How to Characterize and Analyze the Computational Thinking Skills of a Learning Game?

Mathieu Muratet

How to characterize and analyze the computational thinking skills of a learning game?

Mathieu Muratet[1,2][0000−0001−6101−5132]

[1] Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
[2] INSEI, 58-60 avenue des Landes 92150 Suresnes, France
mathieu.muratet@lip6.fr
https://webia.lip6.fr/~muratetm/

Abstract. Computational thinking is a discipline poorly mastered by elementary school teachers. To help teachers address these skills with their students, we develop the SPY game. In this paper we conduct a didactic analysis of the game and we propose a tool to describe the competencies targeted by the game based on game mechanics. The proposed tool has been applied to describe 21 of the 26 PIAF skills (a skills base on computational thinking). We show how these descriptions provide to teachers and designers micro information about the active skills in a level or macro information about the evolution of a game scenario complexity.

Keywords: Serious Game · Learning Game · Computational Thinking · Modeling · Competency · Didactic

1 Introduction and positionning

For a few years now, computer science has been teached again in school in France [2], known as computational thinking [4]. According to Wing [16], computational thinking involves five cognitive abilities: (1) algorithmic thinking, (2) abstraction, (3) evaluation, (4) decomposition, and (5) generalization. Based on this definition, Parmentier *et al.* [13] proposed a fine division of computational and algorithmic thinking skills for basic education (the PIAF framework). They also provided pedagogical scenarios helping teachers to teach these skills to their students. This question of the appropriation of these skills by elementary school teachers is complex to address. Indeed, teaching this new discipline requires an important investment on their side [5], notably because of the lack of training (initial and in-service). For instance, teachers have difficulties in constructing their pedagogical scenarios or in judging the relevance of tools to train their students on these skills.

We assume that even if elementary school teachers do not master these skills, they foresee the interest for their students when they are presented in skills base; and thus a tool based on these references would be usable for the teachers.

Our proposal: Proposing a skill-driven learning game that will assist teachers in developing computational thinking learning sessions with their students.

Today, several dozen learning games on the theme of computer science exist [7,10] but very few explain how the underlying skills are worked on and even fewer provide tools for teachers to help them adapt the games to their context [15].

The research problem is: how to help teachers who are not familiar with computational thinking to identify the skills involved in game situations (which they could create themselves)? This research problem includes two research questions:

- How can we formalize the skills base on computational thinking from the ludic features of a learning game?
- How to exploit this formalization to analyze the levels of the learning game and extract the skills involved?

In order to answer these research questions we will present the theoretical foundation in section 2. Then, in section 3, we will present the SPY game we use in this research and outline its didactic features. In section 4, we will propose a formalization to describe computational thinking skills depending on game functionalities. And we will illustrate the application of this formalization on some computational thinking skills in section 5. At the end, we will present results on the SPY game in section 6 before concluding.

2 Theoretical foundation

In 1994, Balacheff [1] identified the "computational transposition" process. It aims at identifying the links between the internal and external[3] universe of the device through an interface. This interface plays a particular role when it becomes "a reference for the user in relation to which knowledge is built"[4] [1]. Thus, the design choices of an interface influence the knowledge build process.

Designing a learning game with an intrinsic metaphor consists in putting learnings at the heart of the gameplay [9]. In this context, the designers of learning games propose gameplay mechanisms in harmony with the content to be taught, building game features which are mostly didactic. However, the computational transposition of the learnings targeted is not always explicit. Branthiome [3] developed the idea of "adidactical situations" in a learning game on programming. In this game, the students can program an initial solution that is not very efficient and, with the feedback from the system, update its solution to switch to a winning procedure. Then, the game features proposed are didactic in order to allow the students to progress in their learning path even if they are not aware of it during the game.

The player interacts with the learning game and develops usage patterns that transform the game from an artifact to an instrument [6]. The schemas

[3] The user is in the external universe of the device

[4] translated from french "une référence pour l'utilisateur relativement à laquelle la connaissance est cons-truite"

thus developed depend on the properties of the interactive artifact (in our case, the game functionalities available in a given situation). The theory of semiotic mediation develops the idea of the semiotic potential of an artifact [8]. This potential is defined by a double link "which may occur between i) an artifact, and the personal meanings emerging from its use to accomplish a task, and ii) at the same time the mathematical meanings evoked by its use and recognizable as mathematics by an expert" [8]. We focused on this second link, where we look for characterizing the links between a learning game (the artifact) and the signs resulting from its use, recognizable as knowledge, in our case computational thinking.

3 Analysis of didactic features of the SPY game

SPY[5] (see Fig. 1) is a learning game on computational thinking. It has been designed for high school students (grades 9-12). It is an open-source project[6] developed by Sorbonne University. The goal of the game is to program the actions of an agent to reach a precise position. These actions are represented as blocks that the player has to arrange in sequences executed by the agent.

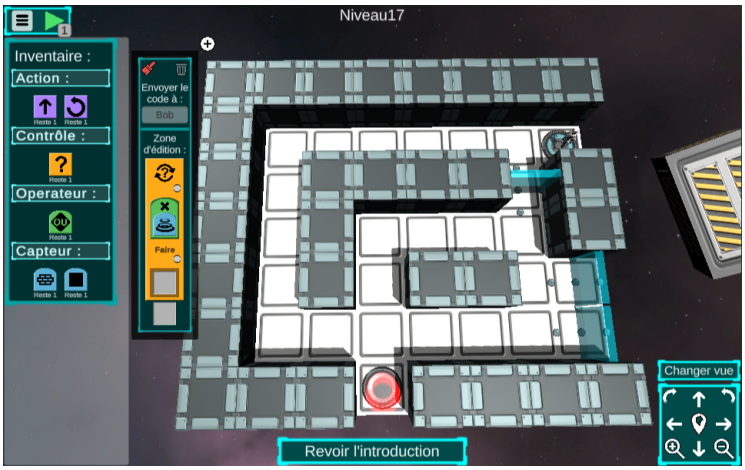


Fig. 1. Example of a SPY level.

The mobilization of computational thinking skills relies on a set of game features. We will detail here the analysis of this artifact to identify all its playful features and to study the links with the knowledge to be taught: computational thinking.

[5] SPY: https://spy.lip6.fr/, accessed on March 6, 2023
[6] SPY source code: https://github.com/Mocahteam/SPY, accessed on March 6, 2023

3.1 Programming blocks

The programming blocks are the fundamental elements of SPY, they constitute the basic bricks of the programs executed by the agents. The programming blocks are divided into four categories:

- **Action blocks:** These blocks allow the player to define the actions executed by the agents: "Move forward", "Turn left", "Turn right", "Wait", "Activate a terminal" and "Turn around". Note that all these actions are atomic except for the "Turn around" action which can be broken down in two actions "Turn right" or "Turn left".
- **Control blocks:** These blocks allow the player to control the action blocks to be executed: "If Then", "If Then Else", "Repeat n times", "While" and "Forever".
- **Sensors:** These blocks give information about the environment around the agents. The sensors return Boolean values that can be used in the "If Then", "If Then Else" and "While" control blocks. The sensors allow the agent to know whether a wall is in front of it, to its left, or its right; whether a pathway is in front of it, to its left, or its right; whether a guard, a guarded area, or a door is in front of it; and whether a terminal or an exit is on its position.
- **Operators:** These blocks allow the player to combine sensors. We find the classic Boolean operators: "No", "Or" and "And". Note that some sensors can be expressed by other sensors using the operators, for example "Wall in front" is equivalent to "No Pathway in front".
- **Blocks limitation:** SPY gives the possibility for each level to define available resources (quantity of each programming blocks). This feature is useful to reduce or increase the complexity of a level. For example, an introductory level could contain only useful blocks for solving the level, thus preventing the player to choose between useless blocks. On the other hand, restricting access to (or the quantity of) certain blocks force the player to use other blocks. For example, we can give only one "Move forward" block to force the usage of "Repeat n times" control block for moving an agent several squares forward.

3.2 Programming area

The programming area is the second fundamental functionality of SPY. It hosts the programming blocks allowing the player to build solutions.

By default, each agent is associated with a programming area. It is however possible to break this association to ask the player to define it. In this case, the player has to indicate in the programming area the name of the agent linked. Thus, on execution, the program will be sent to the linked agent.

A programming area can contain an initial program, which can be optimal, non-optimal, or buggy. The player will therefore have to complete or correct it if necessary.

Finally, SPY allows to associate several agents to a unique programming area. This gives the opportunity to designers to build levels where the player will have to find a solution not only for one agent but for several agents. The player will have to design generic solutions.

3.3 Drag and drop

Drag and drop is the third fundamental functionality in the SPY game because it allows the player to edit programming areas. However, it is possible to disable this feature, in this case, the level has to offer one or more pre-built programs. This feature allows designers to build levels where the player does not have to program but has to read several programs, understand them and choose which one to send to the agent(s) (see Fig. 2).

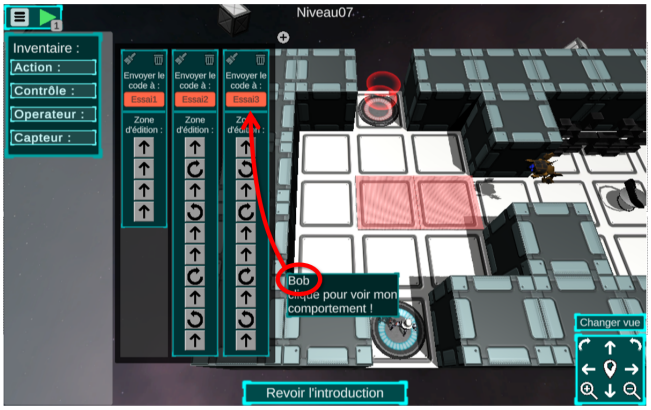


Fig. 2. Example level where drag an drop is disabled. The player has to replace one of the *Essai1*, *Essai2* or *Essai3* with the name of the robot (Bob). In this example the right solution is *Essai3*.

3.4 Obstacles

To solve the different levels of the game, the player must program one or more agents for them to reach a particular position while avoiding obstacles. These obstacles have been designed with a didactic intention.

Guards monitor areas of the maze (see red squares on the ground in Fig. 2). If an agent is detected by a guard, the player loses. Also, as for the agents, the guards can have pre-built program that makes the monitored areas dynamic. The player can select each guard in the game, observe the program that composes it, understand this program, anticipate the guard's movements, and program the agent accordingly.

Doors can be opened and closed using terminals. This feature was introduced to initiate a stepwise resolution process. When a door blocks the pathway, the player must break down his/her solution into steps (objective 1: activate the terminal to open the door; and objective 2: reach the exit). Doors also allow the manipulation of objects with changing state (open or closed).

3.5 Control execution

When the player wants to test his/her solution s/he clicks on the start button. Each programming area sends its program to the target agent if it exists and each agent/guard executes its actions in parallel. The player can follow the execution of the programs by observing the agents/guards moving in the scene. The ongoing action is highlighted in the program execution panel. The player can pause the simulation at any time and execute the programs step by step.

3.6 Number of executions

Finding a program, that moves the agent directly from the starting point to the ending point, in one attempt can be a complex task, especially on levels with moving guards. SPY authorizes to solve a level in several moves. The player can define a first sequence of blocks, execute it, observe the new situation, add new blocks, execute, observe... It is possible to find the solution step by step.

However, it can be relevant to limit this number of executions to force the player to anticipate several actions in advance. For each level of SPY, it is possible to define the authorized number of executions to solve it.

3.7 Fog and briefing

In a default SPY level, the player has an omniscient view of the situation which allows him/her to plan his/her actions to reach the objective described in the briefing. The fog allows modifying this rule by limiting the view of an agent to its close environmentIn this case, the briefing plays a fundamental role, as it should contain clues allowing the player to find the solution. For example, the algorithm can be given in the briefing in natural language and the player has to translate it using the formal language of the game.

3.8 Synthesis of the game functionalities

The analysis of the different functionalities of SPY shows us that each of them is didactic. Some of them are obvious, such as the control blocks that allow to manipulate the associated programming notions. Others are hidden, such as i) the associated program of a guard that will require the player to understand it and to anticipate its execution, or ii) the use of a single programming area to control two agents that will force the player to generalize his/her solution.

From a macro point of view, the different features of SPY allow covering the skills of computational thinking as defined by Wing [16]. The player has to

observe and model the simulation (abstraction), break down his/her strategy into steps (decomposition), determine the best solution (evaluation), plan the actions (algorithmic thinking), and reuse and adapt previous solutions to new problems (generalization). But how do we identify if a particular skill is involved in a given level? How combinations of features influence skills involved?

4 Linking skills to game features

Table 1. Main tags description structuring a SPY level.

Tag	Description
<dragdropDisabled>	If present, disables drag and drop functionality
<blockLimit blockType="X" limit="Y">	If present, defines the quantity Y of blocks of type X available in the inventory (if Y = -1 ⇒ block is unlimited).
<player inputLine="X"> <enemy inputLine="X">	Defines an agent controlled by the player. This agent listens to the communication channel X. Defines a guard. This guard listens to the communication channel X.
<script outputLine="X" editMode="Y" type="Z">	Defines a programming area that will send its content on communication channel X (see <player> and <enemy> tag). The **editMode** property indicates whether or not the player can change the communication channel. The **type** property indicates whether this programming area contains an optimal, non-optimal, buggy, or undefined pre-built program.

To describe a skill we define constraints based on game features that we evaluate with a Boolean expression. If this rule is true because the set of constraints is satisfied, then we consider that the skill is involved in the analyzed level. The rules and constraints defined for each skill are based on the level structure described in XML format. Table 1 shows the tags we reference in this article. A complete and commented level model is given in [11].

The identification of a skill is then based on the evaluation of a set of constraints. Table 2 describes the set of parameters that can be applied to a constraint. In the following section, we give some examples of constraints that we have defined to characterize some PIAF skills.

5 Application of the PIAF skills formalization

As we presented in the introduction, we choose to base our work on the PIAF framework. It is focused on computational thinking and proposes six main skills. Each of these skills is broken down into a set of sub-skills for a total of 26 sub-skills [14].

Table 2. Description of possible constraints.

$TAG \in$ set of XML tags
$ATTR \in$ set of valid attributes for TAG
$OPP \in \{=, <, \leq, >, \geq, \neq\}$
$VAL \in \mathbb{N}$

Constraint	Description
TAG	Filters TAG tags.
$TAG\ ATTR\ OP\ VAL$	Filters TAG tags that have an attribute $ATTR$ equal/different... to an integer value.
$TAG\ ATTR\ include\ SET$	Filters TAG tags that have an attribute $ATTR$ whose value is included in a set of values (SET).
$TAG\ ATTR\ sameValue\ TAG_2\ ATTR_2$	Filters TAG tags that have an attribute $ATTR$ whose value is equal to the value of the attribute $ATTR_2$ of tag TAG_2.
$TAG\ hasChild$	Filters TAG tags that contain at least one child tag.

The six main skills are: C1 - Abstracting away / generalizing; C2 - Compose / decompose a sequence of actions; C3 - Control a sequence of actions; C4 - Evaluate objects or sequences of actions; C5 - Translate between representations; and C6 - Build a sequence of actions iteratively. In the rest of the article we refer to the PIAF skills as follows: CX.Y means that sub-skill Y of skill X is involved, e.g. C2.4 refers to the fourth sub-skill of C2.

The description of all PIAF skills is not exhaustively developed here. The set of rules and constraints used to characterize the PIAF skills according to the game features is available in [12]. Here, we describe with precision three PIAF skills and we summarize the others.

5.1 C1.1 - Name objects and (sequences of) actions

Skill C1.1 is defined as follow: "Being able to give names to objects, actions and sequences of actions" [14, p. 2].

In SPY we consider that this skill is involved when the player can link a programming area with an agent. Indeed, in this case, he has to name his/her program so that it is executed by the right agent.

We describe this skill using the constraint "**TAG ATTR OP VAL**" which is instantiated as follow: "**script editMode = 2**". This constraint means: "filters 'script' tags that have an 'editMode' attribute 'equal' to the value 2". The boolean expression that defines the complete rule is then $C1.1 = Card(script\ editMode = 2) \geq 1$ which means: "Skill C1.1 is involved in the level if the cardinal of the constraint is greater than 0", in other words "if the level contains at least one nameable programming area".

5.2 C1.5 - Predict the outcome of a sequence of actions

Skill C1.5 is defined as: "Being able to tell, from a sequence of actions, what will happen if it is executed. In contrast to competency 1.4, this competency is about

providing a prediction without actually executing the sequence of actions" [14, p. 5].

In SPY, this skill is involved when a level contains a guard with a pre-built program. In this case, the player has to anticipate the movements of the guard. The player will have to propose a first solution to see the guard movements and check his/her predictions.

To describe this skill, it is necessary to check that a programming area is linked with a guard and contains at least one action.

The rule is described using a double parameter: (P1) "**TAG ATTR** sameValue **TAG2 ATTR2**" on one hand and (P2) "**TAG** hasChild" on the other hand.

The first parameter ensures that a guard ("enemy" tag) listens to the same communication line ("inputLine" attribute) as a programming area ("script" tag and "outputLine" attribute). P1 is instantiated as follow: "**enemy inputLine** sameValue **script outputLine**".

The second parameter ensures that the programming area ("script" tag) contains at least one action. P2 is instantiated as follow: "**script** hasChild".

The Boolean expression that defines the complete rule is the intersection of the two parameters: $C1.5 = Card(P1 \cap P2) \geq 1$.

5.3 C2.1 - Order a sequence of actions to reach a goal

Skill C2.1 is defined as: "Given an unordered list of actions and a goal, being able to combine these actions in a valid order to build a sequence that achieves the goal. [...] So, the learner does not need to identify all the parts needed to achieve the goal, but only their correct order" [14, p. 7].

In SPY, this is a situation where the player has only useful blocks available to solve the problem. The player has to simply arrange them in the right order to solve the level.

To describe this skill, it is necessary to fulfil the following constraints: (R1) drag and drop is activated to allow the player to combine actions; (R2) no unlimited block is available; and (R3) there is at least one action available in the inventory.

Rule R1 is described using the constraint "**TAG**" which is instantiated with the tag "**dragdropDisabled**". We want to check that drag and drop is active and therefore that the "dragdropDisabled" tag is not in the level description. R1 is defined as $Card(dragdropDisabled) = 0$.

Rule R2 is described using the constraint "**TAG ATTR OP VAL**" which is instantiated as follow: "**blockLimit limit = -1**". This constraint means: "filters 'blockLimit' tags that have a 'limit' attribute 'equal' to the value -1". We want to check that there is no unlimited amount of blocks, so R2 is defined as $Card(blockLimit\ limit = -1) = 0$.

The rule R3 is described using a double parameter: (P1) "**TAG ATTR OP VAL**" on one hand and (P2) "**TAG ATTR** include **SET**" on the other hand. The first parameter identifies whether a block is available in the inventory ("**blockLimit limit** $\geq$ **1**") and the second whether this block is an action block ("**blockLimit blockType** include **{Forward, TurnLeft, TurnRight,**

Wait, Activate, TurnBack}"). The cardinal of the intersection of these two parameters counts the number of action blocks available in the inventory. R3 is defined as follow: $Card(P1 \cap P2) \geq 1$.

The Boolean expression that defines the complete rule is then: $C2.1 = R1 \;\&\&\; R2 \;\&\&\; R3$.

5.4 Summary of skills encoded

To simplify the description of skills in this section, we name recurring rules and parameters as follows: $NPA()$: number of **N**ameable **P**rogramming **A**rea $\rightarrow Card(script\ editMode = 2)$; $CPA()$: number of **C**orrect **P**rogramming **A**reas $\rightarrow Card(script\ type = 0)$; $NOPA()$: number of **N**on **O**ptimal **P**rogramming **A**reas $\rightarrow Card(script\ type = 1)$; $BPA()$: number of **B**ugged **P**rogramming **A**reas $\rightarrow Card(script\ type = 2)$; $IBQ(B, Q)$: true if level **I**ncludes at least one **B**lock B in **Q**uantity Q, false otherwise $\rightarrow Card((blockLimit\ blockType = B) \cap (blockLimit\ limit \geq Q)) \geq 1$; $ISQ(S, Q)$: true if level **I**ncludes at least one item of the **S**et S in **Q**uantity Q, false otherwise $\rightarrow Card((blockLimit\ blockType\ include\ S) \cap (blockLimit\ limit \geq Q)) \geq 1$; $DD()$: true if **D**rag&**D**rop is enabled, false otherwise $\rightarrow Card(dragdropDisabled) = 0$.

C1.1 Name objects and (sequences of) actions: if a level contains at least one nameable programming area; $C1.1 = NPA() \geq 1$.

C1.2 Differentiate (i) object and action, and (ii) atomic actions and non-atomic actions: if a level contains non-atomic actions ("Turn back" action (IBQ_1)) which can be decomposed into two atomic actions ("Turn left" actions (IBQ_2) or "Turn right" actions (IBQ_3)) or if a level contains actions (ISQ_1) and captors (ISQ_2) that illustrate the concept of expressions and instructions; $C1.2 = (IBQ_1(TurnBack, 1)\ \&\&\ (IBQ_2(TurnLeft, 2)\ ||\ IBQ_3(TurnRight, 2)))\ ||\ (ISQ_1(\{AllActions\},\ 1)\ \&\&\ ISQ_2(\{AllCaptors\}, 1))$.

C1.3 Identify the input parameters of a sequence of actions: No game mechanics for this skill; $C1.3 = \varnothing$.

C1.4 Describe the outcome of a sequence of actions: No game mechanics for this skill; $C1.4 = \varnothing$.

C1.5 Predict the outcome of a sequence of actions: if a level contains a guard with a pre-built program; $C1.5 = Card(enemy\ inputLine\ sameValue\ script\ outputLine\ \cap\ script\ hasChild) \geq 1$.

C1.6 Using objects whose value can change: if a level contains conditional control structure (ISQ_1) and captors (ISQ_2) whose value varies according to the context or if a level makes it possible to control (IBQ) the state of a door ($R1$); **Rule:** $R1 \rightarrow Card(door\ slotId\ sameValue\ slot\ slotId) \geq 1$; $C1.6 = (ISQ_1(\{While, IfThen, IfElse\}, 1)\ \&\&\ ISQ_2(\{AllCaptors\}, 1))\ ||\ (IBQ(Activate, 1)\ \&\&\ R1)$.

C1.7 Recognize existing objects and (sequences of) actions that can be used to reach a similar goal: if a level only provides nameable programming areas ($R1$) of which at least one is a correct pre-filled solution ($R2$) that the payer must recognize and cannot modify (DD); **Rules:** $R1 \rightarrow$

$NPA() = Card(script)$; $R2 \rightarrow CPA() \cap (script\ hasChild)) \geq 1$; $C1.7 = R1\ \&\&\ R2\ \&\&\ !DD$.

C2.1 Order a sequence of actions to reach a goal: if a level allows the player to combine actions (DD) without unlimited blocks ($R1$) and with at least one action available in the inventory (ISQ); **Rule:** $R1 \rightarrow Card\ (blockLimit\ limit = -1) = 0$; $C2.1 = DD\ \&\&\ R1\ \&\&\ ISQ(\{AllActions\}, 1)$.

C2.2 Complete a sequence of actions to reach a simple goal: if a level allows the player to combine (DD) only actions (ISQ_1) without control structures (ISQ_2) and only provides bugged pre-filled solutions ($R1$) for the player to complete; **Rule:** $R1 \rightarrow BPA() \cap (script\ hasChild)) = Card(script)$; $C2.2 = DD\ \&\&\ ISQ_1(\{AllActions\}, 1)\ \&\&\ !ISQ_2(\{ControlList\}, 1)\ \&\&\ R1$.

C2.3 Create a sequence of actions to reach a simple goal: if a level allows the player to combine (DD) only actions (ISQ_1) without control structures (ISQ_2); $C2.3 = DD\ \&\&\ ISQ_1(\{AllActions\}, 1)\ \&\&\ !ISQ_2(\{ControlList\}, 1)$.

C2.4 Create a sequence of actions to reach a complex goal: if a level allows the player to combine (DD) actions (ISQ_1) and control structures (ISQ_2); $C2.4 = DD\ \&\&\ ISQ_1(\{AllActions\}, 1)\ \&\&\ ISQ_2(\{ControlList\}, 1)$.

C2.5 Combine sequences of actions to reach a goal: No game mechanics for this skill; $C2.5 = \varnothing$.

C2.6 Decompose goals into simpler subgoals: if a level does not limit the number of execution to 1; $C2.6 = Card(executionLimit\ amount = 1) = 0$.

C3.1 Repeat a sequence of actions a given number of times: if a level allows the player to combine (DD) For loops (IBQ); $C3.1 = DD\ \&\&\ IBQ(ForLoop, 1)$.

C3.2 Repeat a sequence of actions until a goal has been reached: if a level allows the player to combine (DD) While loops (IBQ) with captors (ISQ_1) and without operators (ISQ_2); $C3.2 = DD\ \&\&\ IBQ(WhileLoop, 1)\ \&\&\ ISQ_1(\{AllCaptors\}, 1)\ \&\&\ !ISQ_2(\{OperatorList\}, 1)$.

C3.3 Integrate a simple condition into a sequence of actions: if a level allows the player to combine (DD) If statements (ISQ_1) with captors (ISQ_2) and without operators (ISQ_3); $C3.3 = DD\ \&\&\ ISQ_1(\{IfThen, IfElse\}, 1)\ \&\&\ ISQ_2(\{AllCaptors\}, 1)\ \&\&\ !ISQ_3(\{OperatorList\}, 1)$.

C3.4 Integrate a complex condition into a a sequence of actions: if a level allows the player to combine (DD) conditionnal statements (ISQ_1) with captors (ISQ_2) and operators (ISQ_3); $C3.3 = DD\ \&\&\ ISQ_1(\{While, IfThen, IfElse\},\ 1)\ \&\&\ ISQ_2(\{AllCaptors\}, 1)\ \&\&\ ISQ_3(\{OperatorList\}, 1)$.

C4.1 Compare two objects according to a given criterion: No game mechanics for this skill; $C4.1 = \varnothing$.

C4.2 Compare two sequences of actions according to a given criterion: if a level only provides nameable pre-filled solutions ($R1$) with at least one correct solution and non-optimal solutions ($R2$) that the player cannot modify (DD); **Rules:** $R1 \rightarrow NPA() = Card(script)$; $R2 \rightarrow CPA() \geq 1\ \&\&\ (CPA() + NOPA()) = Card(script)$; $C4.2 = R1\ \&\&\ R2\ \&\&\ !DD$.

C4.3 Improve a sequence of actions according to a given criterion: if a level only provides pre-filled non-optimal solutions ($R1$) that the player

must fix (DD); **Rule:** $R1 \rightarrow NOPA() \geq 1$ && $NOPA() = Card(script)$; $C4.3 = R1$ && DD.

C5.1 Represent objects or sequences of actions through one formal representation: if a level asks the player to combine (DD) action blocks (ISQ); $C5.1 = DD$ && $ISQ(\{AllActions\}, 1)$.

C5.2 Translate objects or sequences of actions between formal representations: if a level hides the exit position; $C5.2 = Card(fog) \geq 1$ || $Card(hideExits) \geq 1$.

C6.1 Verify if a sequence of actions reaches a given goal: if a level contains a robot to be programmed; $C6.1 = Card(player) \geq 1$.

C6.2 Notice errors in a sequence of actions: No game mechanics for this skill; $C6.2 = \varnothing$.

C6.3 Fix a sequence of actions for reaching a given goal: if a level only provides pre-filled bugged solutions $(R1)$ that the player must fix (DD); **Rule:** $R1 \rightarrow BPA() \geq 1$ && $BPA() = Card(script)$; $C6.3 = R1$ && DD.

C6.4 Extend or modify a sequence of actions to reach a new goal: if a level requires the player to combine (DD) action blocks (ISQ) to control multiple robots with the same program $(R1)$; **Rule:** $R1 \rightarrow Card(player\ inputLine\ sameValue\ player\ inputLine) \geq 2$; $C6.4 = DD$ && $ISQ(\{AllActions\}, 1)$ && $R2$.

6 Results

We have summarized the description of PIAF skills with the proposed formalization and we have illustrated in detail the description of three of them. In addition to the PIAF skills, we have tested our formalization with another skills base. We have described the 5 levels of the "3.4 Programming" skill from the CRCN domain 3 "Content creation"[7]. Finally, we have used the formalization to describe the set of playful functionalities of SPY (called SPY functionalities base). All these descriptions are available in [12].

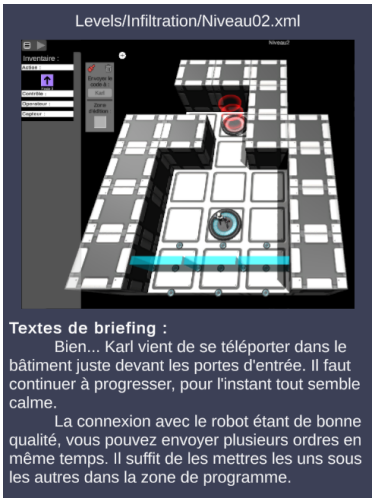
Thus we can analyze each level with respect to the different skills bases. Table 3 illustrates the analysis of one level with the 3 skills bases. This visualization gives to the teacher the skills involved according to the skills base. In this example, the level 1 of CRCN skill 3.4 has been identified. It is defined as "Read and construct an algorithm with simple instructions"[8]. CRCN skills base is less precise than the PIAF ones but gives a overall view of skills involved. With these different levels of information, the teacher will determine if the level is an interesting candidate to integrate his/her pedagogical scenario.

We can also use these descriptions to analyze the complexity of a game scenario. SPY contains two scenarios: an original SPY called "Infiltration" con-

[7] CRCN (*Cadre de Référence des Compétences Numériques*) is a french skills base inspired by the DigComp 2.1: https://eduscol.education.fr/document/20389/download, accessed April 7, 2023

[8] translated from french "Lire et construire un algorithme qui comprend des instructions simples"

Table 3. Analysis of one level with the different skills bases



PIAF skills base: C2.1 - Order a sequence of actions to reach a goal C2.3 - Create a sequence of actions to reach a simple goal C2.6 - Decompose goals into simpler sub-goals C5.1 - Represent objects or sequences of actions through one formal representation C6.1 - Verify if a sequence of actions reaches a given goal
CRCN skills base: Level 1
SPY skills base: F1 - Solve a problem in several steps F10 - unlimited blocks F11 - action block “Move forward”

sisting of 20 levels and a copy of the BlocklyMaze game[9] consisting of 10 levels. Teachers can also build their own scenarios by composing them from the existing level library. We used the PIAF skills base to characterize the didactic dimension of the different scenarios and the SPY functionalities base to characterize their playful dimension. The combination of skills and playful features in each level gives an indicator of its complexity (see Fig. 3). We observe the progressive evolution of the complexity and the appearance of the different skills during the two scenarios. This visualization allows to compare the complexity of scenarios based on common criteria. We can see that Infiltration starts lower and ends higher than BlocklyMaze but the progression is smoother on BlocklyMaze. This visualization is also useful for designers to detect uncovered skills or unbalanced levels. For instance we can see that C4 is only covered in Infiltration level 7. It would be interesting to create levels focused on PIAF skills C4.2 and C4.3. This analysis also reveals some complexity peaks (levels 12 and 14 of Infiltration). These are interesting indicators to study these levels and considering adjustments.

7 Conclusion and perspectives

In this paper we performed a didactic analysis of SPY by exploring the main features of the game in relation to computational thinking. Then, we proposed a formalization to describe skills from game features. We used this formalization to propose a description of i) the 21 PIAF skills involved in SPY, ii) the 5 levels of CRCN skills base and iii) the 35 game features of SPY. All the results are presented in [12]. We have shown that the proposed formalization is independent

[9] BlocklyMaze: https://blockly.games/maze, accessed April 10, 2023

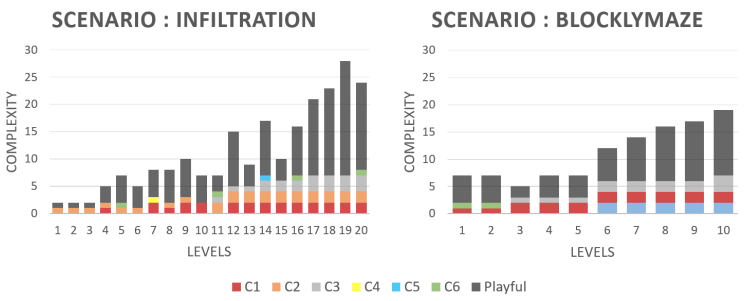


Fig. 3. Analysis of two scenarios.

of skills bases. We have detailed in this article an extract of these results using the C1.1, C1.5 and C2.1 skills of the PIAF base.

Finally, we have shown how these descriptions can be used to build metadata on existing levels. These descriptions provide to teachers and designers micro information about the active skills in a level or macro information about the evolution of the game scenario complexity.

Currently the SPY game does not contain a level editor. The creation or modification of levels is not very accessible (editing XML files by hand). One perspective for this work is to propose an editor allowing teachers to create their own game levels. This will give autonomy to teachers as they will get an automatic skill analysis of their own levels.

Acknowledgements The author acknowledges the support of the French Agence Nationale de la Recherche (ANR), under grant ANR-18-CE38-0008 (project IECARE) and Stéphanie Chane Chick Te for her proofreading of this article.

References

1. Balacheff, N.: La transposition informatique, un nouveau problème pour la didactique. In: Artigue M. and Gras R. and Laborde C. and Tavignot P. and Balacheff N. (ed.) colloque “Vingt ans de didactique des mathématiques en France”, 15-17 juin 1993. pp. 364–370. Recherches en didactique des mathématiques, La Pensée Sauvage, Paris, France (1993), https://telearn.archives-ouvertes.fr/hal-00190646
2. Baron, G.L., Drot-Delange, B., Grandbastien, M., Tort, F.: Computer science education in french secondary schools: Historical and didactical perspectives. ACM Trans. Comput. Educ. **14**(2) (jun 2014). https://doi.org/10.1145/2602486, https://doi.org/10.1145/2602486
3. Branthôme, M.: Pyrates: A Serious Game Designed to Support the Transition from Block-Based to Text-Based Programming. In: European Conference on Technology Enhanced Learning (EC-TEL 2022). Lecture Notes in Computer Science, vol. 13450, pp. 31–44 (2022)
4. Chiprianov, V., Gallon, L.: Introducing Computational Thinking to K-5 in a French Context. In: 21st Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2016). ACM Press, Arequipa,

Peru (Jul 2016). https://doi.org/10.1145/2899415.2899439, https://hal-univ-pau.archives-ouvertes.fr/hal-01908224
5. Kradolfer, S., Dubois, S., Riedo, F., Mondada, F., Fassa, F.: A sociological contribution to understanding the use of robots in schools: The thymio robot. In: Beetz, M., Johnston, B., Williams, M.A. (eds.) Social Robotics. pp. 217–228. Springer International Publishing, Cham (2014)
6. Laisney, P., Chatoney, M.: Instrumented activity and theory of instrument of Pierre Rabardel. In: Philosophy of technology for technology education to Sense/Brill (2018), https://hal-amu.archives-ouvertes.fr/hal-01903109
7. Lindberg, R.S.N., Laine, T.H., Haaranen, L.: Gamifying programming education in k-12: A review of programming curricula in seven countries and programming games. British Journal of Educational Technology **50**(4), 1979–1995 (2019). https://doi.org/https://doi.org/10.1111/bjet.12685, https://bera-journals.onlinelibrary.wiley.com/doi/abs/10.1111/bjet.12685
8. Mariotti, M.A., Maracci, M.: Resources for the Teacher from a Semiotic Mediation Perspective, pp. 59–75. Springer Netherlands, Dordrecht (2012)
9. Marne, B., Wisdom, J., Huynh-Kim-Bang, B., Labat, J.M.: The Six Facets of Serious Game Design: a Methodology Enhanced by our Design Pattern Library. In: Seventh European Conference on Technology Enhanced Learning (EC-TEL 2012). Lecture Notes in Computer Science, vol. 7563, pp. 208–221. Springer Berlin / Heidelberg, Saarbrücken, Germany (Sep 2012). https://doi.org/10.1007/978-3-642-33263-0_17, https://hal.sorbonne-universite.fr/hal-00739124
10. Miljanovic, M.A., Bradbury, J.S.: A review of serious games for programming. In: Göbel, S., Garcia-Agundez, A., Tregel, T., Ma, M., Baalsrud Hauge, J., Oliveira, M., Marsh, T., Caserman, P. (eds.) Serious Games. pp. 204–216. Springer International Publishing, Cham (2018)
11. Muratet, M.: Complete and commented level model in XML format (2023), https://github.com/Mocahteam/SPY/blob/master/Doc/LevelModel.xml, [Online; accessed June 19, 2023]
12. Muratet, M.: Description of PIAF skills using the SPY game features (2023), https://github.com/Mocahteam/SPY/blob/master/Assets/StreamingAssets/Competencies/competenciesReferential.json, [Online; accessed June 19, 2023]
13. Parmentier, Y., Reuter, R., Higuet, S., Kataja, L., Kreis, Y., Duflot-Kremer, M., Laduron, C., Meyers, C., Busana, G., Weinberger, A., Denis, B.: PIAF: Developing Computational and Algorithmic Thinking in Fundamental Education. In: EdMedia+ Innovate Learning, Association for the Advancement of Computing in Education (AACE). pp. 315–322 (06 2020)
14. PIAF Project: The complete list of the 26 PIAF skills and their description (2021), https://piaf.loria.fr/wp-content/uploads/2021/09/PIAF-Referential-of-Competencies-Description-and-Examples.pdf, [Online; accessed June 19, 2023]
15. Saddoug, H., Rahimian, A., Marne, B., Muratet, M., Sehaba, K., Jolivet, S.: Review of the Adaptability of a Set of Learning Games Meant for Teaching Computational Thinking or Programming in France. In: Special Session on Gamification on Computer Programming Learning. vol. 1, pp. 562–569. SCITEPRESS - Science and Technology Publications, Prague, Czech Republic (Apr 2022). https://doi.org/10.5220/0011126400003182, https://hal.science/hal-03668918
16. Wing, J.M.: Computational thinking. Communications of the ACM **49**(3), 33–35 (2006)