Comment caractériser et analyser les compétences de la pensée informatique d'un jeu sérieux ?

Mathieu Muratet

Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
INS HEA, 92150 Suresnes, France
mathieu.muratet@lip6.fr

Résumé. La pensée informatique est un ensemble de compétences mal maîtrisée par les enseignants de l'école obligatoire. Pour aider les enseignants à aborder ces compétences avec leurs élèves, nous développons le jeu SPY, un jeu sérieux dirigé par les compétences. Dans cet article nous conduisons une analyse du jeu SPY et nous proposons un formalisme pour décrire les compétences visées par le jeu à partir des fonctionnalités du jeu. Le formalisme proposé a été appliqué pour caractériser 21 des 26 compétences du PIAF et nous montrons comment ces caractérisations permettent de construire des indicateurs explicitant les compétences en jeu.

Mots-clés : Jeu sérieux, pensée informatique, modélisation, compétence, analyse automatique.

Abstract. Computational thinking is a discipline poorly mastered by elementary school teachers. To help teachers address these competencies with their students, we develop the SPY game, a serious game driven by competencies. In this paper we conduct an analysis of SPY and we propose a formalism to describe the competencies targeted by the game from the game mechanics. The proposed formalism has been applied to describe 21 of the 26 competencies of the PIAF conceptual framework and we show how these descriptions allow us to build indicators explaining the competencies on the line.

Keywords: Serious game, computational thinking, modeling, competency, automatic analysis.

1 Introduction et positionnement

Depuis maintenant quelques années, l'informatique est une discipline des programmes scolaires [1], où l'ensemble des compétences attendues est appelée « pensée informatique ». Selon 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. Sur la base de cette définition Parmentier *et al.*

[3] proposent un découpage fin des compétences de la pensée informatique et algorithmique pour l'enseignement fondamental (le PIAF). Ils proposent également des scénarios pédagogiques pour aider les enseignants à transmettre ces compétences à leurs élèves. Cette question de l'appropriation de ces compétences par les enseignants de l'école primaire est complexe à résoudre. En effet, enseigner cette nouvelle discipline demande un investissement particulièrement important de leur part [4] notamment en raison d'un manque de formation (initiale et continue). Ainsi, par manque de maîtrise de ces compétences les enseignants se trouvent en difficulté pour construire leurs propres scénarios pédagogiques ou juger de la pertinence d'outils pour faire travailler ces compétences à leurs élèves [5].

Nous faisons l'hypothèse que même si les enseignants de l'école primaire ne maîtrisent pas les compétences relatives à la pensée informatique, ils sont capables d'en comprendre les enjeux pour leurs élèves lorsqu'ils sont présentés dans des référentiels ; et qu'un outil basé sur ces référentiels serait lisible pour les enseignants.

Notre proposition est la suivante : Proposer un jeu sérieux d'apprentissage dirigé par les compétences qui assistera les enseignants dans le développement de sessions d'apprentissage de la pensée informatique avec leurs élèves.

Aujourd'hui, plusieurs dizaines de jeux sérieux d'apprentissage sur le thème de l'informatique existent [6, 7] mais très peu explicitent comment les compétences sous-jacentes sont travaillées et encore moins fournissent des outils à destination des enseignants pour les aider à adapter les jeux à leur contexte [8].

La problématique générale est la suivante : comment aider des enseignants non familiers à la pensée informatique à repérer dans des situations de jeu (qu'ils pourraient eux même créer) les compétences en jeu ?

Cette problématique est déclinée en deux questions de recherche :

- Comment formaliser les compétences d'un référentiel de compétence sur la pensée informatique à partir des fonctionnalités ludiques d'un jeu sérieux ?
- Comment exploiter ce formalisme pour analyser les niveaux du jeu sérieux et en extraire les compétences en jeu ?

Afin de répondre à ces questions nous présenterons le cadre théorique dans la section 2. Nous présenterons ensuite dans la section 3 le jeu SPY que nous utilisons dans le cadre de cette étude et nous exposerons ses fonctionnalités didactiques. Nous proposerons dans la section 4 un formalisme pour décrire des compétences de la pensée informatique à l'aide des fonctionnalités de jeu et nous illustrerons l'application de ce formalisme sur quelques compétences de la pensée informatique dans la section 5. Enfin, nous présenterons quelques résultats d'analyses réalisés sur le jeu SPY dans la section 6 avant de conclure.

2 Encrage théorique

Dès 1994, Balacheff [9] identifie le processus de « transposition informatique ». Il vise à identifier les liens entre l'univers interne du dispositif et l'univers externe (dans lequel se trouve l'utilisateur) au travers d'une interface. Cette interface joue un rôle particulier

lorsqu'elle devient « une référence pour l'utilisateur relativement à laquelle la connaissance est construite » [9]. Ainsi, les choix de conception d'une interface influencent le processus de construction de connaissances.

Concevoir un jeu sérieux d'apprentissage à métaphore intrinsèque consiste à placer l'apprentissage au cœur de la jouabilité [10]. Dans ce contexte les concepteurs de jeux sérieux proposent des mécanismes de *gameplay* en harmonie avec les contenus à enseigner. Il y a donc une relation forte entre les fonctionnalités de jeu et le savoir à enseigner, les fonctionnalités de jeu sont alors pour la plupart didactiques. La transposition informatique des apprentissages visés par le jeu sérieux n'est pour autant pas toujours explicite. Branthiome [11] développe l'idée de situations adidactiques dans lesquelles les élèves peuvent envisager des solutions initiales peu efficaces qui, avec les rétroactions du système, permettent de passer sur une procédure gagnante. Pour autant les fonctionnalités de jeu proposées sont didactiques pour permettre aux élèves de progresser dans leurs apprentissages même s'ils n'en ont pas conscience au cours du jeu.

Le joueur est donc en interaction avec le jeu sérieux et développe des schèmes d'utilisation qui transforment le jeu d'un statut d'artefact au statut d'instrument [12]. Les schèmes ainsi développés dépendent des propriétés de l'artefact interactif (dans notre cas des fonctionnalités de jeu disponibles dans une situation donnée). La théorie de la médiation sémiotique développe l'idée du potentiel sémiotique d'un artefact [13]. Ce potentiel est défini par un double lien « qui peut s'établir entre i) un artefact et les significations personnelles émergeant de son utilisation finalisée ; ii) cet artefact et les significations mathématiques évoquées par son usage, reconnaissables comme mathématiques par un expert » [13]. C'est ce second lien qui nous intéresse dans nos travaux où nous cherchons à caractériser les liens entre un jeu sérieux (l'artefact) et les signes issus de l'usage, reconnaissables comme relevant du savoir, dans notre cas la pensée informatique.

3 Analyse des fonctionnalités du jeu SPY

SPY[1] est jeu sérieux d'apprentissage sur le thème de la pensée informatique. Il a été conçu pour un public d'élèves de cycle 3 (CM1, CM2, 6ème). C'est un projet open source[2] développé par Sorbonne Université. Le principe du jeu est de programmer un agent à l'aide d'actions afin de l'aider à sortir d'un labyrinthe (se déplacer du point de départ au point d'arrivée du labyrinthe). Ces actions sont représentées sous forme de blocs que le joueur doit agencer en séquences exécutées par l'agent.

La mobilisation des compétences de la pensée informatique repose sur un ensemble de caractéristiques de jeu. Nous allons détailler ici l'analyse de cet artefact afin d'en identifier tous les ressorts ludiques et étudier les liens avec le savoir à enseigner : la pensée informatique.

[1] SPY : https://spy.lip6.fr, accédé le 09/01/2023
[2] Code source du jeu SPY : https://github.com/Mocahteam/SPY, accédé le 09/01/2023

3.1 Blocs de programmation

Les blocs de programmation sont les éléments fondamentaux de SPY dans la mesure où ils constituent les briques de base composant les programmes exécutés par les agents. Les blocs de programmation sont répartis en quatre catégories :

Les blocs d'action : Ces blocs permettent de définir les actions pouvant être réalisées par les agents à savoir : « Avancer », « Pivoter à gauche », « Pivoter à droite », « Attendre », « Activer un terminal » et « Faire demi-tour ». A noter que toutes ces actions sont atomiques à l'exception de l'action « Faire demi-tour » qui peut être décomposée par deux actions « Pivoter à droite » ou deux actions « Pivoter à gauche ».

Les blocs de contrôle : Ces blocs permettent de contrôler les blocs d'action à exécuter à savoir : « Si Alors », « Si Alors Sinon », « Répéter n fois », « Tant que » et « Répéter indéfiniment ».

Les capteurs : Ces blocs donnent des informations sur l'environnement avoisinant les agents. Les capteurs renvoient des valeurs booléennes qui peuvent être exploitées dans les blocs de contrôle « Si Alors », « Si Alors Sinon » et « Tant que ». Les capteurs permettent à l'agent de savoir si un mur se trouve en face de lui, à sa gauche ou à sa droite ; si un passage se trouve en face de lui, à sa gauche ou à sa droite ; si une sentinelle, une zone surveillée ou une porte se trouve en face de lui ; et si un terminal ou une sortie se trouve sur sa position.

Les opérateurs : Ces blocs permettent de combiner les capteurs. Nous retrouvons les classiques opérateurs booléens : « Non », « Ou » et « Et ». On notera que certains capteurs peuvent être exprimés par d'autres capteurs à l'aide des opérateurs, par exemple « Mur en face » est équivalent à « Non Passage en face ».

Limitation des blocs de programmation : SPY donne la possibilité de paramétrer pour chaque niveau quels blocs de programmation seront disponibles et en quelle quantité. Cette fonctionnalité peut être utile à la fois pour réduire la complexité d'un niveau ou au contraire l'augmenter. Par exemple, un niveau d'introduction pourra ne contenir que les blocs utiles à sa résolution, évitant ainsi au joueur de devoir choisir parmi des blocs potentiellement inutiles. Inversement limiter l'accès à certains blocs (ou leur quantité) peut être utile pour forcer le joueur à utiliser certains blocs (par exemple, demander de faire avancer un agent de plusieurs cases en limitant le nombre de bloc « Avancer » à 1 et en donnant accès au bloc de contrôle « Répéter n fois »).

3.2 Zone de programmation

La zone de programmation est la seconde fonctionnalité fondamentale de SPY. Elle accueille les blocs de programmation permettant de construire les solutions aux problèmes posés.

Par défaut, chaque agent est associé à une zone de programmation. Il est cependant possible de rompre cette association afin de demander au joueur de le faire. Dans ce cas, le joueur doit indiquer dans la zone de programmation le nom de l'agent auquel elle est associée afin que le programme lui soit envoyé lors du lancement de l'exécution.

Une zone de programmation peut être préconstruite. Le programme ainsi proposé au joueur peut être complet, partiel ou bogué. Le joueur devra en conséquence la compléter ou la corriger le cas échéant.

Enfin, SPY offre la possibilité de ne proposer au joueur qu'une seule zone de programmation pour plusieurs agents. Ceci permet de construire des niveaux où le joueur devra imaginer un unique programme permettant de contrôler plusieurs agents pouvant être dans des labyrinthes différents. Le joueur pourra ainsi se confronter à la réalisation de solutions plus génériques.

3.3 Glisser/déposer

Le glisser/déposer est la troisième fonctionnalité fondamentale dans le jeu SPY, car elle permet au joueur de glisser/déposer de nouveaux blocs dans la séquence d'action à exécuter par l'agent. Il est néanmoins possible de désactiver cette fonctionnalité, dans ce cas le niveau doit proposer un ou plusieurs programmes préconstruits que le joueur pourra exécuter sans pouvoir les modifier.

3.4 Obstacles

Pour résoudre les différents niveaux du jeu, le joueur doit programmer un ou plusieurs agents pour les aider à atteindre une position particulière tout en évitant des obstacles. Ces obstacles ont été conçus avec une intention didactique :

- Des sentinelles surveillent des zones du labyrinthe. Si un agent se trouve sur l'une de ces zones surveillées, l'agent est détecté ce qui met fin à la partie. Le joueur doit donc programmer l'agent pour qu'il évite ces zones surveillées. Par ailleurs, comme pour les agents, les sentinelles peuvent contenir un programme préconstruit ce qui rend les zones surveillées dynamiques. Le joueur peut donc sélectionner chaque sentinelle du jeu, observer le programme qui la compose, comprendre cette séquence d'action, anticiper les mouvements de la sentinelle et programmer son agent en conséquence.
- Des portes peuvent être ouvertes et fermées à l'aide de terminaux. Cette mécanique a été introduite pour engager un processus de résolution en étapes. Lorsqu'une porte bloque le passage, le joueur doit décomposer sa solution en sous-étapes (objectif 1 : activer le terminal pour ouvrir la porte ; et objectif 2 : atteindre la sortie). Les portes permettent également de manipuler des objets dont l'état peut changer (ouvert ou fermé).

3.5 Contrôle de l'exécution des programmes

Lorsque le joueur souhaite tester sa solution il clique sur un bouton pour lancer la simulation. Chaque zone de programmation envoie son programme à l'agent ciblé s'il existe et chaque agent/sentinelle exécute ses actions en parallèle. Le joueur peut suivre l'exécution des programmes en observant les agents/sentinelles bouger dans la scène

en fonction des actions en cours d'exécution (mises en évidence dans le panneau d'exécution des programmes). Le joueur peut mettre en pause la simulation à tout moment et exécuter les programmes pas à pas.

3.6 Nombre d'exécutions

Trouver la solution à un problème en un seul coup (trouver un programme qui permet de déplacer l'agent directement du point de départ au point d'arrivée) peut être une tâche complexe notamment sur des niveaux contenant des sentinelles en mouvement. SPY permet donc de résoudre un niveau en plusieurs coups. Le joueur peut définir une première séquence de blocs, l'exécuter, observer la nouvelle situation, ajouter de nouveaux blocs, exécuter, observer… Il est donc possible de construire le programme solution étape par étape.

Cependant, il peut être pertinent de limiter ce nombre d'exécutions pour amener progressivement le joueur à anticiper plusieurs actions en avance. Il est donc possible pour chaque niveau de SPY de définir le nombre autorisé d'exécutions pour le résoudre.

3.7 Brouillard et texte d'introduction

Dans un niveau classique de SPY le joueur a une vue omnisciente de la situation de jeu ce qui lui permet de planifier ses actions en vue d'atteindre l'objectif du niveau décrit dans les textes d'introduction. Le brouillard permet de modifier cette règle en cachant la vue d'un agent à son entourage proche. Dans ce cas, le texte d'introduction présentant l'objectif de la mission joue un rôle fondamental, car il devra contenir des indices permettant au joueur de trouver la solution. Par exemple, l'algorithme peut être donné dans le texte d'introduction en langage naturel et le joueur doit le traduire à l'aide du langage formel du jeu.

3.8 Synthèse des fonctionnalités de jeu de SPY

L'analyse des différentes fonctionnalités de SPY nous montre que chacune d'elles est didactique. Certaines sont évidentes comme les blocs de contrôle qui permettent de manipuler les notions de programmation associées, d'autres sont cachées comme l'association de programmes aux sentinelles qui demandera au joueur de comprendre une séquence d'action et d'anticiper son exécution, ou l'utilisation d'une seule zone de programmation pour contrôler deux agents qui forcera le joueur à généraliser sa solution.

D'un point de vue macro, les différentes fonctionnalités de SPY permettent de couvrir les compétences de la pensée informatique telles que définies par Wing [2]. Le joueur doit observer et modéliser la simulation (abstraction), décomposer sa stratégie en sous-étapes (décomposition), déterminer la meilleure solution (évaluation), planifier les actions à réaliser (pensée algorithmique) et réutiliser et adapter des solutions précédentes sur de nouveaux problèmes (généralisation). Mais comment identifier si telle ou telle compétence est en jeu dans un niveau donné ? Quelles influences les combinaisons de fonctionnalités ont-elles sur les compétences en jeu ?

4 Lier les compétences aux fonctionnalités de jeu

Pour décrire une compétence nous définissons des contraintes sur les fonctionnalités de jeu que nous évaluons sous la forme d'une expression booléenne. Si cette règle est vraie car l'ensemble des contraintes est satisfait, alors nous considérons que la compétence est en jeu dans le niveau analysé.

Les règles et contraintes définies pour chaque compétence s'appuient sur la structure des niveaux décrite au format XML. Le Table 1 présente les balises que nous référençons dans cet article. Un modèle de niveau complet et commenté est donné en [14].

Table 1 : Description des principales balises structurant un niveau SPY

<dragdropDisabled>	Si présente, désactive la fonctionnalité de glisser/déposer
<blockLimit blockType="X" limit="Y">	Si présente, définit la quantité Y de blocs de type X disponibles dans l'inventaire (si Y = -1 bloc en quantité illimitée).
<player inputLine="X">	Définit un agent contrôlé par le joueur. Cet agent écoute le canal de communication X.
<enemy inputLine="X">	Définit une sentinelle. Cette sentinelle écoute le canal de communication X.
<script outputLine="X" editMode="Y" type="Z">	Définit une zone de programmation qui enverra son contenu sur le canal de communication X (voir balise <player> et <enemy>). La propriété **editMode** indique si le joueur peut changer ou non le canal de communication. La propriété **type** indique si cette zone de programmation contient un programme préconstruit optimal, non optimal, bogué ou indéfini.

Une contrainte est un filtre de balises XML qui exclue de l'ensemble des balises définissant un niveau toutes les balises ne respectant pas cette contrainte. Chaque contrainte peut être précisée à l'aide d'un ensemble de paramètres. Le Table 2 décrit l'ensemble des paramètres qui peuvent être appliqués à une contrainte.

Table 2 : Description des contraintes possibles

$$TA^{G} \in ensemble\ de^{S}\ balise^{S}\ XML^{L}\ de\ SP^{Y}$$
$$ATTR^{R} \in ensemble\ de^{S}\ attribut^{S}\ valide^{S}\ pour\ TA^{G}$$
$$OP \in \{=, <, \leq, >, \geq, \neq\}$$
$$VA^{L} \in \mathbb{N}$$

TAG	Filtre les balises TAG
TAG ATTR OP VAL	Filtre les balises TAG qui ont un attribut ATTR égal/différent… à une valeur entière
TAG ATTR include SET	Filtre les balises TAG qui ont un attribut ATTR dont la valeur est incluse dans un ensemble de valeurs (SET)
TAG ATTR sameValue TAG_2 $ATTR_2$	Filtre les balises TAG qui ont un attribut ATTR dont la valeur est égale à la valeur de l'attribut $ATTR_2$ d'une balise TAG_2
TAG hasChild	Filtre les balises TAG qui contiennent au moins une balise fille

L'identification d'une compétence repose alors sur l'évaluation d'un ensemble de contraintes. Nous donnons dans la section suivante quelques exemples de contraintes que nous avons définies pour caractériser quelques compétences du PIAF.

5 Application du formalisme aux compétences du PIAF

Comme nous l'avons présenté en introduction, nous choisissons de baser notre travail sur le référentiel PIAF. Il est focalisé sur la pensée informatique et propose six compétences principales. Chacune de ces compétences est découpée en un ensemble de sous-compétences pour un total de 26 sous-compétences [15].

Les six compétences principales sont : C1 - Définir des abstractions / généraliser ; C2 – Composer / décomposer une séquence d'actions ; C3 - Contrôler une séquence d'actions ; C4 - Évaluer des objets ou des séquences d'actions ; C5 - Manipuler des représentations formelles ; et C6 - Construire une séquence d'actions de manière itérative. Dans la suite de l'article nous faisons référence aux compétences du référentiel PIAF de la manière suivante : CX.Y signifie que la sous-compétence Y de la compétence X est en jeu, par exemple C2.4 désigne la quatrième sous compétence de C2.

Nous ne développons pas ici de manière exhaustive la caractérisation de toutes les compétences du PIAF. L'ensemble des règles et contraintes définies pour caractériser les compétences du PIAF en fonction des fonctionnalités de jeu est accessible en [16]. Néanmoins, nous présentons quelques exemples d'application de notre formalisme sur une sélection de trois compétences du PIAF.

5.1 C1.1 : Nommer des objets et séquence d'actions

La compétence C1.1 est définie ainsi : « Être capable de donner des noms à des objets, des actions et des séquences d'actions » [15, p. 2].

Dans SPY nous considérons que cette compétence est en jeu lorsque le joueur associe une zone de programmation à un agent. En effet, dans ce cas il doit nommer son programme afin qu'il soit interprété par le bon agent.

Nous caractérisons cette compétence à l'aide de la contrainte « **TAG ATTR OP VAL** » qui est instanciée ainsi : « **script editMode = 2** ». Cette contrainte signifie : « filtre les balises "script" qui ont un attribut "editMode" "égal" à la valeur 2 ».

L'expression booléenne qui définit la règle complète est alors « C1.1 = Card(script editMode = 2) > 0 » qui signifie : « La compétence C1.1 est en jeu dans le niveau si le cardinal de la contrainte est supérieur à 0 », autrement dit « si le niveau contient au moins une zone de programmation nommable ».

5.2 C1.5 : Prédire le résultat d'une séquence d'actions

La compétence C1.5 est définie ainsi : « Être capable de dire, à partir d'une séquence d'actions, ce qui se passera si elle est exécutée. Contrairement à la compétence 1.4, cette compétence consiste à fournir une prédiction sans exécuter réellement la séquence d'actions » [15, p. 5].

Dans SPY, cette compétence est travaillée lorsqu'un niveau contient une sentinelle préprogrammée. Dans ce cas le joueur doit anticiper les déplacements de la sentinelle sans pouvoir exécuter son programme. Il devra proposer une première solution pour voir les mouvements de la sentinelle et vérifier ses hypothèses.

Pour décrire cette compétence, il est nécessaire de vérifier dans un niveau qu'une zone de programmation est associée à une sentinelle et qu'elle contient au moins une action.

La règle est décrite à l'aide d'un double paramètre : (P1) « **TAG ATTR** sameValue **TAG_2 $ATTR_2$** » d'une part et (P2) « **TAG** hasChild » d'autre part.

Le premier paramètre permet de s'assurer qu'une sentinelle (balise « enemy ») écoute le même canal de communication (attribut « inputLine ») que celui d'une zone de programmation (balise « script » et attribut « outputLine »). P1 est donc instancié ainsi : « **enemy inputLine** sameValue **script outputLine** ».

Le second paramètre permet de s'assurer que la zone de programmation (balise « script ») contient bien au moins une action. P2 est donc instancié ainsi : « **script** hasChild ».

L'expression booléenne qui définit la règle complète est alors l'intersection des deux paramètres : « C1.5 = Card(P1 ⋂ P2) > 0 ».

5.3 C2.1 : Ordonner une séquence d'actions pour atteindre un objectif

La compétence C2.1 est définie ainsi : « Étant donné une liste non ordonnée d'actions et un but, être capable de combiner ces actions dans un ordre valide pour construire une séquence qui permet d'atteindre ce but. […] Ainsi, l'apprenant n'a pas besoin d'identifier toutes les parties nécessaires pour atteindre l'objectif, mais seulement de les mettre dans le bon ordre » [15, p. 7].

Dans SPY, il s'agit d'une situation où il est proposé au joueur uniquement les blocs utiles à la résolution du problème. Le joueur doit alors simplement les combiner dans le bon ordre pour résoudre le niveau.

Pour décrire cette compétence, il est nécessaire dans un niveau de respecter les contraintes suivantes : (R1) le glisser/déposer est activé pour permettre au joueur de combiner les actions ; (R2) aucun bloc n'est disponible en quantité illimitée ; et (R3) il y a au moins une action disponible dans l'inventaire.

La règle R1 est décrite à l'aide de la contrainte « **TAG** » qui est instanciée avec la balise « **dragdropDisabled** ». Nous souhaitons vérifier que le glisser/déposer est actif et donc que la balise « dragdropDisabled » est absente dans la description du niveau. R1 est donc définie ainsi « Card(dragdropDisabled) = 0 ».

La règle R2 est décrite à l'aide de la contrainte « **TAG ATTR OP VAL** » qui est instanciée ainsi : « **blockLimit limit = -1** ». Cette contrainte signifie : « filtre les balises "blockLimit" qui ont un attribut "limit" "égal" à la valeur -1 ». Nous souhaitons vérifier qu'il n'y a pas de bloc en quantité illimitée, R2 est donc définie ainsi « Card(blockLimit limit = -1) = 0 ».

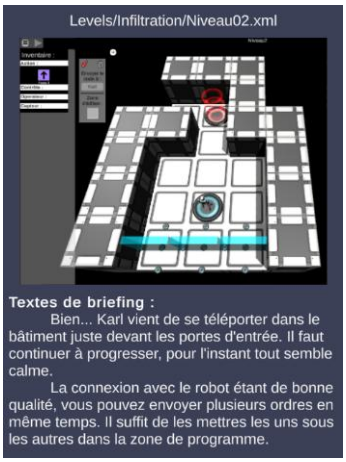
La règle R3 est décrite à l'aide d'un double paramètre : (P1) « **TAG ATTR OP VAL** » d'une part et (P2) « **TAG ATTR** include **SET** » d'autre part. Le premier permet d'identifier si un bloc est disponible dans l'inventaire (« **blockLimit limit > 0** ») et le second si ce bloc est un bloc d'action (« **blockLimit blockType** include **{Forward,TurnLeft,TurnRight,Wait,Activate,TurnBack}** »). Le cardinal de l'intersection de ces deux ensembles compte le nombre de blocs, de type action, disponibles dans l'inventaire. R3 est donc définie ainsi : « Card(P1 ⋂ P2) >=1 ».

L'expression booléenne qui définit la règle complète est alors : « C2.1 = R1 && R2 && R3 ».

6 Résultats

Nous avons illustré de manière détaillée la caractérisation de trois compétences du PIAF à l'aide du formalisme proposé. Sur l'ensemble des 26 compétences du PIAF, 21 sont applicables à SPY et ont été décrites. Outre les compétences du PIAF, nous avons confronté notre formalisme à un autre référentiel, nous avons ainsi décrit les 5 niveaux de la compétence « 3.4 Programmer » du domaine 3 « Création de contenus » du CRCN [17, p. 13]. Enfin nous avons exploité le formalisme pour exposer l'ensemble des fonctionnalités ludiques de SPY (dit référentiel SPY). L'ensemble de ces caractérisations sont accessibles en [16].

Nous pouvons ainsi analyser chaque niveau au regard des différents référentiels. Fig. 1 illustre l'analyse d'un niveau avec les 3 référentiels. Cette visualisation permet à l'enseignant d'obtenir des informations en fonction du référentiel de son choix en vue de déterminer si le niveau est un candidat potentiel pour intégrer sa scénarisation pédagogique.



Référentiel PIAF : C2.1 – Ordonner une séquence d'actions pour atteindre un objectif C2.3 – Créer une séquence d'actions pour atteindre un objectif simple C2.6 – Décomposer des objectifs en sous-objectifs plus simples C6.1 – Vérifier si une séquence d'actions atteint un objectif donné
Référentiel CRCN : Niveau 1
Référentiel SPY : F1 – Résoudre un problème en plusieurs étapes F10 – Blocs limités F11 – Bloc action « Avancer »

Fig. 1. Analyse d'un même niveau avec les différents référentiels

Nous pouvons aussi exploiter ces caractérisations pour analyser l'évolution de la complexité d'un scénario de jeu. SPY contient deux scénarios : un original à SPY intitulé « Infiltration » composé de 20 niveaux et une réplique du jeu BlocklyMaze[3] composé de 10 niveaux. Les enseignants peuvent également construire leurs propres scénarios en les composants à partir de la banque de niveaux existante. Nous avons utilisé le référentiel du PIAF pour caractériser la dimension didactique des différents scénarios

[3] BlocklyMaze : https://blockly.games/maze, accédé le 13/01/2023

et le référentiel SPY pour caractériser leur dimension ludique. Le cumul des compétences et fonctionnalités ludiques dans chaque niveau donne un indicateur sur sa complexité (voir Fig. 2). Nous observons ainsi l'évolution progressive de la complexité des deux scénarios et l'apparition des différentes compétences au cours des deux scénarios. Cette visualisation renvoie aux travaux de Carron *et al.* [18] qui analysent manuellement chaque niveau de jeu sur les dimensions ludiques et pédagogiques. Dans notre cas, l'analyse des niveaux est automatisée.

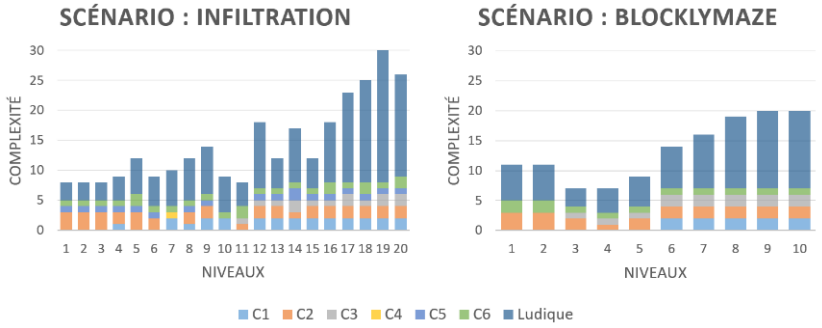


Fig. 2. Analyse des deux scenarios du jeu SPY

7 Conclusion et perspectives

Dans cet article nous avons réalisé une analyse du jeu SPY en explorant les fonctionnalités principales du jeu par rapport à des référentiels de compétences sur la pensée informatique. Nous avons proposé un formalisme pour décrire des compétences par combinaison de fonctionnalité de jeu. Nous avons ensuite exploité ce formalisme pour proposer une caractérisation des 21 compétences PIAF en jeu dans SPY, des 5 niveaux de la compétence 3.4 du CRCN et des 35 fonctionnalités ludiques de SPY. L'ensemble des résultats sont présentés en [16]. Nous avons ainsi montré que le formalisme proposé est indépendant d'un référentiel donné. Nous avons illustré dans cet article un extrait de ces résultats à l'aide des compétences C1.1, C1.5 et C2.1 du PIAF.

Enfin, nous avons montré comment ces caractérisations peuvent être exploitées pour construire des indicateurs sur des niveaux existants et fournir aux enseignants des informations *micros* sur les compétences en jeu dans un niveau ou des informations *macros* sur l'évolution de la complexité d'un scénario de jeu.

Actuellement le jeu SPY ne contient pas d'éditeur de niveaux. La création ou la modification de niveaux reste peu accessible (éditer à la main les fichiers XML). Une perspective pour ce travail est de proposer un tel éditeur, afin de permettre aux enseignants de créer leurs propres niveaux de jeu et d'en obtenir une analyse automatique à partir du référentiel de leur choix.

Une autre piste de recherche que nous avons identifiée consisterait à exploiter la topologie des labyrinthes comme une information susceptible d'être didactique et donc d'être accessible dans le formalisme. Ainsi, en inversant la chaîne, nous pourrions envisager d'automatiquement créer de nouveaux niveaux à partir d'une sélection de compétences visées.

References

1. Baron, G-L., Drot-Delange, B.: L'informatique comme objet d'enseignement à l'école primaire française ? Mise en perspective historique. Revue française de pédagogie Recherches en éducation, 51–62 (2016).
2. Wing, J. M.: Computational thinking. Communications of the ACM, 49(3), 33–35 (2006).
3. Parmentier, Y., Reuter, R., Higuet, S., Kataja, L., Kreis, Y., Duflot-Kremer, M., Laduron, C., Meyers, C., Busana, G., Weinberger, A., Denis, B.: PIAF: developing computational and algorithmic thinking in fundamental education. In: EdMedia+ Innovate Learning, Association for the Advancement of Computing in Education (AACE), pp. 315–322 (2020).
4. Kradolfer, S., Dubois, S., Riedo, F., Mondada, F., Fassa, F.: A sociological contribution to understanding the use of robots in schools: the thymio robot. In: International Conference on Social Robotics, Springer, Cham, pp. 217–228 (2014).
5. Poirson, B.: Appuis et obstacles à l'apprentissage de la pensée informatique à l'école primaire. Education (2020).
6. Lindberg, R. S., Laine, T. H., Haaranen, L.: Gamifying programming education in K-12: A review of programming curricula in seven countries and programming games. British Journal of Educational Technology 50(4), 1979–1995 (2019).
7. Miljanovic, M., Bradbury, J.: A Review of Serious Games for Programming. In: Proc. of the 4th Joint Conference on Serious Games (JCSG 2018), Darmstadt, Germany (2018).
8. Saddoug, H., Rahimian, A., Marne, B., Muratet, M., Sehaba, K., Jolivet, S.: Review of the Adaptability of a Set of Learning Games Meant for Teaching Computational Thinking or Programming in France. In: Proc. of the 14th International Conference on Computer Supported Education, vol. 1: GonCPL, pp 562–569 (2022).
9. Balacheff, N.: La transposition informatique, un nouveau problème pour la didactique. In: Artigue M. et al. (eds) Vingt ans de didactique des mathématiques en France, pp. 364–370 (1994).
10. Marne, B., Huynh-Kim-Bang, B., Labat, J.-L.: Articuler motivation et apprentissage grâce aux facettes du jeu sérieux. In: EIAH 2011, Mons, Belgique, pp. 69–80 (2011).
11. Branthôme, M.: Conception et évaluation du jeu sérieux Pyrates : aspects didactiques dans la construction des niveaux. In: RJC EIAH, Lille, France (2022).
12. Rabardel, P.: Les hommes et les technologies, approche cognitive des instruments contemporains. Paris : Armand Colin (1995).
13. Mariotti, M. A., Maracci, M.: Les artefacts comme outils de médiation sémiotique : quel cadre pour les ressources de l'enseignant ? Ghislaine Gueudet et Luc Trouche. Ressources vives. Le travail documentaire des professeurs en mathématiques., Presses Universitaires de Rennes et INRP, pp. 91–107, Paideia (2010).
14. Modèle de niveau complet et commenté au format XML, https://github.com/Mocahteam/SPY/blob/master/Doc/LevelModel.xml, accédé le 10/01/2023
15. La liste complète des 26 compétences du référentiel PIAF et leur description : https://piaf.loria.fr/wp-content/uploads/2021/09/Examples-PIAF-FR-aout-2021.pdf, accédé le 07/01/2023
16. Description de compétences à l'aide des fonctionnalités du jeu SPY, https://github.com/Mocahteam/SPY/blob/master/Assets/StreamingAssets/Competencies/competenciesReferential.json, accédé le 10/01/2023
17. CRCN, https://eduscol.education.fr/document/20389/download, accédé le 12/01/2023.
18. Carron, T., Muratet, M., Marne, B., Yessad, A.: Analyser et représenter la progression de la difficulté d'un jeu sérieux du point de vue ludique et pédagogique. In: EIAH 2017, Strasbourg, France (2017).