Analyse de l'adaptabilité de jeux pour l'apprentissage de la pensée informatique ou de la programmation

Hajar Saddoug[1], Aryan Rahimian[1], Marne Bertrand[2][0000-0002-4953-9360], Mathieu Muratet[3][0000-0001-6101-5132], Karim Sehaba[4][0000-0002-6541-1877] and Sébastien Jolivet[5][0000-0003-3915-8465]

[1] Sorbonne Université, Place Jussieu, Paris, France
hajar.saddoug@etu.sorbonne-universite.fr,
rahimian_aryan@yahoo.fr
[2] ICAR UMR 5191, Université Lumière Lyon 2, Parvis René Descartes, Lyon, France
bertrand.marne@ens-lyon.fr
[3] Sorbonne Université, CNRS, INS HEA, LIP6, F-75005 Paris, France
mathieu.muratet@lip6.fr
[4] LIRIS - Université Lumière Lyon 2, avenue Pierre Mendès-France, Lyon, France
karim.sehaba@liris.cnrs.fr
[5] IUFE, Université de Genève, rue du Général-Dufour, Genève, Switzerland
sebastien.jolivet@unige.ch

Résumé. Ce travail s'inscrit dans la problématique générale de l'appropriation par les enseignants et les formateurs des jeux sérieux dédiés à l'apprentissage de la pensée informatique ou de la programmation. Pour favoriser cette appropriation, nous souhaitons mettre en œuvre une approche méta-design favorisant la genèse instrumentale. Dans cet article, nous cherchons à tester à quel point une sélection de jeux sérieux se prête à ce type d'approche. Nous y identifions des critères d'analyse destinés à caractériser la capacité à l'instrumentalisation de jeux sérieux et avec ceux-ci nous expertisons en profondeur 10 jeux. Nos résultats montrent que les capacités d'adaptation de la plupart de ces 10 jeux restent faibles et hors de portée de beaucoup d'enseignants et formateurs.

Mots-clés : Jeux sérieux, Méta-design, Genèse instrumentale, Pensée informatique, Programmation.

1 Introduction

La question de l'enseignement de l'informatique, qui remonte aux années 70, se caractérise par un mouvement de balancier entre deux conceptions de ce qu'il faut enseigner. D'un côté, une vision d'une informatique-outil qui promeut la formation à l'usage des outils. De l'autre, une vision de l'informatique en tant qu'objet d'enseignement qui considère qu'il faut avant tout enseigner les concepts et méthodes de l'informatique [1]. Depuis maintenant quelques années, l'informatique disciplinaire réapparaît dans les programmes scolaires, à l'école on parle plus particulièrement de *pensée informatique*.

Selon Jeannette Wing [2], la pensée informatique met en jeu un répertoire de cinq capacités cognitives : (1) la pensée algorithmique, (2) l'abstraction, (3) l'évaluation, (4) la décomposition, et (5) la généralisation. Gilles Dowek [3] quant à lui propose une structuration de l'informatique en quatre concepts afin que les contenus enseignés donnent une image fidèle de la discipline elle-même : (1) l'information numérique, (2) les algorithmes, (3) les langages, et (4) les machines informatiques. Ainsi, l'informatique ne se réduit pas au codage qui ne constitue que la phase finale de traduction dans un langage de programmation. Elle est avant tout une activité conceptuelle. Ainsi conçue, l'informatique quitte le statut d'outil pour ce qu'elle est en réalité : une science ayant ses spécificités et nécessitant un apprentissage propre. Cela étant, enseigner la pensée informatique au primaire et l'informatique au secondaire demande un investissement particulièrement important de la part des enseignants [4] pour aborder cette nouvelle discipline, du fait d'un manque de formation (initiale et continue).

En parallèle de ces évolutions, les jeux sérieux d'apprentissage ont émergé et proposent de nombreux avantages par rapport aux outils d'enseignement classique. En effet, de nombreux auteurs considèrent les jeux sérieux d'apprentissage comme prometteurs, notamment pour augmenter l'engagement et la motivation des apprenants [5], d'autres envisagent ces outils pour favoriser un apprentissage plus constructiviste [6]. Concernant le thème de l'informatique et de la programmation, de nombreux jeux sérieux existent [7, 8], mais leur appropriation par les enseignants reste un enjeu majeur. Nous faisons l'hypothèse qu'une réponse à ce défi est d'impliquer les enseignants à l'aide d'une méthode de conception participative telle que le méta-design.

Le méta-design est une méthode de conception participative avancée, dans laquelle les utilisateurs finaux (« *owners of problems* ») sont impliqués de façon centrale dans les phases initiales de la conception. Mais, et c'est la raison de sa pertinence pour résoudre les problèmes d'appropriation, les utilisateurs doivent aussi avoir les moyens de continuer à être concepteurs durant les phases d'utilisation des artefacts, grâce à *l'underdesign* que Fischer et al. [9] définissent comme : « [...] *underdesign aims to provide social and technical instruments for the owners of problems to create the solutions [of their problems] themselves at use time.* ». Il s'agit donc, dans notre cas de figure, d'identifier si des enseignants et formateurs, considérés comme une des catégories d'utilisateurs finaux de jeux sérieux dédiés à la programmation, parviennent à s'inscrire dans une démarche de méta-design (les apprenants-joueurs sont une autre catégorie d'utilisateurs finaux non traitée ici). Nous faisons l'hypothèse que cette démarche peut favoriser la genèse instrumentale [10] et plus particulièrement l'instrumentalisation, et donc une forme d'appropriation [11].

Néanmoins pour pouvoir mettre en œuvre cette démarche de méta-design, il est nécessaire de disposer d'artefacts adaptables (dans notre cas de jeux sérieux) fournissant les fonctionnalités permettant à ses utilisateurs finaux (enseignants ou des formateurs) d'observer leurs usages à différents niveaux d'abstraction et de les adapter en fonction, en prenant en compte leur contexte et leurs besoins. Dans cet article, nous réalisons une analyse de jeux sérieux sur le thème de la pensée informatique avec comme focale les outils mis à disposition pour favoriser la genèse instrumentale et l'appropriation de ces jeux par les enseignants [11].

Dans la première partie, nous présentons notre méthode de travail, c'est-à-dire comment nous avons élaboré nos critères d'analyse, comment nous avons construit notre sélection de jeux à expertiser et comment nous avons mené l'analyse. Dans la seconde partie, nous présentons et discutons les résultats de cette analyse, avant de conclure dans la dernière partie.

2 Méthode

La littérature propose déjà plusieurs travaux de recensement et d'analyse de jeux sérieux destinés à la programmation. Parmi les travaux que nous avons pu consulter, nous remarquons que l'attention est principalement portée sur les thèmes et les compétences enseignées par les jeux sérieux analysés [7, 8, 12, 13]. Parmi ces travaux, certains comme Lindberg et al. [12], identifient un mauvais alignement entre les compétences enseignées dans les jeux sérieux (29 ont été analysés) et celles présentes dans les programmes (notamment français). On retrouve cette tendance chez Malliarakis et al. et Miljanovic et Bradbury [7, 13] avec respectivement 12 et 49 jeux expertisés (et de nombreux recoupements entre ces travaux). Cependant, ces travaux ne cherchent jamais à caractériser la genèse instrumentale, notamment par l'instrumentalisation. Seuls Vahldick et al. [8] évoquent l'importance que peuvent jouer des éditeurs dans certains cas, sans, pour autant, en faire un critère d'analyse des jeux expertisés (40 jeux analysés).

Nous avons donc entrepris de conduire notre propre analyse en nous concentrant sur la capacité des jeux expertisés à disposer, d'une part, des moyens d'observation permettant à l'enseignant de comprendre les usages du jeu, et d'autre part, des mécanismes d'adaptation lui permettant de réajuster le jeu en fonction de ces usages observés.

2.1 Sélection des jeux

Pour construire la liste des jeux traitant de la pensée informatique à analyser, nous avons associé les jeux que nous avions déjà identifiés dans d'autres travaux [14, 15], à ceux qui proviennent des revues systématiques citées plus tôt, et à d'autres, recherchés sur le web avec les mots-clés « jeux introduction programmation », mais également « jeux codage », « coding game ». Nous n'avons conservé que les jeux sérieux actuellement fonctionnels (par exemple, les jeux utilisant la technologie *Flash* sont désormais très difficilement utilisables), peu chers ou gratuits. Ainsi, nous avons pu recenser 48 jeux permettant l'apprentissage du code ou de la pensée informatique.

Comme nous nous concentrons sur les jeux sérieux destinés à l'apprentissage de la pensée informatique et de la programmation dans un contexte scolaire, nous avons aussi choisi d'exclure les jeux ne présentant aucun accès direct à la programmation par le joueur, ainsi que ceux seulement disponibles sur téléphone mobile ou tablette (dont l'utilisation est plus contrainte en situation scolaire). Nous avons également regroupé les jeux qui proposent un fonctionnement similaire à Scratch [16] ou Blockly [17] qui sont des micromondes [18] proposant une programmation par blocs d'instructions.

Dans un second temps, nous avons passé en revue la liste de jeux, pour les trier et les catégoriser en prenant en compte les éléments suivants :

- Ressemblances et dissemblances entre les jeux ;
- Avantages et inconvénients des outils-auteurs et de suivi présents dans les jeux ;
- Catégories (ex. : âge, discipline, coût, etc.) ;
- Type de formation (lorsque celle-ci est proposée par le jeu) : autoformation, enseignement classique, etc.

Cette catégorisation nous a permis d'affiner la liste pour nous concentrer sur une sélection de 10 jeux (parmi les 48) présentant peu de points communs et correspondant bien à notre objectif : identifier des jeux sérieux centrés sur l'apprentissage de la pensée informatique et de la programmation, qui pourraient être adaptés ou facilement appropriés par des enseignants et des formateurs, c'est-à-dire permettant l'instauration d'un contexte de méta-design. Désormais, chaque jeu retenu pour l'analyse est associé à une catégorie distincte.

Table 1. Liste initiale des 48 jeux, les 10 jeux sélectionnés sont mis en évidence.

Algoblocs	Code hunt	CodeWars	Elevator Saga	Hocus Focus	**Pyrates**	SQL Murder Mystery
AlgoPython	Code Moji	**Codin'Game**	Empire Of Code	Human Resource Machine	**RoboCode**	StarLogo
Bee Bot	Code Monkey	Collabots	Flexbox Defense	Imagi	Ruby Warrior	Tynker
Blockly	Code Monster	**Compute It**	Flexbox Froggy	**KoduGame**	Run Marco	Unruly Splats
Ceebot	CodeCombat	CSS Diner	Gladiabots	Le chevalier de la programmation	Scratch	Untrusted
CheckIO	Codefi	Cyber Dojo	Grid Garden	Pixel	Screeps	Vim adventures
Code	CodeGym	Duskers	Heartbreak	ProgAndPlay	**SPY**	

2.2 Pré-création de la grille d'analyse

Afin d'établir notre grille d'analyse, nous avons identifié plusieurs critères. Dans un premier temps, nous avons identifié trois temporalités liées aux modifications potentielles qui pourraient être opérées par des enseignants ou des formateurs dans les jeux de la liste. À chacune de ces trois temporalités (avant, pendant et après l'instrumentalisation), nous avons associé des types d'assistance à la modification qui nous serviront de critère :

- **Pré-modification** : présence de tutoriels et d'explications de différents formats, concernant la manipulation du jeu et/ou son utilisation à d'autres fins formatives.

- **Durant la modification** : capacité à la détection des erreurs pendant la manipulation du jeu.
- **Post-modification** : possibilité d'avoir une vue d'ensemble des modifications appliquées au jeu et/ou la présence d'une aide complémentaire (maintenance humaine ou numérique interactive type forum, FAQ, etc.)

Dans un second temps, et pour donner suite à cette étape préparatoire, nous avons formulé d'autres critères plus fins et facilement mesurables. Ces critères sont détaillés dans la section suivante.

2.3 Critères d'analyse

Notre but est d'analyser les jeux, non pas pour en sélectionner certains et en disqualifier d'autres, mais de les catégoriser avec des critères fins, afin d'offrir une grille claire rendant compte des convergences et divergences entre jeux adaptables et dotés de moyens de collecte de traces, dédiés à l'enseignement de la programmation ou de la pensée informatique.

Nous avons hésité à utiliser un système de score en raison de sa subjectivité, d'autant plus que l'essentiel dans notre grille était d'indiquer la présence effective ou non du critère défini. Ainsi, nous avons opté pour les appellations « Présence » ou « Absence » du critère, et dans le cas de la présence, nous indiquons toujours en commentaire sous quelle forme le critère est présent. Néanmoins, pour l'un des critères, le « Degré » de scénarisation, nous devions pouvoir être plus précis, sans pour autant indiquer trop de détails inutiles à une analyse rapide. Par conséquent, nous avons opté pour 3 états possibles : un niveau faible, un niveau fort ou une absence de scénario.

Au fur et à mesure que des critères apparaissaient lors de nos explorations, nous avons décidé de les ranger par classe. Nous avons ainsi identifié 7 classes comprenant chacune différents critères dont voici la description.

Table 2. Récapitulatif des 7 classes de critères utilisés dans notre analyse

Classes	Critères associés
Adaptabilité	Code source libre, Profil enseignant, Modification IHM, Modalité d'interaction
Édition	Modification de tâches, Ajout de tâches, Organisation de tâches, Création de scénario, Éditeur fourni
Capacité formative	Guide explicatif (pour jouer), Guide pédagogique (pour éditer), Assistance didactique, Assistance pédagogique
Traces	Progression, Performance, Informations générales, Format de trace
Diversité de choix	Langage de programmation
Communauté	Forum amateurs, Contact avec l'auteur / éditeur
Scénarisation	Degré, Tâches indépendantes

L'adaptabilité est une classe de critères centrale pour nos travaux, elle décrit la capacité du jeu à être modifié. Celle-ci se décompose en 4 critères :

- Code source libre : nous avons cherché à identifier la disponibilité du code source, sa licence, et la présence de ressources numériques ayant été utilisées.
- Profil enseignant : nous avons identifié si un compte et un profil spécifique existaient pour des formateurs ou des enseignants, leur donnant accès à des fonctionnalités spécifiques (créer des leçons, les organiser, visualiser des traces, scores, etc.).
- Modification IHM : nous avons identifié si l'interface du jeu était modifiable pour être adaptée.
- Modalité d'interaction : nous avons identifié s'il était possible de modifier les interactions existantes dans le jeu (par exemple : l'utilisation d'autres outils que le clavier et la souris).

La classe « Édition » se distingue de la classe « Adaptabilité », car elle est moins générale et vise spécifiquement la modification du contenu du jeu (tâche et scénario). Ses critères sont :

- Modification de tâches : la possibilité de modifier le contenu d'une tâche existante, d'un niveau, son niveau de difficulté, etc.
- Ajout de tâches : la possibilité d'ajouter des tâches à celles existantes dans le jeu.
- Organisation de tâches : la possibilité de réorganiser les tâches existantes ou celles que l'on a soi-même créées (quand c'est possible), pour éventuellement créer un scénario.
- Création de scénario : la possibilité de créer un enchaînement de tâches.
- Éditeur fourni : la mise à disposition ou non d'un éditeur facilitant la création, la modification et l'organisation des tâches.

Dans le contexte de nos travaux, nous avons mis en avant une autre classe de critères qui nous paraît importante : « Capacité formative ». Elle a pour but d'identifier, par exemple, si le système forme l'enseignant, ou s'il l'aide à enseigner la programmation. Est-ce que le système lui indique clairement ce qu'il peut faire avec lui ? Si oui, comment le fait-il ? Lui fournit-il des feedbacks durant sa conception pédagogique ? Lui propose-t-il des exercices adaptés aux notions visées ? Le corrige-t-il dans la difficulté de ces derniers ? etc. Donc, il s'agit d'identifier toute information ou tout moyen facilitant la prise en main du système et l'enseignement par celui-ci. Que ce soit avant même d'élaborer et de composer sa leçon dans le système ou pendant son élaboration. Dans cette classe, nous avons 4 critères :

- Guide explicatif (pour jouer) : décrit si sont présentes des informations potentiellement utiles à la compréhension et au bon usage du jeu dans le système.
- Guide pédagogique (pour éditer) : décrit si sont présentes des informations potentiellement utiles à la compréhension et au bon usage de l'enseignement dans le système.
- Assistance didactique : décrit si sont présents des moyens d'aide à l'enseignant pour mieux enseigner les notions visées (en proposant notamment des exercices adaptés)
- Assistance pédagogique : décrit si est présente une assistance qui corrige et guide l'enseignant pendant qu'il compose sa leçon dans le jeu, afin de la rendre plus efficiente.

La classe « Traces » décrit au travers de plusieurs critères les moyens mis à disposition pour collecter, consulter et analyser des traces des actions des apprenants dans le jeu. En effet, l'adaptation d'un jeu par l'enseignant ou le formateur est souvent motivée par des écarts entre des comportements supposés (des apprenants-joueurs) durant la conception du jeu et les pratiques réelles qui peuvent surgir durant son déploiement. De tels écarts ne sont perceptibles, dans certains cas, qu'à travers la mise en place d'un système de collecte de traces, représentant les interactions entre l'utilisateur et le jeu. Les traces ainsi collectées peuvent faire l'objet de transformation afin de mettre en évidence des indicateurs de haut niveau d'abstraction assistant l'enseignant dans la mise en œuvre des adaptations qui s'imposent pour le bon déroulement de l'apprentissage.

Cette classe n'est pas indépendante de la « Capacité formative » ni de « Adaptabilité » (et plus particulièrement son critère « Profil enseignant »). En effet, la présence d'une interface enseignant semble nécessaire à l'accès aux traces. Les critères de cette classe sont :

- Progression : identifie la présence de traces (indicateurs) de la progression des apprenants dans le jeu (les niveaux passés, débloqués, les niveaux rejoués, etc.).
- Performance : identifie la présence de traces (indicateurs) de la performance de chaque apprenant (les scores, les badges obtenus, etc.).
- Informations générales : identifie la présence d'informations sur les apprenants (noms, nombres, classe, groupe, etc.)
- Format de trace : décrit le format des traces fournies (brut ou raffiné). Ici, nous distinguons le format raffiné, à savoir des informations rassemblées et présentées dans le but d'informer l'utilisateur sur les traces ; et le format brut qui s'illustre par des informations partielles et dispersées dans le système.

La classe « Communauté » ne regroupe que deux critères, l'existence d'un lieu d'échanges entre les utilisateurs d'une part (réseau social, forums…) et, d'autre part, celle de coordonnées de contact des auteurs ou de l'éditeur du jeu. Nous avons ajouté cette classe pour identifier si de l'aide extérieure était éventuellement accessible à des enseignants ou des formateurs qui voudraient se lancer dans des modifications.

La classe « Scénarisation » porte sur la scénarisation du jeu et indique le degré de dépendance entre les tâches. Les 2 critères sont :

- Degré : qui indique à quel point la scénarisation est forte, c'est-à-dire l'importance d'une histoire ou d'un contexte qui relie les différentes étapes unitaires du jeu (niveaux, tableaux, exercices, etc.) entre-elles. Ce critère peut avoir 3 valeurs (forte, faible, absence).
- Tâches indépendantes : qui indique le degré de dépendance des tâches (les unités qui composent la scénarisation). Dans le cas d'une scénarisation forte, on s'interrogera sur l'indépendance des tâches du scénario. Peuvent-elles être modifiées séparément et sans affecter la scénarisation générale ?

Un dernier critère ne pouvait être classé ailleurs : « Langage de programmation ». Nous lui proposons donc une classe spécifique « Diversité de choix ». Il s'agit d'indiquer si le jeu permet l'utilisation d'un ou plusieurs langages de programmation. Nous

ne l'avons pas placé dans la classe « Édition », car il s'agit de la diversité de langage pour jouer et non pour éditer. La classe « Diversité de choix » est la seule classe qui est spécifique aux jeux dédiés à l'apprentissage de la pensée informatique et de la programmation. En effet, toutes les autres classes ne contiennent que des critères qui pourraient être utilisés pour analyser l'adaptabilité de jeux sérieux dédiés à d'autres types d'apprentissages.

2.4 Procédure d'analyse

Pour l'analyse de chaque jeu avec la grille, nous avons procédé de la manière suivante : dans un premier temps, le même jeu a été analysé par deux examinateurs séparément, avec une prise de notes des éléments pertinents à retenir. Dans un second temps, la grille a été complétée conjointement. Le temps accordé pour l'examen de chaque jeu fut différent. En effet, nous avons remarqué que selon le type du jeu, le délai d'un examen complet était en moyenne d'une heure. Pour d'autres jeux, étant donné leur caractère simple et leur interface minimaliste, la durée de leur analyse était sensiblement plus courte.

La phase d'analyse n'a pas seulement été de jouer au jeu, mais a aussi nécessité un temps d'appropriation et de maîtrise par les examinateurs : lecture de guides, visionnage de tutoriels avant de jouer et de tester l'éditeur de niveau quand il était présent. Cette analyse a eu comme particularité une approche méta-ludique, dans laquelle les examinateurs devaient être conscients du jeu et d'eux-mêmes en tant que joueurs pendant l'acte de jeu. Cette attitude est nécessaire pour identifier les avantages et les points d'amélioration de chaque jeu et de constater, au fur et à mesure, les points convergents et divergents entre ces derniers.

En effet, l'analyse s'est réalisée par classe de critères, dès que toutes les cases d'une même classe étaient remplies, l'examinateur passait à l'analyse d'une autre classe de la grille. Certaines classes, comme « Adaptabilité » et « Communauté » demandaient plus de temps à l'examinateur que les autres, car des recherches annexes, notamment sur le web, devaient être réalisées pour juger de l'existence de certains critères en dehors du jeu lui-même. Par exemple, la disponibilité du code source libre, la présence de forums dédiés au jeu ou toute information annexe présente sur des blogs d'amateurs.

Une colonne commentaire a été ajoutée à chaque classe, pour indiquer des informations complémentaires sur la présence ou l'absence de certains critères, mais aussi pour ajouter des particularités observées dans le jeu.

3 Résultats et discussion

Par manque de place, nous ne présentons pas ici le détail de l'analyse des 10 jeux retenus, mais la grille d'analyse complète est disponible à l'adresse web suivante : https://recherche.univ-lyon2.fr/meta-dect/Analyse_Adaptabilite_Jeux_Serieux.ods.

Concernant la classe « Adaptabilité », nous constatons que de nombreux jeux ont un code source ouvert (6/10) ce qui est un élément positif pour favoriser leur évolution. En revanche, en l'état, peu de jeux (3/10) fournissent des interfaces spécifiques aux

enseignants (Profil enseignant). Les fonctionnalités proposées sont très différentes entre les jeux : Algoblocs permet à l'enseignant de publier des défis ou d'activer/désactiver des options de commentaires et de forum ; AlgoPython quant à lui permet aux enseignants de créer des classes et d'y associer des élèves ; enfin, le jeu Code permet à l'enseignant d'organiser ses projets, de structurer des cours décomposés en unités et en chapitres. Les deux derniers critères de cette catégorie sont très peu identifiés. Aucun des jeux étudiés n'offre la possibilité de modifier l'interface du jeu et seul KoduGame permet d'utiliser des périphériques d'interaction autre que le couple clavier/souris.

Concernant la classe « Édition », la plupart des jeux ne donnent aucun contrôle sur la scénarisation, ils permettent seulement au joueur de débloquer les niveaux au fur et à mesure de leur progression. Deux jeux sortent du lot : SPY et KoduGame. Le premier offre la possibilité d'ajouter/supprimer/modifier des niveaux du scénario à travers l'édition de fichiers XML. Le second permet aux enseignants de créer des mondes dans lesquels ils pourront définir des tâches, les organiser et préciser des consignes. Nous avons néanmoins noté la possibilité d'ajouter des tâches dans Code, les tâches (appelées unités dans le site) peuvent être assignées à un cours, mais leur contenu ne peut être ni modifié ni réorganisé.

Pour la classe « Capacité formative », 6 jeux sur 10 proposent des guides explicatifs et seulement 3 (Code, PyRates et Ceebot) sont accompagnés de guides pédagogiques qui donnent des informations sur les notions abordées à travers le jeu. Les deux derniers critères de cette classe sont dépendants de la possibilité de créer/modifier une tâche. Aucun des deux jeux identifiés dans la classe « Édition » ne propose d'outils permettant d'assister l'enseignant tant sur les dimensions pédagogiques que didactiques.

Dans la classe « Traces », trois jeux sortent du lot (AlgoPython, CodinGame et Algoblocs). Les actions des joueurs sont tracées afin de construire des tableaux de bord à destination des enseignants affichant des indicateurs de progression et de performance. En revanche, aucun des jeux ne rend accessibles les données tracées rendant impossibles des traitements/analyses personnels.

Pour la classe « Diversité de choix », un seul jeu propose au joueur de pouvoir manipuler différents langages de programmation : Codin'Game. Seuls des langages textuels sont proposés (pas de langages à base de blocs), mais plusieurs paradigmes de programmation sont possibles comme des approches orientées objet, fonctionnelles et impératives. L'enseignant est donc libre de choisir son langage parmi une large palette de 27 langages de programmation différents.

Enfin, la classe « Scénarisation » nous montre que la plupart des jeux (7/10) proposent une forme de scénarisation impliquant des liens entre les différents niveaux de jeu. Ce lien peut être indépendant du contenu des tâches comme dans PyRates où les niveaux sont structurés par une histoire incluant une thématique et des personnages, mais où chaque niveau peut aussi être joué indépendamment des autres niveaux.

4 Conclusion

Nos travaux s'inscrivent dans la problématique générale de l'appropriation par les enseignants et les formateurs des jeux sérieux d'apprentissage de la pensée informatique

et de la programmation. Plus précisément, nous cherchons à caractériser le degré d'adaptabilité de chaque jeu afin de favoriser la dimension instrumentalisation de la genèse instrumentale à travers une approche méta-design.

Dans ce contexte, cet article présente un travail préliminaire d'identification de critères d'adaptabilité de jeux sérieux et leur utilisation pour l'analyse d'une sélection de jeux destinés à ces apprentissages.

Nous présentons une grille d'analyse comportant 7 classes de 1 à 5 critères. Chaque classe décrit une composante d'adaptabilité pour d'éventuels utilisateurs aux profils divers, allant d'un enseignant de primaire qui est novice en informatique à un informaticien expérimenté. À l'exception de l'une des classes et de son critère associé spécifique de la pensée informatique et de la programmation (diversité du choix dans les langages de programmation), nous pensons que l'un des apports principaux de ce travail est de fournir un ensemble structuré de critères d'analyse de l'adaptabilité réutilisables pour n'importe quels types de jeux sérieux d'apprentissage.

Nous avons identifié 48 jeux destinés à l'apprentissage de la pensée informatique et de la programmation. Nous avons réalisé une sélection de 10 jeux sur lesquels nous avons utilisé notre grille d'analyse.

Grâce à cette analyse des 10 jeux sélectionnés avec nos critères, nous avons observé que même s'ils sont nombreux à proposer un code source ouvert (critère « Code source libre »), peu d'entre eux sont facilement adaptables (critères « Modification IHM », « Modalité d'interaction », « Modification de tâches », « Ajout de tâches », « Organisation de tâches », « Création de scénario »). Quand elles sont possibles (ex. SPY, PyRates, Kodu Game), les modifications nécessitent le plus souvent une bonne connaissance de la programmation (éditeurs inexistants, sauf pour Kodu Game) et il n'y a pas de ressources formatives à disposition pour être assisté dans ces modifications.

Au-delà des apports directs de ces travaux (un ensemble de critères d'analyse de l'adaptabilité réutilisables dans d'autres contextes et sa mise en œuvre sur 10 jeux), ces résultats nous permettent d'identifier que les exigences pour avoir un jeu d'apprentissage de la pensée informatique ou de la programmation permettant le méta-design sont élevées et qu'aucun de ceux expertisés ici ne les satisfait complètement. Nos futures recherches seront donc orientées vers l'identification de moyens pour rendre les jeux sérieux destinés à ces apprentissages plus propices à la mise place d'une approche méta-design dans l'optique de favoriser leur appropriation par les enseignants et les formateurs.

Remerciements. Les auteurs remercient l'Université Lumière Lyon 2 pour son soutient financier dans le cadre du programme APPI 2020.

Références

1. Baron G-L, Drot-Delange B (2016) L'informatique comme objet d'enseignement à l'école primaire française ? Mise en perspective historique. Revue française de pédagogie Recherches en éducation 51–62. https://doi.org/10.4000/rfp.5032
2. Wing JM (2006) Computational thinking. Commun ACM 49:33–35. https://doi.org/10.1145/1118178.1118215

3. Dowek G (2011) Les quatre concepts de l'informatique. Athènes : New Technologies Editions, p 21
4. Kradolfer S, Dubois S, Riedo F, Mondada F, Fassa F (2014) A Sociological Contribution to Understanding the Use of Robots in Schools: The Thymio Robot. In: Beetz M, Johnston B, Williams M-A (eds) Social Robotics. Springer International Publishing, Cham, pp 217–228
5. Bouvier P, Sehaba K, Elise L (2014) A trace-based approach to identifying users' engagement and qualifying their engaged-behaviours in interactive systems. Application to a social game. User Modeling and User-Adapted Interaction (UMUAI'14) 45:413–451
6. Bouvier P, Sehaba K, Lavoué É (2014) A trace-based approach to identifying users' engagement and qualifying their engaged-behaviours in interactive systems: application to a social game. User Model User-Adap Inter 24:413–451. https://doi.org/10.1007/s11257-014-9150-2
7. Miljanovic M, Bradbury J (2018) A Review of Serious Games for Programming
8. Vahldick A, Mendes AJ, Marcelino MJ (2014) A review of games designed to improve introductory computer programming competencies. In: 2014 IEEE Frontiers in Education Conference (FIE) Proceedings. IEEE, Madrid, Spain, pp 1–7
9. Fischer G, Giaccardi E, Ye Y, Sutcliffe AG, Mehandjiev N (2004) Meta-design: a manifesto for end-user development. Communications of the ACM 47:33–37
10. Rabardel P (1995) Les hommes et les technologies : une approche cognitive des instruments contemporains. Armand Colin, Paris, France
11. Folcher V, Rabardel P (2004) Hommes, artefacts, activités : perspective instrumentale. Presses Universitaires de France
12. Lindberg RS, Laine TH, Haaranen L (2019) Gamifying programming education in K-12: A review of programming curricula in seven countries and programming games. British Journal of Educational Technology 50:1979–1995
13. Malliarakis C, Satratzemi M, Xinogalos S (2014) Educational games for teaching computer programming. In: Research on e-Learning and ICT in Education. Springer, pp 87–98
14. Marne B, Sehaba K, Muratet M (2021) Vers une méthode de méta-design de jeux sérieux : Application pour l'apprentissage de la programmation à travers Blockly Maze. p 342
15. Nédélec X, Marne B, Muratet M, Sehaba K, Lapostolle J (2021) Une approche méta-design du jeu sérieux pour l'enseignement de l'informatique à l'école élémentaire. In: Broisin J, Declercq C, Fluckiger C, Parmentier Y, Peter Y, Secq Y (eds) Atelier " Apprendre la Pensée Informatique de la Maternelle à l'Université ", dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH). Fribourg, Switzerland, pp 96–106
16. Resnick M, Maloney J, Monroy-Hernández A, Rusk N, Eastmond E, Brennan K, Millner A, Rosenbaum E, Silver J, Silverman B, Kafai Y (2009) Scratch: Programming for All. Communications of the ACM 52:60–67. https://doi.org/10.1145/1592761.1592779
17. Fraser N (2015) Ten things we've learned from Blockly. In: 2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond). IEEE, pp 49–50
18. Papert S (1987) Microworlds: transforming education. In: Artificial intelligence and education. Ablex Publishing Corp.355 Chestnut St. Norwood, NJUnited States, pp 79–94