GTC un formalisme permettant l'analyse automatique des compétences de la pensée informatique dans un jeu sérieux : application au jeu SPY

GTC : A Formalism for the Automatic Analysis of Computational Thinking Skills in a Serious Game : Application to the SPY Game

Mathieu MURATET[1,2]; Sébastien JOLIVET[3]

[1] *Sorbonne Université, CNRS, LIP6, F-75005 Paris, France*
[2] *INS HEA, 92150 Suresnes, France*
[3] *IUFE & TECFA, Université de Genève, Suisse*

Résumé. La pensée informatique est un ensemble de compétences souvent mal maîtrisé par les enseignants de l'école primaire. Pour les aider à aborder ces compétences avec leurs élèves, nous développons le jeu SPY. Dans cet article, nous menons une analyse du jeu SPY et proposons le formalisme GTC (Game To Competency) pour décrire les compétences travaillées, à partir des fonctionnalités du jeu. GTC a été appliqué pour caractériser 21 des 26 compétences d'un référentiel (PIAF). Nous montrons comment ces caractérisations peuvent être exploitées pour construire des indicateurs sur les compétences mobilisées dans chaque mission. Enfin, nous présentons une expérimentation visant à évaluer l'utilité perçue des propositions faites. Elle a mobilisé cinq enseignants d'informatique, en formation à l'Université de Genève, dans le cadre d'un atelier d'échanges dirigés. Les résultats obtenus, très positifs, fournissent un premier niveau de validation des propositions faites et dégagent de nouvelles pistes de recherches.
Mots-clés : Jeu sérieux, pensée informatique, modélisation, compétence, analyse automatique, utilité

Abstract. *Computational thinking is a skillset inadequately mastered by elementary school teachers. To aid teachers in addressing these competencies with their students, we have developed the SPY game. In this paper, we conduct an analysis of SPY and we propose a formalism to describe the competencies targeted by the game based on its mechanics. The proposed formalism has been applied to describe 21 out of the 26 competencies within PIAF conceptual framework. We show how these descriptions allow for the construction of indicators elucidating the competencies involved. Finally, we present an experiment designed to evaluate the usefulness of the proposals made. Five teachers undergoing training at the University of Geneva participated in the experiment within the framework of a supervised workshop. The highly positive outcomes attained offer an initial validation of the proposed methods and suggest new research directions.*
Keywords: *Serious game, computational thinking, modeling, competency, automatic analysis, usefulness*

1. Introduction et positionnement

Depuis maintenant quelques années, l'informatique est devenue une discipline intégrée aux programmes scolaires (Baron *et al*., 2014), englobant un ensemble de compétences désigné sous le terme de « pensée informatique ». Selon Wing (2006), la pensée informatique mobilise cinq capacités cognitives : (1) la pensée algorithmique, (2) l'abstraction, (3) l'évaluation, (4) la décomposition, et (5) la généralisation. Sur cette base, Parmentier *et al*. (2020) proposent un découpage détaillé des compétences de la pensée informatique et algorithmique pour l'enseignement fondamental (le référentiel PIAF). La question de l'appropriation de ces compétences par les enseignants de l'école primaire reste complexe à résoudre. En effet, enseigner cette nouvelle discipline requiert un investissement particulièrement important de leur part (Kradolfer *et al*., 2014), notamment en raison d'un manque de formation (initiale et continue). Ainsi, par manque de maîtrise de ces compétences, les enseignants éprouvent des difficultés pour élaborer leurs propres scénarios pédagogiques ou évaluer la pertinence des outils visant à développer ces compétences chez leurs élèves (Poirson, 2020).

Sigayret *et al*. (2021) identifient trois principaux dispositifs pour initier les élèves à la pensée informatique : les activités débranchées n'utilisant aucun outil numérique, l'approche tangible de la robotique éducative et l'utilisation de logiciels visuels de programmation.

L'informatique débranchée offre une porte d'entrée riche dans la science informatique, n'exigeant pas de matériels coûteux (Alayrangues *et al*., 2017). Cependant, elle demande à l'enseignant une expertise approfondie sur le savoir abordé afin d'être en mesure de guider les élèves dans la construction de leurs connaissances. Ces mêmes auteurs (Alayrangues *et al*., 2017) soulignent que la mise en place de telles activités nécessite des connaissances en informatique ainsi que des compétences en pédagogie. Ce « super prof », maîtrisant ce savoir informatique, reste donc une exception ce qui rend difficile la démocratisation de ce type de scénario pédagogique à l'école primaire.

En ce qui concerne l'approche tangible de la robotique pédagogique, les objets tangibles offrent un artefact fertile sur le plan cognitif pour développer des compétences mathématiques et des formes de pensée algorithmique (Komis et Misirli, 2011). Ces robots programmables sont utilisés dans des scénarios pédagogiques, exigeant de l'enseignant une maîtrise fine des savoirs en jeu pour soutenir les travaux des élèves. Spach (2019) souligne à ce propos les freins au développement de cette approche tangible : « *Le défaut de maîtrise conceptuelle des enseignants est sans doute à l'origine du manque de référencement et du déficit d'institutionnalisation des notions et des concepts abordés dans les situations pédagogiques* ».

Dans ces deux premières approches, l'expertise de l'enseignant sur les savoirs informatiques est donc fortement sollicitée mais fait souvent défaut comme nous avons pu l'évoquer précédemment (Alayrangues *et al*., 2017 ; Spach, 2019). La troisième approche, qui exploite l'utilisation de logiciels visuels de programmation, peut être un élément de réponse à cette problématique car ces environnements peuvent intégrer des systèmes intelligents pour accompagner les élèves et les enseignants. Pour contribuer à répondre à cette problématique, nous proposons un jeu sérieux nommé SPY qui permet aux élèves de travailler les compétences de la pensée informatique et fournit aux enseignants des moyens de comprendre ce qui est travaillé par leurs élèves. Il existe plusieurs dizaines de jeux sérieux d'apprentissage sur le thème de l'informatique (Lindberg *et al*., 2019 ; Miljanovic et Bradbury, 2018). Les activités qu'ils proposent sont généralement structurés autour des notions, mais les compétences mises en oeuvre au sein d'une activité sont rarement identifiées. Par ailleurs, ces jeux fournissent peu d'outils à destination des enseignants pour les aider à adapter les jeux à leur contexte (Saddoug *et al*., 2022). Ce sont ces limitations qui ont motivé la conception du jeu sérieux SPY.

La problématique générale est la suivante : comment aider les enseignants peu familiers avec la pensée informatique à repérer dans des situations de jeu (qu'ils pourraient eux-mêmes créer) les compétences mobilisées ? Dans cet article nous présentons une contribution générique afin de lier des compétences aux fonctionnalités ludiques d'un jeu sérieux. Nous mettons en oeuvre cette contribution dans le jeu sérieux SPY.

Nous fixons dans un premier temps le cadre théorique ainsi que les questions de recherche dans la section 2. Dans la section 3, nous présentons le jeu SPY utilisé dans le cadre de cette étude et détaillons ses variables didactiques. La section 4 est consacrée à la présentation du formalisme GTC (Game To Competency) permettant de décrire les compétences de la pensée informatique à partir des fonctionnalités de jeu, et illustre l'application de ce formalisme pour décrire quelques compétences de la pensée informatique telles que mises en oeuvre dans SPY. Dans la section 5, nous montrons les outils qui ont pu être construits sur la base du formalisme GTC pour offrir à l'enseignant de nouveaux outils centrés sur les compétences de la pensée informatique. Enfin, dans la section 6, nous présentons une expérimentation visant à évaluer l'utilité perçue des outils proposées, dont les résultats sont exposés dans la section 7.

2. Ancrage théorique et questions de recherche

SPY est un jeu conçu pour être utilisé en classe sur la prescription d'un enseignant. Sa pertinence comme élément de réponse à notre problématique (outiller les enseignants, y compris non-experts du domaine, pour enseigner la pensée informatique) nécessite de l'examiner sous différents angles. Pour aborder cet examen nous différencions trois points de vues : celui des *concepteurs* du jeu et ceux des deux catégories d'utilisateurs finaux que sont les *enseignants* et les *apprenants*. Différentes approches théoriques permettent de décrire ou d'éclairer les choix et les effets selon ces trois points de vue. Dans cet article, en l'absence d'expérimentations en conditions écologiques, nous nous concentrons sur les deux premiers points de vue à savoir les *concepteurs* et les *enseignants*.

Tout d'abord, s'agissant de la conception d'un EIAH, en se focalisant sur la question du savoir dans l'environnement, Balacheff introduit en 1994 le concept de *transposition informatique* de la manière suivante : « *aux contraintes de la transposition didactique s'ajoutent, ou plutôt se combinent, celles de modélisation et d'implémentation informatiques : contraintes de la modélisation computable, contraintes logicielles et matérielles des supports informatiques de réalisation. Ce que l'on place habituellement sous le terme d'informatisation ne constitue pas une simple translittération, les environnements informatiques d'apprentissage résultent d'une construction qui est le lieu de transformations nouvelles des objets d'enseignement* » (Balacheff, 1994, p. 364).

De plus, nous travaillons sur un type d'EIAH particulier incluant une forte dimension ludique : un jeu sérieux. Dans notre situation les apprentissages sont placés au cœur de la jouabilité. Marne *et al.* (2012) parlent alors de jeu sérieux d'apprentissage à métaphore intrinsèque ce qui implique que les mécanismes de gameplay soient en cohérence avec les contenus à travailler. Ainsi, il existe une relation forte entre les fonctionnalités de jeu et le savoir à enseigner, elles ont donc, pour la plupart, une fonction didactique.

Dans la théorie de la médiation sémiotique, Mariotti et Maracci (2012) développent l'idée du potentiel sémiotique d'un artefact qui 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* »Ċ'est précisément ce second lien qui est inféré lorsque

l'on fait l'hypothèse que l'usage d'une fonctionnalité du jeu (considérée comme artefact) peut être interprété comme relevant de la mobilisation d'un savoir de la pensée informatique.

Ainsi, lors de la conception d'un tel EIAH, relativement au savoir travaillé, se combinent des enjeux et processus de transpositions didactique, de transposition informatique et une forme de transposition ludique. Bonnat *et al*. (2023) proposent une étude de ce processus qu'ils nomment *ludicisation*.

Par ailleurs, les concepteurs sont amenés à réaliser différents choix, guidés ou contraints par : les objectifs d'apprentissage fixés à l'EIAH, leurs représentations sur l'apprentissage, les possibilités et limites technologiques, etc.

Bruillard (1997) note justement la complexité de conception d'EIAH et souligne la « *difficulté réelle d'incorporer les théories de l'apprentissage et de l'enseignement dans les systèmes informatiques à finalité éducative* ». Finalement, dans la très grande majorité des cas, même pour leur part identifiée et consciente, les choix et contraintes qui dirigent la conception d'un EIAH, ainsi que leurs conséquences sur l'apprentissage lors de leur utilisation, ne sont pas accessibles aux utilisateurs finaux, tels que les enseignants et/ou les apprenants dans notre cas.

Du point de vue de l'utilisateur final enseignant, la connaissance des choix et contraintes qui ont orienté la conception de l'EIAH n'est pas fondamentale. Par contre, il est pertinent qu'il puisse connaitre les effets, de ces choix et des éléments qu'il peut manipuler, sur l'apprentissage.

L'un des enjeux est donc de décrire les effets et potentialités didactiques de l'EIAH et de rendre accessibles ces informations pour l'enseignant. Pour ce faire nous considérons tout d'abord que le jeu s'intègre dans une situation didactique (Brousseau, 1990) et que l'enseignant, en utilisant les potentialités du jeu, va pouvoir définir une situation, se voulant *a-didactique* (Brousseau, 1986), pouvant être traitée sans intervention de l'enseignant par l'élève et nécessitant la mobilisation des savoirs visés pour atteindre l'objectif du jeu. Il est donc nécessaire d'identifier dans le jeu les fonctionnalités pouvant être considérées comme des variables didactiques, c'est-à-dire : « *un élément de la situation sur lequel le maître peut agir ; qui provoque des changements qualitatifs dans les procédures de résolution des élèves ; qui permet d'expliquer les résultats de l'enseignement et d'agir sur eux ; et qui provoque une modification dans l'apprentissage* » (Margolinas, 1992, p. 129).

Ces différents éléments vont être exploités par l'enseignant lors du processus de genèse instrumentale (Rabardel, 1995) qui va faire passer l'EIAH du statut d'artefact au statut d'instrument pour son enseignement (Rabardel, 1999). Pour cela, l'enseignant va, en fonction des apprentissages visés, activer tout ou partie des fonctionnalités de l'artefact. Puis, l'élève-joueur va à son tour engager une genèse instrumentale en interagissant avec l'instance du jeu sérieux définit par l'enseignant, et développer des schèmes d'utilisation l'amenant à construire son propre instrument. Les schèmes pouvant être développés par l'élève dépendent des fonctionnalités, prédéfinies ou activées par l'enseignant dans l'artefact et de leur combinaison. Par exemple, dans le contexte d'un jeu de type « déplacement d'un robot dans un labyrinthe », limiter le nombre d'actions de déplacements et mettre à disposition une structure de contrôle permettant de réaliser des boucles bornées, vont permettre aux élèves de construire des schèmes liés à l'identification d'un schéma à répéter et du nombre d'itérations requis, ainsi qu'à l'agencement des différents composants du programme.

La capacité des enseignants à instrumentaliser l'EIAH pour qu'il instrumente au mieux leur enseignement dépend de divers facteurs parmi lesquels sa formation et sa maîtrise du savoir à enseigner, sur lesquelles nous ne revenons pas dans cet article. Mais aussi de la compréhension qu'ils peuvent avoir des effets, en termes de potentialités d'apprentissages, des choix de développements et des mécaniques de jeu implémentées. Dans cet article, nous

souhaitons donc expliciter et exemplifier, pour l'utilisateur final *enseignant*, ces effets et potentialités dans SPY. Ceci se concrétise par deux questions de recherche :

1. Comment lier les compétences d'un référentiel avec les fonctionnalités ludiques d'un jeu sérieux ?
2. Comment exploiter ce lien pour analyser automatiquement les missions du jeu sérieux SPY et en extraire les compétences de la pensée informatique ?

3. Analyse du jeu SPY

SPY [1] (voir Figure 1) est un jeu sérieux d'apprentissage sur le thème de la pensée informatique. Il est conçu pour un public d'élèves de cycle 3 (9, 10, et 11 ans). C'est un projet *open source* [2] et *open data* [3] développé au sein de Sorbonne Université. Le principe du jeu est de programmer un ou plusieurs robots à l'aide d'actions afin de les faire sortir d'un labyrinthe, c'est-à-dire de les déplacer du point de départ au point d'arrivée du labyrinthe. Ces actions, disponibles dans un inventaire, sont représentées sous forme de blocs que le joueur doit agencer en séquences dans la/les zone(s) d'édition. Il s'agit donc, *in fine*, de réaliser un programme dans un langage par blocs.

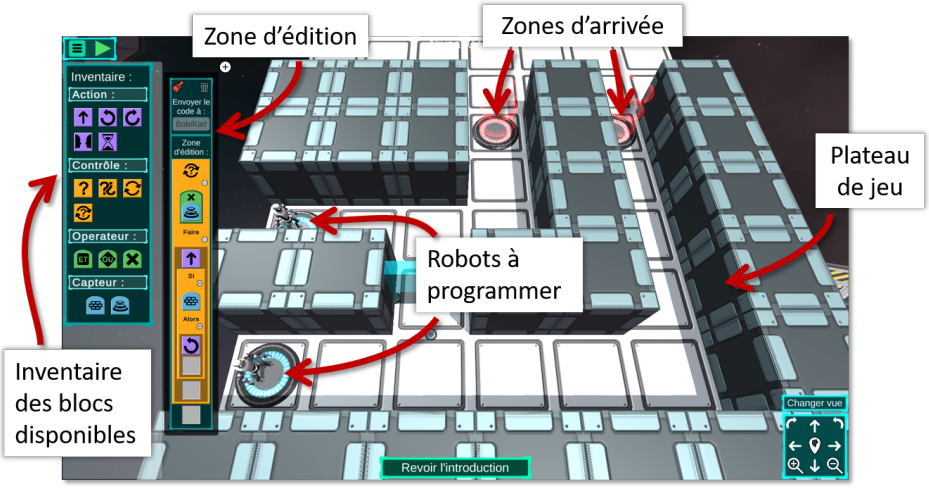


Figure 1 : Interface joueur commentée d'une mission de SPY

SPY est conçu comme une ressource à l'interface de la robotique pédagogique et des environnements visuels de programmation. Du point de vue du joueur, SPY reprend les codes de la robotique pédagogique où des actions simples de déplacement, représentées sous forme de blocs, permettent de programmer un ou plusieurs robots. Ces blocs sont complétés par des blocs plus complexes (incluant des structures de contrôle, des opérateurs logiques et des capteurs) utiles pour résoudre certaines situations du jeu. Cela permet aux élèves une première rencontre avec ces nouveaux concepts avant de les manipuler dans des environnements de programmation plus ouverts comme Scratch, puis dans le cadre de la programmation textuelle.

Du point de vue de l'enseignant, SPY fournit deux éditeurs pour leur permettre d'adapter la ressource à l'usage souhaité :

1. SPY : https://spy.lip6.fr
2. Code source du jeu SPY : https://github.com/Mocahteam/SPY
3. Données ouvertes du jeu SPY : https://spy.lip6.fr/openTraces.html

— *L'éditeur de mission* permet à l'enseignant de créer ses propres missions ou de modifier les missions déjà intégrées au jeu.
— *L'éditeur de scénario* permet à l'enseignant d'agencer et scénariser des missions afin de construire sa propre séquence pédagogique.

La suite de cette section est organisée en deux parties. Tout d'abord, dans la section 3.1, nous présentons de manière détaillée les différents aspects de SPY qui lient les caractéristiques de l'artefact, notamment les ressorts ludiques, avec les notions relatives à la pensée informatique. Puis, dans la section 3.2, nous analysons certaines caractéristiques du jeu en termes de variables didactiques et/ou des éléments jouant sur la complexité des situations.

3.1. Constituants du jeu SPY et leurs fonctions

3.1.1. Blocs de programmation (instructions et expressions)

Les blocs de programmation sont les éléments fondamentaux de SPY, car ils constituent les briques de base permettant d'écrire les programmes exécutés par les robots. Ils sont répartis en quatre catégories :

— **Les blocs d'action :** ils permettent de définir les actions pouvant être réalisées par les agents : « Avancer », « Pivoter à gauche », « Pivoter à droite », « Attendre », « Activer un terminal » et « Faire demi-tour ». À noter que toutes ces actions sont atomiques à l'exception de l'action « Faire demi-tour » qui peut être décomposée en deux actions « Pivoter à droite » ou deux actions « Pivoter à gauche ».
— **Les blocs de contrôle :** ils permettent de contrôler les blocs d'action à exécuter : « Si Alors », « Si Alors Sinon », « Répéter n fois », « Tant que » et « Répéter indéfiniment ».
— **Les capteurs :** ils 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 interdite ou une porte se trouve en face de lui ; et si un terminal ou une sortie se trouve sur sa position.
— **Les opérateurs :** ils 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 ».

Les blocs de programmation sont liés aux notions d'instruction et d'expression. Les choix réalisés en termes de transposition de ces notions dans SPY portent notamment sur les formes associées aux différents blocs. Ainsi, dans le jeu, les instructions (blocs d'action et de contrôle) sont représentées par des formes à l'entête carrée alors que les expressions (blocs capteurs et opérateurs) sont représentées par des formes à l'entête arrondie. Cette représentation visuelle donne implicitement des informations sur la grammaire du langage et donc sur les blocs combinables ou non. La figure 2 présente la grammaire du langage de SPY, nous observons deux types de réceptacles de couleur grise, l'un avec une forme carrée et l'autre avec une entête arrondie. Les instructions de couleur violette (comme l'action « Avancer ») et les structures de contrôle de couleur orange (comme les blocs « Répéter n fois » et « Tant que ») sont de forme carrée et peuvent donc être glissés dans n'importe quel réceptacle carré. Les constituants d'une expression tels que les capteurs de couleur bleu ciel (comme le bloc « Sortie détectée ») et les opérateurs de couleur verte (comme le bloc « Non ») ont une entête de forme arrondie et peuvent donc être glissés dans n'importe quel

réceptacle aux bords supérieurs arrondis.

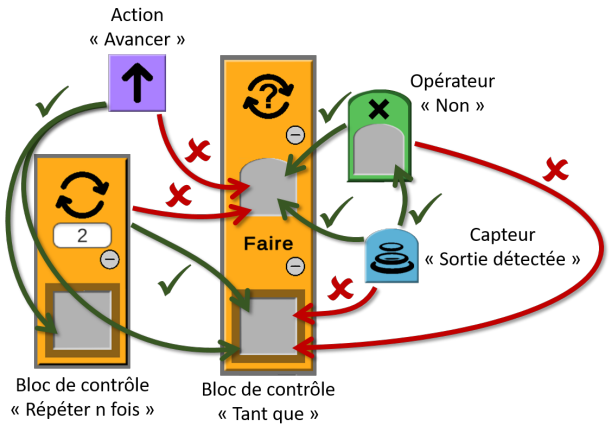


Figure 2 : Illustration de la grammaire des blocs de SPY

Ces différents blocs sont organisés, par glisser/déposer, dans une zone d'édition que nous présentons maintenant.

3.1.2. Zone d'édition

Une zone d'édition accueille les blocs de programmation permettant de construire, pour un robot, les solutions aux problèmes posés. Lorsque le problème implique plusieurs robots, chaque robot a sa propre zone d'édition associée. Une zone d'édition doit contenir au moins une instruction pour pouvoir être exécutée.

À chaque zone d'édition est associé le nom d'un robot, celui qui exécute la séquence d'instructions définie. Ceci fait référence à la notion de nommage et au fait qu'un objet informatique peut être référencé par son nom. Par exemple, dans la figure 1, si le nom du robot indiqué dans le champ « Envoyer le code à : » correspond au nom d'un des robots présents dans la scène, alors le code lui est envoyé pour qu'il l'exécute. Le résultat de cette exécution est visible sur le plateau de jeu.

3.1.3. Plateau de jeu

Dans SPY, le plateau de jeu est défini par un plan pouvant contenir différents éléments de jeu tels que du sol, des murs, des robots programmables, des points d'arrivée, des sentinelles, des portes, etc. Le joueur doit donc programmer un ou plusieurs robots pour les aider à atteindre les points d'arrivée tout en gérant les obstacles.

Les sentinelles, par exemple, interdisent des zones du labyrinthe dans leur ligne de vue. Si un robot se trouve sur l'une de ces zones, il est détecté, ce qui met fin à la partie. Le joueur doit donc programmer ses robots pour qu'ils évitent ces zones interdites. Tout comme pour les robots, les sentinelles peuvent contenir un programme pré-construit par l'enseignant (que le joueur ne peut modifier) ce qui rend les zones interdites dynamiques. Le joueur doit donc sélectionner chaque sentinelle du jeu, vérifier si un programme lui est affectée et, le cas échéant, observer le programme, le comprendre, anticiper les mouvements de la sentinelle et programmer son robot en conséquence.

Les portes sont également des objets interactifs. Elles ont deux états (ouvert ou fermé) et sont contrôlées par des terminaux de contrôle. Les portes et les terminaux de contrôle sont donc un premier exemple de transposition informatique des notions d'état, d'événement et

d'objet modifiable. Ainsi, agir sur un terminal de contrôle a comme conséquence de modifier l'état d'une porte (l'ouvrir ou la fermer).

3.1.4. Outils de contrôle sur l'exécution des programmes

Lorsque le joueur souhaite tester sa solution, il clique sur un bouton pour lancer l'exécution. Chaque zone d'édition envoie son programme au robot ciblé s'il existe, et chaque agent (robot ou sentinelle) exécute ses actions en parallèle. Le joueur peut suivre l'exécution des programmes (voir Figure 3) en observant les agents bouger sur le plateau de jeu en fonction des actions en cours d'exécution (mises en évidence dans le panneau d'exécution des programmes). Le joueur peut mettre en pause l'exécution à tout moment et exécuter les programmes pas à pas. Plusieurs savoirs informatiques sont représentés par ces choix.

Tout d'abord, le choix de l'utilisation d'un panneau d'affichage, appelé « contexte d'exécution » (voir Figure 3), différent de la zone d'édition (voir Figure 1), matérialise la différence entre un programme en cours d'édition et ce même programme chargé en mémoire par les robots et exécuté. Dans l'exemple de la figure 3, un même programme a été envoyé à deux robots différents qui exécutent donc leur programme indépendamment l'un de l'autre. Un premier robot est en train d'exécuter l'action « Pivoter à gauche » car il était face à un mur, alors que le second exécute l'action « Avancer ».

Ensuite, la mise en évidence du résultat de l'évaluation des capteurs renvoie à la notion d'évaluation d'expressions booléennes. Ici, seuls les capteurs notifient le résultat de leur évaluation et non les opérateurs logiques. C'est donc au joueur de reconstruire l'évaluation globale de l'expression booléenne à partir des indices donnés sur chaque capteur.

Enfin, plus classique, la notion de contrôle de l'exécution d'un programme est introduite avec un panneau dédié permettant de mettre en pause l'exécution du programme, d'exécuter pas à pas, de reprendre l'exécution, de l'accélérer ou de l'interrompre.

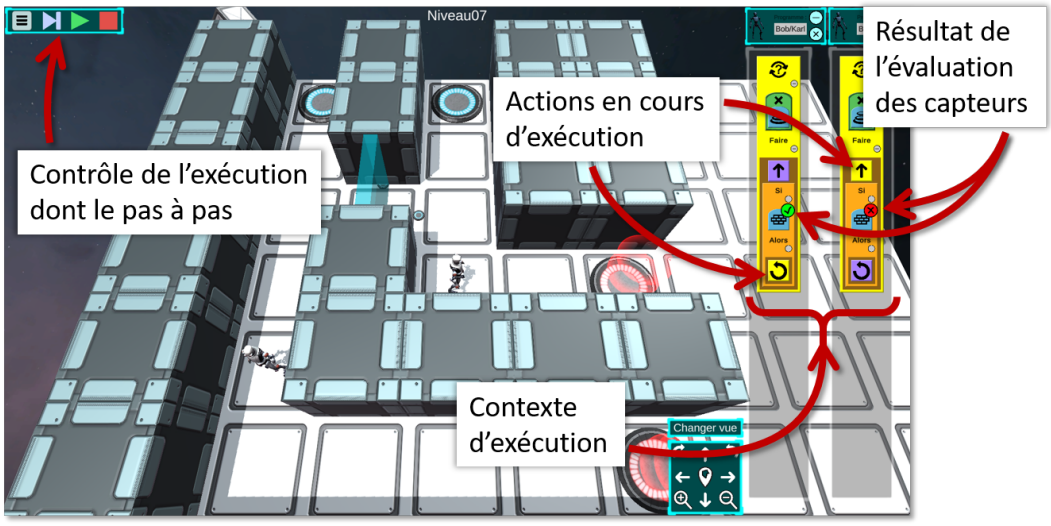


Figure 3 : Vue du jeu SPY en contexte d'exécution

3.1.5. Brouillard et texte d'introduction

Dans une mission 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 de la mission décrit dans les textes d'introduction. Le brouillard permet de modifier cette règle en limitant la vue d'un agent à son entourage proche (voir Figure 4). Dans ce cas, le texte d'introduction

présentant l'objectif de la mission joue un rôle fondamental, car il doit 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 ou sous la forme d'un algorigramme, et le joueur doit le traduire à l'aide du langage formel du jeu. Cette fonctionnalité permet donc de travailler explicitement la conversion de l'information donnée dans un premier registre vers le registre du langage de programmation.

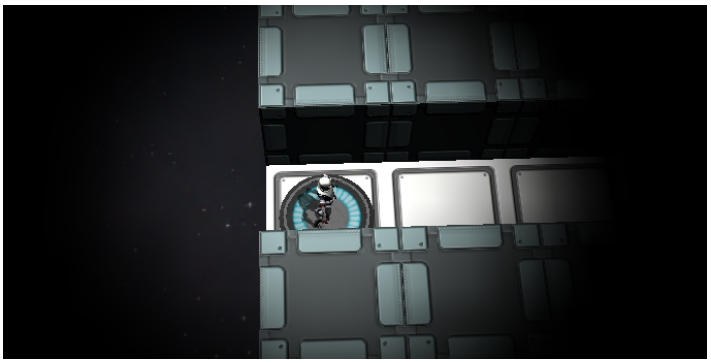


Figure 4 : Exemple de mission où le brouillard est activé.

3.2. VARIABLES DIDACTIQUES ET COMPLEXITÉ

Après avoir présenté les différents éléments constituant le jeu SPY et leur fonction par rapport aux savoirs travaillés nous reprenons certains de ces constituants en les examinant en tant que variables didactiques (Margolinas, 1992). Nous les considérons comme telles dans la mesure où, par le biais de l'éditeur de mission, il s'agit d'éléments pouvant être modifiés par l'enseignant et ayant un effet sur le processus de résolution que les élèves vont adopter pour résoudre les missions. Certains éléments peuvent également, sans nécessairement modifier les procédures de résolution efficaces ou disponibles, agir sur la complexité de l'exercice, par exemple en introduisant des choix à la charge de l'élève.

3.2.1. Blocs de programmation

Chaque bloc de programmation de l'inventaire peut être paramétré pour définir, dans une mission donnée, les blocs disponibles et en quelle quantité. Ainsi, l'accès à certains blocs (ou leur quantité) peut être un moyen pour forcer le joueur à utiliser certains blocs (par exemple, demander de faire avancer un robot de plusieurs cases en limitant le nombre de blocs « Avancer » à 1 et en donnant accès au bloc de contrôle « Répéter n fois »).

Ce paramétrage peut être utilisé pour réduire ou augmenter la complexité d'une mission. Par exemple, une mission d'introduction pourra ne contenir que les blocs utiles à sa résolution, évitant ainsi au joueur de devoir choisir parmi des blocs potentiellement inutiles.

3.2.2. Zone d'édition

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

Une zone d'édition peut également être préconstruite. Le programme ainsi proposé au joueur peut être complet, partiel ou bogué. Le joueur devra le compléter ou le corriger en conséquence.

SPY offre également la possibilité de ne proposer au joueur qu'une seule zone d'édition pour plusieurs robots. Ceci permet de construire des missions où le joueur devra imaginer un unique programme permettant de contrôler plusieurs robots pouvant être dans des labyrinthes différents. Le joueur pourra ainsi se confronter à la réalisation de solutions plus génériques et par exemple, utiliser des boucles non bornées plutôt que des boucles bornées.

Enfin, le glisser/déposer est le mode d'interaction permettant d'ajouter des blocs de l'inventaire à une zone d'édition. Il est possible de désactiver cette fonctionnalité, dans ce cas la mission doit proposer un ou plusieurs programmes préconstruits que le joueur pourra exécuter sans pouvoir les modifier (voir Figure 5). Si cette variable didactique est activée, elle modifie profondément le type de tâche à réaliser par l'élève, en effet, il ne s'agira plus ici de construire un programme mais au contraire de lire et comprendre des programmes préconstruits.

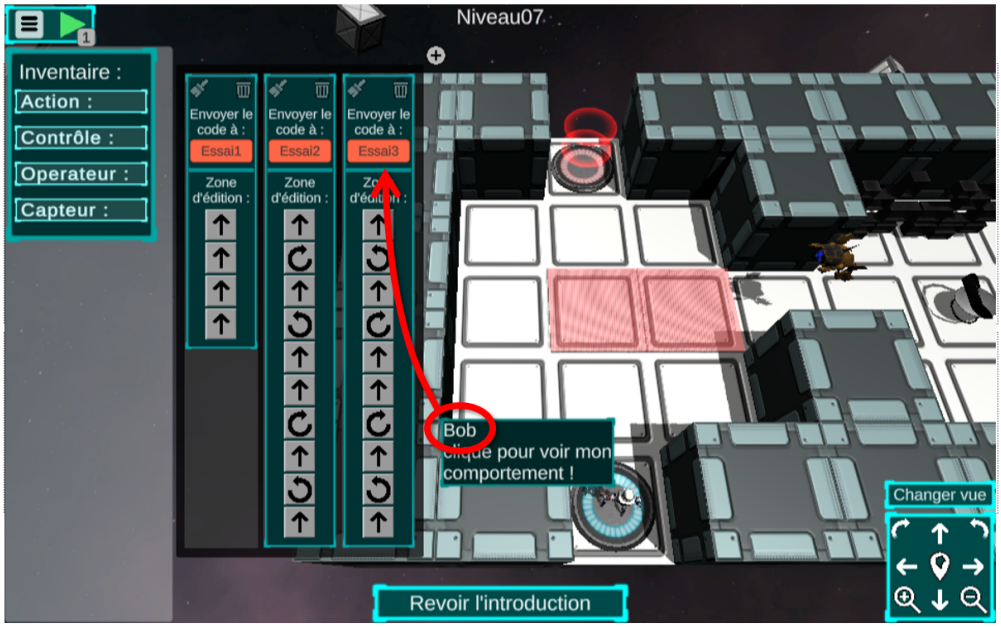


Figure 5 : Exemple de mission où le le glisser/déposer est désactivé. Le joueur doit remplacer l'un des textes « Essai1 », « Essai2 » ou « Essai3 » avec le nom du robot (Bob). Dans cet exemple la bonne solution est « Essai3 ».

3.2.3. Plateau de jeu

Concernant le plateau du jeu, l'enseignant peut modifier sa taille ainsi que le contenu qui le compose. Il peut donc construire des labyrinthes plus ou moins complexes. Par exemple, si l'enseignant positionne l'arrivée à l'extérieur du champ de vue de la caméra, l'élève devra effectuer des actions de déplacement de la caméra afin de recueillir des informations sur la constitution d'une mission avant de commencer à réfléchir à la résolution du problème. Il pourra se poser des questions telles que : où se trouve l'arrivée ? Y a-t-il une sentinelle dans la mission ? etc. Il devra donc mobiliser des procédures relatives à la compétence consistant à comprendre les données du problème avant de tenter de le résoudre.

3.2.4. Contrôle de l'exécution des programmes

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 missions contenant des sentinelles en mouvement. SPY permet donc de résoudre une mission en plusieurs coups. Le joueur peut définir une première séquence de blocs, l'exécuter, observer la nouvelle situation, ajouter de nouveaux blocs, exécuter à nouveau, observer, etc. Il est donc possible de construire une solution étape par étape, ce qui permet aux élèves d'engager un processus de résolution par décomposition du problème.

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

3.3. SYNTHÈSE DES LIENS ENTRE FONCTIONNALITÉS DE JEU ET SAVOIRS

Tableau 1 : Synthèse des liens entre les fonctionnalités du jeu et les savoirs abordés par le jeu

		Fonctionnalités ludiques										
		Forme des blocs	Blocs « Se retourner » *VS* « Pivoter gauche/droite »	Bloc « Répéter n fois »	Bloc « Tant que »	Blocs « Si Alors » et « Si Alors Sinon »	Zone d'édition	Plateau de jeu	Portes/Terminaux	Contrôle de l'exécution	Brouillard	Sentinelles
Savoirs	Grammaire	X										
	Instructions	X										
	Actions (non) atomiques		X									
	Répétition			X	X							
	Test logique				X	X						
	Expressions	X			X	X						
	Edition de programme						X					
	Lecture de code						X					X
	Compréhension de code						X					X
	Nommage						X					
	Comprendre les données du problème							X				
	Etat et évènements								X			
	Débogage									X		
	Traduction										X	
	Généricité						X					
	Décomposition de problèmes									X		

L'analyse des différents constituants de SPY montre que chacun a une fonction didactique. Si certaines sont évidentes, comme les blocs de contrôle qui permettent de manipuler les notions de programmation associées, d'autres sont moins évidentes, comme l'association

de programmes aux sentinelles qui demande au joueur de comprendre une séquence d'actions et d'anticiper son exécution, ou l'utilisation d'une seule zone d'édition pour contrôler deux robots qui force le joueur à généraliser sa solution. Le tableau 1 présente une vue synthétique des liens entre les mécaniques de jeu et les savoir liées.

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 (2006). 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 à de nouveaux problèmes (généralisation). Mais comment identifier, d'un point de vue micro, si telle ou telle compétence est en jeu dans une mission donnée ? Quelles influences les combinaisons de fonctionnalités ont-elles sur les compétences en jeu ? Répondre à ces questions est l'objet de la section suivante.

4. Rétro-ingénierie des compétences à partir des fonctionnalités de jeu : le formalisme GTC

Nous avons montré dans la section 3 comment les savoirs relatifs à la pensée informatique sont transposés dans le jeu SPY. Nous avons également évoqué quelques variables didactiques en relation avec les briques élémentaires du jeu. Dans cette section, nous proposons le formalisme GTC (Game To Competency) afin de décrire le potentiel sémiotique du jeu tel que défini par Mariotti et Maracci (2012) et ainsi répondre à notre première question de recherche : comment lier les compétences d'un référentiel avec les fonctionnalités ludiques d'un jeu sérieux ?

Le formalisme GTC que nous proposons pour répondre à cette question s'appuie sur le métalangage de description générique XML qui est ici utilisé pour décrire les missions d'un jeu. Si une mission contient une fonctionnalité de jeu donnée, la balise XML associée sera ajoutée au fichier XML décrivant la mission. Une mission est donc définie par l'agrégation d'un ensemble de balises XML, chacune décrivant l'état d'une fonctionnalité de jeu et de ses paramètres. Identifier une compétence dans une mission, consiste donc à repérer dans la structure XML de cette mission les fragments XML caractérisant cette compétence. Nous proposons donc de décrire chaque compétence de la manière suivante :

— le nom de la compétence ;
— une liste de filtres, chaque filtre correspondant au nom d'une balise XML et pouvant être complété par une liste de contraintes ;
— une règle qui définit les conditions globales que la compétence doit respecter. Cette règle est une expression booléenne qui combine les résultats des filtres.

Un filtre recherche, dans la structure XML décrivant une mission, toutes les occurrences d'une balise XML spécifique qui vérifient les contraintes définies. L'ensemble des contraintes possibles est présenté dans le Tableau 2.

En résumé, une compétence est décrite avec des filtres sur les balises XML et une règle globale qui orchestre ces filtres. Si cette règle est vraie, alors nous considérons que la compétence est mobilisée dans la mission analysée. Ces filtres s'appuient sur la structure du langage XML et sont indépendants des balises définies dans le jeu SPY. Notre proposition est donc générique pour tout jeu décrivant ses données à l'aide du langage de balisage XML. Trois exemples détaillés sont présentés dans les sections 4.1, 4.2 et 4.3.

L'identification d'une compétence repose donc sur l'évaluation d'un ensemble de filtres. Dans le contexte du jeu SPY, nous présentons dans le Tableau 3 les principales balises XML

Tableau 2 : Description des contraintes possibles sur un filtre.

$TAG \in$ ensemble des balises XML de SPY
$ATTR \in$ ensemble des attributs valides pour TAG
$OPP \in \{=, <, \leq, >, \geq, \neq\}$
$VAL \in \mathbb{N}$

Contraintes applicables sur un filtre ciblant TAG	Description
$\varnothing$	Filtre les balises TAG.
$ATTR\,OP\,VAL$	Filtre les balises TAG qui ont un attribut $ATTR$ égal/différent... à une valeur entière.
$ATTR\ isIncludedIn\ SET$	Filtre les balises TAG qui ont un attribut $ATTR$ dont la valeur est incluse dans un ensemble de valeurs (SET).
$ATTR\ sameValue\ TAG_2\ ATTR_2$	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.
$hasChild$	Filtre les balises TAG qui contiennent au moins une balise fille.

que nous référençons dans cet article. Un modèle de mission complet et commenté est donné dans Muratet (2023a).

Tableau 3 : Description des principales balises structurant une mission SPY.

Tag	Description
<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 disponible dans l'inventaire (si Y = -1 $\Rightarrow$ bloc en quantité illimitée).
<script outputLine="X" editMode="Y" type="Z">	Définit une zone d'édition qui enverra son contenu sur le canal de communication X (voir balise <robot> et <guard>). La propriété **editMode** indique si le joueur peut changer ou non le canal de communication. La propriété **type** indique si cette zone d'édition contient un programme préconstruit optimal, non optimal, bogué ou indéfini.
<robot inputLine="X">	Définit un robot contrôlé par le joueur. Ce robot écoute le canal de communication X.
<guard inputLine="X">	Définit une sentinelle. Cette sentinelle écoute le canal de communication X.

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 (PIAF Project, 2021). 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 indique la sous-compétence Y de

la compétence X, 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 filtres construits pour caractériser les compétences du PIAF en fonction des fonctionnalités de jeu est accessible dans Muratet (2023b). Nous décrivons ici, en détail, l'application de GTC sur une sélection de trois compétences du PIAF et nous résumons les autres en Annexe 8.

4.1. 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 » (PIAF Project, 2021, p. 2).

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

Nous caractérisons cette compétence à l'aide d'un filtre sur la balise « script ». Nous complétons ce filtre avec une contrainte « **ATTR OP VAL** » qui est instanciée ainsi : « **editMode = 2** ». Ce filtre, noté $F1$, définit donc l'ensemble des balises « script » qui ont un attribut « editMode » égal à la valeur 2.

La règle complète pour cette compétence est alors $C1.1 := Card(F1) \geq 1$ qui signifie : « La compétence C1.1 est en jeu dans la mission si le cardinal de l'ensemble des balises sélectionnées par le filtre est supérieur à 0 », autrement dit « si la mission contient au moins une zone d'édition nommable ».

4.2. 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 » (PIAF Project, 2021, p. 5).

Dans SPY, cette compétence est travaillée lorsqu'une mission 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 une mission qu'au moins une zone d'édition est associée à une sentinelle et qu'elle contient au moins un bloc de programmation. Nous caractérisons cette compétence à l'aide d'un filtre sur la balise « script ». Nous notons ce filtre $F1$ et nous le complétons avec deux contraintes :

— $C1$ (de type « **ATTR** sameValue **TAG2 ATTR2** ») définit l'ensemble des zones d'éditions (« script ») qui communiquent sur un canal de communication (ATTR = « outputLine ») identique à celui d'une sentinelle (TAG2 = « guard » et ATTR2 = « inputLine »). $C1$ est donc instanciée ainsi : « **outputLine** sameValue **guard inputLine** » ;
— $C2$ (de type « hasChild ») définit l'ensemble des zones d'éditions (« script ») qui contiennent au moins un bloc de programmation (« hasChild »).

$F1$ peut donc être formellement définit comme l'intersection de $C1$ et $C2$, $F1 := C1 \cap C2$.

Ainsi vérifier dans une mission qu'au moins une zone d'édition est associée à une sentinelle et qu'elle contient au moins un bloc de programmation se résume avec la règle suivante : $C1.5 := Card(F1) \geq 1$.

4.3. 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 » (PIAF Project, 2021, p. 7).

Dans SPY, il s'agit d'une situation où il n'est proposé au joueur que des blocs en quantité limitée. Le joueur doit alors les combiner dans le bon ordre pour résoudre la mission. Pour décrire cette compétence, nous définissons les filtres suivants : ($F1$) le glisser/déposer est activé pour permettre au joueur de combiner les actions ; ($F2$) aucun bloc n'est disponible en quantité illimitée ; et ($F3$) il y a au moins une action disponible dans l'inventaire.

Le filtre $F1$ porte sur la balise « dragdropDisabled ». Nous souhaitons vérifier que le glisser/déposer est actif et donc que la balise « dragdropDisabled » est absente dans la description de la mission. $F1$ définit donc l'ensemble des balises « dragdropDisabled » et nous chercherons à vérifier dans la règle globale que cet ensemble est vide.

Le filtre $F2$ porte sur la balise « blockLimit ». Nous complétons ce filtre avec une contrainte « **ATTR OP VAL** » qui est instanciée ainsi : « **limit = -1** ». $F2$ définit donc l'ensemble des balises « blockLimit » qui ont un attribut « limit » égal à la valeur -1. Ce filtre définit l'ensemble des blocs disponibles en quantité illimité, nous chercherons donc à vérifier dans la règle globale que cet ensemble est vide.

Le filtre $F3$ porte lui aussi sur la balise « blockLimit ». Nous complétons ce filtre avec deux contraintes :

— $C1$ (de type « **ATTR OP VAL** ») définit l'ensemble des blocs de programmation (« blockLimit ») dont le nombre d'exemplaires (ATTR = « limit ») est supérieur ou égal (OP = « $\geq$ ») à 1 (VAL = 1). $C1$ est donc instanciée ainsi : « **limit** $\geq$ **1** » ;

— $C2$ (de type « **TAG ATTR** isIncludedIn **SET** ») définit l'ensemble des blocs de programmation (« blockLimit ») dont le type (ATTR = « blockType ») est inclus dans l'ensemble des blocs d'action (SET = $\{Forward, TurnLeft, TurnRight, Wait, Activate, TurnBack\}$). $C2$ est donc instanciée ainsi : « **blockType** isIncludedIn **{Forward,TurnLeft,TurnRight,Wait,Activate,TurnBack}** ».

$F3$ peut donc être formellement définit comme l'intersection de $C1$ et $C2$, $F3 := C1 \cap C2$.

Ainsi, vérifier dans une mission qu'aucun bloc d'action n'est donné en quantité illimitée se résume avec la règle suivante, $C2.1 := F1 = \varnothing \;\&\&\; F2 = \varnothing \;\&\&\; Card(F3) \geq 1$. À noter ici que nous cherchons bien à vérifier que les ensembles $F1$ et $F2$ sont vides à savoir que le glisser/déposer n'est pas désactivé et qu'il n'y a pas de blocs en quantité illimitée.

5. INTÉGRATION DU MODULE D'ANALYSE AUTOMATIQUE

Pour répondre à notre première question de recherche, nous avons proposé GTC, un formalisme générique pour décrire des compétences à l'aide de fonctionnalités ludiques de jeu. Nous avons détaillé la description de trois compétences du PIAF. Parmi les 26 compétences du PIAF, 21 peuvent être appliquées à SPY et sont résumées dans l'Annexe 8. En plus des compétences du PIAF, nous avons confronté GTC 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 [4] (Cadre de Référence des Compétences Numériques). Enfin, nous

4. CRCN, p. 13, https://eduscol.education.fr/document/20389/download, consulté le 08/11/2023

Tableau 4 : Analyse d'une même mission avec les différents référentiels

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 C5.1 - Représenter des objets ou séquences d'actions au moyen d'une représentation formelle 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 »

avons exploité GTC pour décrire l'ensemble des dimensions ludiques de SPY (dit référentiel SPY), ici une dimension ludique est souvent associée à une, et une seule, fonctionnalité ludique mais certaines dimensions nécessitent plus de finesse. Par exemple, décrire le principe d'ouverture et de fermeture de porte, ne se limite pas à la présence de portes dans le niveau (qui ne peuvent être que des éléments de décors si elles ne sont pas contrôlables). Dans ce cas, nous décrivons à l'aide de notre formalisme GTC, que le principe ludique de porte interactive est conditionné par la présence d'une porte connectée à un terminal et la présence d'au moins un bloc « Activer un terminal » dans l'inventaire du joueur. Toutes ces caractérisations sont accessibles dans Muratet (2023b).

Nous sommes ainsi en mesure d'analyser automatiquement chaque mission (ainsi que de nouvelles missions créées par l'enseignant) en fonction de différents référentiels. Le Tableau 4 illustre l'analyse d'une mission selon les trois référentiels. Cette visualisation fournit à l'enseignant des informations selon le référentiel de son choix. Dans cet exemple, le niveau 1 de la compétence 3.4 du CRCN est identifié. Cette compétence est définie comme « Lire et construire un algorithme qui comprend des instructions simples ». Le CRCN est moins précis que le PIAF mais offre une vue plus globale des compétences en jeu.

5.1. Intégration de l'analyseur à SPY

Le jeu SPY est composé de 4 modules : un *launcher*, un *player* et deux éditeurs (voir Figure 6). Le *launcher* est le point d'entrée pour les élèves, il leur propose de choisir la mission à jouer à l'intérieur d'un scénario. Lorsque l'élève a sélectionné la mission du scénario qu'il souhaite jouer, le *launcher* lance le *player* qui exécute la mission sélectionnée.

Les enseignants, quant à eux, peuvent manipuler deux éditeurs pour créer ou modifier des scénarios ainsi que des missions de jeu. Afin de répondre à la seconde question de recherche (Comment exploiter ce lien [GTC] pour analyser automatiquement les missions du jeu sérieux SPY et en extraire les compétences de la pensée informatique ?), nous avons intégré

notre analyseur de compétence à deux niveaux :

— dans le *launcher*, lors de la sélection d'un scénario, il informe l'élève et indirectement l'enseignant sur les compétences mobilisées dans ce scénario ;
— dans l'éditeur de scénario, selon une double modalité, il filtre les missions par compétence et analyse les compétences d'une mission sélectionnée.

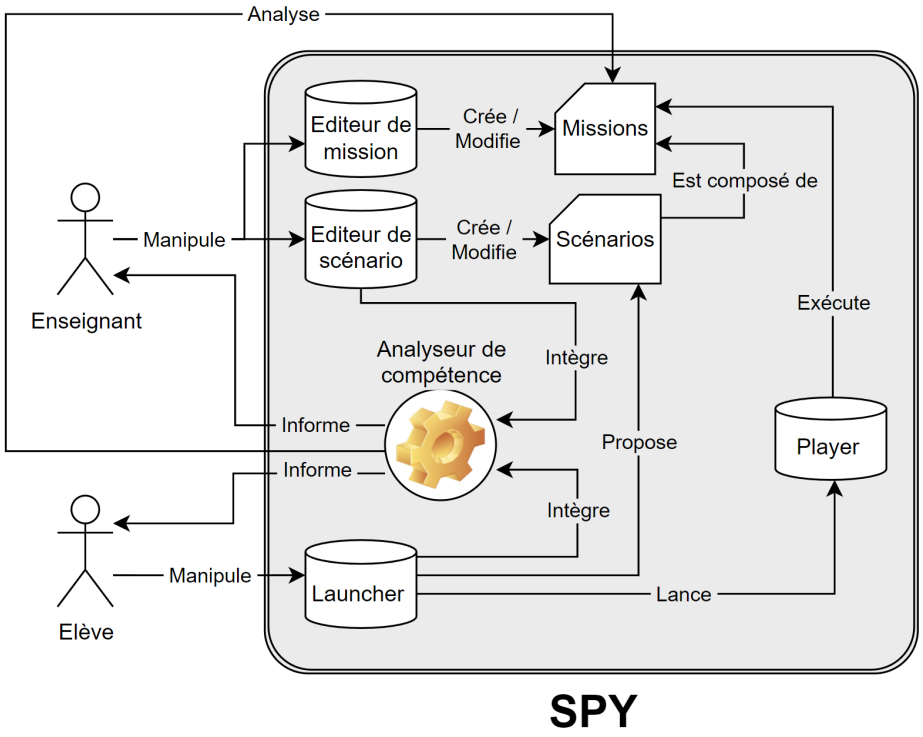


Figure 6 : Vue schématique de l'architecture logicielle de SPY

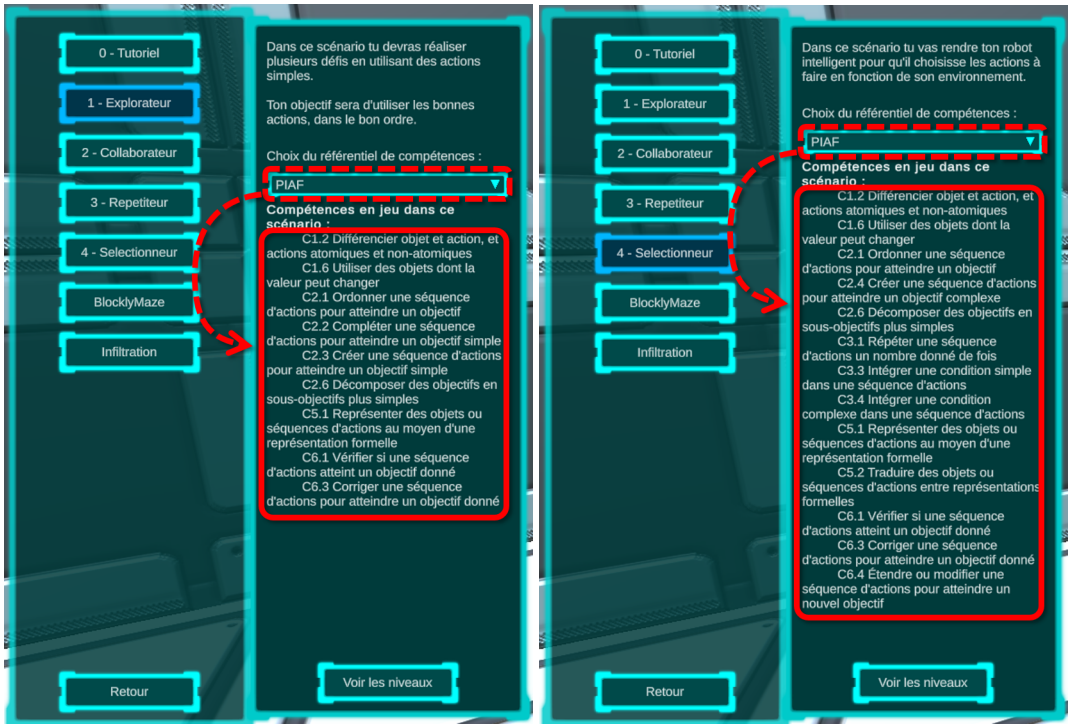
5.1.1. Sélection d'un scénario

Lorsqu'un utilisateur (enseignant ou élève) sélectionne un scénario, la liste des compétences travaillées dans ce scénario lui est proposée (voir Figure 7). Cette liste est construite automatiquement à partir de l'analyse des missions constituant le scénario. L'utilisateur a la possibilité de sélectionner le référentiel de son choix parmi ceux intégrés dans le jeu. Cette analyse a été pensée pour les enseignants afin de leur permettre d'avoir une synthèse rapide des compétences travaillées dans un scénario sans avoir à jouer eux-mêmes à toutes les missions d'un scénario et en déduire les compétences travaillées.

5.1.2. Composition d'un scénario

SPY contient un éditeur de scénario (voir Figure 8). Cet éditeur permet à l'enseignant de consulter, d'éditer et de créer des scénarios. Le panneau de gauche contient l'ensemble des missions de la base de données de SPY, le panneau central présente les informations sur la mission actuellement sélectionnée (dont l'analyse automatique des compétences mobilisées dans cette mission en fonction du référentiel sélectionné), et le panneau de droite affiche l'ensemble des missions intégrées dans le scénario.

L'analyse automatique des compétences est donc une information complémentaire donnée à l'enseignant pour lui permettre de déterminer si la mission sélectionnée est un candidat potentiel pour être intégré dans sa scénarisation pédagogique.



(a) Scénario « Explorateur » (b) Scénario « Sélectionneur »

Figure 7 : Analyse automatique de deux scénarios à l'aide du référentiel PIAF. Les compétences identifiées pour chaque mission dans l'encadré à trait plein sont calculées dynamiquement par le système en fonction du choix du référentiel sélectionné (encadré en pointillé).

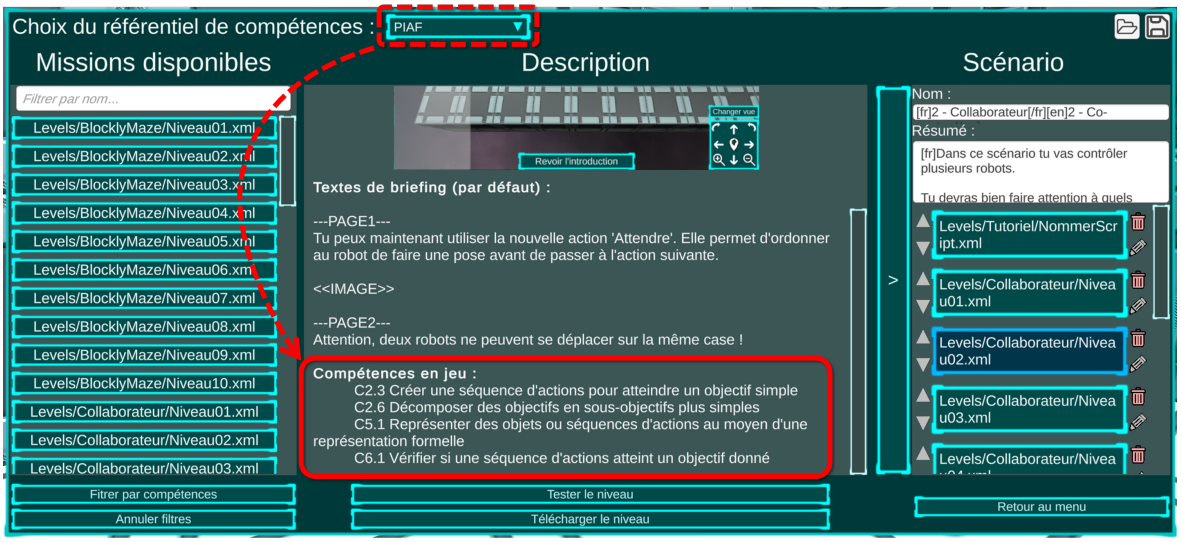


Figure 8 : Vue de l'éditeur de scénario intégré à SPY. La partie encadrée en trait plein est le résultat de l'analyse automatique de la 3ème mission du scénario collaborateur à l'aide du référentiel PIAF (encadré en trait pointillé).

5.1.3. Système de filtre

Comme nous l'avons vu dans la section précédente, l'éditeur de scénario permet aux enseignants d'avoir accès à l'ensemble des missions de la base de données de SPY. Cependant,

identifier les missions permettant de travailler une compétence, ou un ensemble de compétences, n'est pas aisé compte tenu du grand nombre de missions disponibles (près de 60 missions). Nous avons donc détourné l'analyseur automatique de scénario pour créer un filtre de missions par compétences. Cette fonctionnalité permet de sélectionner une ou plusieurs compétences d'un référentiel et de filtrer les missions pour ne proposer à l'enseignant que celles respectant cette contrainte (voir Figure 9).

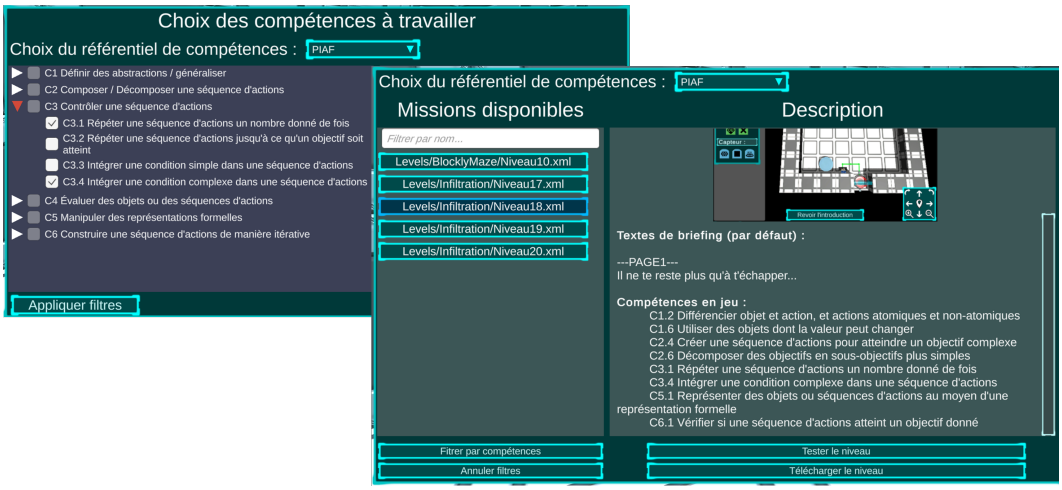


Figure 9 : Vues de l'éditeur permettant de filtrer les missions par compétences. Dans cet exemple les compétences C3.1 et C3.4 du PIAF ont été sélectionnées. Après application du filtre, seules 5 missions de la base de donnée correspondant à la sélection sont proposées à l'enseignant.

5.2. Exploitation de l'analyseur à l'extérieur de SPY

Nous avons également exploité les résultats de l'analyseur automatique des missions pour étudier l'évolution de la complexité d'un scénario de jeu. Nous avons réalisé ce travail sur deux scénarios de SPY : le scénario « Infiltration » composé de 20 missions et le scénario « BlocklyMaze », une réplique du jeu du même nom [5], composé de 10 missions. Nous avons utilisé le référentiel du PIAF pour caractériser la dimension didactique des différents scénarios et le référentiel SPY pour caractériser leur dimension ludique. Le cumul des compétences et fonctionnalités ludiques dans chaque mission donne un indicateur de sa complexité (voir Figure 10). 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.* (2017) qui analysent manuellement chaque mission de jeu selon les dimensions ludiques et pédagogiques. Dans notre cas, l'analyse des missions est automatisée. Cette représentation graphique permet ainsi de comparer la complexité de différents scénarios selon les mêmes critères d'analyse. Nous constatons que le scénario Infiltration commence avec un niveau de complexité moindre que BlocklyMaze et se termine avec un niveau de complexité plus élevé. En revanche, la progression de la complexité semble plus régulière dans BlocklyMaze. Cette visualisation est également utile aux concepteurs pour détecter les compétences non traitées dans un scénario ou pour identifier les missions mal équilibrées ou mal positionnées dans le scénario. Par exemple, dans le scénario Infiltration, la compétence C4 n'est traitée que dans la mission 7. Il pourrait donc être intéressant de créer de nouvelles missions mobilisant les compétences

5. BlocklyMaze : https://blockly.games/maze, consulté le 08/11/2023

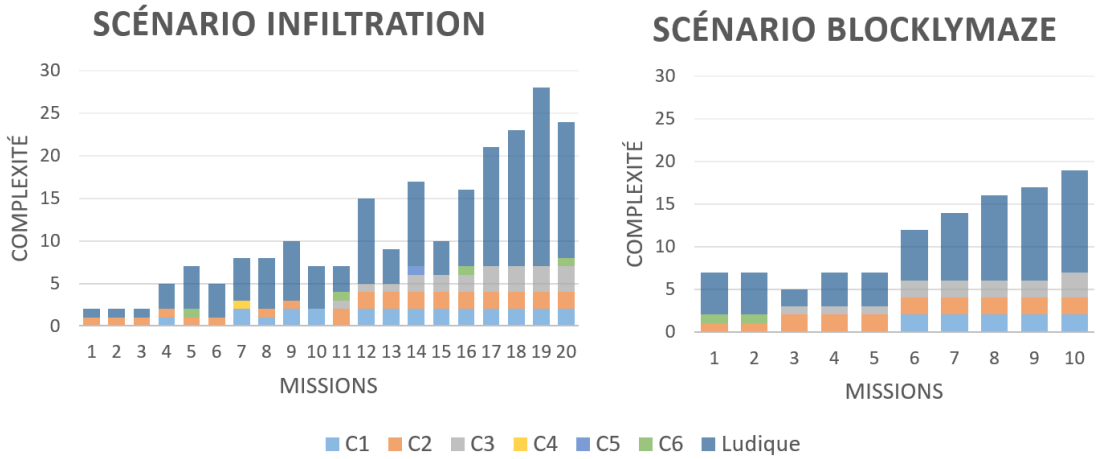


Figure 10 : Analyse des deux scénarios du jeu SPY. Chaque bâton représente un indicateur de la complexité d'une mission obtenue par l'addition du nombre de compétences mobilisées et du nombre de fonctionnalités ludiques impliquées.

C4.2 et C4.3 du PIAF. Cette analyse révèle aussi des pics de complexité (missions 12 et 14 du scénario Infiltration) qui invitent peut-être à étudier ces missions plus en détail et à envisager des ajustements.

6. Expérimentation : contexte et protocole expérimental

Nous avons présenté dans la section 5 comment le module d'analyse automatique a été intégré à SPY pour développer de nouvelles fonctionnalités (analyse d'un scénario, analyse d'une mission, filtrage). Ce module constitue donc une réponse à notre deuxième question de recherche : comment exploiter ce lien [GTC] pour analyser automatiquement les missions du jeu sérieux SPY et en extraire les compétences de la pensée informatique ?

Après un premier temps d'implémentation du module d'analyse automatique sur la base de l'expertise de l'équipe de conception du jeu, il est nécessaire d'interroger la qualité des résultats produits. Afin d'aborder cette question, nous avons conduit une expérimentation en collaboration avec des enseignants en informatique. En effet, si la finalité du module est de contribuer à la problématique générale énoncée dans l'introduction (aider des enseignants peu familiers avec la pensée informatique à repérer, dans des situations de jeu, les compétences mobilisées), l'évaluation de la qualité des résultats produits par l'analyseur automatique de mission nécessite une certaine expertise en informatique et en son enseignement.

6.1. Profil des participants

Plus précisément, l'expérimentation a été menée avec un groupe de cinq étudiants (une femme et quatre hommes), dans le cadre de la deuxième année d'une formation d'enseignants d'informatique dans le canton de Genève en Suisse. Elle a été organisée dans le cadre d'une séance du séminaire "Recherche de didactique de l'informatique". Cette séance a donc participé à la formation à la recherche des étudiants en leur permettant de vivre une situation de recherche. En raison de leur participation à cette formation, tous les participants partagent des caractéristiques communes :

1. une formation universitaire de niveau Master 2 en informatique ou jugée équivalente ;

2. une année minimum d'enseignement de l'informatique au secondaire 1 (équivalent du collège français) ;
3. des enseignements d'informatique assurés au secondaire 2 (équivalent du lycée français) depuis la rentrée scolaire 2023.

Pour compléter ce profil, nous avons posé différentes questions aux participants. Tout d'abord, en plus de leur formation initiale en informatique, quatre des cinq participants ont une expérience professionnelle dans le domaine de l'informatique, pour des durées allant de 2 à 25 ans. Concernant leurs expériences d'enseignement, aucun n'a enseigné l'informatique au primaire. Cependant, trois d'entre eux ont entre quelques mois et 5 années d'expérience d'enseignement dans le secondaire (en plus de celle acquise pendant leur formation).

Par rapport à l'artefact SPY sur lequel ils ont eu à travailler durant l'expérimentation :

— tous les participants ont déjà utilisé au moins un environnement de programmation par blocs pour enseigner l'informatique (entre un et trois parmi Scratch, blockly-games, microbit, code.org, Future Engineer) ;
— tous les participants estiment que la programmation par bloc est une bonne métaphore pour initier les élèves à la programmation ;
— tous les participants signalent utiliser l'approche par compétences pour définir des objectifs d'apprentissage et analyser / concevoir des ressources.

Ces éléments confirment une familiarité des participants avec un environnement du type de SPY et avec l'enseignement de l'informatique. Nous pouvons donc faire l'hypothèse qu'ils présentent un profil pertinent pour évaluer la qualité des résultats produits par l'analyseur de compétences.

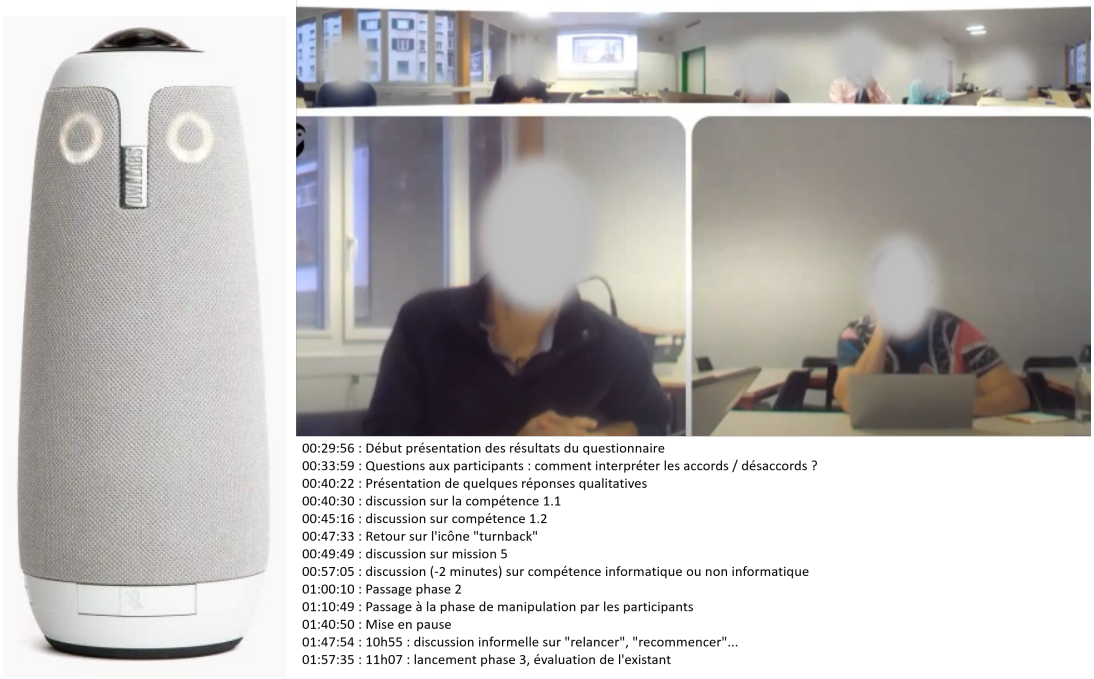
6.2. Protocole expérimental

Pour mener cette expérimentation, nous avons conçu un scénario spécifique (*ScenExp*) composé d'une sélection de six missions. Ce scénario joue sur les différentes fonctions et variables présentées dans la section 3.1, afin de permettre l'identification de compétences variées.

Le protocole expérimental a été décomposé en deux phases. La première phase était organisée à distance en modalité asynchrone. Chaque participant avait un travail individuel à réaliser : jouer le scénario *ScenExp* et procéder à une double analyse des missions. La deuxième analyse n'était accessible qu'à l'issue de la première, celle-ci n'étant plus modifiable.

— Première analyse : chaque participant devait décrire les compétences qui lui semblaient mobilisées dans chaque mission avec leurs propres mots. Il leur était demandé de décrire chaque compétence sous la forme « être capable de + verbe action + complément » et de justifier l'identification de cette compétence à partir d'éléments présents dans la mission.
— Seconde analyse : le référentiel PIAF, avec une courte description de chacune des compétences, était fourni aux participants. Ils devaient, pour chaque mission, indiquer la présence ou l'absence de ces compétences et justifier leurs choix. Cette phase avait comme objectif à la fois de faire découvrir le jeu SPY à tous les participants, et également d'introduire le référentiel PIAF afin de créer un espace de réflexion commun pour la phase 2. Les participants ont eu une semaine pour réaliser cette double analyse des missions.

La seconde phase a eu lieu en présentiel, lors d'une session de travail de 3 heures. Les deux auteurs de l'article étaient présents lors de cette session, l'un (*auteur1*) en tant que concepteur de SPY et animateur de la séance, l'autre (*auteur2*) en tant que responsable du



(a) Caméra de visioconférence 360° Meeting Owl 3 utilisée lors de l'expérimentation

(b) Vue de la session enregistrée dans Zoom avec la prise de notes synchronisée

Figure 11 : Dispositif de captation de la session de travail

séminaire recherche et avec une posture plus externe au déroulé de la séance. Des traces audio et vidéo ont été produites à l'aide d'un dispositif de prise de vues 360 (voir Figure 11a) couplé au logiciel de visioconférence Zoom (voir Figure 11b). Le module de discussion de Zoom a été utilisé par *auteur2* comme outil d'annotation. Cette prise de notes en temps réel est ainsi automatiquement synchronisées avec les flux vidéo et audio de l'enregistrement de la séance.

Cette phase a été découpée en plusieurs temps : accueil (5 min), retour collectif sur le travail engagé dans la phase 1 et discussions (30 min), pause (10 min), présentation dans SPY de l'intégration de l'analyseur automatique (10 min), manipulation individuelle de l'éditeur de scénario (45 min), évaluation de l'utilité perçue des fonctionnalités liées à l'analyseur automatique (30 min), discussions informelles et clôture (30 min).

7. Résultats et analyses

7.1. Premier temps de la phase 2 : retour collectif sur le travail de la phase 1

Lors du premier temps de discussion avec les participants, nous revenions sur la deuxième partie du questionnaire. Chaque participant devait signaler quelles compétences du PIAF il estimait être mobilisées dans chaque mission. Sur les cinq participants prenant part à l'expérimentation, quatre ont complété le questionnaire proposé pour la phase 1. Le cinquième répondant (R5) a bien joué au scénario proposé et a complété le questionnaire, mais une session expirée a entraîné la perte de ses données. Il disposait tout de même de tous les éléments

lui permettant de participer de manière pertinente à la phase 2.

Pour servir de base à la discussion, nous présentons les résultats de ce questionnaire (les quatre répondants sont notés R1 à R4 et l'analyse produite par le système, nommé Sys) à l'aide du tableau présenté dans la figure 12. Il présente les accords mutuels entre les différents juges sur la présence ou non des différentes compétences du PIAF pour chaque mission. Chaque ligne représente l'accord entre deux juges. Nous considérons que deux juges sont d'accord s'ils ont tous les deux identifié une même compétence sur une même mission ou au contraire s'ils ont jugé une compétence absente sur une même mission. Par exemple, le tableau indique un accord moyen de 0,5 entre R1 et R2 sur la compétence C1.3 car R1 a identifié C1.3 sur les missions 1, 2, 3, 5 et 6, alors que R2 l'a identifiée seulement sur les mission 5 et 6. Ils sont donc en accord sur les missions 4 (absence), 5 (présence) et 6 (présence) et en désaccord sur les missions 1, 2 et 3, soit un accord moyen de 0,5. Les colonnes grisées représentent les compétences reconnues par le système comme étant présentes dans au moins une mission du scénario. La colonne « Moyenne globale » représente l'accord moyen entre deux juges sur l'ensemble des compétences (toutes les colonnes), la colonne « Moyenne + » représente l'accord moyen sur les seules compétences identifiées comme actives par le système (colonnes grisées), et la colonne « Moyenne - » représente l'accord moyen sur les seules compétences identifiées comme absentes par le système (colonnes blanches). Dans ces trois dernières colonnes, les meilleurs accords sont mis en évidence sur fond vert.

		C1.1	C1.2	C1.3	C1.4	C1.5	C1.6	C1.7	C2.1	C2.2	C2.3	C2.4	C2.5	C2.6	C3.1	C3.2	C3.3	C3.4	C4.1	C4.2	C4.3	C5.1	C5.2	C6.1	C6.2	C6.3	C6.4	Moyenne globale	Moyenne +	Moyenne -
R1	R2	0	0	0,5	1	1	0,5	0,7	0,2	0,8	1	0,8	0,8	0,5	1	1	0,8	1	0,7	0,3	0,3	1	1	1	0	1	1	0,69	0,79	0,56
R1	R3	1	0	0,8	0	0,5	1	1	0	1	1	0,8	0,7	0,7	1	0,3	0,8	1	0,7	0,8	1	1	0,8	1	0	1	0,3	0,71	0,73	0,67
R1	R4	1	0,2	0,8	0	0,7	0,5	0,7	0,2	0,5	0,7	0,7	0,7	0,5	1	0,7	1	0,8	0,7	0,3	0,3	0,5	0,7	0,7	0,2	0,2	0,7	0,56	0,59	0,53
R1	Sys	0	0,5	0,2	1	1	0,3	0,3	0,7	1	1	1	0,5	0,5	1	0,8	0,8	1	1	1	1	0,8	0,8	1	1	1	0,8	0,78	0,82	0,71
R2	R1	0	0	0,5	1	1	0,5	0,7	0,2	0,8	1	0,8	0,8	0,5	1	1	0,8	1	0,7	0,3	0,3	1	1	1	0	1	1	0,69	0,79	0,56
R2	R3	0	1	0,3	0	0,5	0,5	0,7	0,8	0,8	1	1	0,8	0,8	1	0,3	1	1	1	0,5	0,3	1	0,8	1	1	1	0,3	0,72	0,86	0,53
R2	R4	0	0,8	0,3	0	0,7	1	1	1	0,7	0,7	0,8	0,8	0,7	1	0,7	0,8	0,8	1	0,7	0,7	0,5	0,7	0,7	0,8	0,2	0,7	0,68	0,73	0,61
R2	Sys	1	0,5	0,7	1	1	0,8	0,7	0,2	0,8	1	0,8	0,7	0,7	1	0,8	1	1	0,7	0,3	0,3	0,8	0,8	1	0	1	0,8	0,75	0,83	0,64
R3	R1	1	0	0,8	0	0,5	1	1	0	1	1	0,8	0,7	0,7	1	0,3	0,8	1	0,7	0,8	1	1	0,8	1	0	1	0,3	0,71	0,73	0,67
R3	R2	0	1	0,3	0	0,5	0,5	0,7	0,8	0,8	1	1	0,8	0,8	1	0,3	1	1	1	0,5	0,3	1	0,8	1	1	1	0,3	0,72	0,86	0,53
R3	R4	1	0,8	1	1	0,8	0,5	0,7	0,8	0,5	0,7	0,8	0,7	0,8	1	0,3	0,8	0,8	1	0,5	0,3	0,5	0,8	0,7	0,8	0,2	0,7	0,72	0,72	0,71
R3	Sys	0	0,5	0	0	0,5	0,3	0,3	0,3	1	1	0,8	0,8	0,8	1	0,2	1	1	0,7	0,8	1	0,8	0,7	1	0	1	0,5	0,62	0,76	0,44
R4	R1	1	0,2	0,8	0	0,7	0,5	0,7	0,2	0,5	0,7	0,7	0,7	0,5	1	0,7	1	0,8	0,7	0,3	0,3	0,5	0,7	0,7	0,2	0,2	0,7	0,56	0,59	0,53
R4	R2	0	0,8	0,3	0	0,7	1	1	1	0,7	0,7	0,8	0,8	0,7	1	0,7	0,8	0,8	1	0,7	0,7	0,5	0,7	0,7	0,8	0,2	0,7	0,68	0,73	0,61
R4	R3	1	0,8	1	1	0,8	0,5	0,7	0,8	0,5	0,7	0,8	0,7	0,8	1	0,3	0,8	0,8	1	0,5	0,3	0,5	0,8	0,7	0,8	0,2	0,7	0,72	0,72	0,71
R4	Sys	0	0,3	0	0	0,7	0,8	0,7	0,2	0,5	0,7	0,7	0,5	0,7	1	0,8	0,8	0,8	0,7	0,3	0,3	0,7	0,8	0,7	0,2	0,2	0,8	0,53	0,66	0,36
Sys	R1	0	0,5	0,2	1	1	0,3	0,3	0,7	1	1	1	0,5	0,5	1	0,8	0,8	1	1	1	1	0,8	0,8	1	1	1	0,8	0,78	0,82	0,71
Sys	R2	1	0,5	0,7	1	1	0,8	0,7	0,2	0,8	1	0,8	0,7	0,7	1	0,8	1	1	0,7	0,3	0,3	0,8	0,8	1	0	1	0,8	0,75	0,83	0,64
Sys	R3	0	0,5	0	0	0,5	0,3	0,3	0,3	1	1	0,8	0,8	0,8	1	0,2	1	1	0,7	0,8	1	0,8	0,7	1	0	1	0,5	0,62	0,76	0,44
Sys	R4	0	0,3	0	0	0,7	0,8	0,7	0,2	0,5	0,7	0,7	0,5	0,7	1	0,8	0,8	0,8	0,7	0,3	0,3	0,7	0,8	0,7	0,2	0,2	0,8	0,53	0,66	0,36
																										Moyenne		0,68	0,75	0,58

Figure 12 : Accord mutuel entre les différents juges (système inclus)

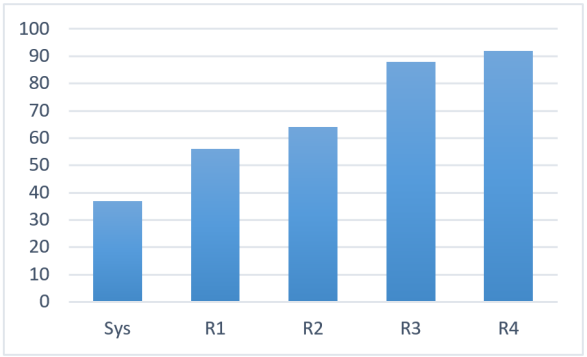


Figure 13 : Nombre de compétences identifiées par chaque juge sur les 6 missions du scénario

Nous remarquons (toujours dans la figure 12) que les répondants 1 et 2 sont plutôt en accord avec le système avec une moyenne globale de 0,78 pour R1 et 0,75 pour R2. Le répondant 3, lui, est plutôt en accord avec les autres juges humains. Le répondant 4 est plutôt en accord avec le répondant 3. Enfin nous remarquons que la moyenne des accords sur la présence des compétences (colonne « Moyenne + ») est plus importante (0,75) que sur l'absence des compétences (colonne « Moyenne - » ; 0,58). Pour comprendre cette tendance nous comparons dans la figure 13 le nombre de compétences identifiées par chaque juge et nous voyons que les juges humains (R1 à R4) ont eu tendance à identifier plus de compétences que le système, ou que le système a eu tendance à éliminer plus de compétences que les juges humains, ce qui explique un désaccord plus fort sur la colonne « Moyenne - ».

L'objectif annoncé aux répondants lors de ce premier échange n'était pas de chercher le consensus à l'issue de la discussion, mais de permettre à chacun d'exprimer son point de vue, de mettre en évidence le côté non homogène des analyses produites et d'identifier des causes de cette non-homogénéité. C'est sur ces éléments qualitatifs que nous revenons maintenant [6].

Par exemple, la compétence C1.1 (« Être capable de donner des noms à des objets, des actions et des séquences d'actions ») du PIAF n'est pas interprétée de la même manière entre les répondants. Une première interprétation faite par le répondant R1 est de considérer que dans le jeu, tous les blocs sont nommés et que, par conséquent, cette compétence est présente dans toutes les missions puisque les joueurs manipulent des blocs. À l'inverse, R2 indique « *Moi je l'avais compris à l'inverse, pour moi celle-ci j'avais mis qu'elle n'était jamais travaillée dans aucune mission parce que tel que j'ai compris l'intitulé de la compétence, c'est être capable pour l'apprenant de donner des noms à des choses, et ça on ne le fait jamais finalement* ». La règle intégrée dans le système d'analyse est construite selon la même compréhension de la compétence que ce dernier répondant, ce qui explique l'accord de 1 entre R2 et Sys pour la compétence C1.1 en opposition à l'accord de 0 entre les autres répondants et le système sur cette même compétence.

Certaines compétences n'ont tout simplement pas été comprises par certains répondants, comme la compétence C1.2 (« Différencier objet et action, et actions atomiques et non-atomiques », la définition d'action atomique n'étant pas claire pour eux, R3 demande « *ça veut dire quoi action atomique ?* »). Elle est donc absente des compétences identifiées dans les 6 missions, alors même que cette compétence fait partie des compétences identifiées par le système. R2 indique « *Moi je l'ai compris comme ça, de comprendre le fait que chaque bloc finalement est lié à une action, à un truc particulier et de savoir finalement les différencier pour utiliser les bons blocs en fonction de ce qu'on veut faire, donc pour moi c'est vrai que s'en était une qui elle à l'inverse de la une était présente souvent* ». L'interprétation de la compétence pour ce répondant est donc différente de la règle définie pour le système qui cherche à identifier la présence d'actions non-atomiques « Turn back » équivalentes à deux actions atomiques « Turn Left/Right » ou la présence d'actions et de capteurs donnant des informations sur l'état du monde.

Lors des réponses données à la première partie du questionnaire (description des compétences sans s'appuyer sur le PIAF), certaines compétences sont identifiées sans pouvoir être décrites dans le PIAF. Ainsi, le répondant R3 identifie la compétence suivante « *Être capable de se repérer dans l'espace, parce que là faut être capable à l'avance quand même de voir où le bonhomme va se déplacer, savoir s'il doit tourner à gauche pour lui mais du coup ça a rien avoir avec l'informatique* », R2 relance « *Non mais c'est un truc auquel on se heurte, moi je m'y suis heurté en début d'année avec mes élèves où on a fait du turtle [...] la représentation dans l'espace c'était une catastrophe [...] finalement c'est une complexité que j'avais pas anticipée* ». On voit donc ici que même si cette compétence n'est pas jugée

6. Toutes les citations sont issues du verbatim de la séance d'expérimentation.

comme relevant de l'informatique, c'est une compétence qui est travaillée à l'occasion de mises en situation provoquées par l'intention de travailler des compétences en informatique. Sur un tout autre sujet, le répondant R5 identifie une autre compétence : « *Savoir découvrir le fonctionnement de la caméra parce qu'on n'a pas toujours la bonne vue pour savoir ce qu'il faut faire [...] Si on voulait vraiment faire le truc du premier coup juste, il fallait manipuler la caméra pour savoir où il fallait aller* ». Dans ce cas, cette compétence est jugée comme relevant de l'informatique, car elle consiste à prendre des informations sur la situation avant de commencer à programmer. Cette compétence est jugée absente du PIAF et n'est d'ailleurs pas intégrée aux règles présentées dans l'Annexe 8. Pour autant, cette remarque donne une piste pour ajouter une nouvelle règle qui pourrait décrire une facette de la compétence C1.3 (« Identifier les paramètres d'entrée d'une séquence d'actions »). Si une mission donne une vue partielle de la situation, elle invite le joueur à déplacer la caméra pour en comprendre tous les éléments et ainsi identifier tous les paramètres du problème avant de construire sa solution.

Nous voyons donc ici, au travers de ces quelques exemples, que l'interprétation d'un référentiel de compétences pour décrire une tâche donnée est une activité complexe sujette à l'expertise et l'interprétation de chaque enseignant. Pour avoir une compréhension plus précise des éventuels liens entre l'expertise des enseignants, l'analyse qu'ils produisent et l'analyse du système, il conviendrait de poursuivre des expérimentations similaires avec un panel de participants plus important.

7.2. Évaluation de l'utilité perçue des fonctionnalités liées à l'analyseur automatique

Suite aux échanges sur les résultats du questionnaire, un temps de manipulation libre de l'éditeur de mission a été proposé aux participants. L'objectif de ce temps et des échanges qui ont suivi, était d'évaluer l'utilité perçue des fonctionnalités liées à l'analyseur automatique. Pour aider au recueil des retours des apprenants, nous avons proposé un questionnaire composé de sept questions. Les réponses étaient données à l'aide d'une valeur dans une échelle de Likert à 5 niveaux allant de « 1 - pas du tout » à « 5 - tout à fait ». La progression dans le questionnaire était contrôlée par les animateurs de la séance, les répondants passaient à la question suivante une fois la discussion sur la question courante terminée. Les questions 1, 2 et 3 portaient sur l'évaluation des fonctionnalités déjà intégrées au jeu et les questions 4, 5, 6 et 7 portaient sur des propositions de nouvelles fonctionnalités potentielles. Nous présentons maintenant les réponses à ces différentes questions accompagnées de quelques éléments qualitatifs tirés du verbatim de la séance.

7.2.1. Évaluation de l'utilité perçue des fonctionnalités déjà intégrées

Q1 : Pour vous, la visualisation des compétences au niveau d'un scénario vous semble utile. Tous les répondants (5/5) ont voté 5 (« tout à fait ») :

- R1 : « *Moi je trouve très bien parce que ça permet à l'enseignant de voir quelles compétences il peut directement viser pour l'élève* » ;
- R5 : « *Il peut aussi faire une progression pour vérifier que ce n'est pas toujours les mêmes compétences travaillées* » ;
- R2 : « *La seule chose qui m'a manqué là-dessus, c'est qu'on a l'agrégation complète de tout ce qui a dans le scénario et on n'a pas la séquence, ça aurait été intéressant d'avoir, alors je ne sais pas sous quelle forme le présenter parce que ça pourrait vite être lourd, peut-être sous forme graphique si c'est possible, de voir dans mon scénario la progression des compétences, ça, ça serait un vrai vrai plus* ».

La remarque du répondant R2 est particulièrement intéressante car elle anticipe directement la question 7 du questionnaire.

Q2 : Pour vous, pouvoir filtrer les missions par compétence vous semble utile. Tous les répondants (5/5) ont voté 5 (« tout à fait ») :

— R2 : « *Quand on veut utiliser quelque chose ou [...] si on veut s'inspirer de quelque chose, pouvoir filtrer c'est hyper intéressant* » ;

— R5 : « *Si on repère que quelque chose n'a pas été acquis par la plupart des élèves, on peut proposer des exercices complémentaires liés à ça* », R1 surenchérit « *exactement* ».

On voit donc ici que la fonction de filtre semble utile pour : (1) rechercher des missions existantes qui pourront servir d'inspiration à la création de nouvelles missions ; et (2) sélectionner un ensemble de missions complémentaires à proposer aux élèves pour les faire travailler une compétence précise jugée non encore acquise. On notera dans la remarque de R5 que l'adaptation envisagée est pensée au niveau du groupe classe (« *[...] la plupart des élèves [...]* ») et non pas à un niveau individuel.

Q3 : Pour vous, l'analyse automatique des compétences d'une mission vous semble utile. Trois répondants ont voté 4 et deux ont voté 5 (« tout à fait ») :

— R2 : « *C'est cool* », R1 surenchérit « *Je pense que c'est moins pertinent, mais c'est cool [rires]* » ;

— R5 : « *Moi je mets 4 parce que ça dépend vraiment de comment est déterminé tout ça, par exemple certaines choses je les trouve détectées de manière assez généreuse* », R2 complète « *Oui, c'est super utile, mais il faut probablement affiner la manière de les détecter* ».

La remarque de R5 est intéressante car il juge le système « généreux ». En d'autres termes, il considère que certaines compétences du PIAF ont été identifiées par le système et ne le devraient peut-être pas. Dans le même sens, R2 suggère de réviser les règles afin de les rendre plus précises. Pour autant, nous avons vu dans la figure 13 que le système est le plus strict des juges avec 37 occurrences de compétences identifiées contre 64 pour R2 et plus de 90 pour R4. Les règles intégrées dans le système d'analyse automatique sont donc plus restrictives que les règles implicites de chaque participant. Indépendamment donc de la réelle finesse d'une analyse, deux questions sous-jacentes sont mises en évidence par les commentaires des répondants.

La première porte sur la confiance portée sur l'analyse et la capacité à expliquer les raisons qui mènent le système à produire un résultat. R2 indique à ce sujet « *Si l'analyse on peut lui faire confiance, parce que les règles sont bien posées, là ça a une vraie valeur, parce que c'est vrai qu'on n'a pas tous les critères en tête quand on crée une mission [...] je pense que c'est une vraie valeur ajoutée pour autant qu'on puisse lui accorder une confiance raisonnable* ».

La seconde porte sur le positionnement des compétences entre elles au sein d'une même mission. R5 propose « *On pourrait avoir des référentiels qui pourraient être structurés autrement et pour lesquels ce n'est pas juste vrai ou faux [présence/absence de la compétence] mais où il y aurait un niveau pour indiquer si elle est plus ou moins présente [...] Indiquer la compétence qui est vraiment centrale à la résolution du problème et une qui est vraiment annexe, et ça on ne le voit pas [...] elles sont toutes à plat* ».

Ces trois premières questions portaient donc sur les fonctionnalités actuellement intégrées dans le jeu SPY. Les évaluations données, ainsi que les remarques formulées, montrent que ces fonctionnalités sont unanimement perçues comme utiles par les cinq répondants. La dernière proposition est légèrement moins plébiscitée que les deux premières mais des pistes d'amélioration intéressantes sont dégagées (explication des règles et positionnement

des compétences entre elles).

7.2.2. Évaluation de l'utilité *a priori* de nouvelles fonctionnalités

Les quatre questions suivantes portent maintenant sur des pistes de développements à venir dans SPY.

Q4 : Actuellement, l'analyseur de compétences par mission est disponible dans l'éditeur de scénario. Disposer également de l'analyseur de compétences dans l'éditeur de mission vous semblerait utile. Quatre répondants ont voté 4 et un a voté 5 (« tout à fait »).

R2 indique à ce sujet « *À chaque fois qu'on ajoute ou qu'on enlève un truc qu'il actualise [les compétences], ouais ça, ça serait un vrai vrai plus* ». Une discussion s'engage sur l'usage a priori d'un tel dispositif et la nécessité pour l'enseignant de garder le contrôle. L'analyseur devrait rester un outil d'aide à la validation de ce qui est pensé par l'enseignant pour confirmer l'intention initiale ou faire apparaître d'éventuels écarts et inviter l'enseignant à retravailler sa mission. R2 synthétise ainsi « *Pour ne pas tomber dans un biais où on va aller un peu au hasard, si on sait qu'on veut faire un scénario qui développe telle compétence et qu'on a moyen de se rafraîchir les idées en affichant les règles, pour cette compétence [...] je vais pouvoir imaginer mon scénario en sachant finalement ce que j'inclus dedans* ».

Q5 : Actuellement, le résultat de l'analyse des missions est présenté comme une liste de compétences. Vous semblerait-il utile de pouvoir disposer d'informations expliquant pourquoi telles ou telles compétences ont été identifiées. Deux répondants ont voté 4 et trois ont voté 5 (« tout à fait »).

Cette question fait directement écho aux réflexions qui ont émergé pour la question 3 sur la confiance portée à l'analyseur automatique et aux règles sous-jacentes. Ces résultats confirment la pertinence d'engager des travaux sur l'explicabilité à l'enseignant des décisions du système.

Q6 : Vous semblerait-il utile dans l'éditeur de mission de disposer d'une fonctionnalité capable de suggérer des modifications à apporter à la mission pour pouvoir travailler une compétence visée. Deux répondants ont voté 2, un répondant a voté 3 et les deux derniers ont voté 5.

Cette proposition est la moins bien notée par les répondants. Elle est jugée moins prioritaire que les propositions précédentes.

Q7 : Actuellement, l'analyse des compétences d'un scénario est une simple agrégation des compétences de chaque mission. Disposer d'une analyse plus fine montrant les compétences utilisées dans chaque mission vous semblerait-elle utile. Associée à cette question, l'illustration de la figure 10 était présentée aux répondants. Tous les répondants (5/5) ont voté 5 (« tout à fait »).

L'évaluation de cette proposition est cohérente avec les retours formulés par le répondant R2 dans la question Q1.

7.2.3. Retour sur les discussions informelles

La dernière étape de notre protocole laissait la place à un temps de discussion informel autour du jeu. Ce temps-là a permis de mettre en évidence des limites et des potentialités d'usages de la ressource. Nous en reprenons quelques éléments marquants dans les lignes suivantes.

Concernant les limites, la critique principale a porté sur des choix de développement des interfaces du jeu, en relevant des améliorations possibles d'interactions. R5 note « *Dans les*

moins c'est l'UI, il y a plein d'éléments d'UI qui posent problèmes, les choses liées fonctionnellement ne sont pas toujours liées visuellement ». Ici R5 fait référence à la zone d'édition apparaissant à la gauche de l'écran de jeu (voir Figure 1) et au panneau d'exécution apparaissant sur la droite de l'écran de jeu (voir Figure 3). C'est deux interfaces sont, pour lui, liées fonctionnellement mais ne le sont pas visuellement car disposées à l'opposée l'une de l'autre. Pour autant ce choix n'était pas innocent. Il permet d'une part de différencier le programme édité, du programme chargé en mémoire par le robot (le programme exécuté) ; ces deux zones d'affichage distinctes ont été pensées dans ce sens. D'autre part, différencier la zone d'édition du contexte d'exécution permet d'allouer autant de contextes d'exécution que de robots dans la mission. Ainsi, comme nous le montrons dans la figure 3, un même programme (défini dans une et une seule zone d'édition) peut être exécuté par deux robots différents où chaque robot a son propre programme chargé en mémoire et l'exécute indépendamment. Ces choix sont, encore une fois, liés aux savoirs à enseigner. Pour autant, dans cet exemple relevé par R5, ces choix ne sont pas lisibles par l'enseignant et méritent donc d'être retravaillés.

Concernant les potentialités, nous en avons identifié de deux ordres : l'exploitation des fonctionnalités en place et le détournement de fonctionnalités à des fins pédagogiques.

Concernant l'exploitation des fonctionnalités en place, la diversité des objets de jeu interactifs est notée. R5 indique « *Clairement par rapport à un truc comme Blockly, il y a déjà des plus comme le fait qu'on peut programmer différents objets qui interagissent voire de manipuler des terminaux [pour ouvrir des portes] ou des trucs comme ça, là il y a de quoi faire plus donc je trouve ça vraiment intéressant* ». R2 complète en insistant sur la dimension ludique de certains scénarios : « *Moi, j'avais regardé le scénario Infiltration qui a une vingtaine de missions qui sont liées par une histoire finalement avec un vrai récit et je pense que, en créant des scénarios comme ça, il y a une vraie possibilité [...] comme tu dis, il y a beaucoup d'interactions possibles dès qu'on met deux robots, dès qu'on rajoute des drones, des objets manipulables, des pièces à ramasser, des terminaux, etc. Il y a beaucoup de choses finalement faisables [...] et le gros plus de cette plateforme c'est de jouer là-dessus, la différentiation* ».

Concernant les détournements à des fins pédagogiques, une première idée est formulée par R2 : « *Moi, il y a un truc que je me suis tout de suite dit quand j'ai vu des scénarios avec deux robots et deux scripts différents, je me suis dit là ce qui serait génial c'est la coopération, deux élèves se connectent sur la même session et chacun programme son robot et ils doivent discuter ensemble [...] pour se dire, ok, attend, moi mon robot il doit faire trois étapes pour ouvrir la porte donc toi il faut que tu attendes trois fois. Dès que j'ai vu deux robots avec deux scripts distincts je me suis dit mais là il y a un potentiel avec les élèves qui est juste incroyable* ». Une discussion s'en est suivie sur la complexité de développer une telle fonctionnalité. Face à cette difficulté technique, R2 propose alors une solution originale « *après je pense qu'on peut y arriver de manière un peu détournée en mettant deux élèves sur le même ordinateur et que chacun programme son robot* ».

Un second détournement concerne l'éditeur de mission par les élèves. Cet éditeur, initialement conçu pour les enseignants afin de leur permettre d'ajouter des missions originales à leur scénario, pourrait être manipulé par les élèves afin qu'ils créent leurs propres missions, tentent de les résoudre et les proposent à leurs camarades de classe. Ce type d'activité pourrait être dirigé en demandant aux élèves de créer une mission qui nécessite de mobiliser une compétence précise. Dans ce cas, la tâche demandée à l'élève serait au-delà de la simple maîtrise de la compétence mais viserait à comprendre ce qu'est une situation qui nécessite la manipulation de la compétence. Le répondant R2 synthétise ces échanges en scénarisant ce détournement de l'usage de l'éditeur de mission sur les différents niveaux scolaires :

« *L'usage classique de jouer des missions au niveau du cycle [collège français], et au niveau du lycée aller sur de la conception de missions là ça pourrait être plus intéressant effectivement* ».

Enfin, dans la continuité des discussions portant sur le détournement de l'éditeur de mission, un dernier détournement est évoqué sur l'exploitation des fichiers XML décrivant les missions de jeu. Présenter ces fichiers aux élèves pourrait permettre d'aborder le thème des langages de balisage pour décrire une ressource (dans notre cas, une mission de jeu) et comprendre que l'éditeur de mission n'est qu'une interface permettant de créer ces fichiers qui peuvent être ouverts et modifiés avec un simple éditeur de texte. Là encore, le répondant R2 synthétise les échanges : « *Ça, les élèves adorent, d'aller taper du code finalement et de voir le résultat dans un produit fini, qui est là qui est bien léché je veux dire et de voir que simplement en allant modifier quelques trucs dans un fichier texte comme ça et bien on a un impact direct sur un jeu, ça je pense que ça devrait plaire* ».

8. 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 le formalisme générique GTC basé sur la structuration du langage de balisage XML pour décrire des compétences par combinaison de fonctionnalités de jeu (question de recherche 1). Nous avons ensuite exploité GTC pour proposer une caractérisation des 21 compétences PIAF présentes dans SPY, des 5 niveaux de la compétence 3.4 du CRCN et des 35 fonctionnalités ludiques de SPY. L'ensemble des résultats est présenté dans Muratet (2023b). Ainsi, nous avons montré que GTC 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. Dans ce travail de recherche, les règles spécifiques à SPY ont été conçues par les chercheurs impliqués sur le projet. Le résultat n'est donc pas l'ensemble des règles proprement dit mais le formalisme générique permettant de décrire ces règles : GTC. Un travail reste à engager pour réviser les règles proposées pour SPY et les soumettre à un processus collectif de validation impliquant des enseignants. Ceci devrait également contribuer à mieux expliquer les décisions du système et donc à augmenter la confiance accordée par les enseignants aux informations fournies par celui-ci.

Aussi, nous avons montré comment ces caractérisations peuvent être exploitées pour construire des indicateurs sur des missions existantes et fournir aux enseignants et concepteurs des informations micros sur les compétences en jeu dans une mission, ou des informations macros sur l'évolution de la complexité d'un scénario de jeu (question de recherche 2). Ces résultats ont été intégrés au jeu SPY à travers trois fonctionnalités : pour l'analyse d'une mission dans l'éditeur de scénario ; pour l'analyse d'un scénario par l'agrégation des compétences de chaque mission composant le scénario ; et par le développement d'une fonction de filtrage permettant de filtrer les missions par compétences dans l'éditeur de scénario.

Ces trois fonctionnalités ont été étudiées lors d'une expérimentation menée avec un groupe d'enseignants en formation à l'Université de Genève. Nous avons pu montrer que l'appropriation d'un référentiel de compétences tel que le PIAF reste complexe, même pour des enseignants formés à la discipline informatique. L'accord entre les différents juges pour identifier les compétences mobilisées dans une sélection de missions du jeu SPY reste hétérogène. Les trois fonctionnalités intégrées au jeu ont été jugées très utiles par les participants à l'expérimentation. L'expérimentation a permis également d'identifier des détournements possibles de certaines fonctionnalités du jeu à des fins pédagogiques, comme la manipulation de l'éditeur de mission par les élèves ou la présentation aux élèves des fichiers XML

décrivant les missions.

Ces premiers retours sont encourageants. Ils permettent d'identifier plusieurs pistes de recherche en rapport avec la contribution de cet article :

— quel passage de l'utilité perçue *a priori* à une utilité vérifiée en contexte réel d'utilisation ?
— quels effets du profil des utilisateurs enquêtés, et en particulier de leurs compétences en informatique ?
— comment favoriser la confiance des utilisateurs dans les informations données par le système notamment en travaillant l'explicabilité des décisions prises et des règles sous-jacentes ?

Plus largement se pose la question de la transférabilité de la démarche d'analyse des compétences à partir des fonctionnalités de jeu et notamment des règles proposées dans d'autres environnements dédiés à l'apprentissage de la pensée informatique.

Enfin, SPY s'inscrit dans une démarche de partage à la fois pour son usage (*free to play*), son code source (*open source*) et ses traces d'interaction (*open data*) (Muratet, 2023c). Les scénarios pré-construits sont utilisables en l'état par des enseignants qui souhaiteraient faire travailler leurs élèves sur les compétences de la pensée informatique. Les deux éditeurs inclus (éditeur de scénario et éditeur de mission) permettent aux enseignants de personnaliser le jeu, voire de créer leurs propres missions et scénarios. Nous avons vu également que l'éditeur de mission pourrait être avantageusement détourné pour le transformer en une ressource pédagogique à destination des élèves en les invitant à créer leurs propres missions. Concernant les chercheurs, l'ouverture de son code source et de ses données ouvre la voie au développement de nouvelles fonctionnalités afin de tester leurs hypothèses en s'appuyant sur des observations en contexte de laboratoire ou écologique et en exploitant les données ouvertes du jeu.

Remerciements

Cette recherche a été menée dans le cadre du projet ANR-18-CE38-0008 (IECARE) financé par l'Agence Nationale de la Recherche (ANR).

Références

Alayrangues, S., Peltier, S., et Signac, L. (2017). Informatique débranchée : construire sa pensée informatique sans ordinateur. *Actes du colloque Mathématiques en Cycle 3 IREM de Poitiers*, (p. 216-226). https://hal.science/hal-01868132

Balacheff, N. (1994). La transposition informatique, un nouveau problème pour la didactique. Dans M. Artigue, R. Gras, C. Laborde et P. Tavignot (dir.), *Actes du colloque "Vingt ans de didactique des mathématiques en France"* (p. 364-370). La Pensée Sauvage. https://telearn.archives-ouvertes.fr/hal-00190646

Baron, G.-L., Drot-Delange, B., Grandbastien, M., et Tort, F. (2014). Computer science education in french secondary schools : historical and didactical perspectives. *ACM Trans. Comput. Educ.*, *14*(2). https://doi.org/10.1145/2602486

Bonnat, C., Sanchez, E., Paukovics, E., et Kramar, N. (2023). Didactic transposition and learning game design. Towards a ludicization model for school visits in museums [Series Title : Transdisciplinary Perspectives in Educational Research]. Dans F. Ligozat, K. Klette et J. Almqvist (dir.), *Didactics in a Changing World* (p. 199-215, T. 6). Springer International Publishing. https://doi.org/10.1007/978-3-031-20810-2_12

Brousseau, G. (1986). Fondements et méthodes de la didactique des mathématiques. *Recherches en didactique des mathématiques*, *7*(2), 33-115.

Brousseau, G. (1990). Le contrat didactique : le milieu. *Recherches en didactique des mathématiques*, *9*(3), 309-336. https://hal.science/hal-00686012/

Bruillard, É. (1997). *Les machines à enseigner*. Editions Hermès.

Carron, T., Muratet, M., Marne, B., et Yessad, A. (2017). Analyser et représenter la progression de la difficulté d'un jeu sérieux du point de vue ludique et pédagogique. *EIAH 2017*. https://hal.science/hal-01515753

Komis, V., et Misirli, A. (2011). Robotique pédagogique et concepts préliminaires de la programmation à l'école maternelle : une étude de cas basée sur le jouet programmable Bee-Bot. Dans G.-L. Baron, É. Bruillard et V. Komis (dir.), *Actes du colloque international DIDAPRO 4 - Dida&Stic* (p. 271-281). New Technologies Editions. https://edutice.hal.science/edutice-00676143

Kradolfer, S., Dubois, S., Riedo, F., Mondada, F., et Fassa, F. (2014). A sociological contribution to understanding the use of robots in schools : the Thymio robot. Dans M. Beetz, B. Johnston et M.-A. Williams (dir.), *Actes de la Conférence Internationle Social Robotics (ICSR)* (p. 217-228). Springer International Publishing.

Lindberg, R. S. N., Laine, T. H., et 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*(4), 1979-1995. https://doi.org/https://doi.org/10.1111/bjet.12685

Margolinas, C. (1992). Eléments pour l'analyse du rôle du maître : les phases de conclusion. *Recherches en didactique des mathématiques*, *12*(1), 113-158.

Mariotti, M. A., et Maracci, M. (2012). Resources for the teacher from a semiotic mediation perspective. Dans G. Gueudet, B. Pepin et L. Trouche (dir.), *From text to 'lived' resources : mathematics curriculum materials and teacher development* (p. 59-75). Springer Netherlands.

Marne, B., Wisdom, J., Huynh-Kim-Bang, B., et Labat, J.-M. (2012). The six facets of serious game design : a methodology enhanced by our design pattern library. *Actes de la 7ème European Conference on Technology Enhanced Learning (EC-TEL 2012)*, (p. 208-221). https://doi.org/10.1007/978-3-642-33263-0_17

Miljanovic, M. A., et Bradbury, J. S. (2018). A review of serious games for programming. Dans S. Göbel, A. Garcia-Agundez, T. Tregel, M. Ma, J. Baalsrud Hauge, M. Oliveira, T. Marsh et P. Caserman (dir.), *Actes de la conférence Serious Games* (p. 204-216). Springer International Publishing.

Muratet, M. (2023a). Complete and commented level model in XML format [[Online ; accessed June 19, 2023]]. https://github.com/Mocahteam/SPY/blob/master/Doc/LevelModel.xml

Muratet, M. (2023b). Description of PIAF skills using the SPY game features [[Online ; accessed June 19, 2023]]. https://github.com/Mocahteam/SPY/blob/master/Assets/StreamingAssets/Competencies/competenciesReferential.json

Muratet, M. (2023c). SPY : un jeu sérieux partagé pour étudier l'apprentissage de la pensée informatique. Dans J. Broisin, C. Declercq, C. Fluckiger, Y. Parmentier, Y. Peter et Y. Secq (dir.), *Atelier "Apprendre la Pensée Informatique de la Maternelle à l'Université", dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH)* (p. 33-40). https://hal.science/hal-04144205

Parmentier, Y., Reuter, R., Higuet, S., Kataja, L., Kreis, Y., Duflot-Kremer, M., Laduron, C., Meyers, C., Busana, G., Weinberger, A., et Denis, B. (2020). PIAF : developing computational and algorithmic thinking in fundamental education. *EdMedia+ Inno-*

vate Learning, Association for the Advancement of Computing in Education (AACE), 315-322.

PIAF Project. (2021). The complete list of the 26 PIAF skills and their description [[Online ; accessed June 19, 2023]]. https://piaf.loria.fr/wp-content/uploads/2021/09/PIAF-Referential-of-Competencies-Description-and-Examples.pdf

Poirson, B. (2020). *Appuis et obstacles à l'apprentissage de la pensée informatique à l'école primaire* [mém. de mast., INSPÉ de Franche-Comté (Vesoul) ; Université de Franche-Comté (UFC)]. https://univ-fcomte.hal.science/hal-02958874

Rabardel, P. (1995). *Les hommes et les technologies. Approche cognitive des instruments contemporains.* Armand Colin.

Rabardel, P. (1999). Eléments pour une approche instrumentale en didactique des mathématiques. *Actes de la 10e école d'été de didactique des mathématiques*, *18 (21)*, 203-213.

Saddoug, H., Rahimian, A., Marne, B., Muratet, M., Sehaba, K., et Jolivet, S. (2022). Review of the adaptability of a set of learning games meant for teaching computational thinking or programming in France. *Special Session on Gamification on Computer Programming Learning*, *1*, 562-569. https://doi.org/10.5220/0011126400003182

Sigayret, K., Tricot, A., et Blanc, N. (2021). Pensée informatique et activités de programmation : quels outils pour enseigner et évaluer ? Dans J. Broisin, C. Declercq, C. Fluckiger, Y. Parmentier, Y. Peter et Y. Secq (dir.), *Actes de l'atelier "Apprendre la Pensée Informatique de la Maternelle à l'Université", dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH)* (p. 68-75). https://hal.science/hal-03241689

Spach, M. (2019). Activités robotiques à l'école : approches de pratiques d'enseignement et effets sur les apprentissages : *Recherches en didactiques*, *N° 28*(2), 68-87. https://doi.org/10.3917/rdid.028.0068

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35.

ANNEXES

RÉSUMÉ DES COMPÉTENCES ENCODÉES

Nous présentons dans cette annexe un résumé des compétences du PIAF encodées. Nous rappelons pour chaque compétence, son intitulé, une mise en contexte de cette compétence dans SPY, ainsi que la description de la compétence à l'aide de GTC sous une forme encadrée. Pour simplifier la description des compétences dans cette annexe, le tableau 5 répertorie les fonctions récurrentes utilisées.

C1.1 Nommer des objets et séquence d'actions : si une mission contient au moins une zone d'édition nommable.

$$C1.1 := ZEN() \geq 1$$

C1.2 Différencier objet et action, et actions atomiques et non-atomiques : si une mission contient une action non-atomique (l'action « Turn back » (IBQ_1)) qui peut être décomposée en deux actions atomiques (les actions « Turn left » (IBQ_2) ou « Turn right » (IBQ_3)) ou si une mission contient des actions (IEQ_1) et des capteurs (IEQ_2) qui illustrent les concepts d'expressions et d'instructions.

$$C1.2 := (IBQ_1(TurnBack, 1)\ \&\&\ (IBQ_2(TurnLeft, 2)\ ||\ IBQ_3(TurnRight, 2)))\ ||(IEQ_1(\{AllActions\},\ 1)\ \&\&\ IEQ_2(\{AllCaptors\}, 1))$$

C1.3 Identifier les paramètres d'entrée d'une séquence d'actions :

Tableau 5 : fonctions récurrentes

Fonction	Description
$ZEN()$	nombre de **Z**one d'**E**dition **N**ommable $\rightarrow Card(script\ editMode = 2)$
$ZEC()$	nombre de **Z**one d'**E**dition **C**orrecte $\rightarrow Card(script\ type = 0)$
$ZENO()$	nombre de **Z**one d'**E**dition **N**on **O**ptimale $\rightarrow Card(script\ type = 1)$
$ZEB()$	nombre de **Z**one d'**E**dition **B**oguée $\rightarrow Card(script\ type = 2)$.
$IBQ(B, Q)$	vrai si la mission **I**nclut au moins un **B**loc B en **Q**uantité Q, faux sinon $\rightarrow Card((blockLimit\ blockType = B) \cap (blockLimit\ limit \geq Q)) \geq 1$
$IEQ(S, Q)$	vrai si la mission **I**nclut au moins un item de l'**E**nsemble S en **Q**uantité Q, faux sinon $\rightarrow Card((blockLimit\ blockType\ isIncludedIn\ S) \cap (blockLimit\ limit \geq Q)) \geq 1$
$DD()$	vrai si le glisser/déposer (**D**rag&**D**rop) est activé, faux sinon $\rightarrow Card(dragdropDisabled) = 0$

Aucune mécanique de jeu pour cette compétence.

C1.4 Décrire le résultat d'une séquence d'actions :

Aucune mécanique de jeu pour cette compétence.

C1.5 Prédire le résultat d'une séquence d'actions : si une mission contient une sentinelle avec un programme défini.

$C1.5 := Card(script\ outputLine\ sameValue\ guard\ inputLine \cap script\ hasChild) \geq 1$

C1.6 Utiliser des objets dont la valeur peut changer : si une mission contient une structure de contrôle conditionnelle (IEQ_1) et des capteurs (IEQ_2) dont la valeur varie en fonction du contexte ou si une mission permet de contrôler (IBQ) l'état d'une porte ($R1$).

$R1 \rightarrow Card(door\ slotId\ sameValue\ slot\ slotId) \geq 1$
$C1.6 := (IEQ_1(\{While, IfThen, IfElse\}, 1)\ \&\&\ IEQ_2(\{AllCaptors\}, 1))\ ||$
$(IBQ(Activate, 1)\ \&\&\ R1)$

C1.7 Reconnaître, parmi des objets et séquences d'actions connus, lesquels peuvent être utilisés pour atteindre un nouvel objectif : si une mission propose seulement des zones d'édition nommables ($R1$) dont au moins une contient une solution pré-construite correcte ($R2$) que le joueur doit reconnaître et ne peut modifier (DD).

$R1 \rightarrow ZEN() = Card(script)$
$R2 \rightarrow Card((script\ type = 0) \cap (script\ hasChild)) \geq 1$
$C1.7 := R1\ \&\&\ R2\ \&\&\ !DD()$

C2.1 Ordonner une séquence d'actions pour atteindre un objectif : si une mission permet au joueur de combiner des actions (DD) en quantité limitée ($R1$) et avec au moins une action disponible dans l'inventaire (IEQ).

$R1 \rightarrow Card\ (blockLimit\ limit = -1) = 0$
$C2.1 := DD()\ \&\&\ R1\ \&\&\ IEQ(\{AllActions\}, 1)$

C2.2 Compléter une séquence d'actions pour atteindre un objectif simple : si une mission permet au joueur de combiner (DD) seulement des actions (IEQ_1) sans structures de contrôle (IEQ_2) et en fournissant seulement des zones d'édition pré-construites mais boguées ($R1$) que le joueur doit compléter.

$R1 \rightarrow Card((script\ type = 2) \cap (script\ hasChild)) = Card(script)$
$C2.2 := DD()\ \&\&\ IEQ_1(\{AllActions\}, 1)\ \&\&\ !IEQ_2(\{ControlList\}, 1)\ \&\&\ R1$

C2.3 Créer une séquence d'actions pour atteindre un objectif simple : si une mission permet au joueur de combiner (DD) seulement des actions (IEQ_1) sans structures de contrôle (IEQ_2).

$C2.3 := DD()\ \&\&\ IEQ_1(\ \{AllActions\}, 1)\ \&\&\ !IEQ_2(\{ControlList\}, 1)$

C2.4 Créer une séquence d'actions pour atteindre un objectif complexe : si une mission permet au joueur de combiner (DD) des actions (IEQ_1) et des structures de contrôle (IEQ_2).

$C2.4 := DD()\ \&\&\ IEQ_1(\ \{AllActions\}, 1)\ \&\&\ IEQ_2(\{ControlList\}, 1)$

C2.5 Combiner des séquences d'actions pour atteindre un objectif : Aucune mécanique de jeu pour cette compétence.

Aucune mécanique de jeu pour cette compétence.

C2.6 Décomposer des objectifs en sous-objectifs plus simples : si une mission ne limite pas le nombre d'exécution à 1.

$C2.6 := Card(executionLimit\ amount = 1) = 0$

C3.1 Répéter une séquence d'actions un nombre donné de fois : si une mission permet au joueur de combiner (DD) des boucles « Répéter n fois » (IBQ).

$C3.1 := DD()\ \&\&\ IBQ(ForLoop, 1)$

C3.2 Répéter une séquence d'actions jusqu'à ce qu'un objectif soit atteint : si une mission permet au joueur de combiner (DD) des boucles « Tant que » (IBQ) avec des capteurs (IEQ_1) et sans opérateurs (IEQ_2).

$C3.2 := DD()\ \&\&\ IBQ(WhileLoop, 1)\ \&\&\ IEQ_1(\{AllCaptors\}, 1)\ \&\&$
$!IEQ_2(\{OperatorList\},\ 1)$

C3.3 Intégrer une condition simple dans une séquence d'actions : si une mission permet au joueur de combiner (DD) la structure de contrôle « Si Alors » (IEQ_1) avec des capteurs (IEQ_2) et sans opérateurs (IEQ_3).

$C3.3 := DD()\ \&\&\ IEQ_1(\{IfThen, IfElse\}, 1)\ \&\&$
$IEQ_2(\{AllCaptors\}, 1)\ \&\&\ !IEQ_3(\{OperatorList\},\ 1)$

C3.4 Intégrer une condition complexe dans une séquence d'actions : si une mission permet au joueur de combiner (DD) des structures de contrôle conditionnelles (IEQ_1) avec des capteurs (IEQ_2) et des opérateurs (IEQ_3).

$C3.3 := DD()\ \&\&\ IEQ_1(\{While, IfThen,\ IfElse\},\ 1)\ \&\&$
$IEQ_2(\{AllCaptors\}, 1)\ \&\&\ IEQ_3(\ \{OperatorList\}, 1)$

C4.1 Comparer deux objets selon un critère donné : Aucune mécanique de jeu pour cette compétence.

Aucune mécanique de jeu pour cette compétence.

C4.2 Comparer deux séquences d'actions selon un critère donné : si une mission ne contient que des zones d'édition pré-construites nommables ($R1$) avec au moins une solution correcte et d'autres solutions non-optimales ($R2$) que le joueur ne peut modifier (DD).

$R1 \rightarrow ZEN() = Card(script)$
$R2 \rightarrow ZEC() \geq 1\ \&\&\ (ZEC() + ZENO()) = Card(script)$
$C4.2 := R1\ \&\&\ R2\ \&\&\ !DD()$

C4.3 Améliorer une séquence d'actions par rapport à un critère donné : si une mission ne contient que des solutions pré-construites non-optimales ($R1$) que le joueur doit améliorer (DD).

$R1 \rightarrow ZENO() \geq 1 \;\&\&\; ZENO() = Card(script)$
$C4.3 := R1 \;\&\&\; DD()$

C5.1 Représenter des objets ou séquences d'actions au moyen d'une représentation formelle : si une mission demande au joueur de combiner (DD) des blocs d'action (IEQ).

$C5.1 := DD() \;\&\&\; IEQ(\{AllActions\},\ 1)$

C5.2 Traduire des objets ou séquences d'actions entre représentations formelles : si une mission cache la position du téléporteur de sortie.

$C5.2 := Card(fog) \geq 1 \;||\; Card(hideExits) \geq 1$

C6.1 Vérifier si une séquence d'actions atteint un objectif donné : si une mission contient un robot à programmer.

$C6.1 := Card(robot) \geq 1$

C6.2 Repérer des erreurs dans une séquence d'actions : aucune mécanique de jeu pour cette compétence.

Aucune mécanique de jeu pour cette compétence.

C6.3 Corriger une séquence d'actions pour atteindre un objectif donné : si une mission propose seulement des zones d'édition pré-construites boguées ($R1$) que le joueur doit corriger (DD).

$R1 \rightarrow ZEB() \geq 1 \;\&\&\; ZEB() = Card(script)$
$C6.3 := R1 \;\&\&\; DD()$

C6.4 Étendre ou modifier une séquence d'actions pour atteindre un nouvel objectif : si une mission demande au joueur de combiner (DD) des blocs d'action (IEQ) pour contrôler plusieurs robots avec une même zone d'édition ($R1$).

$R1 \rightarrow Card(robot\ inputLine\ sameValue\ robot\ inputLine) \geq 2$
$C6.4 := DD() \;\&\&\; IEQ(\{AllActions\}, 1) \;\&\&\; R1$