent la compréhension des rôles de l'interface tangible : une boucle avec une flèche circulaire et une structure conditionnelle avec un point d'interrogation. Leurs formes se ressemblent, elles se différencient sur les symboles et les fonctionnements. Une boucle ne fonctionne qu'avec un nombre (Figure 7.7) et une structure conditionnelle ne fonctionne qu'avec une condition (Figure 7.8), par exemple une comparaison entre deux nombre etc.

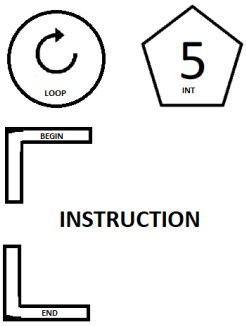


FIGURE 7.7 – Assemblage de blocs d'une boucle

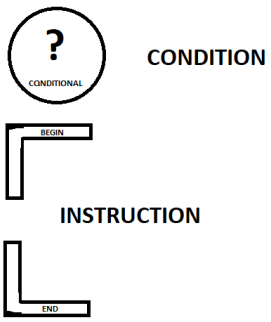


FIGURE 7.8 – Assemblage de blocs d'une structure conditionnelle

7.2 Aide à l'utilisation

Le support d'utilisation suivant est une information complémentaire en plus de ce qui est déjà dite sur les différents concepts de programmation dans la partie présentation. L'interface tangible de programmation consiste à créer un programme en assemblant les différents blocs comme sur la figure 7.9. L'assemblage des blocs doit respecter certaines règles inspirées des règles des langages de programmation.

- L'assemblage se fait de gauche à droite (une instruction) et de haut en bas (le haut est interprété en premier).
- Chaque bloc a un marqueur en bas du symbole, il permet à l'utilisateur d'identifier l'orientation du placement des blocs.
- Une affectation se fait avec un opérateur égale (=), seuls les inputs variable et valeur de même forme peuvent s'affecter.
- Les inputs BEGIN et END sont utilisés qu'avec une boucle ou une structure conditionnelle.
- L'idée de la compilation d'un programme est la suivante : après que les blocs du programme soient assemblés, l'utilisateur le prend en photo. Ensuite, la photo va être compilée par l'ordinateur et traduite en texte sur l'écran.

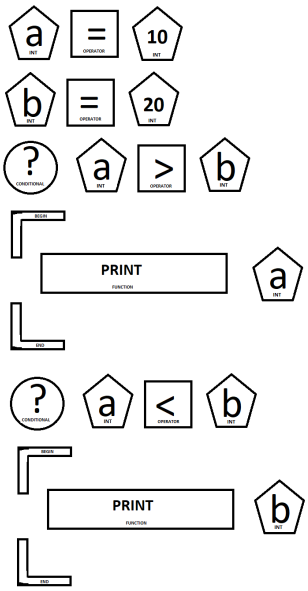


FIGURE 7.9 – Exemple de programme modélisé au moyen du prototype.

7.3 Evaluation

La nouvelle interface tangible de programmation va être évaluée sur sa position dans les différentes classifications, sur son utilisabilité et sur son utilisé.

7.3.1 Position de la nouvelle interface tangible dans les classifications

Pour faire une première évaluation de cette nouvelle interface tangible de programmation, nous allons utiliser les différentes classifications utilisées dans ce travail. Les classifications sont complétées avec les informations détaillées avant.

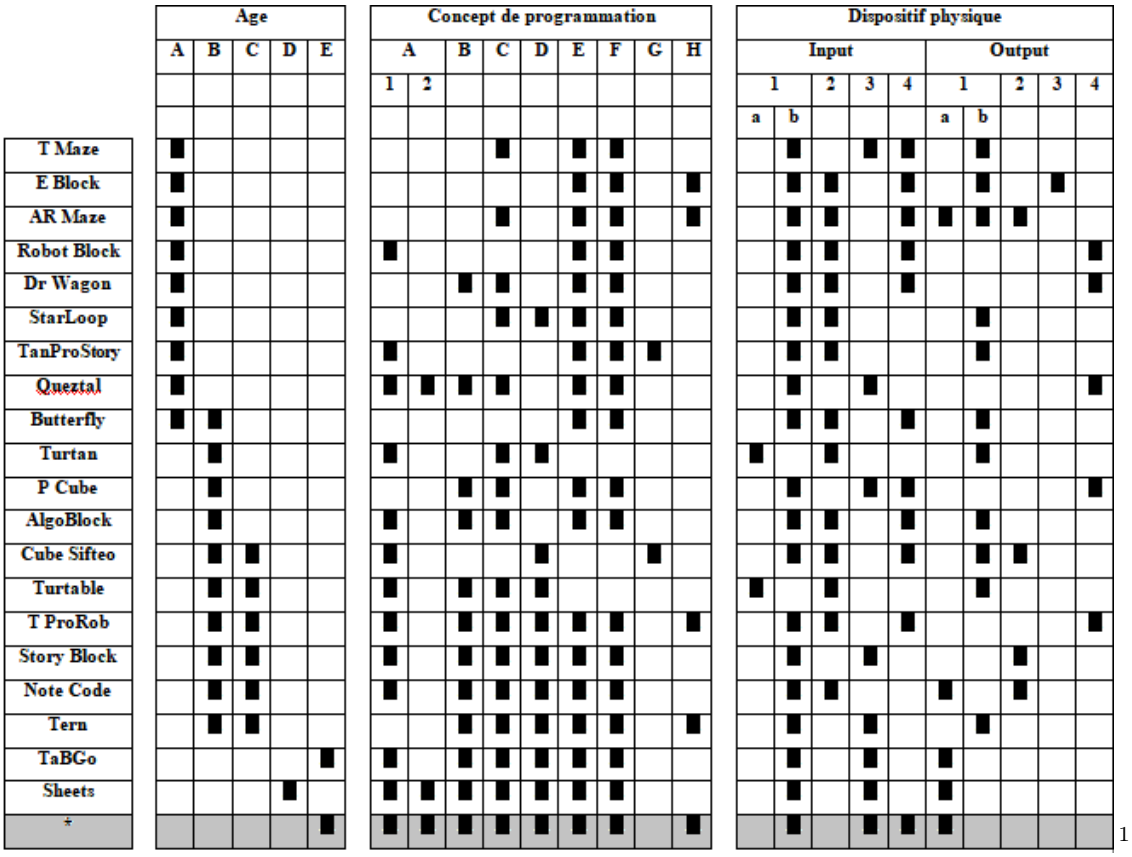


FIGURE 7.10 – Classification de Bodart [19] modifiée avec la nouvelle interface tangible de programmation

1. La nouvelle interface tangible est en fond gris.

	Incarnation				Métaphore			
	Complète	Proche	Environnementale	Distante	Nom	Verbe	Nom & Verbe	Complète
T Maze				■		■		
E Block			■	■		■		
AR Maze			■	■		■		
Robot Block				■		■		
Dr Wagon				■		■		
StarLoop		■		■		■		
TanProStory				■		■		
Quextal				■		■		
Butterfly				■		■		
Turtan		■					■	
P Cube				■		■		
AlgoBlock				■		■		
Cube Sifteo	■		■			■		
Turtable		■					■	
T ProRob				■		■		
Story Block			■			■		
Note Code			■	■		■		
Tern				■		■		
TaBGo				■		■		
Sheets				■		■		
*				■		■		

1

FIGURE 7.11 – Classification de Fishkin [18] avec la nouvelle interface tangible de programmation

	Significatif	Non Significatif	Spécifique	Non spécifique
T Maze	■			■
E Block	■			■
AR Maze	■			■
Robot Block	■			■
Dr Wagon	■			■
StarLoop	■	■	■	■
TanProStory		■	■	
Quextal		■	■	
Butterfly		■	■	
Turtan	■			■
P Cube	■			■
AlgoBlock		■	■	
Cube Sifteo		■	■	
Turtable		■	■	
T ProRob	■			■
Story Block		■	■	
Note Code		■	■	
Tern		■	■	
TaBGo		■	■	
Sheets	■	■	■	■
*	■			■

1

FIGURE 7.12 – Classification de Palaude [20] avec la nouvelle interface tangible de programmation

1. La nouvelle interface tangible est en fond gris.

	Positionnement	Orientation	Forme
T Maze		▮	
E Block	▮	▮	
AR Maze	▮	▮	
Robot Block	▮	▮	
Dr Wagon	▮	▮	
StarLoop	▮		
TanProStory	▮	▮	
Quetzal	▮		
Butterfly	▮		
Turtan	▮		
P Cube	▮		▮
AlgoBlock	▮	▮	
Cube Sifteo	▮		
Turtable	▮		
T ProRob	▮		
Story Block	▮		▮
Note Code			
Tern	▮		▮
TaBGo	▮		▮
Sheets		▮	
*		▮	

1

FIGURE 7.13 – Classification des contraintes avec la nouvelle interface tangible de programmation

Analyse et discussion sur la position de la nouvelle interface tangible dans les classifications

Sur base des quatre classifications ci-dessus (Figure 7.10, Figure 7.11, Figure 7.12, Figure 7.13), nous pouvons voir que la nouvelle interface tangible de programmation suit la tendance des interfaces tangibles de programmation existantes. L'âge cible de la nouvelle interface tangible de programmation peut être déterminé par le croisement des paramètres de classifications. Si nous nous référons à la classification de Sheets [37], la nouvelle interface tangible de programmation remplit les conditions pour être classifiée de la même manière. Le dispositif physique et l'assemblage (ou l'utilisation) des blocs sont similaires et la nouvelle interface tangible de programmation contient tous les concepts de base de la programmation. Avec ces éléments, nous pouvons déduire qu'elle est destinée aux jeunes de plus de dix-huit ans. Néanmoins, la nouvelle interface tangible de programmation peut être utilisée par des jeunes de plus de quinze ans si nous adaptons son utilisation à leur capacité de compréhension.

En plus de faciliter la compréhension des concepts de base de la programmation aux étudiants, la nouvelle interface tangible de programmation a l'avantage d'être facile à produire. La construction des blocs (dispositifs physiques) n'est pas couteuse et les blocs sont réutilisables. Elle peut être aussi utilisée en groupe parce que contrairement à un ordinateur, l'utilisation d'une interface tangible de programmation favorise la

1. La nouvelle interface tangible est en fond gris.

collaboration entre les utilisateurs. Ils peuvent s'entraider pour résoudre un problème.

Cependant, une évaluation sur l'utilisabilité et l'utilité de cette nouvelle interface tangible doit être faite. Cette évaluation permettra aussi de collecter des informations qui pourront améliorer la nouvelle interface tangible.

7.3.2 Test d'utilisabilité et d'utilité

Nous avons accueilli des volontaires afin d'observer s'ils déterminent facilement le lien entre chaque interface tangible et le concept de programmation associé, d'une part, et la relation entre les interfaces tangibles et la syntaxe d'un langage de programmation, d'autre part.

L'évaluation est conçue pour les personnes qui ont déjà une connaissance en programmation. Ce choix permet de savoir aisément si ces personnes se familiarisent facilement à la nouvelle interface tangible de programmation et si son utilisation est intuitive pour eux.

Dans le temps imparti, l'évaluation n'a pu être menée qu'avec un public plus large que le public-cible. Nous avons donc élargi la sélection à toutes personnes volontaires qui ont déjà fait de la programmation. Les volontaires sont au nombre de dix (huit garçons et deux filles), leurs profils et leurs expériences en programmation sont variés :

— un étudiant en première année en science informatique,
— un étudiant en deuxième année en bioinformatique,
— un étudiant en troisième année en science informatique (horaire décalé),
— un étudiant en troisième année en bio informatique,
— trois étudiants en master en science informatique,
— un étudiant en master en ingénierie civil,
— une étudiante en master en ingénierie de gestion,
— un développeur front end.

Déroulement du test

D'abord, nous avons donné une explication aux volontaires sur l'objectif de l'évaluation et sur les interfaces tangibles sans entrer dans les détails afin de ne pas les influencer dans leurs réflexions.

Par la suite, les volontaires ont fait trois exercices (voir - Annexes B.1 Exercices). Les deux premiers exercices consistaient à traduire un assemblage de blocs en pseudo code et le troisième exercice était de programmer avec des blocs pour résoudre un problème. Avec les deux premiers exercices, les volontaires identifiaient le rôle et la signification de chaque bloc. Le troisième exercice mettait en pratique ce qu'ils ont retenu des exercices précédents. Durant tout le test, les volontaires pouvaient poser des questions en cas de blocage.

A la fin, ils ont répondu aux questionnaires (voir - Annexes B.2) dont une partie sur l'interface tangible et une autre partie sur l'utilisabilité et l'utilité [48].

Durant l'évaluation, nous avons enregistré les volontaires puis nous avons transcrit l'enregistrement (voir – Annexes B.3).

Résultats

*La note que vous accordez à ce symbole par rapport à la signification que vous venez de donner. * en bref, ce symbole est-il bien choisi ? Représente-t-il bien ce qu'il devrait représenter selon vous ?*

(1) Pas du tout d'accord, (2) Pas d'accord, (3) Indifférent, (4) D'accord, (5) Tout à fait d'accord.

— Bloc d'un nombre (Figure 7.3) :
  La moyenne est de 3.2.
  Les notes des participants sont 4, 2, 3, 3, 3, 3, 4, 4, 3, 3.
— Bloc d'un booléen (Figure 7.4) :
  La moyenne est de 3.
  Les notes des participants sont 4, 4, 3, 3, 3, 2, 1, 4, 3, 3.
— Bloc d'une chaine de caractère (Figure 7.2) :
  La moyenne est de 4.2.
  Les notes des participants sont 4, 5, 4, 3, 5, 5, 4, 5, 4, 3, 5.
— Bloc d'une boucle (Figure 7.7) :
  La moyenne est de 4.4.
  Les notes des participants sont 5, 5, 4, 5, 5, 5, 3, 5, 3, 4.
— Bloc d'une structure conditionnelle (Figure 7.8) :
  La moyenne est de 4.1.
  Les notes des participants sont 5, 2, 4, 5, 5, 5, 3, 5, 3, 4.
— Bloc d'une fonction (Figure 7.6) :
  La moyenne est de 3.4.
  Les notes des participants sont 5, 3, 4, 3, 3, 5, 3, 3, 3, 2.

Selon vous, est-il envisageable d'utiliser des blocs différents pour un même concept(le concept de variable : nombre, boolean, string) ? OUI – NON

Quatre volontaires ont répondu oui et six volontaires ont répondu non.

L'information textuelle en bas de chaque bloc vous aide t-elle ?

Deux volontaires ont considéré que l'information textuelle est une contrainte. Huit volontaires ont répondu que l'information textuelle donne une précisons sur le concept de programmation.

Évaluez l'utilisabilité du système (à savoir l'ensemble des blocs avec symboles) en complétant la grille ci-dessous :

(1) Pas du tout d'accord, (2) Pas d'accord, (3) Indifférent, (4) D'accord, (5) Tout à fait d'accord.

— J'ai trouvé ce système inutilement complexe.
La moyenne est de 2.
Les notes des participants sont 2, 1, 3, 3, 4, 1, 2, 2, 1, 2.
— J'ai trouvé ce système facile à utiliser.
La moyenne est de 4.
Les notes des participants sont 4, 4, 3, 4, 2, 4, 3, 4, 5, 4.
— Je pense que j'ai besoin d'un support technique pour être capable d'utiliser ce système.
La moyenne est de 3.
Les notes des participants sont 1, 1, 4, 4, 1, 5, 5, 1, 2, 1.
— J'ai trouvé qu'il y avait trop d'incohérence dans ce système.
La moyenne est de 2.
Les notes des participants sont 3, 1, 3, 3, 1, 2, 1, 3, 1, 2.
— Je suppose que la plupart des gens apprendraient très rapidement à utiliser ce système.
La moyenne est de 4.
Les notes des participants sont 5, 4, 4, 5, 4, 4, 3, 5, 5, 4.
— J'ai trouvé ce système très contraignant à utiliser.
La moyenne est de 2.
Les notes des participants sont 4, 1, 2, 1, 4, 1, 1, 2, 2, 2.

Évaluez l'utilité du système en complétant la grille ci-dessous :
(1) Pas du tout d'accord, (2) Pas d'accord, (3) Indifférent, (4) D'accord, (5) Tout à fait d'accord.

— De manière générale, je trouve ce système facile à utiliser.
La moyenne est de 3.9.
Les notes des participants sont 4, 4, 4, 4, 2, 5, 3, 4, 5, 4.
— Ce système est simple à utiliser.
La moyenne est de 4.3.
Les notes des participants sont 4, 4, 5, 4, 2, 4, 5, 5, 5, 5.
— Je peux comprendre facilement les concepts de base de programmation en utilisant ce système.
La moyenne est de 3.9.
Les notes des participants sont 4, 4, 3, 4, 4, 4, 3, 5, 5, 3.
— Je suis à l'aise avec l'utilisation de ce système.
La moyenne est de 4.
Les notes des participants sont 4, 4, 4, 4, 4, 4, 3, 4, 4, 5.
— Cela a été facile d'apprendre à utiliser ce système.
La moyenne est de 4.
Les notes des participants sont 3, 3, 2, 4, 5, 5, 5, 4, 4, 5.

Temps d'exécution des exercices

La moyenne (en minute) est de 15.
Les temps (en minute) des participants sont 13, 13, 22, 14, 13, 22, 25, 10, 12, 11.

Analyse et discussion du test

Les trois exercices sont exécutés en moyenne en quinze minutes. Ce temps allant de dix à vingt-cinq minutes, selon l'expérience des volontaires en programmation. Les personnes qui ont l'habitude de programmer effectuaient les exercices en dix à treize minutes tandis que celles qui ont eu un ou deux cours de programmation les effectuaient en une vingtaine de minute. Neuf volontaires ont réussi les deux premiers exercices. Comme dit auparavant, l'objectif de ces deux exercices est de permettre aux volontaires d'identifier le rôle de chaque bloc et de savoir comment les utiliser ou les assembler. Sur le troisième exercice, six volontaires ont réussi, deux ont bien utilisé les blocs mais leurs solutions ne sont pas complètes et deux ont échoué. Les deux volontaires avec une solution incomplète voulaient utiliser un bloc ELSE ; par conséquent, ils assemblaient les blocs en pensant que derrière un bloc de structure conditionnelle IF, il y a l'ELSE. Pour les deux volontaires qui n'ont pas réussi à faire le troisième exercice, l'un d'entre eux n'arrivait pas à identifier les blocs à utiliser. L'autre n'a pas réussi les trois exercices pour diverses raisons : il a eu un semestre de cours de programmation (ce semestre) et a encore des difficultés avec la programmation. Toutefois, tout ce qu'il a compris en programmation, il l'a relié avec les blocs.

Les exercices étaient faciles pour les volontaires qui sont à l'aise en programmation. C'est l'identification du rôle de chaque bloc qui leur a pris un peu de temps, ce qui est normal car c'est comme s'ils apprennent une nouvelle syntaxe de programmation. L'absence de la structure conditionnelle ELSE a perturbé certains volontaires parce qu'ils ont l'habitude d'utiliser le IF AND ELSE. L'idée d'utiliser successivement deux structures conditionnelles IF n'était pas dans leur réflexion. Pour la personne qui ont échoué l'exercice, si l'explication était détaillée ou un support d'utilisation a été donné, il aurait pu réussir.

Interfaces tangibles

Tout le monde était d'accord sur le choix des formes qui était arbitraire. A part pour le string qui évoque un texte, ils étaient au moins tous d'accord sur la forme du bloc qui évoque un mot ou une phrase. La forme des blocs est choisie arbitrairement parce qu'il n'y a pas de forme spécifique ou idéale pour représenter un nombre ou un booléen. Les formes sont bien expliquées dans le support pour que les utilisateurs les comprennent. Ce qui n'est pas le cas pour les exercices d'évaluation.

Les volontaires qui ont appris la programmation avec un langage non typé, ont trouvé les différentes formes pour les types (nombre, booléen, string) bizarres. Par contre, les volontaires qui ont appris la programmation avec un langage typé, n'avaient pas ce problème. La compréhension des différentes formes pour les types dépendent du langage de programmation enseigné ou appris aux volontaires. L'utilisation des formes pour les types ont un avantage et un inconvénient. Son avantage est de permettre aux étudiants de différencier, dès l'utilisation de l'interface tangible, les différents types. Son inconvénient est de rendre l'utilisation de l'interface plus complexe pour les novices parce qu'ils doivent comprendre à la fois l'affectation des variables et les différents types.

Concernant l'affectation des variables, ils trouvaient qu'utiliser la même forme pour le nom de variable et la valeur n'est pas intuitive.

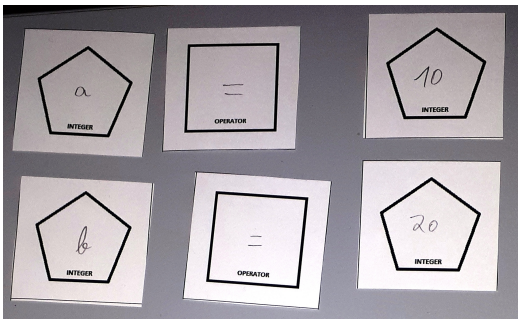


FIGURE 7.14 – Affectation

Dans leur esprit, écrire dans le bloc signifie que la valeur est affectée à une variable et le fait d'utiliser un autre bloc, plus le bloc avec le = n'est pas évident, même si en langage texte, l'affectation se fait « a = 20 ».

Le choix des symboles était bien choisi puisque les volontaires étaient tous d'accord et n'avaient pas de remarques négatives sur les deux concepts. Pour la boucle, la flèche qui tourne représente une répétition. Pour le point d'interrogation, il montre qu'il a un choix à faire. Ce résultat confirme l'étude de Palaude [20] sur les symboles par rapport à la boucle et la structure conditionnelle.

Pour la fonction, la remarque sur la forme du bloc était la même que pour la variable. Le choix de la forme était arbitraire. Sur l'utilisation, les volontaires n'avaient pas compris du premier coup qu'il fallait écrire le nom de la fonction sur le bloc. Ils cherchaient un bloc de fonction avec un mot print déjà imprimé. Le fait d'écrire sur le bloc de fonction peut complexifier l'utilisation de la fonction parce que l'utilisateur doit connaitre et mémoriser les noms de fonction qui peuvent être utilisés.

La question, pour vous est-il envisageable d'utiliser des blocs différents pour le concept de type (nombre, booléen, string) ?

Leurs réponses se basent sur leurs expériences en programmation. Comme déjà discuté dans l'interface tangible de variable, la compréhension des différentes formes pour les types dépendent du langage de programmation enseigné ou appris aux volontaires.

L'information textuelle en bas de chaque bloc vous aide-t-elle ?

Deux participants ont donné comme réponse le rôle de l'information textuelle en tant que contrainte. Les huit autres ont répondu que l'information textuelle donne une précision sur le concept ou le rôle du bloc. Néanmoins, les volontaires ont bien fixé l'orientation des blocs. Sans se rendre compte, ils les ont placés dans le bon sens. La présence de cette information textuelle joue deux rôles : une contrainte qui aide à bien orienter chaque bloc et une information supplémentaire pour leur utilisation.

Utilisabilité et utilité

Les réponses des participants reflètent leurs expériences sur l'évaluation. Les participants qui ont réussi les exercices ont un avis positif sur la nouvelle interface tangible de programmation. Ceux qui ont moins réussi ont répondu de manière mitigée.

Même s'ils n'avaient pas d'explication détaillée ou un support d'utilisation à disposition, avant de faire les trois exercices, ils ont bien su utiliser l'interface tangible de programmation. Ce qui prouve que l'utilisation de cette interface tangible de programmation n'est pas difficile.

En parallèle avec le cours standard de programmation, cette interface tangible de programmation peut être une aide supplémentaire pour les novices qui ont des difficultés à comprendre les concepts de base de programmation. Avec un bon support et une bonne explication sur son utilisation aux débutants, ces derniers n'auront pas de problèmes à l'utiliser.

7.3.3 Amélioration de la nouvelle interface tangible de programmation

Suite à l'évaluation et les analyses qui viennent d'être discutées, quelques améliorations ont été apportées à l'interface tangible de programmation.

Variable

Il y a deux choix pour la déclarer : typée et non typée. Le nom d'une variable et sa valeur sont mis dans un seul bloc. Avec ce système, le concept de variable est bien présenté en un bloc avec un nom qui contient une valeur. Donc pour accéder à la valeur d'une variable, nous utilisons simplement son nom.

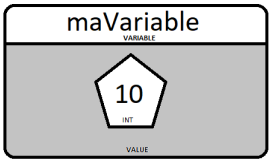


FIGURE 7.15 – Variable typée

Avec cette technique, le concept de type est gardé. Lorsque nous affectons une variable, nous plaçons la valeur typée à l'intérieur.

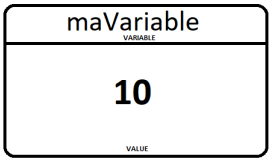


FIGURE 7.16 – Variable non typée

Avec cette technique, le concept de variable est simplifié parce qu'il n'y a plus de type à gérer. La confusion qui peut se présenter lorsque nous affectons une variable est résolue car la déclaration et l'affectation d'une variable se passent sur un seul bloc.

Structure conditionnelle

FIGURE 7.17 – Bloc d'une structure conditionnelle IF

FIGURE 7.18 – Bloc d'une structure conditionnelle ELSE

Désormais, notre structure conditionnelle contient l'IF AND ELSE. La structure conditionnelle IF est représentée par le même symbole qu'auparavant c'est à dire un point d'interrogation. La structure conditionnelle ELSE est représentée par un point d'interrogation à l'envers. Il implique que la condition acceptée est le contraire de celle qu'est déclarée par la structure conditionnelle IF.

Fonction

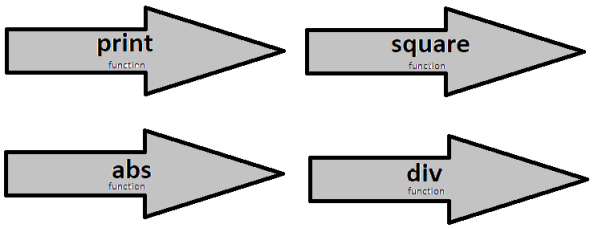


FIGURE 7.19 – Blocs de fonction

La forme du bloc de fonction a changé afin de la différencier de celle de la variable et d'indiquer que la fonction s'applique sur les paramètres qui la suivent.

L'idée est de fournir des blocs déjà imprimés avec le nom des fonctions pour faciliter son utilisation. Avec cette technique, les utilisateurs connaissent les fonctions qu'ils peuvent utiliser et qui sont reconnues par le système.

Utilisation de couleur

Les blocs avec un fond de couleur blanc sont les blocs où nous pouvons écrire dessus, les blocs avec un fond de couleur gris sont les blocs avec des symboles ou textes déjà imprimés.

Exemple d'assemblage

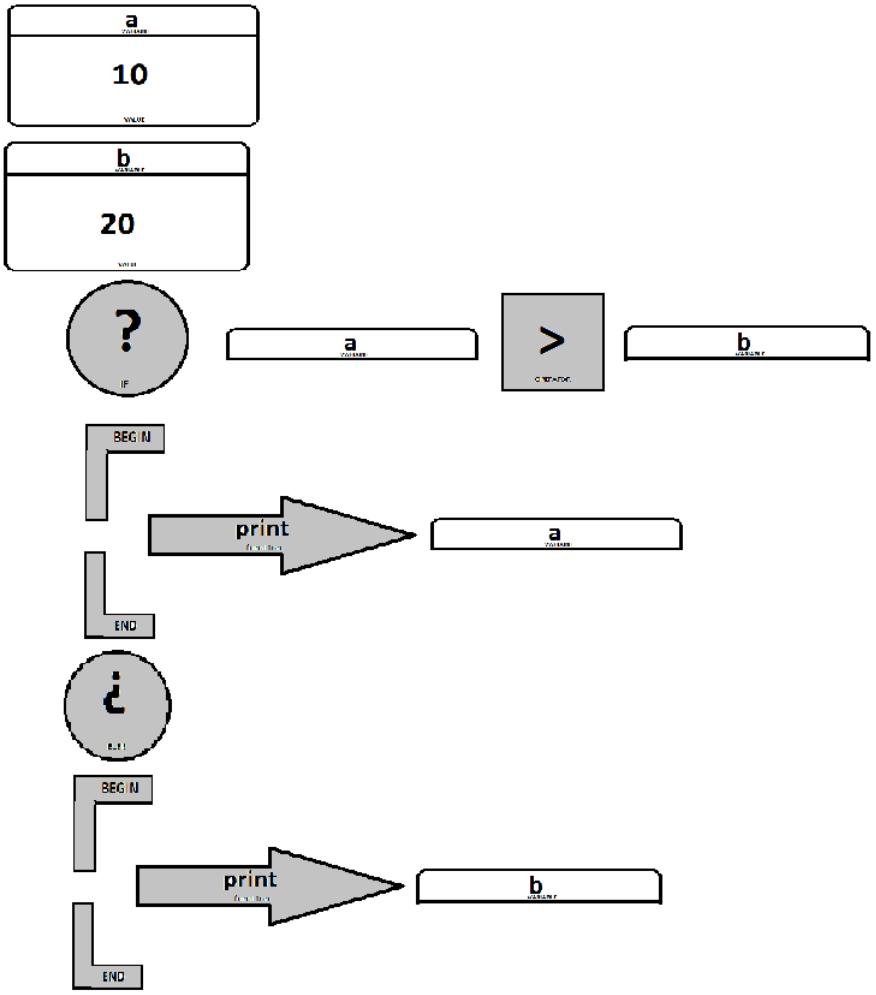


FIGURE 7.20 – Exemple de programme modélisé au moyen du prototype amélioré

8. Conclusion et travaux futurs

Ce mémoire est réalisé dans le but de trouver une réponse à la problématique « Quelles caractéristiques d'interfaces utilisateur tangibles permettent d'apprendre la programmation à des novices ? ».

Afin de répondre à cette problématique, nous avons cherché des interfaces tangibles de programmation destinées aux jeunes de plus de douze ans. Nous avons ensuite présenté chaque interface tangible de programmation sur quatre facteurs : l'âge des personnes à qui elle est destinée, le dispositif physique, l'utilisation et les concepts de programmation. Sur la base de ces informations, nous avons complété la classification de Fishkin [18], la classification de Palaude [20], la classification de Bodart [19] et la classification des contraintes que nous avons établie. Ces taxonomies et ces analyses nous ont permis de mettre en évidence les tendances d'interfaces tangibles de programmation. Sur la base des taxonomies et des tendances, une nouvelle interface tangible de programmation a été créée et validée par dix personnes.

Cela nous amène à répondre à la problématique. L'input et l'output d'une interface tangible doivent être séparés afin de bien distinguer la partie programmation et la partie exécution. La création d'un programme doit être un assemblage de bloc et chaque bloc doit être associé à un seul concept de programmation. Ces caractéristiques permettent à un utilisateur de bien identifier les concepts de programmation et de voir l'exécution qui correspond à un bloc ou un concept de programmation. L'exécution de l'interface tangible de programmation doit ressembler à un jeu afin d'attirer plus de monde. De la même manière, chaque concept de programmation doit être associé à une exécution spécifique. Les symboles utilisés doivent être des symboles significatifs pour faciliter l'interprétation. Les contraintes doivent permettre d'indiquer les erreurs dès l'assemblage pour que l'utilisateur les identifie rapidement. Malheureusement, ces contraintes sont chères ou difficile à mettre en place.

En réponse à ce travail, plusieurs travaux peuvent être effectués dans le futur.

Ainsi, concernant la nouvelle interface tangible de programmation, une évaluation doit être reproduite avec un groupe issu uniquement du public cible. Avec cette nouvelle évaluation, nous pourrons vérifier si grâce à l'interface tangible de programmation, les personnes qui ont du mal à comprendre avec l'apprentissage standard peuvent comprendre les concepts de base de la programmation.

La sélection des interfaces peut être élargie, notamment vers des outils commerciaux, et non forcément issus de la recherche scientifique. Eventuellement avec d'autres interfaces tangibles de programmation, nous pourrons trouver d'autre façon de présenter les concepts de programmation.

Il y a également le développement d'une application qui convertira la photo de l'en-

semble des blocs en ligne de code pour l'exécution, qui prend une photo d'assemblage de bloc puis la compile en ligne de code. Le but du compilateur est de reconnaître les formes de blocs et les symboles qu'ils représentent afin de savoir le concept utilisé et sa signification. Cette application peut être créée en se basant sur les interfaces tangibles de programmation qui présentent le même système (AR Maze [27], P Cube [34], Quetzal [35], Sheets [37], Story Block [39], T Maze [40], TaBGo [41], Tern [43]).

Après cette application de compilation, une application de jeu qui exécute le code peut être créée, comme dans codingame [49] qui est une application de programmation ludique. Les utilisateurs écrivent du code pour résoudre uneénigme sous forme de jeu d'animation.

A. Bibliographie

[1] Fahima Djelil. *Design and evaluation of an Object-Oriented Programming microworld based on a 3D construction and animation game*. Theses, Université Blaise Pascal - Clermont II, December 2016.

[2] Nicolas Guibert, Laurent Guittet, and Patrick Girard. Apprendre la programmation par l'exemple : méthode et système. In *Technologies de l'Information et de la Connaissance dans l'Enseignement Supérieur et de l'Industrie*, pages 345–352, Compiègne, France, October 2004. Université de Technologie de Compiègne. Les actes papiers peuvent être commandés à l'adresse suivante : http ://www.utc.fr/tice2004/commande_actes_tice2004.doc.

[3] Martinha Piteira and Carlos Costa. Computer programming and novice programmers. In *Proceedings of the Workshop on Information Systems and Design of Communication*, ISDOC '12, pages 51–53, New York, NY, USA, 2012. ACM.

[4] Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming : A review and discussion. *Computer Science Education*, 13 :137–, 06 2003.

[5] Julie Henry and Bruno Dumas. Apprentissage de la programmation par des novices : étude de l'adéquation entre concepts, outils d'apprentissage et besoins des utilisateurs. 03 2018.

[6] Janine Rogalski. Enseignement de méthodes de programmation dans l'initiation à l'informatique. In Georges-Louis Baron, Jacques Baudé, and Philippe Cornu, editors, *Colloque francophone sur la didactique de l'informatique*, pages 61–72, Paris, France, September 1988. Association EPI.

[7] A. Mayrhauser and A. Marie Vans. Program understanding - a survey. 09 1994.

[8] Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. A survey of literature on the teaching of introductory programming. In *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '07, pages 204–223, New York, NY, USA, 2007. ACM.

[9] Phit-Huan Tan, Choo-Yee Ting, and Siew-Woei Ling. Learning difficulties in programming courses : Undergraduates' perspective and perception. In *Proceedings of the 2009 International Conference on Computer Technology and*

Development - Volume 01, ICCTD '09, pages 42–46, Washington, DC, USA, 2009. IEEE Computer Society.

[10] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. A study of the difficulties of novice programmers. *SIGCSE Bull.*, 37(3) :14–18, June 2005.

[11] James C. Spohrer and Elliot Soloway. Novice mistakes : Are the folk wisdoms correct ? *Commun. ACM*, 29(7) :624–632, July 1986.

[12] David Weintrop and Uri Wilensky. To block or not to block, that is the question : Students' perceptions of blocks-based programming. In *Proceedings of the 14th International Conference on Interaction Design and Children*, IDC '15, pages 199–208, New York, NY, USA, 2015. ACM.

[13] *TEI '15 : Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction*, New York, NY, USA, 2015. ACM.

[14] B. Ullmer and H. Ishii. Emerging frameworks for tangible user interfaces. *IBM Systems Journal*, 39(3.4) :915–931, 2000.

[15] Sophie Lepreux, Julien Castet, Nadine Couture, Emmanuel Dubois, Christophe Kolski, Sébastien Kubicki, Valérie Maquil, and Guillaume Riviere. Tangible interaction on tabletop, définitions ans models. *Journal d'Interaction Personne-Système*, Volume 5, Number 1(1) :1–21, November 2016.

[16] Javier Pereda. Querying european linked data through tangible user interfaces. https ://trinkermedia.wordpress.com/tag/tangible-user-interface/, 2014.

[17] Ehud Sharlin, Benjamin Watson, Yoshifumi Kitamura, Fumio Kishino, and Yuichi Itoh. On tangible user interfaces, humans and spatiality. *Personal Ubiquitous Comput.*, 8(5) :338–346, September 2004.

[18] Kenneth P. Fishkin. A taxonomy for and analysis of tangible interfaces. *Personal Ubiquitous Comput.*, 8(5) :347–358, September 2004.

[19] Julie Henry, Bruno Dumas, and Antoine Bodart. Tangible Programming for Children : Survey of Kits, classification and opportunities. In AFIHM, editor, *30eme conférence francophone sur l'interaction homme-machine*, TeC-Travaux en Cours, page 9p, Brest, France, October 2018.

[20] Axel Palaude. Symbolic relevance of tangible user interfaces for learning programming. 2018.

[21] Junnan Yu and Ricarose Roque. A survey of computational kits for young children. In *Proceedings of the 17th ACM Conference on Interaction Design and Children*, IDC '18, pages 289–299, New York, NY, USA, 2018. ACM.

[22] Sophie Lepreux, Julien Castet, Nadine Couture, Emmanuel Dubois, Christophe Kolski, Sebastien Kubicki, Valérie Maquil, and Guillaume Riviere. Tangible interaction on tabletop, an illustrated federating framework. *Journal d'Interaction Personne-Système*, Volume 5, Number 1(1) :23 – 59, September 2017.

[23] Peter Dalsgaard and Kim Halskov. Tangible 3d tabletops : Combining tangible tabletop interaction and 3d projection. In *Proceedings of the 7th Nordic Conference on Human-Computer Interaction : Making Sense Through Design*, NordiCHI '12, pages 109–118, New York, NY, USA, 2012. ACM.

[24] 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, pages 159–162, New York, NY, USA, 2007. ACM.

[25] Brygg Ullmer, Hiroshi Ishii, and Robert J. K. Jacob. Token+constraint systems for tangible interaction with digital information. *ACM Trans. Comput.-Hum. Interact.*, 12(1) :81–118, March 2005.

[26] Hideyuki Suzuki and Hiroshi Kato. Algoblock : a tangible programming language, a tool for collaborative learning. In *Proceedings of 4th European Logo Conference*, pages 297–303, 1993.

[27] Qiao Jin, Danli Wang, Xiaozhou Deng, Nan Zheng, and Steve Chiu. Ar-maze : A tangible programming tool for children based on ar technology. In *Proceedings of the 17th ACM Conference on Interaction Design and Children*, IDC '18, pages 611–616, New York, NY, USA, 2018. ACM.

[28] Theodosios Sapounidis and Stavros Demetriadis. Touch your program with hands : Qualities in tangible programming tools for novice. In *Proceedings of the 2011 15th Panhellenic Conference on Informatics*, PCI '11, pages 363–367, Washington, DC, USA, 2011. IEEE Computer Society.

[29] José María Rodríguez Corral, Antón Civit Balcells, Arturo Morgado Estévez, Gabriel Jiménez Moreno, and María José Ferreiro Ramos. A game-based approach to the teaching of object-oriented programming languages. *Computers & Education*, 73 :83 – 92, 2014.

[30] David Merrill, Emily Sun, and Jeevan Kalanithi. Sifteo cubes. In *CHI '12 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '12, pages 1015–1018, New York, NY, USA, 2012. ACM.

[31] 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*, IDC '13, pages 561–564, New York, NY, USA, 2013. ACM.

[32] Danli Wang, Yang Zhang, and Shengyong Chen. E-block : A tangible programming tool with graphical blocks. *Mathematical Problems in Engineering*, 2013, 02 2013.

[33] Vishesh Kumar, Tuhina Dargan, Utkarsh Dwivedi, and Poorvi Vijay. Note code : A tangible music programming puzzle tool. In *Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '15, pages 625–629, New York, NY, USA, 2015. ACM.

[34] S. Kakehashi, T. Motoyoshi, K. Koyanagi, T. Oshima, H. Masuta, and H. Kawakami. Improvement of p-cube : Algorithm education tool for visually impaired persons. In *2014 IEEE Symposium on Robotic Intelligence in Informationally Structured Space (RiiSS)*, pages 1–6, Dec 2014.

[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, pages 159–162, New York, NY, USA, 2007. ACM.

[36] 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*, IDC '12, pages 98–105, New York, NY, USA, 2012. ACM.

[37] Jiro Tanaka Kazuki Tada. Tangible programming environment using paper cards as command objects. In *6th International Conference on Applied Human Factors and Ergonomics nd the Affiliated Conferences, AHFE*, 2015.

[38] 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*, ISS '17, pages 300–305, New York, NY, USA, 2017. ACM.

[39] Varsha Koushik and Shaun K. Kane. Tangibles + programming + audio stories = fun. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '17, pages 341–342, New York, NY, USA, 2017. ACM.

[40] 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*, IDC '11, pages 127–135, New York, NY, USA, 2011. ACM.

[41] Jean-Baptiste Marco, Nadine Jessel-Baptiste, and Philippe Truillet. TaBGO : Programming with tangibles blocks. In AFIHM, editor, *30eme conférence francophone sur l'interaction homme-machine*, TeC-Travaux en Cours, page 7p, Brest, France, October 2018.

[42] Yunfeng Qi, Danli Wang, Lan Zhang, and Yining Shi. Tanprostory : A tangible programming system for children's storytelling. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '15, pages 1001–1006, New York, NY, USA, 2015. ACM.

[43] 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, pages 1965–1970, New York, NY, USA, 2007. ACM.

[44] D. Gallardo, C. F. Julia, and S. Jorda. Turtan : A tangible programming language for creative exploration. In *2008 3rd IEEE International Workshop on Horizontal Interactive Human Computer Systems*, pages 89–92, Oct 2008.

[45] Sébastien George, Iza Marfisi-Schottman, and Marc Leconte. TurtleTable : apprendre les bases de la programmation avec des interfaces tangibles. In *Atelier Apprentissage Instrumenté de l'Informatique, Orphee RDV*, Font-Romeu, France, January 2017.

[46] Jorma Sajaniemi. The roles of variables. http ://saja.kapsi.fi/var_roles/, 2008.

[47] 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*, CHI '05, pages 859–868, New York, NY, USA, 2005. ACM.

[48] J. Lewis. Ibm computer usability satisfaction questionnaires : psychometric evaluation and instructions for use. In *International Journal of Human-Computer Interaction*, pages 57–78, 1995.

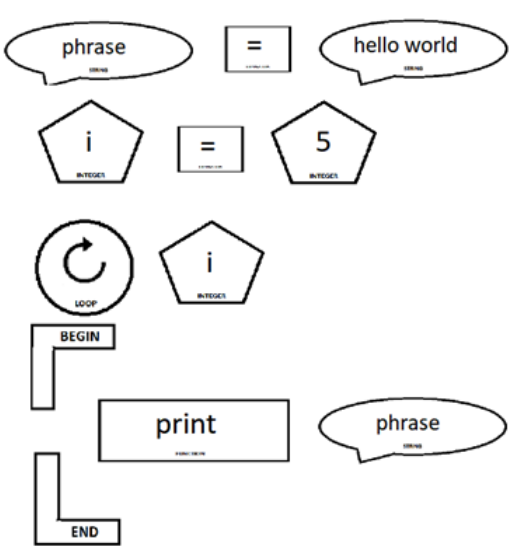
[49] Prins Butt. Students' perceptions of game-based learning using codingame. 2016.

B. Annexes

B.1 Exercices

Exercice 1

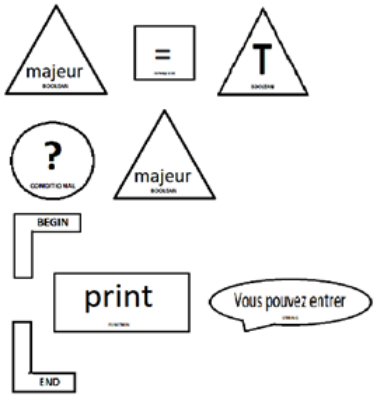
Traduire cet ensemble de bloc en langage python ou en pseudo code



Résultat

Exercice 2

Traduire cet ensemble de bloc en langage python en pseudo code



Résultat

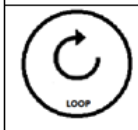
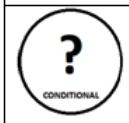
Exercice 3

Assembler les blocs afin de résoudre le problème suivant :
— Déclarer deux nombres a et b avec les valeurs a = 10 et b = 20
— Comparer ces deux nombres et afficher le plus grand des deux

B.2 Questionnaires

Nom et Prénom :

Dans le tableau ci-dessous, merci de donner pour chaque symbole :
— la (les) signification(s) du symbole selon vous (et pourquoi ce symbole vous inspire cette signification)
— la note que vous accordez à ce symbole par rapport à la signification que vous venez de donner
* en bref, ce symbole est-il bien choisi ? Représente-t-il bien ce qu'il devrait représenter selon vous ?
1 : Pas du tout d'accord ; 2 : Pas d'accord ; 3 : Indifférent ; 4 : D'accord ; 5 : Tout à fait d'accord
— Selon la réponse donnée pour la crédibilité, justifiez ou suggérez mieux.

	Signification	Note					Justification/Suggestion
		1	2	3	4	5	
							
							
							
							
							
							

Selon vous, est-il envisageable d'utiliser des blocs différents pour un même concept (le concept de variable : nombre, booléen, string) ? OUI – NON

Si oui, donnez des exemples qui justifient votre réponse.

L'information textuelle en bas de chaque bloc vous aide t-elle ? Si oui, comment ?

Évaluez l'utilisabilité du système (à savoir l'ensemble des blocs avec symboles) en complétant la grille ci-dessous :

1 : Pas du tout d'accord ; 2 : Pas d'accord ; 3 : Indifférent ; 4 : D'accord ; 5 : Tout à fait d'accord

	1	2	3	4	5
J'ai trouvé ce système inutilement complexe					
J'ai trouvé ce système facile à utiliser					
Je pense que j'ai besoin d'un support technique pour être capable d'utiliser ce système					
J'ai trouvé qu'il y avait trop d'incohérence dans ce système					
Je suppose que la plupart des gens apprendraient très rapidement à utiliser ce système					
J'ai trouvé ce système très contraignant à utiliser					

Évaluez l'utilité du système en complétant la grille ci-dessous :

1 : Pas du tout d'accord ; 2 : Pas d'accord ; 3 : Indifférent ; 4 : D'accord ; 5 : Tout à fait d'accord

	1	2	3	4	5
De manière générale, je trouve ce système facile à utiliser					
Ce système est simple à utiliser					
Je peux comprendre facilement les concepts de base de programmation en utilisant ce système					
Je suis à l'aise avec l'utilisation de ce système					
Cela a été facile d'apprendre à utiliser ce système					

B.3 Transcriptions

Scéance 1

— Introduction et une brève explication
— .....
— Participant : je dois le traduire en pseudo code ou en code ?
— Moi : commet tu veux, tu peux le traduire en pseudo code, tu peux le traduire en code
— .....
— Moi : Qu'est ce que tu penses de l'interface tangible de programmation
— Participant :
- c'est un peu bizarre de ne pas avoir un bloc else
- c'est un peu bizarre d'utilisé plusieurs forme pour le variable

— .....

Scéance 2

— Introduction et une brève explication
— .....
— Participant : Est-ce qu'on doit utiliser le bloc opérateur pour ... (affectation d'une variable)
— Moi : Normalement oui
— .....
— Moi : Un commentaire par rapport à l'utilisation de l'interface tangible
— Participant : C'est flou sur la distinction du type et du variable elle-même, ça crée une confusion. Tu peux aussi utiliser des couleurs pour les différenciés.
— .....

Scéance 3

— Introduction et une brève explication
— .....
— Participant 1 : Il y a quelque chose qui n'est pas cohérent sur l'interface tangible
— .....
— Participant 1 : je peux expliquer comment je le comprends
— Participant 2 : je ne me souviens plus comme écrire cela en code
— .....
— Participant 1 : Ou sont les valeurs ?
— Moi : Tu peux écrire les valeurs sur les blocs
— Participant 1 : Si je veux écrire un nombre
— Moi : je l'ai montré un exemple
— Participant 1 : comment les étudiants puissent savoir les types, t'as du crée un type n'importe quoi
— .....
— Participant 1 et 2 : elle est ou la structure conditionnelle else
— .....
— Participant 1 : Est-ce qu'on peut faire une opération ternaire ?
— Moi : on peut faire mais il faut bien utiliser les blocs
— .....
— Participant 1 : Est qu'il a des négations
— Moi : non, l'interface tangible essaie de faciliter la compréhension de la programmation mais il n'est pas créer pour prendre la place des langages
— .....
— Participant 2 : Comment je fais s'il n'y a pas de else et je veux tester deux conditions
— Moi : il vérifie les deux conditions mais il exécute la partie du code qui est vraie
— .....
— Moi : tu peux écrire directement sur les blocs
— Participant 2 : c'est quoi un opérateur ?
— Moi : *, +, =, > ...
— .....

Scéance 4

— Introduction et une brève explication
— .....
— Participant 1 : Est-ce que j'explique l'ensemble de bloc ou je traduis le code ?
— Moi : Soit en pseudo code, soit en code
— .....
— Participant 2 : J'ai du révisé avant ...
— .....
— Participant 2 : Il est marqué quoi en dessous du bloc
— Moi : Le concept qu'il représente
— .....
— Participant 1 : C'est quoi le forme carré ?
— Moi : Opérateur
— .....
— Participant 3 : le troisième exercice doit ressembler à l'assemblage de l'exercice un et deux
— Participant 2 : Je peux expliquer l'exercice un et deux parce que je ne me souviens pas comment programmer
— Moi : Tu peux expliquer
— .....
— Participant 1 : c'est quoi le majeur
— Moi : C'est une variable
— .....
— Participant 2 : c'est quoi le triangle
— Moi : c'est un boolean
— .....
— Participant 2 : Le but est de faire l'exercice de manière instinctive
— Moi : oui, l'idée c'est ça, c'est pour ça que je n'ai pas donné une explication détaillé
— .....
— Moi : Est-ce que vous avez des remarques par rapport au test, par rapport à l'interface tangible
— Participant 1 : C'est claire comme même, j'ai eu de difficulté sur l'utilisation de la fonction
— .....

Scéance 5

— Introduction et une brève explication
— .....
— Participant 1 : Le triangle signifie un boolean, c'est toi qui a ajouté le nom et le motif
— Moi : la forme représente un concept et les valeurs peuvent être écrites dessus
— Participant 2 : Le triangle représente quoi
— Moi : C'est marqué en bas des blocs leur signification
— .....
— Participant 3 : il est où l'opérateur égale

— Participant 1 : l'opérateur
— Moi : oui, tu dois marquer égale dessus
— .....
— Participant 1 : J'ai pensé à écrire directement dans le bloc, après il faut utiliser un autre bloc pour l'affectation, c'est un peu bizarre
— .....
— Participant 1 : le begin et le end ne sont pas instinctifs
— .....