

A theoretical framework for the design and analysis of computational thinking problems in education

Giorgia Adorni · Alberto Piatti · Engin Bumbacher · Lucio Negrini ·
 Francesco Mondada · Dorit Assaf · Francesca Mangili · Luca
 Gambardella

Abstract The field of computational thinking education has grown in recent years as researchers and educators have sought to develop and assess students' computational thinking abilities. While much of the research in this area has focused on defining computational thinking, the competencies it involves and how to assess them in teaching and learning contexts, this work takes a different approach. We provide a more situated perspective on computational thinking, focusing on the types of problems that require computational thinking skills to be solved and the features that support these processes.

We develop a framework for analysing existing computational thinking problems in an educational context. We conduct a comprehensive literature review to identify prototypical activities from areas where computational thinking is typically pursued in education. We identify the main components and characteristics of these activities, along with their influence on activating computational thinking competencies. The frame-

work provides a catalogue of computational thinking skills that can be used to understand the relationship between problem features and competencies activated.

This study contributes to the field of computational thinking education by offering a tool for evaluating and revising existing problems to activate specific skills and for assisting in designing new problems that target the development of particular competencies. The results of this study may be of interest to researchers and educators working in computational thinking education.

Keywords Computational thinking · Competence development · Educational framework · Learning contexts · Situated cognition

1 Introduction

Computational thinking (CT) has emerged as a crucial skill for students to acquire in the 21st century. As a result, there has been an increased effort to integrate computer science education into K-12 classrooms (Weintrop et al. 2021). This initiative was triggered by Jeannette Wing's introduction of the term CT in (Wing 2006). Despite the impressive growth in tools, activities and curricula for teaching CT, significant challenges remain in successfully integrating this new concept into schools. One critical challenge is the lack of a precise, universally accepted definition of CT, a relatively ambiguous concept (Weintrop et al. 2021). Instead, different definitions with varying purposes, each focusing on different aspects of CT, have been proposed (Lafuente Martínez et al. 2022). This lack of consensus has made it difficult for the field to advance beyond the exploratory stage. To address this issue and systematically develop and evaluate different approaches to teaching, developing, and assessing CT, it is necessary to have a precise and comprehensive formalisation

G. Adorni · F. Mangili · L. Gambardella
 Dalle Molle Institute for Artificial Intelligence (IDSIA), Università della Svizzera Italiana and University of Applied Sciences and Arts of Southern Switzerland (USI-SUPSI), Lugano, Switzerland

A. Piatti · L. Negrini
 Department of Education and Learning (DFA), University of Applied Sciences and Arts of Southern Switzerland (SUPSI), Locarno, Switzerland

E. Bumbacher
 Haute école pédagogique du canton de Vaud (HEP-VD), Lausanne, Switzerland

F. Mondada
 Mobile Robotic Systems Group (MOBOTS), Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland

D. Assaf
 School of Education, University of Applied Sciences and Arts Northwestern Switzerland (FHNW), Windisch, Switzerland

of the concept and to identify widespread best practices. Therefore, efforts are needed to establish a clear and standardised definition of CT that reflects the diverse perspectives and purposes of the field. However, there is no easy way to define CT (Shute et al. 2017). Prevailing approaches have focused on decomposing it into sub-dimensions and explicitly specifying each, e.g., Brennan and Resnick (2012); Grover and Pea (2017). These dimension-based approaches have been used to categorise existing assessment tasks based on analysing the underlying skills associated with a task, e.g., Lafuente Martínez et al. (2021). However, creating tasks for developing and assessing these sub-dimensions has been difficult. A recent study developed a reliable and validated CT assessment for adults by combining existing items (Lafuente Martínez et al. 2022). While experts identified several CT sub-dimensions to be addressed by the items, statistical analyses suggested a one-dimensional model as the best solution. A core problem is that sub-dimensions such as decomposition, generalisation, or pattern recognition are closely interwoven and hard to separate (Lafuente Martínez et al. 2021). Similar issues are observed with other complex constructs, such as scientific inquiry or practices, as noted by previous studies (Osborne 2014; Ford 2015).

In this article, we propose an alternative, more situated approach by focusing on the types of problems that require CT to be solved, which we call computational thinking problems (CTPs). This approach is based on the idea that CT can be better understood by examining the problems that elicit it and the features that support CT processes rather than attempting to define CT itself. The idea behind incorporating a situated approach in this context is that CT is not just a collection of skills. Still, it is also closely tied to the specific context in which it is being applied and allows understanding the complexity of CT. The theories of situated learning proposed by (Roth and Jornet 2013; Heersmink 2013) stress that learning is most effective within authentic and meaningful contexts. They argue that knowledge is constructed through interactions with the environment and the community rather than abstract instruction.

Our framework for analysing, evaluating and revising existing CTP and designing new CTP is first based on the theory proposed by Piatti et al. (2022). They emphasise that CT should be considered a situated activity and contextualised and embedded in real-world problem-solving situations. They also highlight that CT should be seen as a dynamic and adaptive problem-solving process rather than a fixed set of competencies or skills. Their view combines the original definition of CT from Wing (2006) – “Computational thinking is the

thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that an information-processing agent can effectively carry out.” – which has been widely cited and considered the foundation of the field of CT, with the situated theories of learning (Roth and Jornet 2013; Heersmink 2013).

To avoid creating a novel CT competence model, we have adopted state-of-the-art frameworks (Brennan and Resnick 2012; Weintrop et al. 2016; Shute et al. 2017) and reviewed relevant literature (Tikva and Tambouris 2021; Bocconi et al. 2016, 2022). Our approach resulted in a catalogue of commonly recognised CT competencies widely used to develop and assess student abilities in CT. These competencies, also called dimensions or skills, represent the essential capacities students need to develop to solve CTPs effectively. We established a direct link between our catalogue of CT competencies and our framework by showing how specific skills are more likely to be activated when solving CTPs with specific characteristics. Finally, the framework is applied to a variety of prototypical problems from standard CT domains, demonstrating that it allows for systematically analyse and evaluate CTPs based on the competencies they are intended to elicit and to assess student abilities in a targeted and efficient manner.

Answering research questions such as “What are the characteristics of problems requiring CT to be solved?”, “Which characteristics should a CTP have to activate one (or more) CT competencies?” and “Which CT dimensions are activated in a CTP with certain characteristics?”, can provide a better understanding of how to design and assess CTPs that effectively develop CT skills. Our proposed framework offers a systematic approach for identifying the characteristics of CTPs and the corresponding CT competencies required to solve them. This approach enables the selection or design of tasks effective in developing and assessing the desired skills, ultimately improving the quality of the tasks and assessment process.

This article is divided into four main sections. Section 2 presents a theoretical framework for analysing, evaluating, revising and designing CTPs. This approach is based on a catalogue of CT skills, which are commonly used to develop and evaluate student abilities. To determine how the characteristics of a CTP impact the development of CT skills, we defined a mapping between skills and characteristics, which is discussed in detail in Appendix A. Section 3 validates the framework through a qualitative analysis of various CTPs found in the literature, focusing on unplugged, robotic, and virtual activities. It is important to note that readers are not required to read the entire section. They can choose

to read one example per category or skip this section altogether and move directly to Sections 4 and 5. Additionally, Appendix B provides graphical templates that offer a visual representation of how each CTP discussed in the article aligns with the CT competencies outlined in the study. Section 4, includes an overview of the relationship between CT competencies and different activity domains. It outlines the strengths, weaknesses and areas for improvement of each category analysed. Section 5, summarises the study’s key contributions, limitations and implications for future research and practice in the field of CT competencies development.

2 Methods

This section presents a theoretical framework for analysing, evaluating, revising and designing computational thinking problems (CTPs) in educational contexts.

Through a thorough analysis of prototypical activities from various classic areas of the literature where computational thinking (CT) is typically pursued, we identify the main characteristics and components of these problems. Relying on existing frameworks and literature on CT competencies, we present a catalogue of skills typically used to assess student abilities in this area. We then define a mapping between the identified characteristics of the CTPs and these competencies to determine how these features influence the development of CT abilities.

2.1 Computational thinking problems (CTPs)

The increasing emphasis on the development and assessment of students’ CT skills has led to a need for a structured approach to analyse, evaluate, revise, and design CTPs. Although several initiatives have been launched to introduce computer science education in classrooms, there is currently no universally accepted definition of CT. This lack of a standardised definition presents significant challenges in creating appropriate assessment tasks and identifying the various sub-dimensions of CT. Therefore, it is crucial to establish a clear and precise definition of CT that reflects the diverse perspectives and purposes of the field to facilitate the development and assessment of effective CT pedagogical interventions.

To address this issue, we followed the approach in Piatti et al. (2022), which is more closely aligned with the complex, multi-faceted settings in which CT is typically activated, such as educational environments or situations involving multiple people and rich physical environments. They combine the original view of Wing

(Wing 2006) with the situated theories of learning (Roth and Jornet 2013; Heersmink 2013). According to Piatti et al. (2022), CT activities are shaped by the physical and social context in which they occur and involve external cognitive artefacts. Piatti et al. (2022) proposed

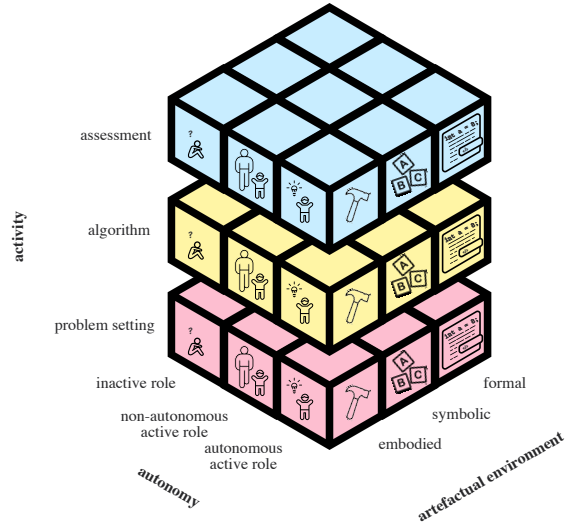


Fig. 1 Visualisation of the computational thinking cube (CT-cube) (From Piatti et al. (2022)). This model considers the type of activity (problem setting, algorithm, assessment), the artefactual environment (embodied, symbolic, formal), and the autonomy (inactive role, non-autonomous active role, or autonomous active role).

a framework called computational thinking cube (CT-cube), illustrated in Fig. 1, for the design of CTPs and the assessment of CT. This model considers the type of CT activity being performed or required (problem setting, algorithm, assessment), the artefactual environment in which the activities occur, represented by the tools used (embodied, symbolic, formal), and the social interactions and individual’s level of autonomy a priori and/or during the task (inactive role, non-autonomous active role, or autonomous active role). It is important to note that the steps in the activity dimension may be iterated as needed to arrive at a satisfactory solution.

Based on the formulation of CT from Piatti et al. (2022), we define CTPs by considering the context in which the activity is being performed. According to our framework, every CTP consists of several components: the system, comprising the environment and the agent, the problem solver, and the task.

The *environment* is a physical and/or a virtual space, characterised by one or more variables, called “descriptors”, which may change over time according to the dynamics of this space.

The *agent* is a human, robotic or virtual being capable of performing “actions” on the environment to

change the value of its descriptors and therefore alter its state. An *algorithm* is a finite set of instructions that an agent should follow to perform actions in the environment to solve the task. Algorithms for different types of agents can take various forms, such as code for a virtual agent, behaviour for a robot, or a verbal or written set of instructions for a human agent.

The *problem solver* is a human or group of people who can solve tasks that require the use of algorithms, such as designing, implementing, or communicating them to an agent to change the state of an environment. They have access to *reasoning tools*, which are cognitive artefacts used to think about the task, for example, whiteboards employed to organise ideas and understand the logic of a problem or solution. Some of these tools, known as *interaction tools*, also allow the problem solver to interface with the system. An example is a programming platform used to write a program that controls a robotic arm. In this case, the tool serves as both a reasoning tool, enabling the problem solver to plan and design the code, and as an interaction tool, allowing the execution of the algorithm and the observation of its effect on the system. The set of all tools is collectively known as *artefactual environment* and is described in the works of Heersmink (2013) and Piatti et al. (2022).

The *task* is the activity that the problem solver performs to find one or more solutions to a CTP. A solution is a combination of initial states, algorithms, and final states that meets the system’s requirements for a particular environment, with its set of states, and a given agent, with its set of algorithms. The initial state is the starting configuration of the environment, while the final state is the state of the environment after the algorithm is performed. For a solution to be valid, the execution of the algorithm on the initial state must produce the final state.

Fig. 2 conveys the framework’s components and their interactions. Each component is depicted in a distinct colour consistently throughout the article.

The proposed framework, in addition to defining CTPs and their components, allows classify them according to their characteristics, which are relevant for eliciting and assessing CT skills in educational contexts.

Definition 1 (Artefactual environment) Tools and resources used by a problem solver to reason, understand a problem or interact with a system.

The first characteristic we defined in our framework is the artefactual environment, in line with the definition from Piatti et al. (2022) and the model of the three

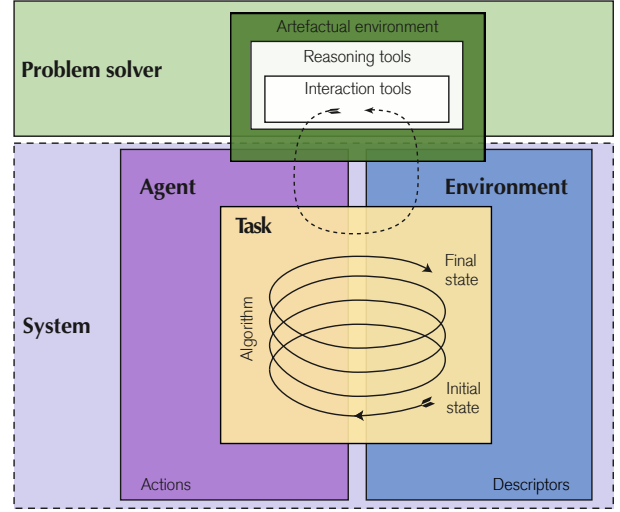


Fig. 2 Visualisation of the components of a CTP. According to the framework, a CTP includes: (1) the problem solver (in green) characterised by the artefactual environment, i.e., the set of reasoning and interaction tools, (2) the system, which consists of an environment with its descriptors (in blue) and an agent with its actions (in violet), and (3) the task (in yellow) characterised by the set of initial states, algorithms and final states.

worlds of mathematics by Tall (2006, 2013, 2020). In particular, tools can be distinguished into “embodied” or ecological and iconic representational cognitive artefacts based on embodiment and perception, “symbolic” cognitive artefacts used to conceive and apply procedures and rules, and “formal” cognitive artefacts used to create, generalise and represent structures.

Definition 2 (Tools functionalities) Specific features and functions provided by a tool or resource that enable a problem solver to express a wide range of instructions and operations to the agent.

Another aspect of CTPs that is closely related to the previous one is the set of tools functionalities. These functionalities can include defining and manipulating “variables”, using different types of “operators”, creating “sequences” of actions, “repeating” actions, using “conditional” statements, defining “functions”, executing tasks in “parallel” and triggering “events”. For example, a symbolic artefact, such as a block-based programming platform, may have many functionalities, such as sequences, repetitions, conditionals, etc. In contrast, the programming interface may have limited functionalities during a robotic activity, for example, it could only consent to the use of operators (like moving forward) or events.

Definition 3 (Problem domain) The category of an activity depending on the nature of the agent and of the environment in which the task is performed.

The domain classification of a CTP is useful for gaining insights into the task's context and identifying its specific characteristics, challenges, and considerations. Three main categories of domains are commonly recognised in cognitive task paradigms, these include: “unplugged” activities, which involve a human agent and a physical environment, “robotic” activities, in which the agent is a robot and the environment is physical, and “virtual” activities, where both agent and environment are virtual, such as a simulated entity. It is worth noting that the agent may be “embedded” in the environment, so its descriptors may likewise be used to describe it. Furthermore, in some cases, the problem solver and agent may be “overlapped”, meaning they are the same entity.

Definition 4 (System resettability) The property of a system to be restored to its initial state. It can be achieved either through the direct intervention of the problem solver on the system (agents and environment) or indirectly through the reversibility of actions within the system.

Resettability is another characteristic of CTPs. An action is reversible if it is possible to undo its effects, allowing the system to be restored to a previous state. A system is considered non-resettable if it cannot be returned to its initial state. This characteristic is crucial in educational contexts as it enables the problem solver to experiment, make mistakes and try different solutions without being constrained by the previous actions. Imagine a task where the problem solver must draw a picture on a piece of paper using a pencil. If he makes a mistake, he can easily erase it and start over. The system is directly resettable because the problem solver can directly intervene and reset the system to its initial state. Alternatively, the system is not resettable if the problem solver can only use a pen. If he makes a mistake, it can not be erased, and the problem solver must continue with the picture in its current state. An example of indirect resettability would be if the problem solver is drawing a picture on a digital tablet. He can use the undo button to wipe out the previous action and return to a previous state.

Definition 5 (System observability) The property of a system that pertains to the ability of the problem solver to observe the effects of actions taken within the system and their impact on the system's state.

Observability is another aspect that can vary between CTPs. Systems can be classified as partially observable, in which only the aggregate effects of a limited number of actions can be perceived, totally observable, in which every single action and its consequences are visible, or not observable, in which the problem solver is unable to directly see the results of the agent's actions and the system's state and must infer it from other information. For example, in a chess game, the problem solver can see the state of the board and the pieces all the time, making the system fully observable. In a game of poker, conversely, the problem solver can only see his own cards and the community cards, but not those held by the other players, making the system partially observable. In a scenario where the problem solver remotely controls a robot to explore an underground cave, the person cannot directly see the environment and robot's actions and must rely on sensor data to determine its location and progress. As a result, the system is non-observable, and decisions are based on limited information.

Definition 6 (Task type) The category of an activity influenced by both the number and type of objectives that need to be achieved to solve the task.

The last important set of CTP features are those related to the task. Each element that composes the task (initial state, algorithm, final states) can be “given” or is “to be found”. The tasks are divided into six categories, which differ depending on the number of objectives. Tasks with a single objective are classified into the following types: (1) *find the initial state*: given the final state and the algorithm that produced it, the problem solver must infer the initial state on which the algorithm was applied; (2) *find the algorithm*: given the initial and the final states, the problem solver must devise and describe an algorithm, or a part of it, that the agent can execute to transform the system from the initial to the final state; (3) *find the final state*: given the initial state and an algorithm, the problem solver must derive the final state. Tasks with multiple objectives fall into the following types: (4) *creation act*: given an initial state, the problem solver must determine a desired final state and an algorithm that the agent can

use to transform the system from the initial to the final state; (5) *application act*: given an algorithm, the problem solver must identify one or more pairs of initial and final states on which the algorithm can be applied successfully; (6) *project act*: given a desired final state, the problem solver must define an initial state and an algorithm that the agent should use to transform the system from the initial to the final state.

Definition 7 (Task cardinality) The proportion between the number of given elements to the number of elements to be found to solve the task.

In a task, it is possible for each of the given elements and elements that need to be found to be singular or multiple. The relationship between the number of given elements and the number of elements to be found can be “one-to-one”, “many-to-one” or “many-to-many”. For example, a task with a one-to-one cardinality can be one where the problem solver is provided with a single initial and final state and is expected to find a single algorithm that transforms the initial state into the final state. In contrast, a task with a many-to-one cardinality can be one where multiple initial states are given, and the problem solver is expected to find a single algorithm that transforms all the initial states into a single final state. Finally, a task with a many-to-many cardinality can be one where the problem solver is provided with multiple initial states and a single final state and is expected to find multiple algorithms that transform all possible initial states into the desired final state. This type of task can be traced back to many tasks with a many-to-one cardinality.

Definition 8 (Task explicitness) The level of explicitness or implicitness in the presentation of the given task’s elements.

The way the elements of the task are stated can be used to distinguish a CTP by its level of explicitness. A task with “explicitly” specified elements is directly usable in the problem-solving process, while a task with “implicitly” expressed through constraints requires additional interpretation to be understood. For example, in a task where a robot must turn on its lights as soon as it finds a ball in a playground, the given elements are the initial state (the robot lights switched off, the robot and the ball positions) and the final state (the robot lights turned on and the robot positioned in front of the ball), while the element to be found is the algorithm (the set of actions the robot should perform to find the ball).

The position of the ball can be described either explicitly using coordinates, or implicitly by stating that it is located in the playground.

Definition 9 (Task constraints) The limitations or specific requirements that must be adhered to on the elements of a task to be found for the solution to be considered as valid.

A CTP can be distinguished by the type of constraints on the elements of the task that need to be found. In particular, the elements that have to be found are distinguished in “unconstrained”, meaning they can be freely chosen from all possible states and algorithms, without any limitations or specific requirements that need to be met to consider the solution valid, and “constrained”, meaning they must belong to a specified subset of the respective universe set of states or algorithms. Referring to the previous example, the algorithm to be found can be unconstrained if the robot can perform any action to find the ball (moving in a random direction, using sensors to detect the ball, or following a predefined path) or constrained if the programming platform limits the commands the robot can execute (moving only in specific directions, using only specific sensors or following a particular set of predefined paths to find the ball).

Definition 10 (Algorithm representation) The mean by which an algorithm is conveyed.

Finally, how the algorithm is represented is a significant task characteristic. An algorithm can be “manifest” if it is directly expressed or “latent” if it is not stated but is tacit or inferred by the problem solver. Manifest algorithms can be further classified as “written” if it is represented by an external and persistent representation of it, such as through a programming language, or “not written” if it is communicated verbally or through other non-permanent means. When the problem solver and the agent are the same entity, the algorithm used to solve the task is often not explicitly expressed or written down, as the problem solver is carrying out the steps of the algorithm directly through their actions as the agent.

Fig. 3 graphically represent the components and characteristics of a CTP according to our framework.

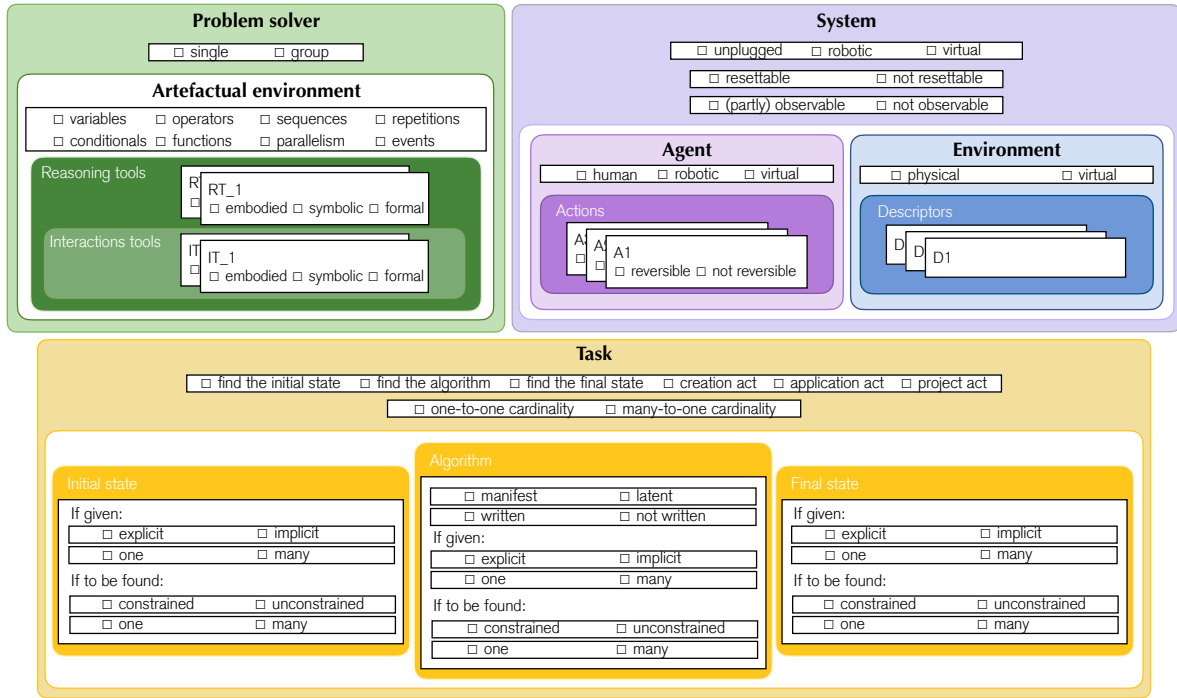


Fig. 3 Graphical template for the analysis of CTPs. Template suitable for graphically analysing the components of any CTP according to our framework. Colours represent the CTP components and characteristics following the same colour scheme of Fig. 2.

2.2 A catalogue of computational thinking (CT) skills

In this study, we have identified a set of CT competencies commonly used to assess student abilities in CT. These competencies, also called dimensions or skills, represent the fundamental abilities students need to develop to solve CT problems effectively. We decided to draw from various state-of-the-art competency models to select and define our taxonomy of CT competencies rather than relying on a single model. Our selection process was inspired by the literature reviews of Tikva and Tambouris (2021) and Bocconi et al. (2016, 2022), which provides a comprehensive overview of CT skills and their potential for compulsory education. Of great importance is the framework of Brennan and Resnick (2012). They proposed a list of CT skills divided into three dimensions: computational concepts, practices, and perspectives. This characterisation is often used in literature, but it was limited for our purposes because it only considered virtual activities, while we also investigated robotics and unplugged ones. Therefore, we extended this list by partially following the taxonomy of CT in Science, Technology, Engineering and Math (STEM) courses proposed by Weintrop et al. (2016). This classification consists of four main categories: data practices, modelling and simulation practices, computational problem-solving, and systems thinking practices. Another work we based on is that of Shute et al.

(2017). They developed a competence model based on a review of the relevant definitions of CT in the literature, including those of Brennan and Resnick (2012) and Weintrop et al. (2016).

To facilitate the assessment of competencies and ensure that it is focused and efficient, we have organised CT skills into a hierarchy with layers of dimensions and sub-dimensions, depicted in Fig. 4. The skills taxonomy is based on the activity dimension of the CT-cube framework by Piatti et al. (2022), introduced in the previous section, representing the individual's role in the cognitive system at each moment of the task. The activity sub-dimensions have been developed mainly using the frameworks of Brennan and Resnick (2012); Shute et al. (2017); Weintrop et al. (2016). Competencies related to the activity dimension involve a wide range of operations and cognitive processes. For example, the “problem setting” competence may require recognising and understanding various components of the framework within the given CTPs, as well as modelling the problem. The “algorithm” competence may involve the comprehension and exploitation of different instructions with varying difficulty levels, often influenced by the type of artifactual environment involved. The “assessment” competence may consider determining whether a solution is correct or its quality is satisfactory. Table 1 summarises the competencies of the

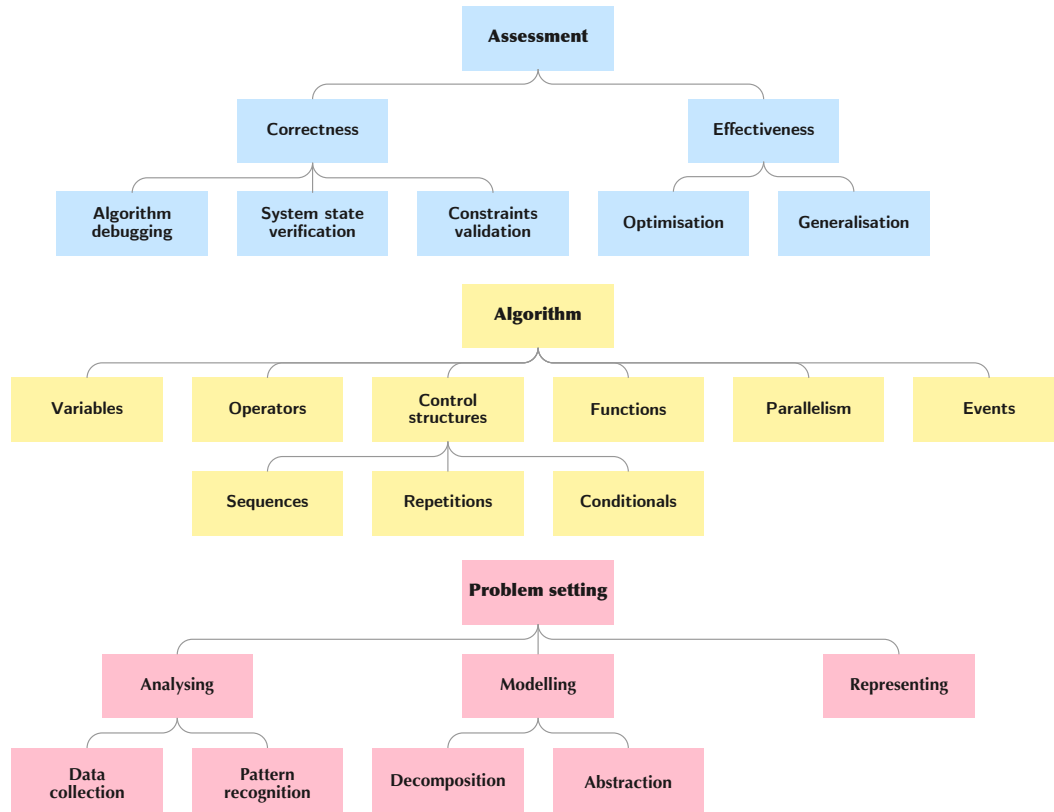


Fig. 4 Visualisation of our taxonomy of CT competencies. The overall structure is based on the CT-cube (Piatti et al. 2022). The sub-skills are derived from validated CT models (Brennan and Resnick 2012; Weintrop et al. 2016; Shute et al. 2017). The three skills groups are represented with the same colour scheme used for the CT-cube dimensions in Fig. 1.

first level of the hierarchy, defining all the possible values that the activity dimension can assume.

Table 1 Main competencies of the framework and their definition. The skills listed are based on the values of the activity dimension of the CT-cube framework (Piatti et al. 2022).

Competence	Definition
<i>Problem setting</i>	Recognise, understand, reformulate or model a CTP and its components so that its solution can be computed. ^a
<i>Algorithm</i>	Conceive and represent a set of agent's actions that should be executed by a human, artificial or virtual agent to solve the task. ^b
<i>Assessment</i>	Evaluate the quality and validity of the solution in relation to the original task. ^c

^a See Table 2 for sub-competencies.

^b See Table 3 for sub-competencies.

^c See Table 4 for sub-competencies.

Tables 2 to 4 provide a detailed breakdown of the sub-dimensions for the three possible values of the CT-

Table 2 Problem setting sub-competencies and their definition. The skills listed are based on leading-edge competence models (Brennan and Resnick 2012; Shute et al. 2017; Thalheim 2000; Weintrop et al. 2016; Wing 2011; Bocconi et al. 2016; Selby and Woollard 2013; Angeli et al. 2016; Csizmadia et al. 2015; Selby 2014; Barr and Stephenson 2011).

Competence	Definition
<i>Analysing</i>	Collect, examine and interpret data about the system: environment descriptors and agent actions.
<i>Data collection</i>	Gather details about the system.
<i>Pattern recognition</i>	Identify similarities, trends, ideas and structures within the system.
<i>Modelling</i>	Restructure, clean and update knowledge about the system.
<i>Decomposition</i>	Divide the original task into sub-tasks that are easier to be solved.
<i>Abstraction</i>	Simplify the original task, focus on key concepts and omit unimportant ones.
<i>Representing</i>	Illustrate or communicate information about the system and the task.

cube activity class (problem setting, algorithm, assessment). In the tables, each row represents a specific skill. The parent skill is distinguished from the lower-level skills by a dashed line, while the lower-level competencies of a particular competence are separated by dotted lines. This helps to visually organise the table to differentiate between the different skill levels and make it easier to understand their relationships.

Table 3 Algorithm sub-competencies and their definition. The skills listed are based on leading-edge competence models (Brennan and Resnick 2012; Cui and Ng 2021; Rodríguez-Martínez et al. 2020; Bocconi et al. 2016, 2022; Shute et al. 2017). The following definitions are about the algorithmic concepts related to the skill since the skill definition varies based on the artefactual environment: in embodied contexts, it involves recognising or describing concepts through senses; in symbolic contexts, it involves applying them; and in formal contexts, it involves understanding properties to structure complex algorithms.

Competence	Definition
<i>Variables</i>	Entity that stores values about the system or intermediate data.
<i>Operators</i>	Mathematical operators (such as addition (+), subtraction (−) etc.), logical symbols (such as and (&), or (), and not (!)) or for comparison (such as equal to (=), greater than (>), and less than (<)), or even specific commands or actions (such as “turn left” or “go straight”).
<i>Control structures</i>	Statements that define the agent actions flow’s direction, such as sequential, repetitive, or conditional.
<i>Sequences</i>	Linear succession of agent actions.
<i>Repetitions</i>	Iterative agent actions.
<i>Conditionals</i>	Agent actions dependent on conditions.
<i>Functions</i>	Set of reusable agent actions which produce a result for a specific sub-task.
<i>Parallelism</i>	Simultaneous agent actions.
<i>Events</i>	Variations in the environment descriptors that trigger the execution of agent actions.

It is important to note that the activation of the activity sub-dimensions, especially the algorithmic one, is closely related to the type of artefactual environment being considered. The task-related artefactual environment combines the tools given and the type of problem involved. The activation of a competence may vary depending on the context in which it is being applied or whether the task is performed with embodied, symbolic, or formal artefacts. When considering algorithmic sub-

Table 4 Assessment sub-competencies and their definition. The skills listed are based on leading-edge competence models (Brennan and Resnick 2012; Shute et al. 2017; Weintrop et al. 2016; Bocconi et al. 2016).

Competence	Definition
<i>Correctness</i>	Assess whether the task solution is correct.
<i>Algorithm debugging</i>	Evaluate whether the algorithm is correct, identifying errors and fixing bugs that prevent it from functioning correctly.
<i>System states verification</i>	Evaluate whether the system is in the expected state, detecting and solving potential issues.
<i>Constraints validation</i>	Evaluate whether the solution satisfies the constraints established for the system and the algorithm, looking for and correcting eventual problems.
<i>Effectiveness</i>	Assess how effective is the task solution.
<i>Optimisations</i>	Evaluate whether the solution meets the standards in a timely and resource-efficient manner, and eventually identify ways to optimise the performance.
<i>Generalisation</i>	Formulate the task solution in such a way that can be reused or applied to different situations.

competencies, for example, to activate these skills it is necessary to consider the tool used to represent the algorithm and the type of abstraction of the reasoning required. In embodied environments, knowledge is represented through sensory experiences, such as seeing, hearing, or touching. In this context, these skills may be activated simply by recognising and describing algorithmic concepts using physical interactions. In symbolic environments, knowledge is represented using symbols, such as words and numbers, or languages, such as natural or formal languages. Common types of formal language include those used to code, such as block-based and textual programming languages. In this context, reasoning requires the problem solver to be able to apply these competencies to solve problems and accomplish tasks. In formal environments, knowledge is represented with abstractions, such as mathematical models, logical systems and proofs. In this context, it is necessary to have a deeper understanding of how these skills work and what their properties are to be able to structure and apply them effectively, creating a complex system.

2.3 Map CTP characteristics to CT skills

To formalise our framework, clarifying the role of CTP characteristics in CT skills assessment, we utilise the previously established definition of CTP to identify the components of these problems and their characteristics and to understand how different features may influence the assessment of CT competencies. We establish a direct link between our catalogue of CT competencies and the proposed framework, demonstrating how particular CT dimensions are more likely to be activated when solving CTPs with specific characteristics. The link between the two is discussed in detail in Appendix A, which explains how the various characteristics of CTPs can impact the development of CT competencies and how certain skills are more likely to be used when solving CTPs with specific characteristics. This approach enables the assessment of student abilities in a targeted and efficient manner.

Our analysis of the relationships between characteristics and competencies revealed four different types of connections. From the perspective of the features, a certain characteristic can either be required to activate a certain competence, prevent its activation, trigger its activation or be irrelevant to its activation. Conversely, from the perspective of the competencies, a skill is activated if it has all required features and none of the preventing ones, can be encouraged by triggering features or is inhibited if it has a preventing feature or only irrelevant features. The relationship between features and skills can vary based on the type of environment in which the problem is presented. Table 5 provides a comprehensive explanation of the symbols used to represent the relationship between the characteristics of a CTP and CT competencies.

Table 6 has been realised to offer a practical understanding of the relationship between different features of CTPs we identified and our catalogue of CT competencies enabling us to assess student abilities in a targeted and efficient manner. Each column is a characteristic of the CTP and every row represents a skill. Overall, this table provides a clear visual representation of how different features of the CTP can influence the assessment of CT competencies and how the different dimensions of CT skills are more likely to be activated when solving CTPs with specific characteristics.

This framework, supported by Table 6, is intended to be used as a tool to analyse existing CTPs, understand which CT competencies can be assessed by the available features, evaluate the problem effectiveness for intended educational purposes and eventually revise it if it is unsuitable. To use the framework, the first step is to identify the CTP profile and list all of its features.

Table 5 Overview of the notation for representing the relationship between characteristics and skills.

Symbol	Meaning
✓	The characteristic is required for the competence activation
✗	The characteristic prevents the competence activation
+	The characteristic promotes the competence activation
<i>none</i>	The characteristic is irrelevant for the competence activation
✓*	The characteristic is one of the possible characteristics required for the competence activation
✓ ^{SF} / ✗ ^{SF}	The characteristic is required for / prevents the competence activation in the symbolic and formal environments
✓ ^F / ✗ ^F	The characteristic is required for / prevents the competence activation in the formal environment

If the CT competencies to be measured have not been specified in the activity, the framework can be used to determine which skills can be assessed from the available features. On the other hand, if the CT competencies are outlined, the framework can be used to compare the problem's characteristics with those associated with the specified competencies and determine the CTP effectiveness for intended educational purposes. The effectiveness of a CTP can be classified into five categories based on the relationship between its features and the CT competencies.

Definition 11 (Minimal CTP) A complete and optimised CTP in which its characteristics perfectly match the features essential to activate the CT competence it is intended to elicit.

In other words, the problem is designed to activate the intended CT skill with the minimum required features without any irrelevant or distracting elements.

Definition 12 (Extensive CTP) A complete and rich CTP that, in addition to having all of the necessary characteristics to activate a certain CT competence, also has features that act as a stimulus to activate that skill or others.

These additional features make the problem more engaging and provide additional opportunities for stu-

Table 6 Comprehensive overview of the relationship between different CTP characteristics and CT competencies. The table shows the relationship between the characteristics of CTPs (columns) and CT competencies (rows). The CTP features considered include the tools' functionalities, the system's property, and the task trait. The meaning of the symbols used is provided in Table 5. The same colour schemes used for the CT-cube dimensions in Fig. 1 and for the CTP components in Fig. 2 are employed to present the skills and features, respectively.

	Tool functionalities							System				Task												
	Variables	Operators	Sequences	Repetitions	Conditionals	Functions	Parallelism	Events	System resettable	System not resettable	System (partially) observable	System not observable	Initial or final state to be found	Algorithm to be found	One-to-one cardinality	Many-to-one cardinality	Explicit elements	Implicit elements	Unconstrained elements	Constrained elements	Algorithm manifest	Algorithm latent	Algorithm written	Algorithm not written
Data collection	✓							+	+	+	+	+			+	+		+		+	+	+	+	+
Pattern recognition	+		+	✓*	+	✓*			+	+	+	+			+	+	+	+	+	+	+	+	+	+
Decomposition	+	+	✓*	+	+	✓*	+		+	+		+			+	+		+	+	+	+	+	+	+
Abstraction	✓		+	+	+	✓				+		+			+	+	+	+	+	+	+	+	+	+
Data representation	✓		+	+	+	+				+		+			+	+		+	+	+	+	+	+	+
Variables	✓	+	+	+	+	+	+	+			+	+	✓ ^F	+	+	+	+	+	+	+	+		+	+
Operators	+	✓	+	+	+	+	+	+			+	+	✓ ^F	+		+	+	+	+	+	+		+	+
Sequences	+	+	✓	+		+					+	+	✓ ^F			+	+	+	+	+	+		+	+
Repetitions	+	+	+	✓		+					+		✓ ^F		+	+	+	+	+	+	+		+	+
Conditionals	+	+			✓			+			+	+	✓ ^F	+	+	+	+	+	+	+	+		+	+
Functions	+	+	+	+		✓					+		✓ ^F	+	+	+	+	+	+	+	+		+	+
Parallelism	+	+					✓				+		✓ ^F		+	+		+		+	+		+	+
Events	+	+			+			✓			+		✓ ^F				+		+		+		+	+
Algorithm debugging	+	+	+	+	+	+	+	+	✓	✗	+		✓				+				✓	✗	✓ ^F	✗ ^F
System state verification									✓	✗	+		✓				+			✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F	
Constraints validation	+	+	+	+	+	+	+	+	✓	✗	+								✗	✓				
Optimisation	+	+	+	+	+	+	+	+	✓	✗	+													
Generalisation	✓		+	+	+	✓	+		✓	✗	+				+		+		+					

dents to demonstrate their CT abilities. The problem is considered complete and rich, as the additional features enhance the overall learning experience and provide a more comprehensive evaluation of the student's skills.

Definition 13 (Unfocused CTP) A CTP in which some of the features are irrelevant to activate the skill it is intended to elicit.

These irrelevant features can create confusion and distract the students from focusing on the essential aspects of the problem, hindering the optimisation of the problem in activating the desired competencies and affecting the accuracy of the assessment. An unfocused CTP should be revised to remove these unnecessary features to improve its effectiveness.

Definition 14 (Adjustable CTP) A CTP where some essential features required to activate a certain competence are missing.

This type of problem can be modified to include the missing features, optimising it to activate the relevant CT skills. This can be done by modifying the problem statement, the system, or the artefactual environment.

Definition 15 (Unsuitable CTP) A CTP that is not appropriate for its intended purpose, which lacks some essential features but has some unwanted.

This type of problem makes it difficult or impossible to effectively used to assess certain CT competencies and

it would require too many changes to be useful, and it may be better to design or choose another problem instead.

Additionally, Table 6 can be used when creating or choosing a new CTP to assess specific CT competencies. The first step, in this case, would be to define the skills to be evaluated and then use the table to list the necessary features and those that are not needed. The table can also be used to find existing problems that match the list of characteristics or to design a new activity from scratch.

3 Results

To examine and validate our method in-depth, we applied the proposed framework to a range of computational thinking (CT) activities that are widely recognised as representatives in educational settings. We focused on three categories of computational thinking problems (CTPs), unplugged, robotic, and virtual activities, to provide a more nuanced understanding of each type of activity’s unique features and challenges. The selection of activities serves as a means to demonstrate the effectiveness and practicality of the framework and is not exhaustive. Through this analysis, we aim to provide a comprehensive account of the framework’s applicability and identify best practices and areas for improvement in the design and assessment of CT activities, with the ultimate goal of contributing to the field of CT education.

For each CTP presented in this section, we provided in Appendix B the graphical template used for their analysis, describing the component and the characteristics of the CTP, and the table that summarises the relationship between the characteristics of the CTPs and the CT competences that can be activated or not.

3.1 Unplugged activities

In the context of CT, an unplugged activity is an activity that does not involve the use of a computer or technology (Brackmann et al. 2017; Del Olmo-Muñoz et al. 2020). As per our framework, unplugged activities refer to CTPs where the agent is a human rather than a virtual or robotic agent. These activities have a common feature of involving physical and non-technological artefactual environments, as they require manipulating physical objects rather than using technology for manipulation. Unplugged activities are designed to help students develop CT skills often related to problem setting, such as pattern recognition and fundamental com-

puter science concepts through hands-on and tangible activities before they are introduced to more abstract, technology-based activities (Bell et al. 2009). Examples of unplugged CTPs include puzzles, games, and other activities that involve manipulating physical materials, such as blocks or cards.

3.1.1 Cross Array Task

The Cross Array Task is an unplugged activity designed by Piatti et al. (2022) using the CT-cube to assess the development of algorithmic skills in compulsory schools.

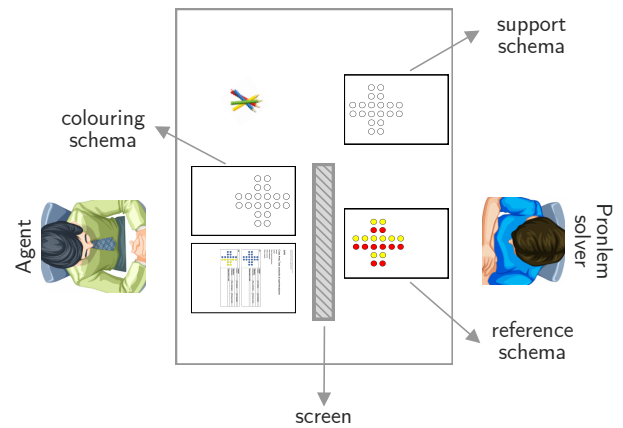


Fig. 5 The Cross Array Task activity adapted from Piatti et al. (2022). The task requires the problem solver to instruct the agent to reproduce a reference schema solely through verbal communication, with the option of supplementing instructions via gesturing on a support schema if deemed necessary. A removable screen separates the participant and researcher to regulate potential visual cues.

Components The Cross Array Task, illustrated in Fig. 5, involves a student and a researcher in a classroom setting, seated at a table and separated by a removable screen. The student has to communicate an algorithm to colour a white cross array to match a reference schema.

- *Problem solver*: the student who has to communicate an algorithm corresponding to the sequence of instructions to reproduce the colouring of the reference schema. The artefactual environment comprises cognitive tools such as the support and the colouring schemes, available to the problem solver to reason about the task. Additionally, the problem solver can interact with the system to communicate the algorithm. This can be achieved using a natural language such as the voice (symbolic) or gestures (embodied) on the empty cross array. Moreover, by removing the screen that separates

the problem solver from the agent, he can have visual feedback (embodied) of the cross array being coloured.

- *Agent*: the researcher, executor of the problem solver’s instructions, responsible for filling the colouring schema according to the problem solver’s algorithm. The agent’s actions are not resettable.
- *Environment*: the cross array to be coloured, whose state is described by the colour of each dot (white, yellow, blue, green, or red).
- *Task*: find the algorithm. The system’s state is defined by the colouring cross status, initially white and, at the end the same as the reference schema. The algorithm is the set of agent instructions to achieve this transformation.

Characteristics The characteristics of this activity have been analysed using the graphical templates shown in Fig. B.1 in Appendix B.

- *Tool functionalities*: voice and gestures provide various functionalities associated with algorithmic concepts, suitable to design the algorithm, including (i) *variables* can represent different colours of the cross array dots; (ii) *operators* are used to change the colour of the dots performing actions such as colouring a dot, a row, a square and so on; (iii) *sequences* determine the order in which the actions should be executed to achieve the desired outcome; (iv) *repetitions* allow for repeating specific sequences of operations, such as colouring the first column in red and repeating it every two columns; (v) *functions* consist of operations that perform a specific task and can be applied to different inputs, for example, creating a pattern of alternating red and yellow dots in a square and applying it to different positions of the cross array; (vi) *parallelism* involves executing multiple actions simultaneously and can be associated with using symmetries to describe the pattern.
- *System resettability*: the system is not resettable since it is impossible to reverse the agent’s actions.
- *System observability*: the system is partially observable since the cross array being coloured by default is not seen until the end of the task unless the problem solver demands otherwise.
- *Task cardinality*: the task has a one-to-one mapping, with given one initial and one final state, and an algorithm to be found.
- *Task explicitness*: all elements are given explicitly.
- *Task constraints*: the algorithm is unconstrained.
- *Algorithm representation*: the algorithm is represented through voice commands or gestures. It is considered manifest, because it can be seen, but not written since it is not stored in a permanent format.

Enabling features for competencies development The relationship between features and skills is summarised in Appendix B in Table B.1. This paragraph explores the enabling characteristics that support the development of competencies within the task.

- *Problem setting*: all competencies can be activated thanks to the presence of variables, sequences, repetitions and functions in the tool functionalities. The presence of many tool functionalities, the non-resetability of the system and the algorithm representation positively affect and boost problem setting skills. The system observability supports data collection and pattern recognition. The one-to-one cardinality, in addition, stimulates decomposition. The explicit and unconstrained definition of the task elements also promotes pattern recognition, decomposition and abstraction.
- *Algorithm*: all competencies associated with the algorithmic concepts enabled by the tool functionalities, meaning variables, operators, sequences, repetitions and functions, can be activated in all three artefactual environments and promote one another. The form of representation of the algorithm, the system observability, and the explicit and unconstrained definition of the task elements further enhance these. The one-to-one cardinality helps to enhance some of these skills as well.
- *Assessment*: since the system is not resettable, no assessment skills can be developed.

Inhibiting features for competencies development

- *Conditionals and events*: non-activable as these functionalities are unavailable in the platform. A way to make conditionals available in the tool functionalities would be allowing the problem solver to change a dot colour, for example by communicating instructions such as: “if the dot is red, then colour it yellow”. By doing this, the problem solver engages with the concept of conditionals and can develop their algorithmic skills. The completion of each row in the cross array can be considered an event. The problem solver can specify that they want to fill the cross line by line, and once a line is complete, the researcher will move on to the next line. This allows the problem solver to list only the sequence of colours without repeating the instructions for where to go. The change in the environment (completing a row) triggers the researcher to move to the next row. Using conditionals and events can greatly enhance the complexity of the solutions that can be generated and help develop advanced CT skills.
- *Assessment skills*: the inability to reset the system impairs the development of the student’s skills. One

possible solution to this issue is enabling the student to reset the colouring scheme using a voice command. This would return the schema to its initial blank state, allowing the student to start the task from the beginning and practice their assessment skills. To develop system state verification, it is also essential to not reveal the initial or final states. Moreover, constraints should be imposed on the algorithm to develop constraint validation skills, for example, limiting the use of specific operators or the number of times they can be used, allowing the problem solver to develop the ability to think about the constraints and limitations in their algorithms.

3.1.2 Graph Paper Programming

Graph Paper Programming is an unplugged activity from Code.org (2015), a nonprofit organisation that aims to provide students with the opportunity to learn computer science as part of their education, offering various activities designed to increase diversity in computer science and reach students at their skill level and in ways that inspire them to continue learning.

The Graph Paper Programming activity can be divided into two parts, each with a different task. In the first part, the student is given a 4×4 grid of white and black squares and asked to write explicit instructions for another classmate to reproduce the image without letting the other person see the original drawing. In the second part, the same student follows the instructions they previously wrote to reproduce the image. By dividing the activity into these two parts, we can gain a deeper understanding of the cognitive processes and skills involved in each task and understand the potential for this activity to support the development of CT abilities.

Use the symbols to write a program that would draw the image.

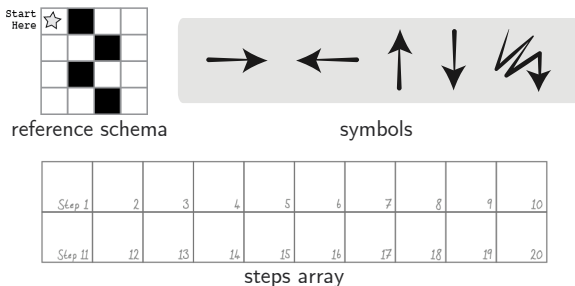


Fig. 6 The first part of the Graph Paper Programming (GPP) activity, adapted from Code.org (2015). The task requires the problem solver to instruct the agent to reproduce the reference schema with instructions written on a steps array using a predefined set of arrow symbols.

Components (part 1) The first part of the activity is illustrated in Fig. 6.

- *Problem solver:* the student who writes the set of instructions for the agent to follow. The artefactual environment comprises cognitive tools such as the reference schema (embodied) and the support and the set of arrow symbols (embodied) available to the problem solver to reason about the task. Additionally, the problem solver can interact with the system to communicate the algorithm, writing the arrow symbols in the steps array (symbolic). These can be considered as a programming language and its programming platform.
- *Agent:* the other student who executes the problem solver's instructions by filling the colouring schema accordingly. Its actions are not resettable.
- *Environment:* the schema to be coloured, described by the colour of each square (white or black).
- *Task:* find the algorithm. The system's state is defined by the colouring schema status, initially white and, at the end the same as the reference schema. The algorithm is the set of agent instructions to achieve this transformation.

Read the program above and draw the image that it describes.

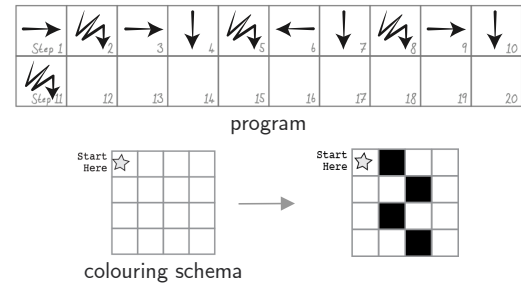


Fig. 7 The second part of the Graph Paper Programming (GPP) activity, adapted from Code.org (2015). The task requires the problem solver to fill the empty colouring schema following the program provided. The figure illustrates the expected final state.

Components (part 2) The second part of the activity is illustrated in Fig. 7.

- *Problem solver and Agent:* they overlap and correspond to the student who follows the instructions to recreate the image. The only action the agent can perform is to paint the colouring schema without the possibility of undoing it. The artefactual environment comprises cognitive tools such as the colouring schema (embodied) available to the problem solver to reason about the task. As before, the arrow symbols on the steps array (symbolic) are also used to interact with the system. Moreover, being

the agent and the problem solver overlapped, visual feedback (embodied) is always given.

- *Environment*: the schema to be coloured, described by the colour of each square (white or black).
- *Task*: find the final state. The system's state is defined by the colouring schema status, initially white. However, the final state is not given and has to be found using the provided algorithm.

Characteristics The characteristics of this activity have been analysed using the graphical template presented in Appendix B. Fig. B.2 refers to the first part of the activity, while Fig. B.3 refers to the second.

- *Tool functionalities*: in both parts of the activity, the tools provide various functionalities associated with algorithmic concepts suitable to design the algorithm, including (i) *variables* can represent different colours of the schema squares; (ii) *operators* correspond to the arrow symbols used to instruct the agent to move from one square to another, determining whether a square is coloured black or white; (iii) *sequences* determine the order in which the actions should be executed to achieve the desired outcome; (iv) *repetitions* allow for repeating specific sequences of operations; (v) *functions* can be represented by a group of instructions that perform a specific task, such as colouring a particular shape on the grid, that can be used multiple times.
- *System resettability*: the system is not resettable since it is impossible to reverse the agent's actions.
- *System observability*: in the first part of the activity, the system is not observable, as there is no visual feedback about the agent's actions; in the second part of the activity, the visual feedback consented thanks to the problem solver and agent overlapping, makes the system totally observable.
- *Task cardinality*: both tasks have a one-to-one mapping, with one initial state, final state and algorithm.
- *Task explicitness*: the task elements are explicit.
- *Task constraints*: the final state is unconstrained.
- *Algorithm representation*: the algorithm is manifest and written, represented through the arrow symbols written in the steps array.

Enabling features for competencies development This paragraph explores the enabling characteristics that support the development of competencies within this CTP. The relationship between features and skills in the two parts of the activity are summarised in Appendix B in Tables B.2 and B.3.

- *Problem setting*: all competencies can be activated thanks to the presence of variables, sequences, repetitions and functions in the tool functionalities.

The presence of many tool functionalities, the non-resettability of the system and the algorithm representation positively affect and boost problem setting skills. In the first part of the activity, the inability to observe the systems further supports the development of all these competencies, while in the second part, the system observability sustains only data collection and pattern recognition. The one-to-one cardinality in addition stimulates data collection and pattern recognition but also decomposition. The explicit and unconstrained definition of the task elements also promotes pattern recognition, decomposition and abstraction.

- *Algorithm*: the competencies associated with algorithmic concepts, including variables, operators, sequences, repetitions, and functions, can be developed in the first part of the activity through the use of the tool functionalities in all artefactual environments. However, in the second part of the activity where the algorithm is given, these competencies cannot be developed in the formal environment but only in the embodied and symbolic environments. The algorithm representation, the system observability, and the explicit and unconstrained definition of the task elements further enhance all these skills.
- *Assessment*: since the system is not resettable, no assessment skills can be developed.

Inhibiting features for competencies development

- *Conditionals, parallelism and events*: non-activable as these functionalities are unavailable in the platform. Activating conditionals would be allowed by providing a new arrow symbol that determines the colour of a square based on certain conditions, for example, the colour of the square above. Parallelism can be enabled by operating a new arrow symbol instructing the agent to colour two squares simultaneously. An event that could be taken into account is that every time a cell is filled with black, the instructions in the steps array move to the next row rather than continuing from that specific cell.
- *Assessment skills*: the inability to reset the system impairs the development of the student's skills. In the first part of the activity, where the problem solver is writing the set of instructions for the agent, to solve this issue the system can be reset by simply starting over with a new blank graph array and writing a new set of instructions, or simply offering the possibility to use pencil and eraser. In the second part of the activity, where the problem solver and the agent are the same people, the system can be reset by either using a new colouring schema or

erasing the previously produced image and starting over.

3.1.3 Triangular Peg Solitaire

Triangular Peg Solitaire is a strategy game that can be played in two variants: the classic, on a triangular board with 15 holes and pegs, and the paper and pencil modality. The board is initially filled with pegs, except for one hole, which is left empty (see the top of Fig. 8). This game is often used to teach problem-solving, logic and strategy skills, requiring the player to determine the most efficient sequence of moves to remove all the pegs on the board except one. Research has shown that the second activity variant can effectively promote problem-solving skills even in older students (Barbero 2020). The game is played by following the rules, which dictate that a peg can only be moved by jumping over a neighbouring peg on the diagonal or horizontal lines (see the bottom of Fig. 8). By analysing the two versions of this game, we aim to understand their impact on promoting problem-solving and critical thinking skills, evaluating advantages and limitations and providing insights into how they can be used for CT development.

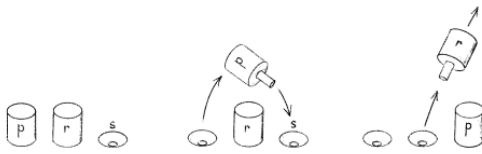
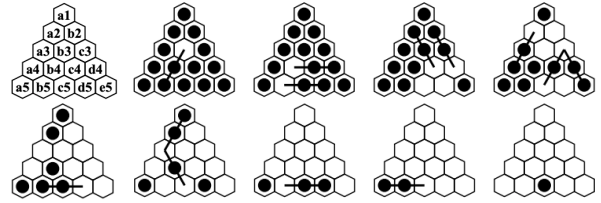


Fig. 8 The Triangular Peg Solitaire. The game is played on a board containing 15 spots, with 14 pegs placed on it at the start of the game (top). The task requires the problem solver to strategically move one peg at a time to eliminate all other pegs on the board until only one remains (bottom), jumping a peg over a neighbouring peg on the diagonal or horizontal lines, with the constraint that there must be a free landing spot for the jumping peg (adapted from Berlekamp et al. (2004)).

Components (board variant) The first variant of the activity, illustrated in Fig. 8, requires solving the game on a physical board.

- *Problem solver and Agent:* they overlap and correspond to the player who must determine the most efficient sequence of moves to remove all the pegs on the board. The only action the agent can perform is to move the pegs on the board without the possibility of undoing it. The artefactual environment comprises tools for reasoning and interacting with the system. Being the agent and the problem solver overlapped, visual feedback (embodied) of the state of the board is always given. Moreover, the problem solver can physically interact with the system by moving the pegs on the board (embodied).
- *Environment:* the wooden board, described by the number of pegs on it.
- *Task:* find the algorithm. The system's initial state is the board with 14 pegs. The final state is the board with one peg. The algorithm to be found specifies the sequence of moves to remove all the pegs.



vacate b3
b5-b3, d4-b4, d5-b5, b2-d4, a2-c4, a4-a2,
e5-c3-c5, b5-d5, a1-a3-c5, d5-b5, a5-c5
11 moves

Fig. 9 A Triangular Peg Solitaire solution adapted from Bell (2007, 2008) and Barbero and Gómez-Chacón (2018). The task requires the problem solver to solve the game using paper and pencil by meticulously documenting their entire thought process. The solution can be presented in multiple ways, such as graphically using a Cartesian notation (top) or by numbering the boxes progressively and expressing the movements used (bottom).

Components (paper & pencil variant) The first variant of the activity, illustrated in Fig. 9, requires solving the game with paper and pencil by documenting the thought process and devising a winning strategy.

- *Problem solver and Agent:* they overlap and correspond to the player who must determine the most efficient sequence of moves to remove all the pegs on the board. The only action the agent can perform is to write the thinking process and strategy to remove the pegs on the board without the possibility of undoing it. The artefactual environment comprises tools for reasoning and interacting with the system. Being the agent and the problem solver overlapped, visual feedback (embodied) of the state

is always given. Moreover, the problem solver can physically interact with the system by writing the thinking process (symbolic).

- *Environment*: the board drawn in the thinking process, described by the number of pegs on it.
- *Task*: find the algorithm. The system's initial state is the board with 14 pegs. The final state is the board with one peg. The algorithm to be found specifies the sequence of moves to remove all the pegs.

Characteristics The characteristics of this activity have been analysed using the graphical template presented in Appendix B. In particular, Fig. B.4 refers to the board version of the activity, while Fig. B.5 refers to the paper and pencil variant.

- *Tool functionalities*: in both variants of the activity, the tools provide various functionalities associated with algorithmic concepts suitable to design the algorithm, including (i) *variables* can represent the state of the board and in particular the number of pegs on it; (ii) *operators* correspond to the moves made by the player to change the state of the board by removing or moving pegs from one hole to another; (iii) *sequences* determine the order of moves made by the player; (iv) *repetitions* allow for repeating certain moves or sequences of moves; (v) *conditionals* refer to the possible decisions that the plays may need to make, such as where to jump in one direction or another; (vi) *functions* can be represented by a group of instructions that perform a specific task, for example, a function to delete pegs in a row which can be applied to several rows.
- *System resettability*: in the board variant, the system is not resettable meaning that once the player has made a move, it cannot be undone. However, the system is resettable in the paper and pencil version, even though the player's actions are not reversible. This is because the informal setting, in which the player documents their thought process, allows for experimentation and exploration without fear of judgement or negative consequences. In other words, the player can freely make mistakes, express uncertainty, and experiment with different strategies without permanently impacting the game.
- *System observability*: in both variants, the system is observable because the agent and problem solver, that overlap, can see the state of the board anytime.
- *Task cardinality*: both activity variants have a one-to-one mapping, with one initial state, one final state and one algorithm.
- *Task explicitness*: the initial state of the system is given explicitly, while the final one is given implicitly

since the task instruction does not specify which is the position of the last remaining peg.

- *Task constraints*: the algorithm is constrained by the game rules which dictate that a peg can only be moved by jumping over a neighbouring peg on the diagonal or horizontal lines and that there must be a free landing spot for the jumping peg.
- *Algorithm representation*: in the board variant of the game, the algorithm is latent since it is performed physically through the player's moving the pegs, it is not permanently recorded and cannot be revisited. On the other hand, in the paper and pencil modality, the player writes down the algorithm, and it becomes a permanent record that can be reviewed and used as a reference. This allows the player to experiment freely and change their approach without having to start over each time. The written representation of the algorithm in the paper and pencil modality provides a clear and concrete way to represent the player's thought process and strategy.

Enabling features for competencies development This paragraph explores the enabling characteristics that support the development of competencies within this CTP. The relationship between features and skills in the two activity variants are summarised in Appendix B in Tables B.4 and B.5.

- *Problem setting*: all competencies can be activated in both variants of the game thanks to the presence of variables, sequences, repetitions and functions in the tool functionalities. The presence of many tool functionalities, the non-resettability of the system (in the board version of the game), the implicit and constrained definition of the task elements and the algorithm representation positively affect and boost problem setting skills. The system's observability sustains data collection and pattern recognition skills, while the one-to-one cardinality and the resettability of the system (in the paper and pencil variant of the game) also stimulate decomposition.
- *Algorithm*: the competencies associated with algorithmic concepts, including variables, operators, sequences, repetitions, conditionals and functions, can be developed in all artefactual environments. The system observability, the implicit and constrained definition of the task elements, and the manifest written representation of the algorithm (in the paper and pencil variant of the game) can further enhance these skills.
- *Assessment*: no assessment skills can be developed in the board variant of the activity since the system is not resettable. In the paper and pencil version of the activity, algorithm debugging can be activated

in all artefactual environments due to the resetability of the system and the manifest and written representation of the algorithm; constraint validation can be developed because there are constraints on the algorithm that can be verified since the system is resettable; optimisation can be activated as it only requires the resetability of the system; generalisation can be activated through the system's resetability and the presence of variables and functions. Tool functionalities as well as the system observability help develop these competencies.

Inhibiting features for competencies development

- *Parallelism and events*: non-activable as these functionalities are unavailable in the platform. It is possible to develop parallelism and events in the game by incorporating a variant where multiple pegs can be moved simultaneously or by introducing multiple players who can make moves simultaneously. A way to incorporate events would be to use technology such as a computer program or app to play the game, creating programmed events that the player's actions could trigger. For example, if the player removes a certain peg, the computer could trigger an event that changes the board's appearance or displays a message. However, it is important to note that these changes would result in a different game with a different set of objectives and challenges and might not necessarily have the same educational benefits as the original Triangular Peg Solitaire.
- *Assessment skills*: the inability to reset the system impairs the development of the student's skills in the first variant of the activity. In the paper and pencil version, only system state verification is non-activable because both the initial and final states are provided. The task can be adjusted by modifying the game such that only the initial state is provided and the final state is unknown. For example, letting the player determine the specific peg position for the last peg. This would make the task more challenging and require the player to develop their skills in system state verification.

3.1.4 Computational Thinking Test (CTt)

The Computational Thinking Test (CTt) is an assessment tool designed to evaluate the CT skills of students between the ages of 12 and 13 (Román-González 2015; Román-González et al. 2017b,a; Román-González et al. 2018). It aligns with the work of researchers such as Brennan and Resnick (2012); Kalelioğlu (2015), who have identified key computational concepts related to algorithmic skills. Additionally, the CTt is designed to

align with the standard interfaces used by organisations universally recognised in this context, such as Code.org, which utilise visual blocks to teach coding. The test consists of 28 multiple-choice questions. However, we will only analyse two.

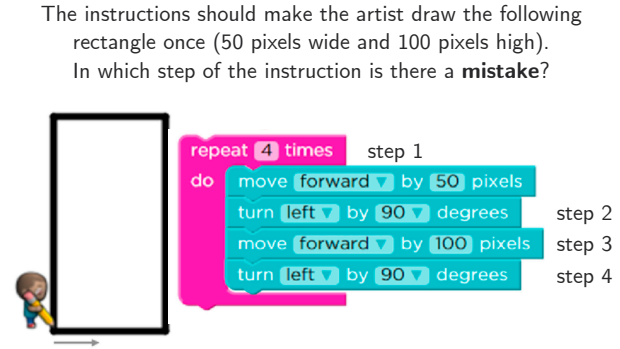


Fig. 10 Item 7 of the Computational Thinking Test (CTt) adapted from Román-González et al. (2017b). The task requires the problem solver to correct a set of instructions that should make the agent draw a rectangle.

Components (item 7) The first CTP analysed, called “item 7”, is depicted in Fig. 10 and has been designed to evaluate the student's ability to identify and fix errors in code, in a script that does not involve computational nesting concepts but only the concept of repetitions.

- *Problem solver*: the student taking the test presented with a wrong code script that must be fixed. The artefactual environment comprises only cognitive tools, thus is impossible to interact with the system, which is considered static. The reasoning tool provided includes the sketch in the problem description (embodied) and the visual blocks interface (symbolic), which allows the student to think about the instructions and imagine to test different solutions.
- *Agent*: the artist responsible for drawing the rectangle according to the instructions provided. It is an abstract representation, not a physical entity that can be observed or interacted with. The actions it can perform are moving and turning, considered reversible since they must be corrected.
- *Environment*: the place where the imaginary rectangle should be drawn, described by the number of drawn rectangle segments, length and orientation.
- *Task*: find the algorithm. The system's initial state is the imaginary rectangle not yet being drawn, while the final state is the 50 × 100 pixels rectangle drawn. The algorithm is not valid, for this reason, the final and correct version of it has to be found.

constrained definition, and its written representation promote the development of all problem setting skills. The system resettability, the one-to-one cardinality and the explicit representation of elements support other competencies.

- *Algorithm*: the competencies associated with algorithmic concepts provided by the tool functionalities can be developed in all artefactual environments. The non-observability of the system, the explicit and constrained definition of the task elements, and the manifest written representation of the algorithm can further enhance these skills.
- *Assessment*: algorithm debugging can be activated in all artefactual environments due to the resettability of the system and the written algorithm; since the system can be reset, also the constraints on the algorithm can be checked and corrected, allowing for the development of constraint validation; similarly optimisation can be activated since the resetting capability is sufficient; generalisation can be developed thanks to the system’s resettability and the presence of variables and functions. The tool functionalities available further encourage the development of these competencies.

Inhibiting features for competencies development

- *Conditionals*: non-activable in the first activity since this functionality is not present in the visual blocks provided to the students.
- *Parallelism and events*: non-activable in both activities, as before, because they are not available in the visual blocks provided to the students. To activate these skills, the visual blocks must include the tools for creating parallelism and events in the algorithm.
- *System state verification*: non-activable because the initial and final states are provided.

3.2 Robotics activities

In robotics, educational robotics and physical computing activities involve using physical robotic hardware equipped with controllers, sensors, and actuators. These robotic agents are programmed to perform specific behaviours in response to the environment. To achieve this, problem solvers are typically provided with a programming platform allowing interaction with the robot.

There are numerous commercially available programming platforms for educational robots, each offering its own set of hardware and programming platforms (Bravo et al. 2017). The agents can be programmed using formal textual programming languages, such as Python (Noone and Mooney 2018), or symbolic visual programming languages, which are often based on blocks,

like Blockly or Scratch (Shin et al. 2014). Some platforms also allow for embodied physical interactions, enabling users to manipulate the robot through touch buttons or tangible symbols that are scanned and executed (Bers and Horn 2010; Mussati et al. 2019).

In this section, we aim to analyse various activities based on different types of agents, including the Thymio II, the Ozobot, and the Micro:bit.

3.2.1 Thymio Lawnmower Mission

Thymio Lawnmower Mission is an educational robotics activity designed by Chevalier et al. (2020) to promote the development of students’ CT skills through the Thymio II robot. The Thymio II, for instance, is a widely used educational robot equipped with various sensors, including proximity sensors, an accelerometer, a remote control receiver, motors, a speaker, and LEDs, distributed throughout its body (Riedo et al. 2013; Shin et al. 2014). The authors aimed to address the issue of students spending excessive time programming and not enough time problem-solving, referred to as the trial-and-error loop, by conducting an instructional intervention on two groups of primary school students.

Components In the Thymio Lawnmower Mission, illustrated in Fig. 12, the Thymio II robot must systematically traverse all green lawn squares, much like a lawnmower would mow a lawn.

- *Problem solver*: the group of students performing the task who must program the agent’s lawnmower behaviour. The artefactual environment comprises tools designed for reasoning and interaction, including a graphical programming environment called VPL (symbolic), which allows for the creation of sensor-action relationships to determine the robot’s behaviour, the agent (embodied) and the visual feedback (embodied).
- *Agent*: the Thymio II, whose actions consist in moving around the playground, by changing velocity and orientation, and using sensors to detect obstacles. All actions are considered irreversible.
- *Environment*: the playground, i.e., a lawn area surrounded by a fence, divided into eight green squares and one grey square, the garage. Its state is defined by the grass condition or the number of squares passed over by the agent.
- *Task*: find the algorithm. The initial state is the lawn with tall grass, meaning the robot is not passed over any of the squares that compose it. The final state is the same lawn with the grass mowed, meaning the robot passes over all green squares. The algorithm is the set of moving actions to reach the system’s final state from the initial.

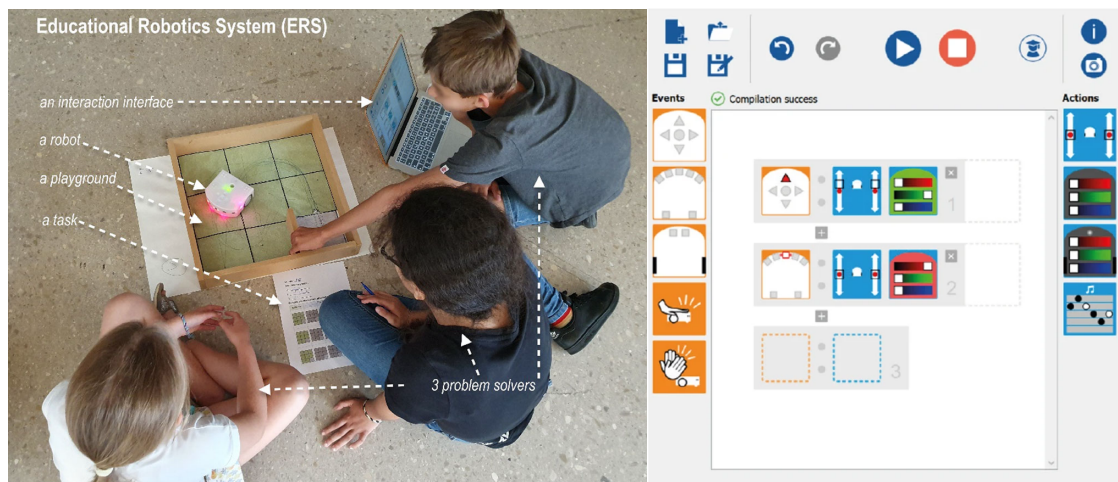


Fig. 12 The Thymio Lawnmower Mission adapted from Chevalier et al. (2020). A group of pupils must program the Thymio II robot to pass over all eight green lawn squares and avoid the fence (left). A special visual programming language platform, with graphical icons that are straightly interpretable, is used for this task (right).

Characteristics In the Thymio Lawnmower Mission, the students who participated in the activity were divided into two groups, the control group and the test group, to which different conditions were imposed. The control group was allowed to complete the task without any constraints. In contrast, the test group was subjected to an instructional intervention that blocked the programming interface at certain times to overcome the trial-and-error loop. As a result, the two activity variants have distinct characteristics, analysed using the graphical templates shown in Figs. B.8 and B.9 in Appendix B.

- *Tool functionalities*: the system provides a comprehensive set of tools for the problem solver to create and control the behaviour of the agent to solve the task, including (i) *variables* such as the values of sensors or the state of the robot; (ii) *operators* represent the basic actions that the agent can perform; (iii) *sequences* are not represented by a specific block in the VPL interface, but can be created by arranging blocks in a specific order; (iv) *functions* are not represented by blocks in the VPL but refer to the possibility of conceptually grouping blocks of code associated with a particular behaviour to produce outputs given inputs; (v) *events* are directly represented in the graphical interface by sensor-action relationships, allowing the robot to perform actions in response to stimuli, such as detecting an obstacle.
- *System resetability*: in the control group, the problem solvers can reset the system directly by physically moving the Thymio II agent in the environment and restarting the task by repositioning it in the garage. On the other hand, those in the test group do not have this option as they cannot di-

rectly interact with the agent and cannot modify the algorithm since it has to be first written and then executed.

- *System observability*: the platform provides real-time visual feedback, making the system observable.
- *Task cardinality*: the task has a one-to-one mapping, with an initial and final state and an algorithm.
- *Task explicitness*: the elements of the task are given explicitly, as the student is provided with clear instructions on what the outcome should look like.
- *Task constraints*: the algorithm is unconstrained.
- *Algorithm representation*: the algorithm is written in the workspace and expressed by the set of blocks and their connections.

Enabling features for competencies development This paragraph explores the enabling characteristics that support the development of competencies within the task. The relationship between features and skills in the two activity variants are summarised in Appendix B in Tables B.8 and B.9.

- *Problem setting*: all competencies can be activated thanks to the presence of variables, sequences and functions in the tool functionalities. The presence of many tool functionalities positively affects and boosts problem setting skills. The manifest and written representation of the algorithm can further encourage the development of these skills. In the control group, being the system resettable, data collection, pattern recognition and decomposition are promoted, while in the test group, abstraction and data representation are also encouraged. The one-to-one cardinality of data elements also facilitates data collection, pattern recognition and decompo-

sition. The system observability also supports data collection and pattern recognition.

- *Algorithm*: all competencies associated with the algorithmic concepts enabled by the tool functionalities, meaning variables, operators, sequences, functions and events, can be activated in all three artefactual environments and promote one another. These are further enhanced by the manifest and written algorithm representation, system observability, and the explicit and unconstrained definition of the task elements. The one-to-one cardinality helps to develop variables and operators further.
- *Assessment*: regarding the control group, algorithm debugging can be activated in all artefactual environments since the algorithm has to be found, the system is resettable, and the algorithm is manifest and written; optimisation can be developed thanks to the resettable of the system; generalisation can be activated through the system's resettable and the presence of variables and functions. The tool functionalities available and the system observability help develop these skills as well. On the other hand, the system is non-resettable in the test group, and no assessment skills can be developed.

Inhibiting features for competencies development The lack of specific features may hinder the development of particular skills.

- *Repetitions, conditionals and parallelism*: non-activable in both control and test groups, as these functionalities are unavailable in the VPL programming platform. Therefore, to develop these skills, it is necessary to change the programming language to a textual programming language such as ASEBA.
- *Algorithm debugging*: non-activable in the test group since the system is not resettable.
- *System state verification*: non-activable, in both control and test groups, because the initial and final states of the system are given, and in the test group, also because the system is not resettable.
- *Constraint validation*: non-activable, in both control and test groups, due to the absence of constraints on the algorithm, and in the test group because the system is not resettable.
- *Optimisation*: non-activable in the test group because the system is not resettable.
- *Generalisation*: non-activable in the test group due to the non-resettable of the system.

3.2.2 Remote Rescue with Thymio II (R2T2)

Remote Rescue with Thymio II (R2T2) is another collaborative educational robotics activity, presented by

Mondada et al. (2016) to promote Science, Technology, Engineering and Math (STEM) education in schools and encourage students towards careers in these fields.

Components The R2T2 activity, illustrated in Fig. 13, is a rescue operation on a Mars station, whose goal is to assess the damage of the power plant and restart the main generator by remotely controlling 16 Thymio II robot. The activity is divided into five phases, each with a specific objective. In the first phase, the robots must enter the station and push away an obstacle blocking the main door. In the second phase, the robots must stand on control spots to activate access to the generator. In the third phase, the robots must look into the generator through a small window. In the fourth phase, the robots must turn on a light when detecting the generator rotor using proximity sensors and off when it is no longer visible, thus estimating the generator speed. In the final phase, the generator is restarted, and the mission is completed.

- *Problem solver*: the group of students performing the task who must program the agents' behaviours to restart the main generator. The artefactual environment comprises tools designed for reasoning, such as paper and pencils and another robot that is physically accessible. Tools available also to interact with the system include the programming platform (symbolic) with the two available programming environments, VPL and ASEBA, a textual programming language (Magenat et al. 2011), and the five webcams, installed around the playground, provide a delayed continuous visual feedback (embodied) through YouTube video streams.
- *Agent*: the 16 remote-controlled Thymio II, which can move around the playground accelerating and rotating, use proximity sensors and turn some lights on and off. All actions are considered irreversible.
- *Environment*: the playground, i.e., the Mars station, characterised by different descriptors used in the different mission stages, such as the obstruction by the obstacle, covering of the control spots and finally, the restart of the generator.
- *Task*: find the algorithm. In the initial state, the generator is not working, while it has been restarted at the end. The algorithm is the set of moving actions to reach the system's final state from the initial.

Characteristics Fig. B.10 in Appendix B provides the graphical template used to analyse the task components and characteristics.

- *Tool functionalities*: the system provides a comprehensive set of tools for the problem solver to create and control the agent's behaviour to solve the

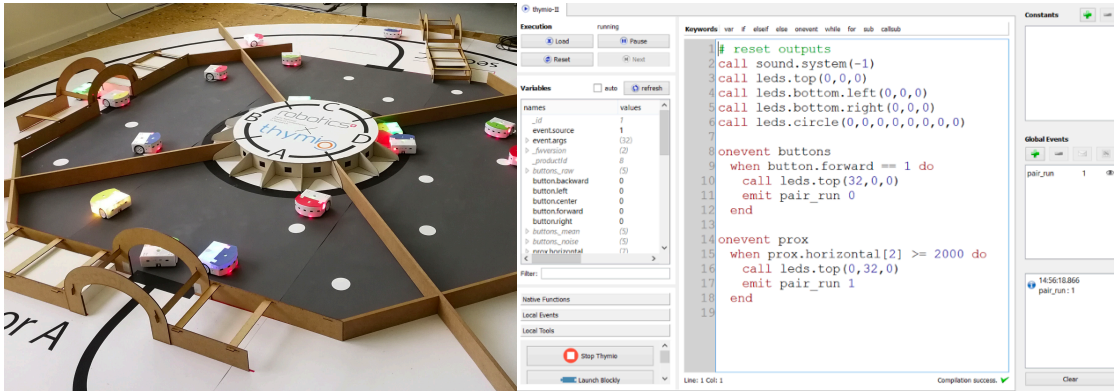


Fig. 13 The Remote Rescue with Thymio II (R2T2) mission on Mars adapted from Mondada et al. (2016). Sixteen worldwide teams of pupils collaborate with 16 Thymio to restart the main generator of a simulated damaged power Mars station (left) in five phases using a visual programming language or textual programming language programming platforms (right).

task, depending on the programming environment used. VPL offers the possibility to use *variables*, *operators*, *sequences*, *functions* and *events*. Additionally, ASEBA offers control flows such as *repetitions* and *conditionals*. Furthermore, *parallelism* is possible since it refers to the ability to run multiple processes simultaneously, in this case, the Thymio II robots performing the rescue operation in parallel. The agents can execute their tasks concurrently without waiting for each other to complete them.

- *System resettability*: the system cannot be reset due to the irreversible nature of the actions carried out by the robots. Once the robots take an action, the change in the system’s status is permanent and cannot be undone. Furthermore, the physical separation between the problem solvers and the system means no immediate way to reset the system.
- *System observability*: the delayed but continued system visual feedback makes it totally observable.
- *Task cardinality*: the task has a one-to-one mapping, with an initial and final state and an algorithm.
- *Task explicitness*: all elements are given explicitly.
- *Task constraints*: the algorithm is unconstrained.
- *Algorithm representation*: the algorithm is manifest and written in the programming platform.

Enabling features for competencies development The relationship between features and skills is summarised in Appendix B in Table B.10.

- *Problem setting*: all competencies can be activated thanks to the presence of variables, sequences and functions in the tool functionalities. The presence of all tool functionalities, the manifest written representation of the algorithm and the non-resettability of the system further encourage the development of these skills. The system observability supports data collection and pattern recognition. The one-to-one

cardinality, in addition to these skills, stimulates decomposition. The explicit and unconstrained definition of the task elements also promotes pattern recognition, decomposition and abstraction.

- *Algorithm*: competencies across all artefactual environments can be triggered since the tool’s functionalities enable all algorithmic concepts. The system observability, the explicit and unconstrained definition of the task elements, and the manifest written algorithm representation further enhance these.
- *Assessment*: since the system is not resettable, there are no assessment skills activable.

Inhibiting features for competencies development The inability to reset the system restricts the activation of all assessment skills. The task can be adjusted by incorporating a mechanism for resetting the system to a previous state, enabling the problem solver to correct errors made during the implementation of the algorithm, explore different solutions, and learn from their mistakes. Additionally, it is necessary to omit the initial or final states to verify the system’s state. Furthermore, to develop the constraint validation skill, it is necessary to incorporate constraints into the algorithm.

3.2.3 Ozobot maze

The Ozobot Maze activity is a screenless robotics task proposed by Bryndová and Mališů (2020) aimed at teaching primary school students in the Czech Republic CT skills. The educational robot used in this task is the Ozobot, a small programmable robot, used to introduce students to coding, equipped with sensors to follow black lines and read colour patterns called Color Codes to change speed, direction and movements.

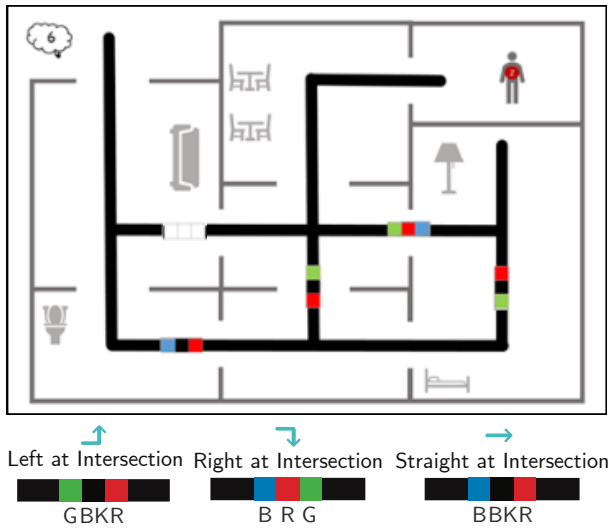


Fig. 14 The Ozobot maze adapted from Bryndová and Mališů (2020). The task requires the pupil to instruct the Ozobot to cross a maze avoiding obstacles and reaching the room where the red person is. Commands such as increasing the speed, changing direction and making some cool movements (spinning like a tornado) are given in Color Codes.

Components In the Ozobot Maze activity, illustrated in Fig. 14, the robot should be guided through a maze to reach the room where the red person is.

- *Problem solver:* the student who creates a suitable sequence of instructions using Color Codes to guide the Ozobot through the maze. The artefactual environment comprises tools for reasoning and interacting with the system. Predefined stickers or markers to fill the empty Codes with colour sequences (embodied) are used to give the robot the correct instructions to achieve the goal. The visual feedback (embodied) lets the problem solver observe the agent and its movements in the playground.
- *Agent:* the Ozobot agent, which can move around in the playground by changing velocity and orientation. This action is not reversible.
- *Environment:* the playground, i.e., the house map, whose state is defined by the agent's position relative to the red person.
- *Task:* find the algorithm. The initial state is the empty maze with the Ozobot positioned near the starting point. The system's final state is the Ozobot reaching the end of the maze, in the room with the red person, and all the Color Codes being filled. The algorithm is the set of agent instructions, shown by the Color Codes, to reach the system's final state from the initial.

Characteristics The characteristics of this activity have been analysed using the graphical templates shown in Fig. B.11 in Appendix B.

- *Tool functionalities:* the system provides a comprehensive set of tools for the problem solver to create and control the behaviour of the agent to solve the task, including (i) *variables* can be used to store values such as the position of the robot in the maze; (ii) *operators* are basic actions that the agent can perform, represented by Direction Codes such as moving straight, turning left or right; (iii) *sequences* represent the set of instructions used to control the behaviour of the robot in a step-by-step manner and that the Ozobot must follow to complete the task; (iv) *repetitions* are a way of repeating the same instructions multiple times and refer to the possibility of repeating the same Color Code multiple times, for example, if the agent encounters the same type of intersection repeatedly in the path and the same Color Code is used to specify the direction the agent should take; (v) *conditionals* are used to make decisions based on certain conditions, for example understanding what to do at an intersection; (vi) *functions* can be reflected in the different types of Color Codes that can be reused in different situations and map inputs to outputs, such as mapping a specific type of intersection to a specific direction.
- *System resettability:* the system is not resettable since it is impossible to change the Color Codes once they have been filled in.
- *System observability:* the real-time visual feedback makes the system observable.
- *Task cardinality:* the task has a one-to-one mapping.
- *Task explicitness:* the elements of the task are given explicitly, as the student is provided with clear instructions on what the outcome should look like.
- *Task constraints:* the algorithm is unconstrained.
- *Algorithm representation:* the algorithm is manifest and written, expressed by the set of Color Codes.

Enabling features for competencies development The relationship between features and skills is summarised in Appendix B in Table B.11. This paragraph explores the enabling characteristics that support the development of competencies within the task.

- *Problem setting:* all competencies can be activated thanks to tool functionalities such as variables, sequences and functions. The manifest and written algorithm representation, as well as the non-resettability of the system, can further encourage the development of these skills. The system observability supports data collection and pattern recognition. The one-to-one cardinality, in addition, stimulates decomposition. The explicit and unconstrained definition of the task elements promotes pattern recognition, decomposition and abstraction.

- *Algorithm*: all competencies associated with the algorithmic concepts enabled by the tool functionalities, meaning variables, operators, sequences, repetitions, conditionals and functions, can be activated in all three artefactual environments and promote one another. These are further enhanced by the manifest and written representation of the algorithm, the system observability, and the explicit and unconstrained definition of the task elements. The one-to-one cardinality helps to enhance some of these skills as well.
- *Assessment*: since the system is not resettable, there are no assessment skills activable.

Inhibiting features for competencies development

- *Parallelism and events*: non-activable since the related features are missing in the tool functionalities. To develop these skills, it is possible to switch to a different interaction tool, such as OzoBlockly, a visual programming language designed to code Ozobots Evo and includes these functionalities.
- *Assessment skills*: The non-resettable feature of the system hinders the development of assessment abilities. In this sense, by allowing the problem solver to reset the system to a previous state, for example by letting them change the Color Codes stickers and move the robot back to the starting position, the activity can be improved. This way, the problem solver can correct any mistakes made during the implementation of the algorithm, experiment with different solutions, and learn from their mistakes. To develop system state verification, it is also essential to not reveal the initial or final states. Moreover, constraints should be imposed on the algorithm to develop constraint validation skills.

3.2.4 Mini-golf challenge with micro:bit

In robotics, physical computing activities involve using microcontrollers, sensors, and other electronic components to build and program interactive systems. To enhance the learning experience, various off-the-shelf robotic kits have been developed that allow students to construct robots easily and control them through a graphical user interface.

In these activities, students are often engaged in an initial phase of actively constructing the system using recycled materials, electronic circuits and programming the robot. These activities evaluate the students' understanding of algorithmic concepts, problem-solving skills, knowledge of physics and engineering, creativity, and ability to work collaboratively.

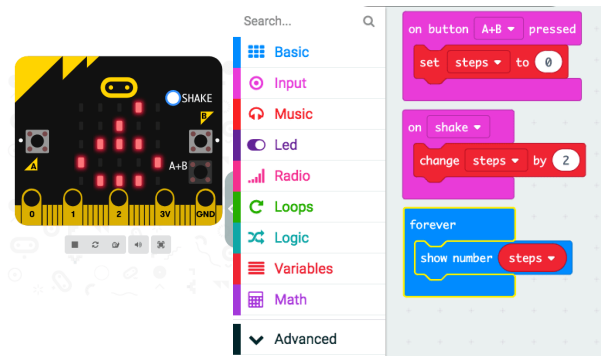


Fig. 15 The BBC micro:bit (left) and its block programming interface (right).

One such physical computing activity is the Mini-golf challenge, proposed by Assaf et al. (2021). In this activity, students are tasked with programming a mini-golf lane's moving and interactive elements using the BBC micro:bit (Ball et al. 2016; Microbit 2016). The micro:bit, depicted in Fig. 15, is a pocket-sized computer that can be programmed using Microsoft's MakeCode editor (Makecode 2016), which provides a user-friendly interface with colour-coded blocks similar to Scratch and the ability to switch to JavaScript to view the text-based code.

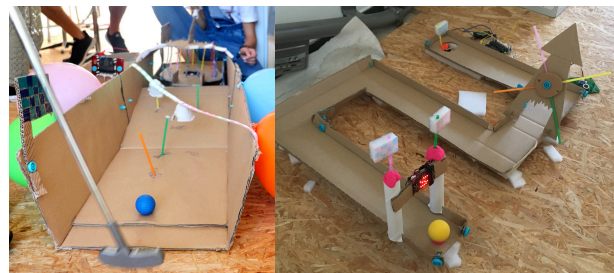


Fig. 16 The Mini-golf challenge challenge adapted from Assaf et al. (2021). The task requires a group of pupils to define the behaviour of the mini-golf lane movable obstacles, sounds, and lights by programming the BBC micro:bit.

Components In the Mini-golf challenge, illustrated in Fig. 16, the objective is to program a mini-golf lane's moving and interactive elements.

- *Problem solver*: the group of students who must program the micro:bit. The artefactual environment disposed of paper and pencil (embodied), a cognitive tool to support the thinking phase. Other tools are provided to interact with the system, including the toolkit (embodied) whose components can be assembled and disassembled at will, the visual programming language offered by the MakeCode editor (symbolic), and the visual feedback (embodied).

- *Agent*: the micro:bit, which can use sensors, control the movement and actions of the mini-golf elements, and turn LEDs and speakers on and off. All actions are considered not reversible.
- *Environment*: the assembled toolkit, which consists of various components, including a ball, speakers, and lights. Its state is described by the state of its elements, including the ball position, lights illumination and speakers ignition.
- *Task*: creation act. The initial state is given by the toolkit assembled. The system's final state and the students' algorithm are open-ended, which defines the mini-golf station elements' behaviour.

Characteristics The characteristics of this activity have been analysed using the graphical templates shown in Fig. B.12 in Appendix B.

- *Tool functionalities*: the MakeCode editor allows using all the tool functionalities we defined.
- *System resettability*: the system can be directly reset by physically moving the ball back to its starting position, and resetting the state of the lights and speakers, for example, by turning them off. Additionally, the MakeCode editor includes a convenient button to streamline the agent's reset process.
- *System observability*: the real-time visual feedback makes the system observable.
- *Task cardinality*: the task has a one-to-one mapping.
- *Task explicitness*: the initial state of the toolkit, including the ball's position, the state of the lights, and the speakers, is not specified.
- *Task constraints*: no constraints are imposed on the two elements to be found.
- *Algorithm representation*: the algorithm is manifest and written in the programming platform.

Enabling features for competencies development The mapping between features and skills is summarised in Appendix B in Table B.12. This passage analyses the characteristics that support the development of competencies within the task.

- *Problem setting*: all competencies can be activated thanks to the presence of variables, sequences and functions in the tool functionalities. The availability of numerous tool functionalities, the written algorithm representation and the non-resettability of the system encourage the development of problem setting skills. The implicit description of the task elements creates an environment of uncertainty, allowing for multiple interpretations and solutions, stimulating problem setting skills. The system's observability allows the development of data collection

and pattern recognition, while the one-to-one cardinality also encourages decomposition. The unconstrained definition of task elements fosters pattern recognition, decomposition, and abstraction.

- *Algorithm*: all competencies can be activated since the tool functionalities enable all algorithmic concepts in all three artefactual environments. The system observability, the implicit and unconstrained definition of the task elements and the algorithm's manifest written representation further enhance these. The one-to-one cardinality helps to enhance some algorithmic skills as well.
- *Assessment*: algorithm debugging and system state verification can be developed in all artefactual environments by the direct resettability of the system and the written representation of the algorithm; optimisation can be activated as the resettability of the system alone is sufficient; generalisation is enabled through the system's resettability and the presence of variables and functions. Tool functionalities, system observability and the implicit definition of the task elements further support their development.

Inhibiting features for competencies development The constraint validation skill cannot be developed as the algorithm and the final state to be found are unconstrained. To encourage the development of this competence, the activity can be adapted by introducing defined constraints, such as a maximum number of moves for the ball to reach the hole or that certain elements of the mini-golf station must be activated in a specific order. These constraints will require students to evaluate the feasibility of their solutions within the specified limitations and assess their adherence to the established criteria.

3.3 Virtual activities

The last domain of educational activities analysed includes CTPs characterised by the presence of a virtual system. These activities typically involve programming a virtual agent to perform a specific or a set of tasks in a virtual environment. In contrast to CTPs with a physical environment, virtual activities always provide a virtual interface, often including a comprehensive programming platform that allows the problem solver to use different types of programming languages, such as textual programming language and visual programming language. In some virtual games, the problem solver can interact directly with the agent by clicking on it. This allows for greater flexibility in terms of how the problem solver can program the virtual agent. Additionally, virtual activities often include debugging tools, which

allow the problem solver to identify and fix errors in their code.

3.3.1 Classic Maze

In addition to unplugged activities, Code.org is a platform that offers many coding activities for children based on Blockly, a Google framework for block-based programming (Lovett 2017). The Classic Maze is part of the Hour of Code offered by Code.org, a worldwide effort to broaden participation in the computer science field (Studio.code.org 2020b). Participants must use block-based programming to guide different characters through a maze in this activity. The creatures include ones from popular franchises such as Angry Birds, Plants vs Zombies, and Scrat from Ice Age. In this way, they learn the foundations of computer science and algorithmic concepts by successfully guiding the characters through the maze. We decided to analyse two activities of the Classic Maze. Appendix B includes the graphical template used to analyse the components and characteristics of the task, found in Figs. B.13 and B.14, and the mapping between the CTP features and the competencies in Tables B.13 and B.14.

Components In the first activity of the Classic Maze, presented in Román-González et al. (2018) and illustrated in Fig. 17, the Angry Bird should be guided through a maze to reach and hit a Green Pig.

- *Problem solver*: the student performing the task who must program the agent's behaviour. The artefactual environment comprises tools designed for reasoning and interacting with the system simultaneously, including the programming platform composed of the virtual scenario (embodied artefact), the blocks and the workspace (symbolic artefacts). The system also furnishes various hints to users (embodied artefact), including video tutorials, guidance on how to use the platform, command recommendations, suggestions on the number of blocks required to solve the task, and feedback on the problem solver's progress towards a solution.
- *Agent*: the Angry Bird, programmed to navigate a maze and hit the other character. The agent's actions comprise moving forward, turning left, and right. Moving forward is considered a non-reversible action, whereas the turning is reversible, as a turn left can be easily undone by a turn right, and vice versa.
- *Environment*: the virtual scenario where the two creatures are located, described by their positions.
- *Task*: find the algorithm. The initial state corresponds to the animals' initial positions, while in the final, the characters are in the same position. The algorithm is the set of moving actions to reach the system's final state from the initial.

Characteristics

- *Tool functionalities*: the programming platform enables problem solvers to reason about the task at hand and program the movements of the red bird by employing a set of predefined blocks, each representing a specific action the agent is authorised to perform: (i) *variables*, while not explicitly delivered in the blocks of the programming platform, can be inferred from the visual feedback provided by the system, allowing the problem solver to store values such as the position of the characters; (ii) *operators* represent the basic actions that the agent can perform, such as moving or turning and are represented by distinct blocks in the visual programming language depicted in cyan by the platform; (iii) *sequences* are a series of blocks to be executed in a specific order and are implicitly conveyed by the collection of blocks; (iv) *functions*, which are self-contained blocks of code that perform a specific task and can be executed multiple times with different inputs (e.g., different initial positions of characters), are a concept of relative complexity. It is not certain that the problem solver will recognise them as such rather than just blocks.
- *System resettability*: the platform provides a direct means of resetting the task through the "start over" button, even if some agent actions are irreversible. This allows the problem solver to start over and try a different approach if necessary.
- *System observability*: the system provides real-time visual feedback through animations and graphical representations of the system state and its changes, making it observable. The problem solver can monitor the effects of the agent's actions on the system, allowing him to have a complete understanding of the system's current state and to make informed decisions in their problem-solving process.
- *Task cardinality*: the task has a one-to-one mapping, with only one starting position for the animal elements, one final position for the Angry Bird to be placed, and only one algorithm to be found.
- *Task explicitness*: the elements of the task are given explicitly through the depiction of the scenario that clearly shows the animals' positions.
- *Task constraints*: there are no constraints on the elements to be found. All the blocks provided are available without limitations.

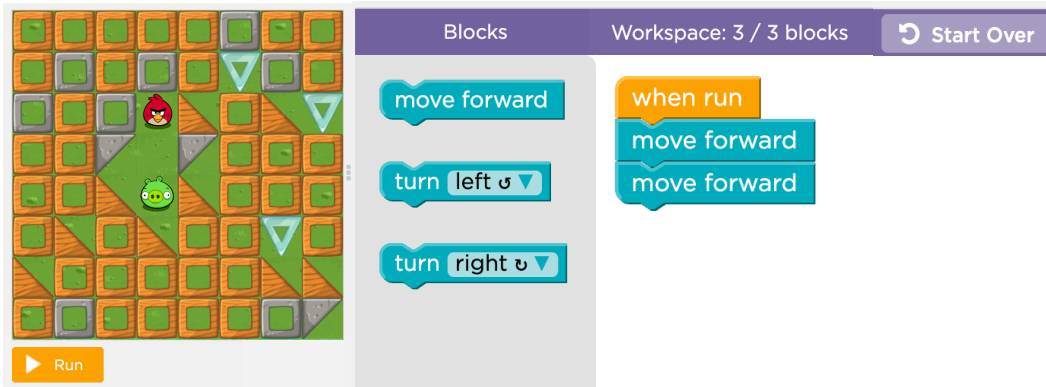


Fig. 17 The Angry Bird hitting the Green Pig maze adapted from Studio.code.org (2020a). The problem solver must write a program to get the Angry Bird through the maze to hit the Green Pig (left) by selecting the instruction blocks (middle) and assembling them in the workspace (right).

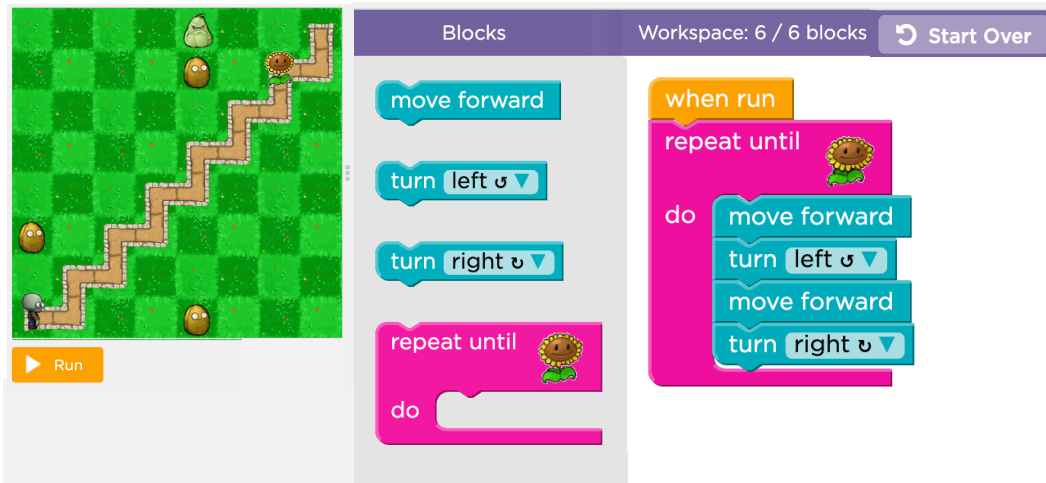


Fig. 18 The Plants vs Zombies maze adapted from Studio.code.org (2020c). The problem solver must write a program to get the Zombie through a maze to eat the plant (left) by selecting the instruction blocks (middle) to be assembled in the workspace (right).

- *Algorithm representation*: the algorithm is written in the workspace and expressed by the set of blocks and their connections.

Enabling features for competencies development This paragraph explores the enabling characteristics that support the development of competencies within the task. Certain features play a crucial role in activating skills, while others can further enhance and encourage the development of these competencies.

- *Problem setting*: all competencies can be activated thanks to the presence of variables, sequences, and functions in the tool functionalities. The manifest and written algorithm representation can further encourage the development of these skills. The resetability of the system and the one-to-one cardinality of data elements facilitate data collection, pattern recognition and decomposition. Observing the system also supports data collection and pattern

recognition. Using variables boosts pattern recognition and decomposition, while explicit and unconstrained elements, in addition to these two skills, encourage abstraction. Sequences positively affect pattern recognition, abstraction and data representation, which is also facilitated by functions. Operators also encourage decomposition.

- *Algorithm*: all competencies associated with the algorithmic concepts enabled by the tool functionalities, meaning variables, operators, sequences, and functions, can be activated in all three artefactual environments. Features such as variables, operators, sequences, and functions help activate the algorithmic skills in a task. These are further enhanced by the manifest and written algorithm representation, system observability, and the explicit and unconstrained definition of the task elements. The one-to-one cardinality enhances variables and operators.

- *Assessment*: algorithm debugging can be activated in all artefactual environments due to the resettability of the system and the manifest and written representation of the algorithm; optimisation can be activated as it only requires the resettability of the system; generalisation can be activated through the system's resettability and the presence of variables and functions. Features such as variables, operators, sequences, functions and system observability help develop algorithm debugging and optimisation, while sequences can also foster generalisation.

Inhibiting features for competencies development This paragraph explores the impact of missing features on skill activation, focusing on how adjusting the task can enable the development of certain competencies.

- *Repetitions, conditionals, parallelism, and events*: non-activable due to the absence of specific tool functionalities. These competencies cannot be triggered until these functionalities are added to the tool.
- *System state verification*: non-activable because both the initial and final state are provided. The task can be adjusted to activate this skill by making one of these states to be found. For instance, by not providing the position of the Green Pig to the problem solver, who must then determine its location, the system's final state becomes unknown, and the system state verification activable.
- *Constraint validation*: non-activable because the algorithm to be found is unconstrained. Some constraints can be imposed on the blocks used in the task's programming platform to activate this skill, such as limiting the agent's ability to turn right.

Comparison of two Classic Maze activities The activities proposed in the Classic Maze exhibit a progressive increase in difficulty. For example, the Plants vs Zombies maze, illustrated in Fig. 18, is a similar task requiring finding the algorithm. The main differences between this task and the previous are the characters involved, the difficulty of the path that the agent, in this case, the Zombie, must traverse to reach the plant, and the set of actions available to the agent. The programming platform for this activity also includes the possibility of using repetitions, represented in pink, adding another layer of difficulty. The features of the CTP remain unchanged from the previous task, except for the addition of the repetition functionality in the tool, which enables the activation of the related competence.

3.3.2 Store the Marbles

Store the Marbles is a virtual programming activity, presented by Algorea, an online resource designated

by France-IOI, for learning the basics of programming (ALGOREA 2020; France-IOI.org 2004). The activity is designed to teach students problem-solving skills and programming concepts using a visual block-based programming language. The activity is part of a series of progressive difficulty courses and exercises available on the France-IOI website.

Components In this activity, illustrated in Fig. 19, the robot should pick up the marble on his path and drop it into a hole.

- *Problem solver*: the student who must program the agent's behaviour. The artefactual environment comprises tools designed for reasoning and interacting with the system simultaneously, including the programming platform composed of the virtual scenario (embodied artefact), the blocks and the workspace (symbolic artefacts). The system also furnishes suggestions on the number of blocks required to solve the task (embodied artefact), but it does not provide additional hints beyond this information.
- *Agent*: the virtual robot, programmed to move on the path to collect the marble and drop them in the hole. The agent's actions comprise moving eastward, picking up a marble, and dropping a marble. The simple movement is considered irreversible, whereas picking up and dropping a marble are considered reversible actions, as they are the reverse of each other. The agent's actions comprise moving forward, turning left, and right. Moving forward is considered a non-reversible action, whereas the turning is reversible, as a turn left can be easily undone by a turn right, and vice versa.
- *Environment*: the virtual scenario where the robot and the marble are located, which is described by their respective positions.
- *Task*: find the algorithm. The initial state corresponds to the robots' initial positions and the maze distribution of marbles. The final state corresponds to the robot positioned on the hole and the marbles inside it. The algorithm is the set of moving actions to reach the system's final state from the initial.

Characteristics Fig. B.15 in Appendix B provides the graphical template used to analyse the task components and characteristics.

- *Tool functionalities*: the system provides a comprehensive set of tools for the problem solver to implement the algorithm to solve the task, including (i) *variables* allows the problem solver to store values such as the position of the robot, marble, and hole; (ii) *operators* represent the basic actions that the agent can perform, such as moving or turning and

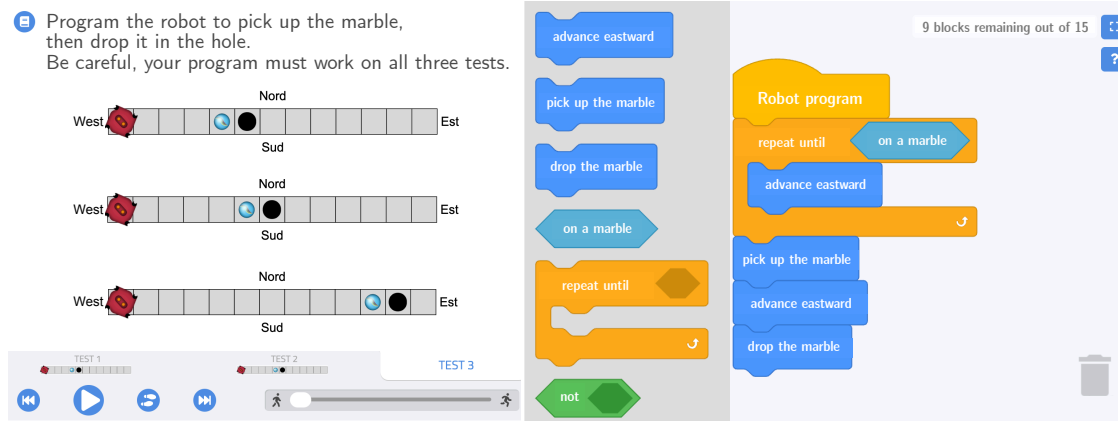


Fig. 19 The Store the Marbles activity adapted from ALGOREA (2020). The task requires the problem solver to program the robot to produce an algorithm valid for different situations using a visual programming language.

are represented by the blue blocks of the visual programming language; (iii) *sequences* are obtained by concatenating a series of blocks to be executed in a specific order, allowing the problem solver to plan and execute a series of actions to reach the goal; (iv) *repetitions* are offered by a specific orange block and allow the problem solver to repeat a sequence of actions until a specific condition is met; (v) *functions* are self-contained blocks of code that perform a specific task and can be executed multiple times, providing a flexible and reusable component in the algorithm design. The platform places importance on the design of the algorithm as a function that can adapt to changing inputs, such as the different starting conditions of the task.

- *System resettability*: the platform allows for resetting the task through a button, enabling the problem solver to start over and try a different approach.
- *System observability*: the platform provides real-time visual feedback, making the system observable.
- *Task cardinality*: the task has a many-to-one cardinality, with many initial and final states given and only one algorithm to be found. The challenge in this activity is for the program to work with multiple initial states, as the problem solver is given three different configurations of marble and hole positions. The solution must successfully sort the marbles into the correct locations.
- *Task explicitness*: the task elements are given explicitly through the depiction of the scenario that clearly shows the positions of the robot, marble, and hole.
- *Task constraints*: the algorithm is unconstrained.
- *Algorithm representation*: the algorithm is manifest and written, expressed through a set of blocks.

Enabling features for competencies development This paragraph explores the enabling characteristics that support the development of competencies within the task. The relationship between features and skills is summarised in Appendix B in Table B.15.

- *Problem setting*: all competencies can be activated thanks to the presence of variables, sequences, repetitions and functions in the tool functionalities. The manifest and written representation of the algorithm and the many-to-one cardinality can further encourage the development of these skills. The resettability of the system facilitates data collection, pattern recognition and decomposition. The system observability supports data collection and pattern recognition. Using variables boosts pattern recognition and decomposition, while explicit and unconstrained elements in addition to these two skills encourage abstraction. The use of sequences positively affects pattern recognition, abstraction and data representation, which is also facilitated by functions. Operators also encourage decomposition abstraction and data representation.
- *Algorithm*: all competencies associated with the algorithmic concepts enabled by the tool functionalities, i.e., variables, operators, sequences, repetitions and functions, can be activated in all three artefactual environments. Features such as variables, operators, sequences, repetitions and functions help activate the algorithmic skills in a task. These are further enhanced by the manifest and written representation of the algorithm, system observability, the explicit and unconstrained definition of the task elements, and the many-to-one cardinality.
- *Assessment*: algorithm debugging can be activated in all artefactual environments due to the resettability of the system and the manifest and written

representation of the algorithm; optimisation can be developed as it only requires the resettability of the system; generalisation can be activated through the system's resettability and the presence of variables and functions. Features such as variables, operators and functions help develop algorithm debugging and optimisation. Sequences, repetitions and system observability foster algorithm debugging, optimisation, and generalisation. The many-to-one cardinality influences the development of generalisation as well.

Inhibiting features for competencies development The absence of certain features can limit the activation of certain skills.

- *Conditionals, parallelism, and events*: non-activable as the platform does not supply them.
- *System state verification*: non-activable because the initial and final states of the system are given. The skill can be promoted by making the initial or final state to be found.
- *Constraint validation*: non-activable due to the algorithm's absence of constraints. The task is adjustable by imposing constraints on it.

3.3.3 Zoombinis Allergic Cliffs Puzzle

The Zoombinis activity is a popular CT learning game where the player must guide little blue creatures with distinct personalities and appearances through different puzzles to escape imprisonment (Zoombinis 2021; Rowe et al. 2021).

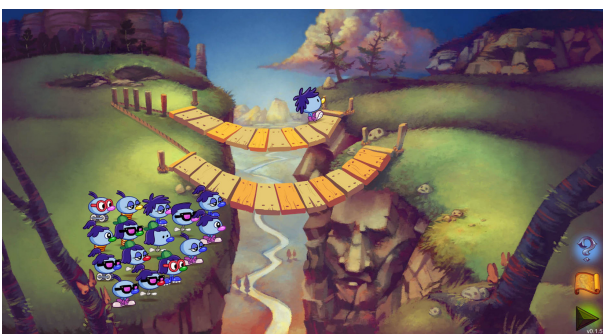


Fig. 20 The Allergic Cliffs Puzzle from Zoombinis (2021). The player must find out which characteristics allow the Zoombinis to cross the bridge without being sent back. The bottom cliff does not accept creatures with flat hair, while the top cliff rejects all others.

Components In the Allergic Cliffs Puzzle, depicted in Fig. 20, the task is to find a procedure that allows all

Zoombinis to cross the bridge. Each cliff accepts different attributes of the creatures, who must find the correct combination of Zoombini attributes that will allow each creature (the agents) to cross the bridge

- *Problem solver*: the player who must find the correct combination of Zoombini attributes that will allow each creature to cross the bridge. The artefactual environment comprises tools designed for reasoning and interacting with the system simultaneously, including the click and drag platform (embodied artefact) and the visual feedback of the virtual scenario (embodied artefact). The system provides feedback on whether the Zoombini's attributes match the requirements of the bridge (embodied artefact), allowing the player to adjust their strategy accordingly.
- *Agent*: the Zoombinis character, which can be dragged to the entrance of a bridge and the only action is allowed to do is to cross it or be sent back in case of failure. This action cannot be reset.
- *Environment*: the virtual scenario described by the Zoombinis' positions and the number of Zoombinis that have crossed a bridge.
- *Task*: find the algorithm. the state of the system depends on the number of creatures that crossed the bridge, initially none and at the end all. The algorithm is the set of moving actions to reach the final state. The problem solver can try different combinations of attributes, testing one attribute's value by holding one attribute constant and testing the others, or even proceed by changing values and attributes until all Zoombinis have crossed the bridges.

Characteristics Fig. B.16 in Appendix B provides the graphical template used to analyse the task components and characteristics.

- *Tool functionalities*: (i) *variables* are used to refer to the attributes of the Zoombinis such as hair colour or eye shape, which can have different values and must be combined correctly to cross the bridge; (ii) *operators* can be used to combine the conditions for the Zoombinis to cross the bridge, for example the player could use a logical AND operator to require that the creature have two specific attributes in order to cross the bridge; (iii) *sequences* can be the order in which the Zoombinis are moved to cross the bridge; (iv) *repetitions* are used to try multiple combinations of Zoombini attributes until the problem solver find the correct solution; (v) *conditionals* are reflected in the requirement of the bridges to accept certain attributes of the Zoombinis, as the player must determine if the Zoombini's attributes match the requirements of the bridge; (vi) *functions* are the combination of Zoombini attributes that allows

all creatures to cross the bridges, as it maps inputs (the Zoombini attributes) to outputs (the success or failure of the Zoombinis crossing the bridges); (vii) *events* correspond to the Zoombini being sent back as it occurs as a result of a wrong player's combination of Zoombini attributes.

- *System resettability*: the platform provides a direct means of resetting the game.
- *System observability*: the system provides real-time visual feedback through animations and graphical representations of the system state and its changes, making the system observable.
- *Task cardinality*: the task presented in the scenario is a one-to-one mapping.
- *Task explicitness*: the system's state is explicit and is represented by the number of Zoombinis that have crossed the bridge.
- *Task constraints*: there are no constraints on the elements to be found. All the blocks provided are available without limitations.
- *Algorithm representation*: the algorithm is latent because the problem solver only performs actions such as dragging the characters across the bridge, but the underlying logic and steps to reach the final goal of having all Zoombinis cross the bridge are not explicitly defined by the player.

Enabling features for competencies development Certain features play a crucial role in activating skills, while others can further enhance and encourage the development of these competencies. The relationship between features and skills is summarised in Appendix B in Table B.16.

- *Problem setting*: all competencies can be activated thanks to the presence of variables, sequences, and functions in the tool functionalities. The latent algorithm can further encourage the development of these skills. The resettability of the system and the one-to-one cardinality facilitate data collection, pattern recognition and decomposition. Observing the system also supports data collection and pattern recognition. Explicit and unconstrained elements boost pattern recognition, decomposition and abstraction. Moreover, all tool functionalities have a positive impact on the problem setting learning process, helping the problem solver to understand the problem and find a solution more effectively.
- *Algorithm*: all competencies associated with the algorithmic concepts enabled by the tool functionalities can be activated in all three artefactual environments, and foster the development also of the other algorithmic concepts related skills. These are further enhanced by the system observability and the

explicit and unconstrained definition of the task elements. The one-to-one cardinality helps to enhance variables, operators and conditionals further.

- *Assessment*: optimisation can be activated by the resettability of the system while generalisation can be developed through the system's resettability and the presence of variables and functions. The tool's functionalities and the system's observability aid in developing these skills.

Inhibiting features for competencies development This activity has some missing features on skill activation, with a focus on how adjusting the task can enable the development of certain competencies.

- *Parallelism*: non-activable due to the absence of the specific tool functionalities. The design of the activity could be altered to include multiple Zoombinis acting at the same time, allowing for the simultaneous execution of multiple actions.
- *Algorithm debugging*: non-activable because the algorithm is implicit and in the form of dragging and dropping the characters to cross the bridge. Without the ability to debug, the problem solver must rely on their understanding of the rules and ability to identify any issues through trial and error. To integrate debugging, the algorithm needs to be made explicit, either by writing down the steps as text or in code form. Then the user can test and debug the algorithm by checking if each step is performed correctly and if the final result matches expectations, thereby improving their debugging skills.
- *System state verification*: non-activable because both the initial and final state are provided. The task can be adjusted to activate this skill by making one of these states to be found.
- *Constraint validation*: non-activable since the algorithm to be found is unconstrained. Constraints can be imposed to activate this skill, such as the order in which the Zoombinis can cross the bridge.

4 Discussion

This study provides a theoretical framework for analysing, evaluating and revising existing computational thinking problems (CTPs) and designing new ones for educational contexts. By analysing prototypical activities from various classic areas of the literature, we identified the principal characteristics of these problems and the competencies they can activate.

4.1 Relating problem domain features to the activation of CT skills

This section aims to provide a comprehensive overview of computational thinking (CT) skills and their relationship to different activity domains, including unplugged, robotics, and virtual. It is important to note that the characteristics and design of CTPs can significantly vary, impacting the activation of CT skills and the effectiveness of the CTP in achieving its intended competencies. By exploring these areas, we aim to gain a deeper understanding of the unique features and challenges of each type of problem domain. Ultimately, this will provide insights into how CT skills can be developed across various contexts and how the characteristics of CTP can influence the activation of competencies. Our goal is to identify the problem domains that are most suitable for activating specific CT competencies, facilitating the choice and design of more effective CTPs in educational contexts.

4.1.1 Unplugged activities

Unplugged activities are a popular teaching method in computer science education, used to foster the development of CT competencies without relying on a computer. These activities can involve various materials such as paper & pencil, physical objects and manipulatives, which allow learners to develop their problem-solving and algorithmic skills in a hands-on and tangible way.

To better understand the relationship between unplugged activities' features and CT competencies' activation, we analysed four prototypical unplugged activities from the literature, including the Cross Array Task, Graph Paper Programming, Triangular Peg Solitaire, and Computational Thinking Test (CTt). Through this analysis, we identified the key characteristics of these problems and provided insights into how different aspects of a CTP can influence the development of CT competencies.

- *Tool functionalities*: unplugged activities often rely on simple tools that can help develop problem setting skills such as abstraction, decomposition and pattern recognition.
- *System resettability*: it is common for unplugged activities to lack system resettability. In such instances, students cannot easily undo their actions or reset the system to a previous state, which can positively or negatively affect the activation of CT skills. On the one hand, the lack of reset functionality can encourage students to plan and think more carefully about their actions, leading to a more thoughtful approach to problem setting that can activate competencies such as abstraction, decomposition, and algorithmic thinking. On the other hand, it may make it more difficult for students to experiment and explore different approaches to a problem, hindering the activation of competencies related to assessment and evaluation such as debugging.
- *System observability*: unplugged activities have varying degrees of observability. Some CTPs may be highly observable, with clear steps and processes that are easy to follow and understand, while others may be more abstract or open-ended, making it more challenging to observe and analyse the processes involved. In activities with a higher level of observability, learners may easily identify and apply CT competencies. Conversely, activities with a lower level of observability may require learners to engage in more exploratory, creative thinking, potentially activating problem setting competencies.
- *Task type*: typically, unplugged activities are designed with a single objective that requires the learner to find one specific element, such as the algorithm, initial state, or final state. Different aspects of CT competencies can be developed depending on the task's specific objective. For example, tasks that focus on the development of algorithms are particularly effective in fostering learners' algorithmic thinking skills. In a formal learning environment, such tasks can give learners a deeper understanding of algorithm design principles and an opportunity to practice algorithm creation and refinement. Conversely, tasks that focus on identifying the initial or final state may be particularly useful in developing learners' problem setting skills. By recognising and defining a problem's key elements, learners can develop the necessary skills to construct effective solutions and assess them.
- *Task cardinality*: unplugged activities often involve tasks with low cardinality, meaning they can be completed in small steps. This can be advantageous for CT skill development, as learners can focus on developing specific skills.
- *Task explicitness*: in unplugged activities, the given task elements are often explicit and easy to understand, which can be beneficial for novice learners who are in the early stages of developing their CT skills. This feature can aid in developing algorithm design and problem setting skills.
- *Task constraints*: unplugged activities often have few constraints, allowing learners to experiment and explore different solutions. This can help develop computational problem setting competencies as well as algorithmic concepts.

- *Algorithm representation*: in unplugged activities, the algorithm is often represented visually or physically, which can encourage the development of algorithm design and problem setting skills.

Based on the features, specific CT competencies are more likely to be activated in unplugged activities, including problem setting skills, such as pattern recognition, decomposition and abstraction, and fundamental computer science concepts.

However, unplugged activities also have some limitations, such as limited tool functionalities, the frequent lack of a tool to write the algorithm, and the inability to reset the system. These drawbacks can hinder the development of assessment competencies and should be considered when using or designing CT activities to ensure a balance between developing different CT competencies.

4.1.2 Robotics activities

Robotic activities can be an effective means of promoting the activation of CT skills. In this discussion, we will analyse the enabling and inhibiting features of four different robotic activities, namely the Thymio Lawnmower Mission, Remote Rescue with Thymio II, Ozobot maze and Mini-golf challenge with micro:bit, and how these features relate to the activation of CT skills.

- *Tool functionalities*: robotics activities typically involve a wide range of tool functionalities that students can access, spanning from basic programming concepts, such as variables and sequences, to more advanced ones like loops, parallelism, and events. This abundance of functionalities enables diverse problem-solving approaches, but it may also promote a trial-and-error approach, where students resort to trying out various functionalities without comprehending their underlying principles or the most effective ways to use them. To overcome this limitation, robotics activities often encourage the development of problem setting competencies by limiting access to the programming interface. While still allowing students to explore and experiment with the tool functionalities, it can foster deeper engagement with the problem and facilitate cognitive processes related to problem understanding, idea generation, and solution formulation, which are critical components of CT abilities.
- *System resettability*: in the context of robotics activities, it is common for problem solvers to have direct control over the system by repositioning the agent in the environment and restarting the task. Additionally, the programming platforms used for these activities often provide a reset functionality, which enables students to test and refine their designs through iterative experimentation. The ability to reset the system allows for efficient debugging, a critical component of problem-solving and CT. However, it is worth noting that the trial-and-error approach to programming, where the algorithm is developed incrementally based on feedback received, can be time-consuming and may not be an optimal learning strategy for all students.
- *System observability*: in robotics activities, the ability to observe the system is of paramount importance, as it enables students to monitor the robot's behaviour and analyse the results of their algorithms. The system's observability, coupled with the reset feature, allows students to test and refine their algorithms based on the robot's behaviour. Nevertheless, relying solely on this approach can result in an endless cycle of trial and error. By physically separating the problem solvers from the system, and providing limited visual feedback that is asynchronous and delayed, students are encouraged to cultivate problem setting skills in addition to algorithmic ones, overcoming the trial and error limitation.
- *Task type*: the type of task used in educational robotics and physical computing activities can significantly impact the development of students' competencies. While educational robotics tasks typically have a single objective, usually finding the algorithm, physical computing activities often involve multiple objectives, which require students to achieve many goals to complete the task successfully. For example, a physical computing task may require finding both the algorithm and the final state of the robot. Novice learners may find activities with a single objective more beneficial, as they provide a clear and specific goal for the learner to work towards. Such tasks also enable students to apply appropriate strategies and techniques to solve the task, thereby promoting the development of algorithmic thinking skills in a formal environment. However, this approach may limit the development of advanced problem setting skills, which are essential for tackling complex real-world problems. Therefore, physical computing activities with multiple objectives may be more effective in developing these skills, as they require learners to engage in more extensive problem-solving activities and develop their skills in identifying and assessing the critical components of a problem.
- *Task cardinality*: in educational robotic activities, the task cardinality is often one-to-one.
- *Task explicitness*: in educational robotic activities, the tasks are typically characterised by explicit in-

structions and well-defined elements. On the other hand, physical computing activities tend to have fewer explicit elements and may require more critical thinking and problem-solving skills. Tasks with more explicit elements may limit critical thinking and require less problem setting, as students can rely more on following instructions. In contrast, less explicit tasks require students to engage in problem setting, utilising skills such as pattern recognition, abstraction, decomposition, and data representation. Consequently, in the former case, students may be more inclined to jump directly into programming the agent's behaviour, while in the latter case, they may focus more on analysing the task's elements to develop a solution.

- *Task constraints*: robotics activities are typically unconstrained, providing students with more freedom and opportunities to experiment with various approaches to algorithm design without necessarily undergoing a rigorous problem setting phase. Novice learners may benefit from this feature, as it can encourage them to be more creative and exploratory in their problem-solving process. However, it may also limit the development of more advanced competencies.
- *Algorithm representation*: in robotics activities, the algorithm is an essential component, which is typically manifest and written. Engaging students in activities that involve a written algorithm can offer several benefits, such as promoting more logical thinking and problem-solving skills. By requiring the students to read and interpret the algorithm, they are encouraged to think critically about the problem and use logical reasoning to understand the steps required to achieve the task's objectives.

Robotics activities effectively promote the development of a range of CT competencies. The nature of the activity, as well as its goals and constraints, play an important role in developing different types of CT skills. Some activities emphasise the cultivation of problem setting skills, promoting critical and logical thinking, while others prioritise algorithmic thinking, fostered by trial and error approaches.

While robotics activities have many benefits for developing CT skills, they also have some limitations. Robotics activities typically involve specific tasks or challenges that limit students' ability to explore their ideas and creativity. This often leads to a trial and error approach, wherein students focus solely on finding a solution that works rather than developing a robust and efficient algorithm that underlies the problem. Nonetheless, strategies can be implemented to mitigate such issues that encourage students to engage in deeper think-

ing before executing their solutions. Disincentive strategies such as blocking access to the programming interface, physically separating the system from the problem solver, and inserting a time delay in the visual feedback can encourage students to think critically about their solutions and develop a comprehensive understanding of the problem at hand.

4.1.3 Virtual activities

Virtual activities offer a wide range of possibilities for the development of CT skills, making them an ideal tool for CTP implementation. The three virtual activities presented in this study, Classic maze, Store the Marbles, and Zoombinis Allergic Cliffs Puzzle, have different characteristics and features that impact the activation of CT skills and the overall effectiveness of the CTP in eliciting the competencies they are intended to develop.

- *Tool functionalities*: typically, virtual activities offer programming platforms or virtual games, with the former providing more tool functionalities than the latter. Despite the availability of tool functionalities in virtual activities, concepts such as parallelism and events are often overlooked in both cases. Activities that offer a wide range of tool functionalities can enable learners to experiment with various concepts, providing opportunities for them to explore their functions and ultimately develop their CT skills.
- *System resettability*: virtual activities offer the convenience of a reset function, allowing learners to start over the activity easily. This characteristic renders virtual activities useful for developing algorithmic thinking and performing assessment tasks. For instance, learners can utilise this feature to debug the code and verify the solution's correctness. Furthermore, learners can leverage the reset function to explore the generalisation and optimisation of the solution, promoting more advanced competencies. This unique feature offers a significant advantage over other learning activities, which may not offer the same level of flexibility for learners to test different solutions and refine their approaches.
- *System observability*: virtual activities provide learners with a high degree of observability, as they often include features that offer immediate feedback on the outcome of a user's decisions. Such real-time feedback can be highly valuable in promoting the development of CT skills, allowing learners to experiment with different strategies and observe their impact in real time. Moreover, the observability of virtual activities, combined with the ability to reset the system, can also enhance learners' problem

setting skills by enabling them to analyse a given situation and evaluate the effectiveness of their chosen strategy based on the system's outcomes.

- *Task type*: in virtual activities, there is typically only one objective, often centred around finding the algorithm to solve a problem. While this approach can effectively develop learners' algorithmic thinking skills in a controlled environment, it may not fully represent the complexity of real-world problems often requiring multiple objectives.
- *Task cardinality*: virtual activities often involve many-to-one cardinality of tasks, which can provide learners with opportunities to develop and practice different CT skills. This feature can enhance learners' ability to develop problem setting skills such as decomposition and pattern recognition, as multiple scenarios must be considered to achieve a task solution. As a result, learners are encouraged to explore multiple ways of approaching a problem, experimenting with different solutions, practising breaking complex problems into smaller, more manageable parts and identifying common patterns that can be applied to solve related problems. This can help learners develop their abstraction and algorithmic thinking skills by identifying and implementing reusable strategies for solving related problems.
- *Task explicitness*: virtual activities typically provide learners with explicit instructions for completing the task, which can limit their critical thinking and problem setting skills. This is because learners are not required to engage in the process of identifying the problem or generating possible solutions on their own. Instead, they are presented with a specific problem and given clear instructions for solving it.
- *Task constraints*: virtual activities often lack constraints, limiting the development of critical thinking and problem solving skills among learners. Without constraints, learners may not be challenged to think deeply about how to approach a problem and may not consider multiple possible solutions or evaluate the effectiveness of their chosen solution.
- *Algorithm representation*: virtual activities that require learners to engage in coding can provide opportunities for them to develop algorithmic thinking skills and practice debugging. In contrast, virtual games that allow the problem solver to interact directly with the environment without using external tools may limit the debugging abilities. However, they may promote more problem setting skills, such as identifying patterns and developing strategies to tackle complex situations. This is because learners must rely on their own problem-solving skills and

strategies to progress through the game rather than following predefined coding concepts.

When considering the development of CT competencies through virtual activities, the specific competencies that are activated can vary depending on the characteristics of the activity. One advantage of virtual activities is their high degree of observability, allowing for immediate feedback and the opportunity for experimentation and iteration, which can lead to the development of assessment skills such as debugging, system state verification, constraint verification optimisation and generalisation. Virtual coding activities aim primarily at developing learners' algorithmic thinking skills, typically providing learners with tools and functionalities to experiment with different concepts and explore their functions. On the other hand, virtual games are designed to enhance learners' problem setting skills, such as problem decomposition, pattern recognition, and abstraction.

Despite their potential benefits in developing CT skills, virtual activities may have some limitations. Firstly, some virtual activities may require learners to rely on trial and error, limiting their ability to develop problem setting skills. Secondly, the absence of physical objects in virtual activities may undermine learners' involvement and pleasure in the learning process. Finally, virtual activities may provide limited creativity, exploration, and experimentation opportunities. These drawbacks highlight the importance of designing virtual activities that balance the development of various CT competencies while considering learners' engagement, motivation, and enjoyment.

5 Conclusion

In this article, we presented a comprehensive framework for evaluating and refining existing computational thinking problems (CTPs) in education or problems that require computational thinking (CT) skills to be solved and creating new ones, to assess CT competencies effectively.

One of the main contributions of this study is that CTPs consist of several components, including the system, problem solver, and task. The system comprises the environment and the agent, while the problem solver is a human or group of people who can solve tasks requiring algorithms. Our study has identified that the artefactual environment, or the set of reasoning and interaction tools available to the problem solver, plays a crucial role in developing CT abilities.

The proposed framework allows for the systematic analysis of CTPs according to their characteristics, which

are relevant for eliciting and assessing CT skills in educational contexts. These characteristics include tool functionalities, system observability and resettability, task details, such as type, cardinality, constraint and explicitness, and the algorithm representation. Our study provides a catalogue of skills typically used to assess student abilities in this area and defines a mapping between the identified characteristics of the CTPs and these competencies, drawing from existing frameworks and literature. This mapping can be used to determine how these features influence the development of CT abilities, offering a clear understanding of the relationship between the features of the problems and the CT skills they activate, those that can be promoted or cannot be activated. This framework ultimately can be used to define, analyse, revise, and create learning activities.

The developed framework was demonstrated through its application to a diverse set of CTPs from the literature, including unplugged, robotic, and virtual educational activities. The resulting analysis provided a comprehensive understanding of each activity category's unique features and challenges, as well as the specific CT competencies they activate. With this information, it is possible to identify the best category of activities to develop a specific category of expertise.

Unplugged activities are the best-suited category of CTPs for developing problem setting skills due to their hands-on and tangible approach, which allows learners to engage with problems physically and interactively. Virtual games can also effectively develop problem setting skills as they often provide a contextualised and engaging environment. Physical computing activities in the robotics domain effectively develop problem setting skills. These activities necessitate programming robots to attain multiple objectives involving the manipulation and interaction of real-world objects. As a result, learners must use critical thinking when attempting to solve problems and consider all factors of the task at hand. Additionally, the implicit elements within these activities, which may not be immediately obvious, further promote the ability to set achievable goals and identify the necessary steps to attain them.

When it comes to developing algorithmic skills, both robotic and virtual activities are the most effective. Robotic activities, in particular, provide an excellent opportunity to enhance algorithmic thinking by involving learners in designing the algorithmic sequence for the robot to fulfil the given task. Virtual coding activities, on the other hand, focus on developing learners' algorithmic thinking skills by providing them with tools and functionalities to implement algorithms and experiment with different concepts. In contrast, unplugged

activities are not as effective in formally developing algorithmic competencies, as they do not provide the same level of complexity and sophistication as robotic and virtual activities.

For the development of assessment skills, virtual activities are the best choice, as they often provide immediate feedback and allow learners to test and improve their solutions, whereas unplugged and robotic activities do not offer the same level of feedback or refinement options.

In conclusion, the proposed framework enables educators not only to determine which skills are developed by a particular CTP and to revise it appropriately to align with the desired learning objectives but also to select activities that target specific CT skills, thereby facilitating efficient and effective learning. Moreover, the framework can be used to create new CTPs in educational settings.

Notwithstanding the contributions of this study, there are limitations to consider. The presented framework is based on a restricted set of activities, which may not represent all possible scenarios. To improve the framework's validity and applicability, future research should expand this set and further validate and refine the framework. Additionally, the study's focus on a specific set of CT skills, which may not cover the full range of competencies that can be developed through CTPs. Therefore, further empirical studies should aim to validate the framework and the catalogue of competencies. While the framework is primarily designed for CTPs in educational contexts, future research could explore its applicability in other contexts and investigate the impact of different task types, age groups, cultural backgrounds, and educational levels on the framework's validity and generalisability.

Abbreviations

CT	computational thinking
CT-cube	computational thinking cube
CTP	computational thinking problem
CTt	Computational Thinking Test
R2T2	Remote Rescue with Thymio II
STEM	Science, Technology, Engineering and Math

Conflict of interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

This research was funded by the Swiss National Science Foundation (SNSF) under the National Research Program 77 (NRP-77) Digital Transformation (project number 407740_187246).

References

- ALGOREA AFi (2020) Ranger les billes. <https://parcours.algorea.org/contents/4707-4702-1471479157476024035-1312565015631453321-1609762309306083784-4230985342204608/>, [Accessed 24 January 2022]
- Angeli C, Voogt J, Fluck A, Webb M, Cox M, Malyn-Smith J, Zagami J (2016) A k-6 computational thinking curriculum framework: Implications for teacher knowledge. *Journal of Educational Technology & Society* 19(3):47–57
- Assaf D, Betschart S, Moser A, Curtins J, Steiner M, Walker N (2021) «iMake-IT»: Invent, Code, and Shape Your World! a workshop live stream from a science communication project where 5th grade students work in a makerspace on open-ended projects. In: *FabLearn Europe/MakeEd 2021-An International Conference on Computing, Design and Making in Education*, pp 1–3
- Ball T, Protzenko J, Bishop J, Moskal M, De Halleux J, Braun M, Hodges S, Riley C (2016) Microsoft touch develop and the bbc micro: bit. In: *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, IEEE, pp 637–640
- Barbero M (2020) Backward reasoning in problem-solving situations: a multidimensional model. PhD thesis, Universidad Complutense de Madrid, Facultad de Ciencias Matemáticas
- Barbero M, Gómez-Chacón IM (2018) Analysing regressive reasoning at university level. In: *INDRUM 2018*
- Barr V, Stephenson C (2011) Bringing computational thinking to k-12: What is involved and what is the role of the computer science education community? *ACM Inroads* 2(1):48–54, DOI 10.1145/1929887.1929905, URL <https://doi.org/10.1145/1929887.1929905>
- Bell GI (2007) Triangular peg solitaire unlimited. <https://doi.org/10.48550/ARXIV.0711.0486>
- Bell GI (2008) Solving triangular peg solitaire. *Journal of Integer Sequences* 11(2):3
- Bell T, Alexander J, Freeman I, Grimley M (2009) Computer science unplugged: school students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology* 13
- Berlekamp ER, Conway JH, Guy RK (2004) *Winning ways for your mathematical plays*, volume 4. AK Peters/CRC Press, Location New York
- Bers MU, Horn MS (2010) Tangible programming in early childhood. *High-tech tots: Childhood in a digital world* 49:49–70
- Bocconi S, Chiocciariello A, Dettori G, Ferrari A, Engelhardt K, Kampylis P, Punie Y (2016) Developing computational thinking: Approaches and orientations in k-12 education. In: *Ed-Media+ Innovate Learning, Association for the Advancement of Computing in Education (AACE)*, pp 13–18
- Bocconi S, Chiocciariello A, Kampylis P, Dagienė V, Wastiau P, Engelhardt K, Earp J, Horvath MA, Jasutė E, Malagoli C, et al. (2022) Reviewing Computational Thinking in Compulsory Education. Tech. rep., Joint Research Centre (Seville site), DOI 10.2760/126955
- Brackmann CP, Román-González M, Robles G, Moreno-León J, Casali A, Barone D (2017) Development of Computational Thinking Skills through Unplugged Activities in Primary School. In: *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, Association for Computing Machinery, New York, NY, USA, WiPSCE '17, p 65–72, DOI 10.1145/3137065.3137069
- Bravo FA, González AM, González E (2017) A review of intuitive robot programming environments for educational purposes. In: *2017 IEEE 3rd Colombian Conference on Automatic Control (CCAC)*, IEEE, pp 1–6
- Brennan K, Resnick M (2012) New frameworks for studying and assessing the development of computational thinking. In: *Proceedings of the 2012 annual meeting of the American educational research association*, Vancouver, Canada, vol 1, p 25
- Bryndová L, Mališů P (2020) Assessing the current level of the computational thinking within the primary and lower secondary school students using educational robotics tasks. In: *2020 The 4th International Conference on Education and Multimedia Technology*, Association for Computing Machinery, New York, NY, USA, ICEMT 2020, p 239–243, DOI 10.1145/3416797.3416819
- Chevalier M, Giang C, Piatti A, Mondada F (2020) Fostering computational thinking through educational robotics: a model for creative computational problem solving (CCPS). *International Journal of STEM Education* 39, DOI 10.1186/s40594-020-00238-z
- Codeorg (2015) *Instructor Handbook-Code Studio Lesson Plans for Courses One, Two, and Three*. <https://code.org/curriculum/docs/k-5/complete.pdf>, [Accessed 4 May 2022]
- Csizmadia A, Curzon P, Dorling M, Humphreys S, Ng T, Selby C, Woollard J (2015) Computational thinking-a guide for teachers. Tech. rep., Computing at School, URL <https://eprints.soton.ac.uk/424545/>
- Cui Z, Ng OL (2021) The interplay between mathematical and computational thinking in primary school students' mathematical problem-solving within a programming environment. *Journal of Educational Computing Research* 59(5):988–1012
- Del Olmo-Muñoz J, Cózar-Gutiérrez R, González-Calero JA (2020) Computational thinking through unplugged activities in early years of Primary Education. *Computers & Education* 150:103832, DOI 10.1016/j.compedu.2020.103832
- Ford MJ (2015) Educational Implications of Choosing “Practice” to Describe Science in the Next Generation Science Standards. *Science Education* 99(6):1041–1048, DOI <https://doi.org/10.1002/sce.21188>
- France-IOIorg (2004) France-IOI. <http://www.france-ioi.org>, [Accessed 24 January 2022]
- Grover S, Pea R (2017) Computational Thinking: A Competency Whose Time Has Come. *Computer science education: Perspectives on teaching and learning in school* 19:1257–1258
- Heersmink R (2013) A taxonomy of cognitive artifacts: Function, information, and categories. *Review of Philosophy and Psychology* 4(3):465–481, DOI 10.1007/s13164-013-0148-1
- Kalelioğlu F (2015) A new way of teaching programming skills to K-12 students: Code.org. *Computers in Human Behavior* 52:200–210, DOI 10.1016/j.chb.2015.05.047
- Lafuente Martínez M, Lévêque O, Benítez Baena I, Hardebolle C, Dehler Zufferey J (2021) Assessing Computational Thinking: Development and Validation of the Algorithmic Thinking Test for Adults. *Journal of Educational Computing Research* DOI 10.1177/07356331211057819
- Lafuente Martínez M, Lévêque O, Benítez I, Hardebolle C, Zufferey JD (2022) Assessing computational thinking: Development and validation of the algorithmic thinking test for adults. *Journal of Educational Computing Research* p 07356331211057819

- Lovett A (2017) Coding with Blockly. Cherry Lake Publishing, Ann Arbor, Michigan
- Magenat S, Rétornaz P, Bonani M, Longchamp V, Mondada F (2011) Aseba: A modular architecture for event-based control of complex robots. *IEEE/ASME Transactions on Mechatronics* 16(2):321–329, DOI 10.1109/TMECH.2010.2042722
- Makecode (2016) Microsoft MakeCode for Micro:bit. <https://makecode.microbit.org>, [Accessed 4 May 2022]
- Microbit (2016) BBC's Make It Digital initiative, Micro:bit Educational Foundation. <https://microbit.org>, [Accessed 4 May 2022]
- Mondada F, Bonnet E, Davrajh S, Johal W, Stopforth R (2016) R2t2: Robotics to integrate educational efforts in south africa and europe. *International Journal of Advanced Robotic Systems* 13(5):1729881416658165, DOI 10.1177/1729881416658165
- Mussati A, Giang C, Piatti A, Mondada F (2019) A Tangible Programming Language for the Educational Robot Thymio. In: 2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA), IEEE, pp 1–4, DOI 10.1109/IISA.2019.8900743
- Noone M, Mooney A (2018) Visual and textual programming languages: a systematic review of the literature. *Journal of Computers in Education* 5(2):149–174
- Osborne J (2014) Teaching Scientific Practices: Meeting the Challenge of Change. *Journal of Science Teacher Education* 25(2):177–196, DOI 10.1007/s10972-014-9384-1
- Piatti A, Adorni G, El-Hamamsy L, Negrini L, Assaf D, Gambardella L, Mondada F (2022) The CT-cube: A framework for the design and the assessment of computational thinking activities. *Computers in Human Behavior Reports* 5:100166, DOI <https://doi.org/10.1016/j.chbr.2021.100166>
- Riedo F, Chevalier M, Magnenat S, Mondada F (2013) Thymio ii, a robot that grows wiser with children. In: 2013 IEEE workshop on advanced robotics and its social impacts, IEEE, pp 187–193
- Rodríguez-Martínez JA, González-Calero JA, Sáez-López JM (2020) Computational thinking and mathematics using scratch: an experiment with sixth-grade students. *Interactive Learning Environments* 28(3):316–327
- Román-González M (2015) Computational thinking test: Design guidelines and content validation. In: EDULEARN15 Proceedings, IATED, pp 2436–2444
- Román-González M, Moreno-León J, Robles G (2017a) Complementary Tools for Computational Thinking Assessment. In: Proceedings of International Conference on Computational Thinking Education (CTE 2017), S. C Kong, J Sheldon, and K. Y Li (Eds.). The Education University of Hong Kong, pp 154–159
- Román-González M, Pérez-González JC, Jiménez-Fernández C (2017b) Which cognitive abilities underlie computational thinking? criterion validity of the computational thinking test. *Computers in human behavior* 72:678–691
- Román-González M, Pérez-González JC, Moreno-León J, Robles G (2018) Can computational talent be detected? Predictive validity of the Computational Thinking Test. *International Journal of Child-Computer Interaction* 18:47–58, DOI 10.1016/j.ijcci.2018.06.004
- Roth WM, Jornet A (2013) Situated cognition. *Wiley Interdisciplinary Reviews: Cognitive Science* 4(5):463–478, DOI 10.1002/wcs.1242
- Rowe E, Almeda MV, Asbell-Clarke J, Scruggs R, Baker R, Bardar E, Gasca S (2021) Assessing implicit computational thinking in Zoombinis puzzle gameplay. *Computers in Human Behavior* 120:106707, DOI 10.1016/j.chb.2021.106707
- Selby C, Woollard J (2013) Computational thinking: the developing definition. Tech. rep., University of Southampton, URL <https://eprints.soton.ac.uk/356481/>
- Selby CC (2014) How can the teaching of programming be used to enhance computational thinking skills? PhD thesis, University of Southampton
- Shin J, Siegwart R, Magnenat S (2014) Visual programming language for thymio ii robot. In: Conference on interaction design and children (idc'14), ETH Zürich
- Shute VJ, Sun C, Asbell-Clarke J (2017) Demystifying computational thinking. *Educational Research Review* 22:142–158, DOI 10.1016/j.edurev.2017.09.003
- Studiocodeorg (2020a) Angry birds hitting the green pig maze. <https://studio.code.org/hoc/1>, [Accessed 24 January 2022]
- Studiocodeorg (2020b) Classic Maze. <https://studio.code.org/s/hourofcode>, [Accessed 24 January 2022]
- Studiocodeorg (2020c) Plants vs Zombies maze. <https://studio.code.org/hoc/12>, [Accessed 24 January 2022]
- Tall D (2006) A Theory of Mathematical Growth Through Embodiment, Symbolism and Proof. *ANNALES de DIDACTIQUE et de SCIENCES COGNITIVES, IREM de STRASBOURG* 11:195 – 215
- Tall D (2013) How Humans Learn to Think Mathematically: Exploring the Three Worlds of Mathematics. Cambridge University Press, Cambridge, DOI 10.1017/CBO9781139565202
- Tall D (2020) Making Sense of Mathematical Thinking over the Long Term: The Framework of Three Worlds of Mathematics and New Developments. *MINTUS: Beiträge zur mathematischen, naturwissenschaftlichen und technischen Bildung Wiesbaden*: Springer URL <https://homepages.warwick.ac.uk/staff/David.Tall/pdfs/dot2020a-3worlds-extension.pdf>
- Thalheim B (2000) The database design process. In: *Entity-Relationship Modeling*, Springer, Berlin, Heidelberg, pp 13–28
- Tikva C, Tambouris E (2021) Mapping computational thinking through programming in K-12 education: A conceptual model based on a systematic literature Review. *Computers & Education* 162:104083, DOI 10.1016/j.compedu.2020.104083
- Weintrop D, Beheshti E, Horn M, Orton K, Jona K, Trouille L, Wilensky U (2016) Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology* 25(1):127–147
- Weintrop D, Rutstein DW, Bienkowski M, McGee S (2021) Assessing computational thinking: an overview of the field. *Computer Science Education* 31(2):113–116, DOI 10.1080/08993408.2021.1918380
- Wing J (2011) Research notebook: Computational thinking—what and why. *The link magazine* 6:20–23
- Wing JM (2006) Computational thinking. *Communications of the ACM* 49(3):33–35, DOI 10.1145/1118178.1118215
- Zoombinis (2021) Zoombinis. <https://www.terc.edu/zoombinis/about/>, [Accessed 4 May 2022]

A The relationship between CTP and CT skills

This section provides a detailed analysis of the content of Table 6, organised by the main levels of the activity dimension: problem setting, algorithm, and assessment. For each dimension, we first describe the link between skills and the required features, then the link between the skills and the characteristics that act as catalysts.

The computational thinking problem (CTP) features we considered are the functionalities allowed to the problem solver by

the tools, the property of the system, such as resettability and observability, and finally, the trait of the task, including the elements required to be found, the type of cardinality of the elements given and to be found, the presence of constraints and the type of representation of the algorithm.

A.1 Indispensable features to activate problem setting competencies

Starting from the problem setting skills, to activate the “data collection” competence, the only requirement is that the tools available allow the use and recognition of variables. Without variables, there would be nothing to collect data on. The “pattern recognition” competence requires the presence of repetitions or functions since they allow the identification of repeating patterns in the data. The “decomposition” competence requires the presence of functions or sequences that can be used to break down a complex problem into smaller more manageable components. The “abstraction” competence demands the presence of variables to represent key concepts and functions to encapsulate and reuse specific behaviour within a single, self-contained unit, simplifying the original task and allowing the problem solver to reason about the problem at a higher level of abstraction. Finally, the “data representation” competence requires only variables to represent data.

A.2 Features triggering problem setting competencies

Generally, the attributes of the problem not required directly to activate the skills can influence them in some way. In the case of the characteristics of the tools, for example, variables also play a role in “pattern recognition” and “decomposition”, as they can be used to store patterns or parts of a complex problem. Then, operators can be useful for the “decomposition” of the problem into smaller parts. At the same time, sequences can contribute to the processes of “pattern recognition” and “abstraction”, helping the problem solver to identify patterns or regularities in the data, as well as the key concepts or essential elements of a problem, but also in “data representation” to organise and present data in a clear and meaningful way. Repetitions can influence the activation of the problem setting skills of “decomposition”, “abstraction”, and “data representation” because they can make the task more complex thus requiring the problem solver to use these practices. Similarly, conditionals can help to structure and simplify a problem, making it more manageable and easier to solve, enabling “pattern recognition”, “decomposition”, “abstraction” and “data representation”. Functions can influence the activation of “data representation” by helping the problem solver organise and structure data. Parallelism can influence the activation of the problem-solving skill of “decomposition” as it allows for breaking the problem into independent subtasks that can be executed simultaneously. Finally, events can trigger “data collection” at a specific point in time. For the sake of the characteristic of the system, resettability allows the problem solver to start over and try different approaches to solving the problem, thus stimulating can problem setting skills such as “data collection”, “pattern recognition”, and “decomposition”, as they can test different strategies and collect data on their effectiveness. On the other hand, if the system is not resettable, the problem solver may have to rely more on “abstraction” and “data representation” skills to find a solution, as they cannot try different approaches and must work with the information they have available. In general, a resettable system allows more freedom for the problem

solver, giving a chance to explore different solutions. In contrast, a non-resettable system may require more creativity to find a solution. If the system is observable, the problem solver would likely use skills related to “data collection”, as he can directly perceive and then gather information about the system’s state and properties. Additionally, he may use skills related to “pattern recognition”, such as identifying patterns or trends in the data collected. These skills can help the problem solver understand the system’s current state and make informed decisions about how to solve the task. Conversely, suppose the system is not observable. In that case, the problem solver may need to rely on abstract and hypothetical reasoning to devise a solution, activating “pattern recognition”, “decomposition”, and “abstraction” to understand the problem and identify possible solutions. Also “data collection” may be necessary to gather information about the system and its behaviour, even if that information is not directly observable. Additionally, the “data representation” skill can be used to organise and interpret the information they have collected to make sense of the problem and develop a solution. When there is a many-to-one cardinality in the system, it means that there is a large amount of data that needs to be processed, and multiple inputs or sources of information can be used to achieve a single goal or outcome. In this scenario, the “data collection” skill will likely be activated because the problem solver needs to gather a large amount of information to understand the problem and find a solution. Since there are multiple instances of a certain element or pattern, recognising the commonalities and differences among them would be essential to understand the overall system, leading to the use of more complex data collection and analysis strategies, thereby activating the “pattern recognition” competence. The “decomposition” and the “abstraction” skills will also likely be activated as the problem solver needs to break down the problem into smaller manageable parts and find the underlying principles and concepts in the problem to understand the overall system and find a solution. Finally, as there are multiple instances of a specific element, it would be essential to communicate them clearly and concisely, thus activating the “data representation” skill. By contrast, if there is a one-to-one cardinality in the system, the competencies of problem setting that are likely to be activated include “data collection”, “pattern recognition”, and “decomposition”. The skills “abstraction” and “data representation” are less likely to be activated since the direct correspondence between the system elements means there is less need to abstract or represent the information. It can be assumed that with implicit elements, the “data collection”, “pattern recognition”, “decomposition” and “abstraction” competencies may be activated as the problem solver needs to infer information from the context or the environment, understand the underlying concepts or patterns in the task, decompose the problem into smaller sub-problems, and create abstract representations of the system. The same reasoning can be applied to constrained elements. Moreover, it is possible that the competence “data representation” may be activated as implicit or constrained elements may require the problem solver to think about how to represent the data in a way that accurately reflects the underlying information or constraints. Likewise, with explicit elements, the “pattern recognition”, “decomposition”, and “abstraction” competencies may also be activated, as the problem solver needs to understand and make sense of the given information, and the presence of unconstrained elements to be found may allow for more flexibility and creativity in problem-solving, potentially activating these skills, as problem solvers may need to find novel ways to organise or make connections among the elements. In this scenario, the “data collection” and “data representation” competencies may be more straightforward and not as crucial, especially when the elements are explicit and thus the information is already provided in a structured format. Regarding the

representation of the algorithm, overall all problem setting competencies may be activated. Nevertheless, a manifest algorithm makes the problem solver's task easier by providing a clear set of instructions and reducing the need for "pattern recognition" and "decomposition". However, a not manifest algorithm can promote more "pattern recognition", "decomposition", and "abstraction" as the problem solver needs to infer the algorithm from the problem statement and available information and cannot represent it.

A.3 Indispensable features to activate algorithmic competencies

For the algorithm dimension, each competence to be activated requires that the corresponding characteristic of the tool is enabled. For example, to activate the "variable" skill, the tools used by the problem solver should include variables. Moreover, in a formal artefactual environment, the task requires that the algorithm is not given but has to be found. Otherwise, it is possible only to assess the problem solver ability to recognise these skills and apply them, but not create an algorithm from scratch.

A.4 Features triggering algorithmic competencies

Again, some characteristics can also influence the activation of algorithmic competencies. Regarding the characteristics of the tools, for example, the presence of variables may influence the activation of all the other algorithmic skills, since they provide a fundamental building block for creating algorithms and can be used in conjunction with other algorithmic structures. Similarly, operators influence the activation of all algorithmic skills. The presence of sequences may influence the activation of "variables", "operators", "repetitions" and "functions"; repetitions may influence the activation of "variables", "operators", "sequences" and "functions"; the presence of conditionals may influence the activation of "variables", "operators" and "events"; functions may influence the activation of "variables", "operators", "sequences" and "repetitions"; the presence of parallelism may influence the activation of "variables" and "operators"; while events may influence the act "variables", "operators" and "conditionals". The resetability or non-resetability of a system is not relevant for activating or not algorithmic competencies. The system's observability, or the ability to observe the agent's actions and the system's state, allows tracking of how the algorithm is executing and makes it easier for the problem solver to identify these procedures used by the agent. Instead, a non-observable system may activate the skills of "variables", "operators", "sequences", and "conditionals", since the problem solver may need to rely more heavily on their ability to reason about the system and make inferences based on limited information. Regarding the ratio of elements given and to be found, from one side, a one-to-one cardinality may influence the activation of the algorithmic skill "variables" that can be used to define the direct correspondence between the elements in the system and their representations, but also of "operators" and "conditionals" proper to manipulate them and necessary to ensure the correct mapping. On the other side, a many-to-one cardinality can make it more challenging to understand the relationship between the given elements and those to be found, impacting the ability to understand the algorithm and its parts and enforcing the use of certain types of structures. For example, the problem solver can keep track and map multiple instances to a single object using "variables". If the task at hand involves processing multiple pieces of data and producing a single result, a

"repetition" can be used to iterate over the inputs. Similarly, "conditionals" can be used if the task requires selecting one output out of multiple possibilities based on certain conditions. In contrast "functions" can be used to modularise the code and make it more organised and maintainable. Finally, "parallelism" can be used to speed up the processing of multiple inputs by running multiple iterations simultaneously. Further, explicit elements provide clear and specific information about the task that must be solved, allowing the problem solver to use all the algorithmic structures to manipulate and work with that information to achieve the desired outcome. Besides, the presence of implicit elements in the task makes it more difficult for the problem solver to understand and determine the necessary steps to solve the task, thus some algorithmic structures may need to be used to compensate for this shortcoming. For example, "variables" would be necessary to store and track the values of implicit elements, "operators", "sequences", "repetitions", "conditionals", and "functions" would help make decisions and perform actions based on the values of these variables. These algorithmic structures would allow the problem solver to explain the implicit elements effectively and develop more sophisticated and efficient solutions. Similarly, the space for possible solutions is limited when constrained elements are involved in the task and it may be necessary to use some algorithmic structures to ensure those constraints are met. For example, while solving a puzzle, the final state and algorithm have to be found, and they have constraints: the problem solver has to fit several pieces together to form a complete image, pieces must fit together to form a specific figure, and certain pieces can only be placed in certain orientations. To solve this task, the problem solver might use a combination of algorithmic structures such as "variables" to keep track of the current state of the puzzle and the position of the pieces, "operators" to manipulate the pieces and move them around, "sequences" to try different combinations of pieces, "repetitions" to keep trying different combinations until the puzzle is complete, and "conditionals" to check if the current combination of pieces meets the constraints. Additionally, "functions" could also be used to group sets of repeated actions. Finally, how the algorithm is represented can affect the activation of various algorithmic structures depending on the type of representation used. Considering different types of tools, each can be more suited to activating one skill rather than another. If the algorithm is represented in a mathematical notation, the use of "operators" may be more prominent. On the one hand, if the algorithm is represented in a visual block-based programming language, the use of "sequences", "repetitions" and "conditionals" may be more intuitive and easier to activate. On the other hand, if the algorithm is represented in a text-based programming language, the use of "variables" and "functions" may be more natural to activate. Finally, robotic programming languages are usually designed for detecting and responding to "events", such as sensor readings or other inputs. They often have built-in functionalities for concurrent execution of multiple instructions, allowing "parallelism". Overall, the choice of representation can affect the ease and familiarity of activating different algorithmic structures and may also shape the problem solver's understanding and ability to apply them effectively.

A.5 Indispensable features to activate assessment competencies

Finally, in the assessment category, all skills have in common the need for the system to be resettable for the skill to be activated. For example, in "algorithm debugging", if the instruction cannot be reversed, it is impossible to revise and test the previous code

versions. Thus, resettability is necessary to debug the algorithm in a controlled and repeatable environment. The same applies to correcting errors in the state and constraints and improving the solution's performance or generalising it. In the specific case of "algorithm debugging", this skill can be activated in all the artefactual environments if the algorithm has to be found and if it is manifest because it allows the user to understand and check the logic and the flow of the algorithm. This is essential to identify and fix any bugs or errors in the algorithm. While it becomes increasingly important to have a written algorithm as the difficulty level of the artefactual environment rises, it may still be possible to solve the problem without one. However, the absence of a written algorithm may make it more challenging to analyse or modify the solution in a formal setting, as the artefactual environment is more abstract and requires a more in-depth understanding. For this reason, we considered the skill required in this context. The "system state verification" competence can be activated in all three artefactual environments if at least one between the initial and final states must be found. In embodied environments, direct physical interactions with the system provide a way to observe its state without needing a manifest algorithm. However, in symbolic and formal environments, a manifest representation of the algorithm, written in the case of formal environments, is crucial to fully understand its logical flow, verify the system state, and perform formal reasoning about its correctness. This may involve analysing the symbolic representation to understand how it impacts the system state. To activate the "constraints validation" competence, it is blatant that the other necessary characteristic is having constraints on the states to be found. To enable "optimisation", additional features are not required, while for "generalisation", variables and functions are necessary to reuse and apply the task solution to different problems.

A.6 Features triggering assessment competencies

Each tool functionality available to the problem solver can be a potential cause of error in the algorithm. For example, if the problem solver is unfamiliar with one of them or does not understand how to use it correctly, he may not use it at all or misuse it. This can lead to errors in the algorithm and potentially result in the problem not being solved correctly. This is why functionalities of the tools if available can activate "algorithm debugging". Also for "constraints validation", all the characteristics of the tools are influential. Above all, variables, operators, conditional and functions may allow the problem solver to perform various calculations and comparisons to check if the values assigned to the variables meet the specified constraints. Further, it could be that the constraint imposed is precisely on the algorithm and prohibits using some of these structures. The functionality of the tools available to the problem solver can greatly impact the "optimisation" of the algorithm in several ways. Parallelism allows for multiple tasks or processes to be executed simultaneously, which can greatly reduce the overall time required to complete a task. Sequences and other structures, such as loops, can also help to improve efficiency by allowing for the automation of repetitive tasks and the ability to perform actions in a specific order. Additionally, using functions and subroutines can improve the readability and maintainability of the algorithm, making it easier to identify and fix any errors that may occur. However, having access to a wide range of functionalities can make it challenging for the problem solver to choose the appropriate one for a specific task, leading to a revision of the solution to increase efficiency and performance. The competence "generalisation" can also be influenced by other characteristics of the tools. The presence of sequences and repetitions in the toolset enables the problem solver to apply the

same algorithm to different parts of a problem or task. Similarly, the inclusion of conditionals allows for the application of different algorithms depending on the specific conditions of the task. Furthermore, the presence of events in the toolset allows for creating algorithms that can respond to different triggers within the problem, leading to a greater generalisation of the solution and the ability to adapt to changes within the problem. In terms of observability, an observable system allows the problem solver to have a clear understanding of the system's state and the output of the algorithm, which can aid in identifying and addressing errors and inefficiencies and performance issues, as well as recognising patterns or regularities that can be generalised to new or different situations. However, it is essential to note that while observability can aid in all assessment skills, it is not strictly necessary for their activation. For example, one could still perform "algorithm debugging" and "system state verification" on a non-observable system, though it may be more difficult. Similarly, "generalisation" can still occur without perfect observability, but it may be harder to identify patterns and regularities without direct access to the system state. If the system has a many-to-one cardinality, the competence of "generalisation" may be activated as it would be necessary to apply the same algorithm to different inputs or outputs. If the system contains implicit elements, the competencies "algorithm debugging" and "system state verification" may be activated as the problem solver may need to identify and troubleshoot any issues with the algorithm that are not immediately apparent or infer the current state of the system based on the implicit information provided. Also "generalisation" may be activated as the problem solver may need to apply the algorithm to different situations based on the implicit information provided. Finally, suppose in the system there are elements to be found with constraints. In that case, the "generalisation" skill may be activated because it requires the problem solver to adapt the task to the specific constraints and can be intended as solving a new problem using the knowledge acquired in a previous situation and adapting it to a new one.

B Analysis of the 16 prototypical CTPs

This section provides supplementary materials for the 16 activities examination presented in the "Results" section of this study. For this purpose, we adapted the graphical template from Fig. 3 describing the various components and characteristics of the specific computational thinking problems (CTPs). Additionally, we adjusted Table 6 to illustrate the relationship between the CTP characteristics and the computational thinking (CT) competencies. Notably, the tables highlight the columns corresponding to the activity features in light grey, while white columns indicate the absence of a given feature. In addition, the tables highlight the green checkmarks and the red crosses, respectively green and red, to indicate which features enable or inhibit a particular skill. Furthermore, the tables include an additional column absent in the original Table 6. This column provides insight into which CT competencies can be activated based on the various characteristics of the CTP.

By providing a comprehensive analysis of activity features and their potential impact on CT development, this section supports the findings presented in the "Results" section. This information could be useful for educators seeking to integrate CT into their teaching practices, as it offers practical insights into how specific activities can be used to promote the development of CT competencies.

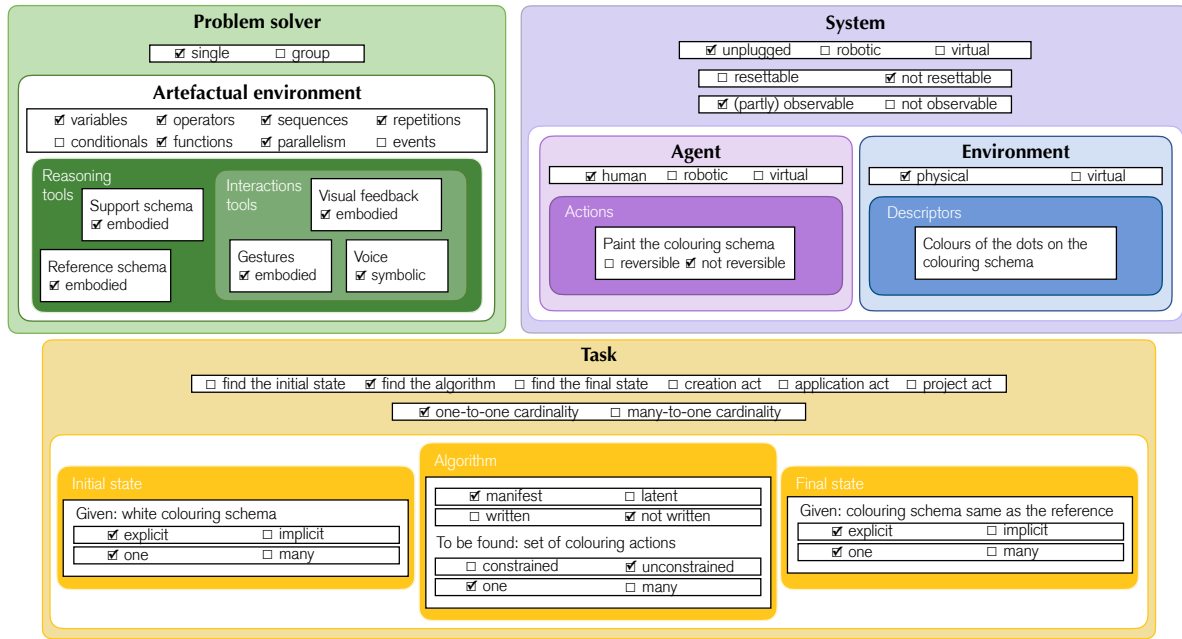


Fig. B.1 Graphical template for analysing the Cross Array Task.

Table B.1 Relationship between the Cross Array Task characteristics and CT competencies.

		Tool functionalities								System				Task											
		Variables	Operators	Sequences	Repetitions	Conditionals	Functions	Parallelism	Events	System resettable	System not resettable	System observable	System not observable	State to be found	Algorithm to be found	One-to-one cardinality	Many-to-one cardinality	Explicit elements	Implicit elements	Unconstrained elements	Constrained elements	Algorithm manifest	Algorithm latent	Algorithm written	Algorithm not written
Data collection	✓	✓						+	+	+	+	+		+	+	+	+	+	+	+	+	+	+	+	
Pattern recognition	✓	+		+	✓*	+	✓*		+	+	+	+			+	+	+	+	+	+	+	+	+	+	
Decomposition	✓	+	+	✓*	+	+	✓*	+	+			+			+	+	+	+	+	+	+	+	+	+	
Abstraction	✓	✓		+	+	+	✓			+	+	+			+	+	+	+	+	+	+	+	+	+	
Data representation	✓	✓		+	+	+	+			+	+	+			+	+	+	+	+	+	+	+	+	+	
Variables	✓	✓	+	+	+	+	+	+	+		+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+	
Operators	✓	+	✓	+	+	+	+	+	+		+	+		✓ ^F	+		+	+	+	+	+	+	+	+	
Sequences	✓	+	+	✓	+		+			+	+	+		✓ ^F			+	+	+	+	+	+	+	+	
Repetitions	✓	+	+	+	✓		+			+	+	+		✓ ^F		+	+	+	+	+	+	+	+	+	
Conditionals	✗	+	+			✓		+			+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+	
Functions	✓	+	+	+	+		✓				+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+	
Parallelism	✓	+	+					✓			+	+		✓ ^F		+	+	+	+	+	+	+	+	+	
Events	✗	+	+		+			✓			+	+		✓ ^F			+	+	+	+	+	+	+	+	
Algorithm debugging	✗	+	+	+	+	+	+	+	✓	✗	+	+		✓				+			✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F	
System state verification	✗		+	+	+	+	+	+	✓	✗	+	+	✓					+		✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F		
Constraints validation	✗	+	+	+	+	+	+	+	✓	✗	+	+							✗	✓					
Optimisation	✗	+	+	+	+	+	+	+	✓	✗	+	+													
Generalisation	✗	✓		+	+	+	✓	+	✓	✗	+	+				+		+		+					

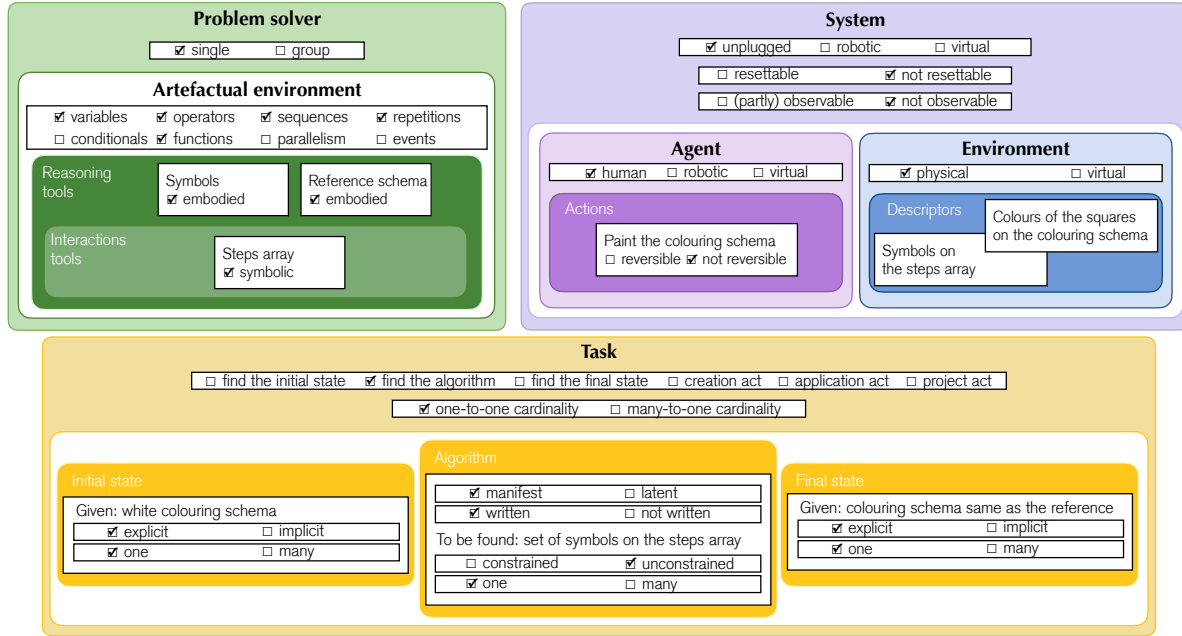


Fig. B.2 Graphical template for the analysis of the Graph Paper Programming (part 1).

Table B.2 Relationship between the Graph Paper Programming (part 1) characteristics and CT competencies.

		Tool functionalities								System				Task											
		Variables	Operators	Sequences	Repetitions	Conditionals	Functions	Parallelism	Events	System resettable	System not resettable	System observable	System not observable	State to be found	Algorithm to be found	One-to-one cardinality	Many-to-one cardinality	Explicit elements	Implicit elements	Unconstrained elements	Constrained elements	Algorithm manifest	Algorithm latent	Algorithm written	Algorithm not written
		✓	✓	✓	✓	□	✓	□	□	□	✓	□	✓	□	✓	✓	□	✓	□	✓	□	✓	□	✓	□
Data collection	✓	✓						+	+	+	+	+			✓	+	+	✓	+	✓	+	+	+	+	
Pattern recognition	✓	+		+	✓*	+	✓*		+	+	+	+				+	+	+	+	+	+	+	+	+	
Decomposition	✓	+	+	✓*	+	+	✓*	+	+	+		+				+	+	+	+	+	+	+	+	+	
Abstraction	✓	✓			+	+	+	+	+	+	+	+				+	+	+	+	+	+	+	+	+	
Data representation	✓	✓			+	+	+	+	+	+	+	+				+	+	+	+	+	+	+	+	+	
Variables	✓	✓	+	+	+	+	+	+	+		+	+		✓ ^F	+	+	+	+	+	+	+	+		+	+
Operators	✓	+	✓	+	+	+	+	+	+		+	+		✓ ^F	+	+	+	+	+	+	+	+		+	+
Sequences	✓	+	+	✓	+	+	+	+	+		+	+		✓ ^F		+	+	+	+	+	+	+		+	+
Repetitions	✓	+	+	+	✓	+	+	+	+		+	+		✓ ^F		+	+	+	+	+	+	+		+	+
Conditionals	✗	+	+	+		✓	+	+	+		+	+		✓ ^F	+	+	+	+	+	+	+	+		+	+
Functions	✓	+	+	+	+		✓	+	+		+	+		✓ ^F	+	+	+	+	+	+	+	+		+	+
Parallelism	✗	+	+	+	+	+		✓	+		+	+		✓ ^F	+	+	+	+	+	+	+	+		+	+
Events	✗	+	+	+	+	+		✓	+		+	+		✓ ^F	+	+	+	+	+	+	+	+		+	+
Algorithm debugging	✗	+	+	+	+	+	+	+	✓	✗	+	+		✓				+			✓ ^{SF}	✗	✓ ^F	✗ ^F	
System state verification	✗	+	+	+	+	+	+	+	✓	✗	+	+	✓					+			✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F	
Constraints validation	✗	+	+	+	+	+	+	+	✓	✗	+	+							✗	✓					
Optimisation	✗	+	+	+	+	+	+	+	✓	✗	+	+													
Generalisation	✗	✓		+	+	+	✓	+	✓	✗	+	+				+		+		+					

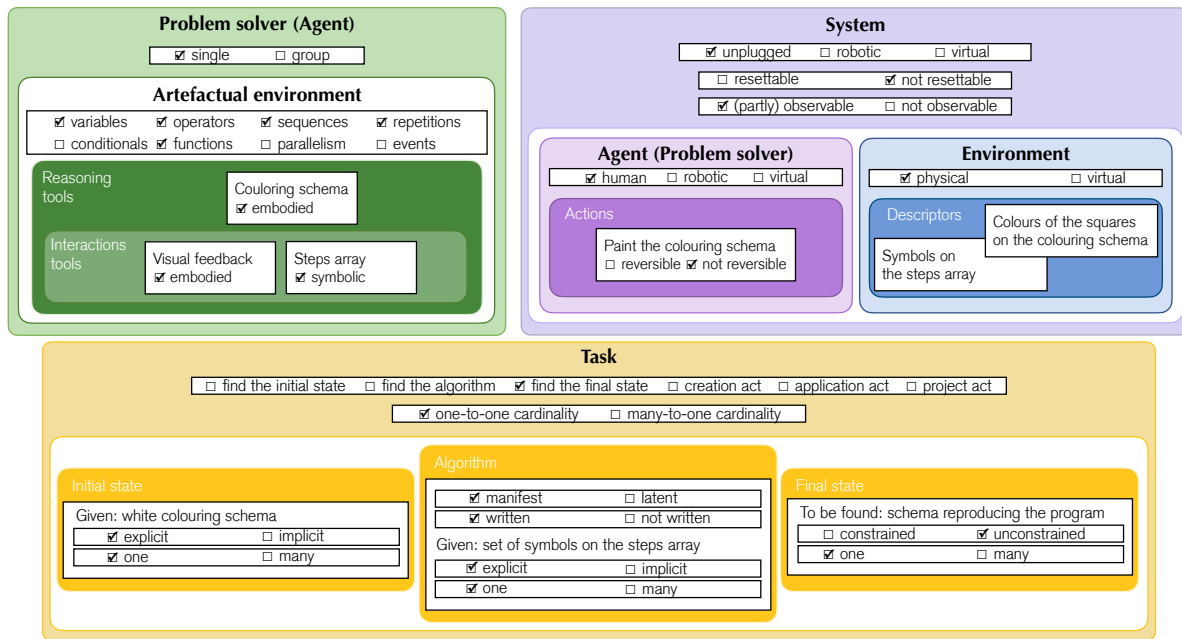


Fig. B.3 Graphical template for analysing the Graph Paper Programming (part 2).

Table B.3 Relationship between the Graph Paper Programming (part 2) characteristics and CT competencies.

		Tool functionalities								System				Task											
		Variables	Operators	Sequences	Repetitions	Conditionals	Functions	Parallelism	Events	System resettable	System not resettable	System observable	System not observable	State to be found	Algorithm to be found	One-to-one cardinality	Many-to-one cardinality	Explicit elements	Implicit elements	Unconstrained elements	Constrained elements	Algorithm manifest	Algorithm latent	Algorithm written	Algorithm not written
		✓	✓	✓	✓	□	✓	□	□	□	✓	□	□	✓	□	✓	□	✓	□	✓	□	✓	□	✓	□
Data collection	✓	✓						+	+	+	+	+	+			+	+		+		+	+	+	+	+
Pattern recognition	✓	+		+	✓*	+	✓*		+	+	+	+	+			+	+	+	+	+	+	+	+	+	+
Decomposition	✓	+	+	✓*	+	+	✓*	+	+	+		+	+		+	+	+	+	+	+	+	+	+	+	+
Abstraction	✓	✓			+	+	✓		+	+	+	+	+			+	+	+	+	+	+	+	+	+	+
Data representation	✓	✓		+	+	+	+			+		+	+			+		+		+	+	+	+	+	+
Variables	✓ ^{ES}	✓	+	+	+	+	+	+	+		+	+	+	✓ ^F	+	+	+	+	+	+	+	+	+	+	+
Operators	✓ ^{ES}	+	✓	+	+	+	+	+	+		+	+	+	✓ ^F	+		+	+	+	+	+	+	+	+	+
Sequences	✓ ^{ES}	+	+	✓	+	+	+		+		+	+	+	✓ ^F			+	+	+	+	+	+	+	+	+
Repetitions	✓ ^{ES}	+	+	+	✓	+	+		+		+	+	+	✓ ^F		+	+	+	+	+	+	+	+	+	+
Conditionals	✗	+	+	+		✓	+	+			+	+	+	✓ ^F	+	+	+	+	+	+	+	+	+	+	+
Functions	✓ ^{ES}	+	+	+	+		✓				+	+	+	✓ ^F	+	+	+	+	+	+	+	+	+	+	+
Parallelism	✗	+	+					✓			+	+	+	✓ ^F		+	+	+		+	+	+	+	+	+
Events	✗	+	+		+			✓			+			✓ ^F			+		+	+		+	+	+	+
Algorithm debugging	✗	+	+	+	+	+	+	+	✓	✗	+		✓				+				✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F	
System state verification	✗								✓	✗	+		✓				+				✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F	
Constraints validation	✗	+	+	+	+	+	+	+	✓	✗	+								✗	✓					
Optimisation	✗	+	+	+	+	+	+	+	✓	✗	+														
Generalisation	✗	✓		+	+	+	✓	+	✓	✗	+				+		+		+	+					

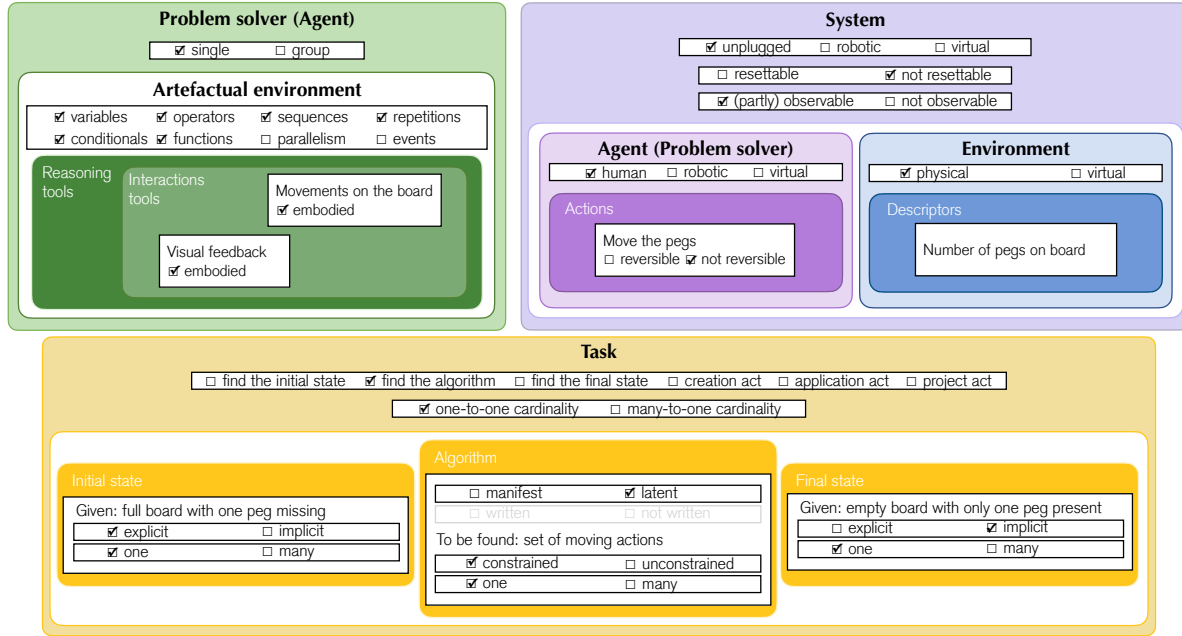


Fig. B.4 Graphical template for analysing the Triangular Peg Solitaire (board variant).

Table B.4 Relationship between the Triangular Peg Solitaire (board variant) characteristics and CT competencies.

		Tool functionalities								System				Task											
		Variables Operators Sequences Repetitions Conditionals Functions Parallelism Events								System resettable	System not resettable	System observable	System not observable	State to be found	Algorithm to be found	One-to-one cardinality	Many-to-one cardinality	Explicit elements	Implicit elements	Unconstrained elements	Constrained elements	Algorithm manifest	Algorithm latent	Algorithm written	Algorithm not written
										☐	☒	☐	☐	☐	☐	☐	☐	☐	☒	☒	☐	☐	☒	☐	☐
Data collection	☒	☒						☐	☐	☐	☒	☒	☐	☐	☒	☒	☒	☐	☒	☐	☒	☐	☒	☐	☐
Pattern recognition	☒	☐		☐	☒	☐	☒			☐	☐	☐	☐			☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Decomposition	☒	☐	☐	☒	☐	☐	☒	☐		☐			☐			☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Abstraction	☒	☒			☐	☐	☒			☐	☐	☐	☐			☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Data representation	☒	☒		☐	☐	☐	☐			☐	☐	☐	☐			☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Variables	☒	☒	☐	☐	☐	☐	☐	☐	☐			☐	☐	☒	☒	☐	☐	☐	☐	☐	☐		☐	☐	☐
Operators	☒	☐	☒	☐	☐	☐	☐	☐	☐			☐	☐	☒	☒	☐	☐	☐	☐	☐	☐		☐	☐	☐
Sequences	☒	☐	☐	☒	☐	☐	☐	☐	☐			☐	☐	☒	☒		☐	☐	☐	☐	☐	☐	☐	☐	☐
Repetitions	☒	☐	☐	☐	☒	☐	☐	☐	☐			☐	☐	☒	☒	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Conditionals	☒	☐	☐	☐	☐	☒	☐	☐	☐			☐	☐	☒	☒	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Functions	☒	☐	☐	☐	☐	☐	☒	☐	☐			☐	☐	☒	☒	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Parallelism	☒	☐	☐				☒	☐	☐			☐	☐	☒	☒	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Events	☒	☐	☐		☐			☒	☐			☐	☐	☒	☒	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Algorithm debugging	☒	☐	☐	☐	☐	☐	☐	☐	☒	☒	☐	☐		☒	☒			☐			☒	☒	☒	☒	☒
System state verification	☒	☐							☒	☒	☐	☐	☒					☐			☒	☒	☒	☒	☒
Constraints validation	☒	☐	☐	☐	☐	☐	☐	☐	☒	☒	☐	☐	☐						☒		☒		☒	☒	☒
Optimisation	☒	☐	☐	☐	☐	☐	☐	☐	☒	☒	☐	☐	☐						☒	☒			☒	☒	☒
Generalisation	☒	☒		☐	☐	☐	☒	☐	☒	☒	☐	☐	☐			☐		☐		☐					

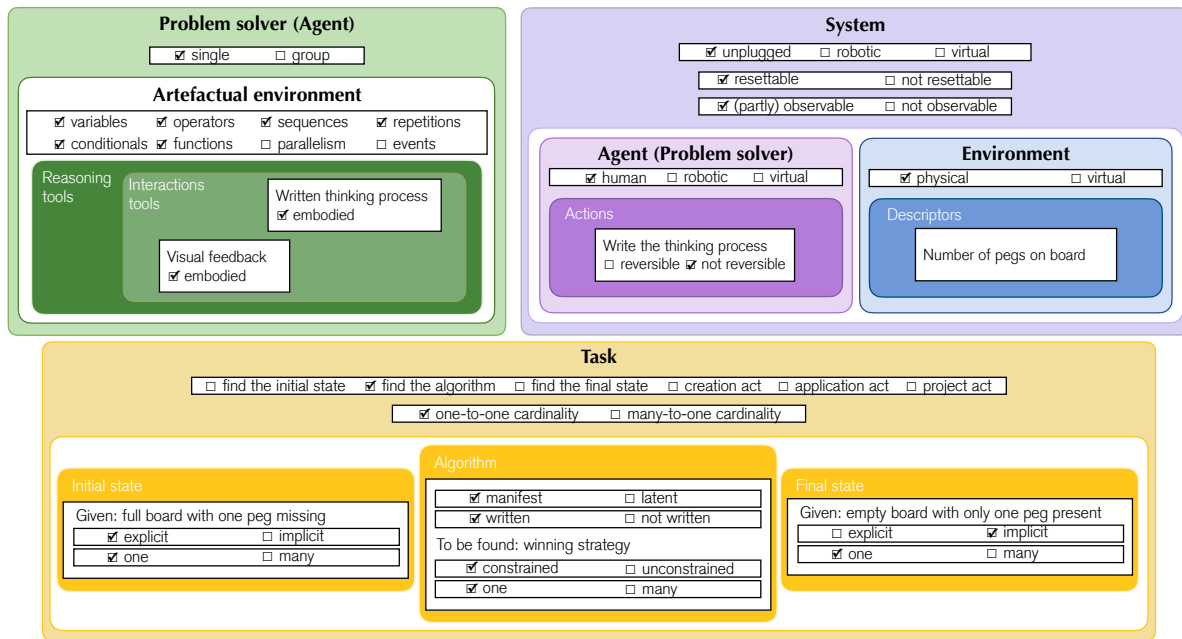


Fig. B.5 Graphical template for analysing the Triangular Peg Solitaire (paper & pencil variant).

Table B.5 Relationship between the Triangular Peg Solitaire (paper & pencil variant) characteristics and CT competencies.

		Tool functionalities								System				Task											
		Variables	Operators	Sequences	Repetitions	Conditionals	Functions	Parallelism	Events	System resettable	System not resettable	System observable	System not observable	State to be found	Algorithm to be found	One-to-one cardinality	Many-to-one cardinality	Explicit elements	Implicit elements	Unconstrained elements	Constrained elements	Algorithm manifest	Algorithm latent	Algorithm written	Algorithm not written
		✓	✓	✓	✓	✓	✓	□	□	✓	□	✓	□	□	✓	✓	✓	□	□	✓	✓	✓	□	✓	□
Data collection	✓	✓							+	+	+	+	+			+	+		+		+	+	+	+	+
Pattern recognition	✓	+		+	✓*	+	✓*			+	+	+	+			+	+	+	+	+	+	+	+	+	+
Decomposition	✓	+	+	✓*	+	+	✓*	+		+			+			+	+	+	+	+	+	+	+	+	+
Abstraction	✓	✓			+	+	+	✓					+				+	+	+	+	+	+	+	+	+
Data representation	✓	✓		+	+	+	+	✓					+			+	+	+	+	+	+	+	+	+	+
Variables	✓	✓	+	+	+	+	+	+	+			+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+
Operators	✓	+	✓	+	+	+	+	+	+			+	+		✓ ^F	+		+	+	+	+	+	+	+	+
Sequences	✓	+	+	✓	+		+					+	+		✓ ^F			+	+	+	+	+	+	+	+
Repetitions	✓	+	+	+	✓		+					+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+
Conditionals	✓	+	+			✓		+				+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+
Functions	✓	+	+	+	+		✓					+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+
Parallelism	✗	+	+					✓				+	+		✓ ^F		+	+	+	+	+	+	+	+	+
Events	✗	+	+			+			✓			+	+		✓ ^F			+		+		+	+	+	+
Algorithm debugging	✓	+	+	+	+	+	+	+	✓	✗	+			✓				+			✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F	
System state verification	✗								✓	✗	+		✓					+			✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F	
Constraints validation	✓	+	+	+	+	+	+	+	✓	✗	+								+	✗	✓				
Optimisation	✓	+	+	+	+	+	+	+	✓	✗	+														
Generalisation	✓	✓		+	+	+	✓	+	✓	✗	+					+		+		+					

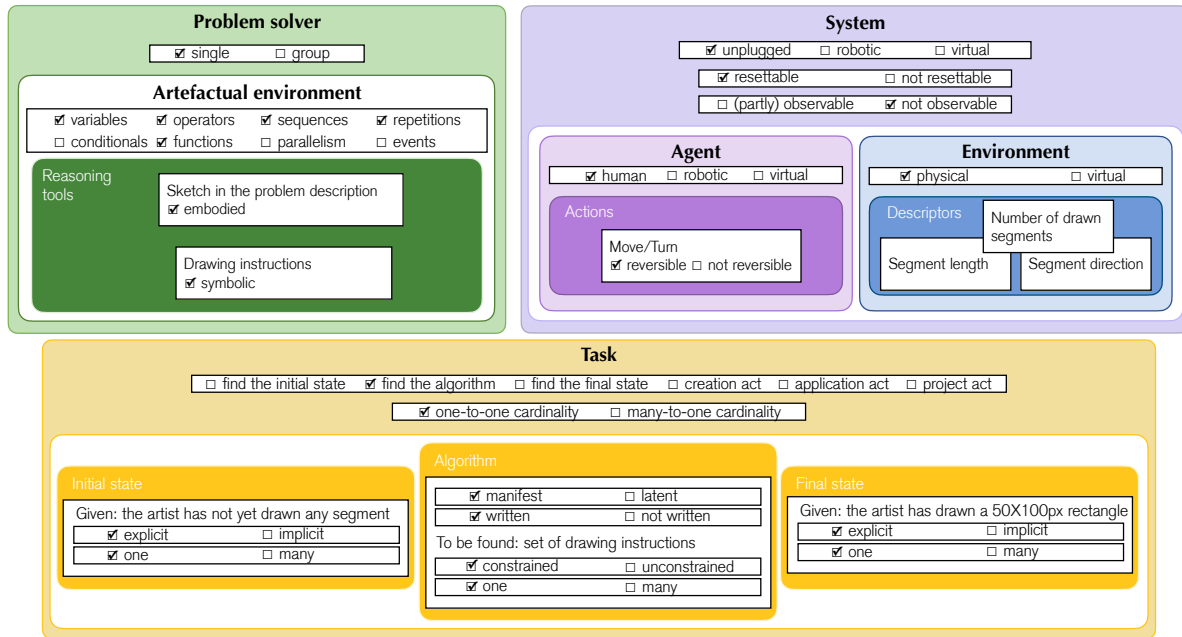


Fig. B.6 Graphical template for analysing the Computational Thinking test (item 7).

Table B.6 Relationship between the Computational Thinking test (item 7) characteristics and CT competencies.

		Tool functionalities								System				Task												
		Variables	Operators	Sequences	Repetitions	Conditionals	Functions	Parallelism	Events	System resettable	System not resettable	System observable	System not observable	State to be found	Algorithm to be found	One-to-one cardinality	Many-to-one cardinality	Explicit elements	Implicit elements	Unconstrained elements	Constrained elements	Algorithm manifest	Algorithm latent	Algorithm written	Algorithm not written	
		✓	✓	✓	✓	□	✓	□	□	✓	□	□	✓	□	✓	✓	□	✓	□	□	✓	□	✓	□	✓	□
Data collection	✓	✓						+	+	+	+	+				+	+		+		+	+	+	+	+	
Pattern recognition	✓	+		+	✓*	+	✓*		+	+	+	+				+	+	+	+	+	+	+	+	+	+	
Decomposition	✓	+	+	✓*	+	+	✓*	+	+			+				+	+	+	+	+	+	+	+	+	+	
Abstraction	✓	✓			+	+	✓		+	+		+				+	+	+	+	+	+	+	+	+	+	
Data representation	✓	✓			+	+	+	+				+				+	+	+	+		+	+	+	+	+	
Variables	✓	✓	+	+	+	+	+	+	+		+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+	+	
Operators	✓	+	✓	+	+	+	+	+	+		+	+		✓ ^F	+		+	+	+	+	+	+	+	+	+	
Sequences	✓	+	+	✓	+		+				+	+		✓ ^F			+	+	+	+	+	+	+	+	+	
Repetitions	✓	+	+	+	✓		+				+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+	+	
Conditionals	✗	+	+			✓		+			+	+		✓ ^F		+	+	+	+	+	+	+	+	+	+	
Functions	✓	+	+	+	+		✓				+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+	+	
Parallelism	✗	+	+					✓			+	+		✓ ^F		+	+	+	+	+	+	+	+	+	+	
Events	✗	+	+		+			✓			+	+		✓ ^F			+		+	+		+	+	+	+	
Algorithm debugging	✓	+	+	+	+	+	+	+	✓	✗	+		✓					+			✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F		
System state verification	✗								✓	✗	+		✓					+			✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F		
Constraints validation	✓	+	+	+	+	+	+	+	✓	✗	+								✗	✓						
Optimisation	✓	+	+	+	+	+	+	+	✓	✗	+															
Generalisation	✓	✓		+	+	+	✓	+	✓	✗	+					+		+		+						

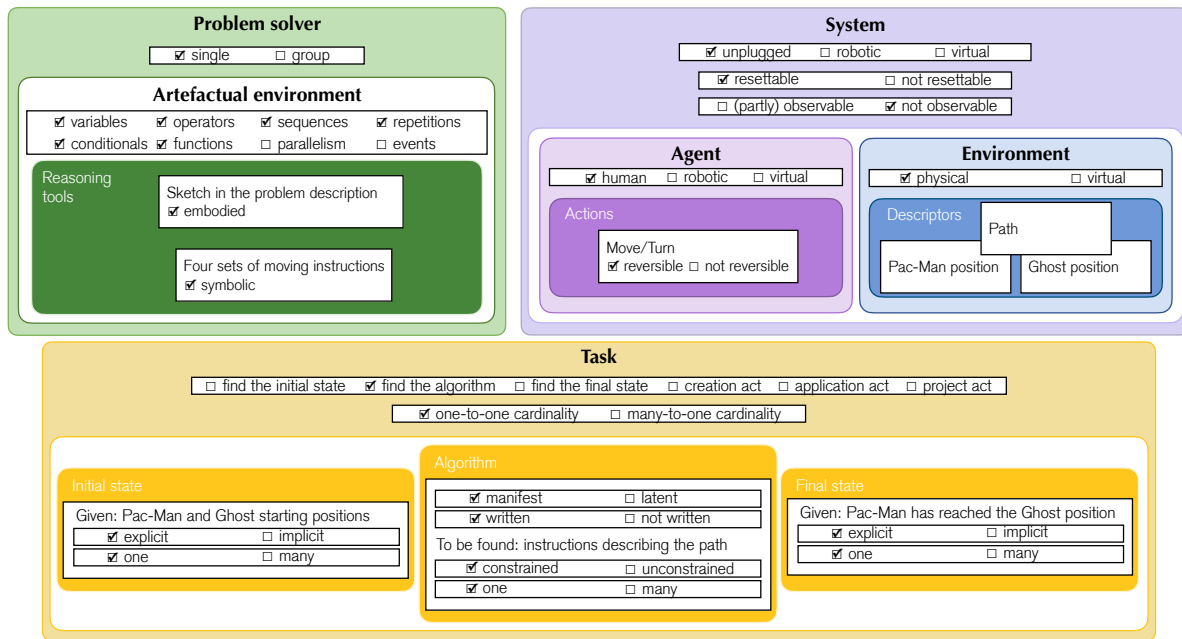


Fig. B.7 Graphical template for analysing the Computational Thinking test (item 14).

Table B.7 Relationship between the Computational Thinking test (item 14) characteristics and CT competencies.

		Tool functionalities								System				Task												
		Variables	Operators	Sequences	Repetitions	Conditionals	Functions	Parallelism	Events	System resettable	System not resettable	System observable	System not observable	State to be found	Algorithm to be found	One-to-one cardinality	Many-to-one cardinality	Explicit elements	Implicit elements	Unconstrained elements	Constrained elements	Algorithm manifest	Algorithm latent	Algorithm written	Algorithm not written	
		✓	✓	✓	✓	✓	✓	□	□	✓	□	□	✓	□	✓	✓	□	✓	□	□	✓	□	✓	□	✓	□
Data collection	✓	✓							+	+	+	+				+	+		+		+	+	+	+	+	
Pattern recognition	✓	+		+	✓*	+	✓*			+	+	+				+	+	+	+	+	+	+	+	+	+	
Decomposition	✓	+	+	✓*	+	+	✓*	+		+	+					+	+	+	+	+	+	+	+	+	+	
Abstraction	✓	✓			+	+	+	✓				+				+	+	+	+	+	+	+	+	+	+	
Data representation	✓	✓			+	+	+	+			+	+				+	+	+	+	+	+	+	+	+	+	
Variables	✓	✓	+	+	+	+	+	+	+			+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+	
Operators	✓	+	✓	+	+	+	+	+	+			+	+		✓ ^F	+		+	+	+	+	+	+	+	+	
Sequences	✓	+	+	✓	+		+					+	+		✓ ^F			+	+	+	+	+	+	+	+	
Repetitions	✓	+	+	+	✓		+					+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+	
Conditionals	✓	+	+			✓		+				+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+	
Functions	✓	+	+	+			✓					+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+	
Parallelism	✗	+	+					✓				+	+		✓ ^F	+	+	+		+	+	+	+	+	+	
Events	✗	+	+			+			✓			+			✓ ^F			+		+		+	+	+	+	
Algorithm debugging	✓	+	+	+	+	+	+	+	+	✓	✗	+			✓				+			✓	✗	✓ ^F	✗ ^F	
System state verification	✗									✓	✗	+		✓					+			✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F	
Constraints validation	✓	+	+	+	+	+	+	+	+	✓	✗	+							+		✗	✓				
Optimisation	✓	+	+	+	+	+	+	+	+	✓	✗	+														
Generalisation	✓	✓		+	+	+	✓		+	✓	✗	+					+		+		+					

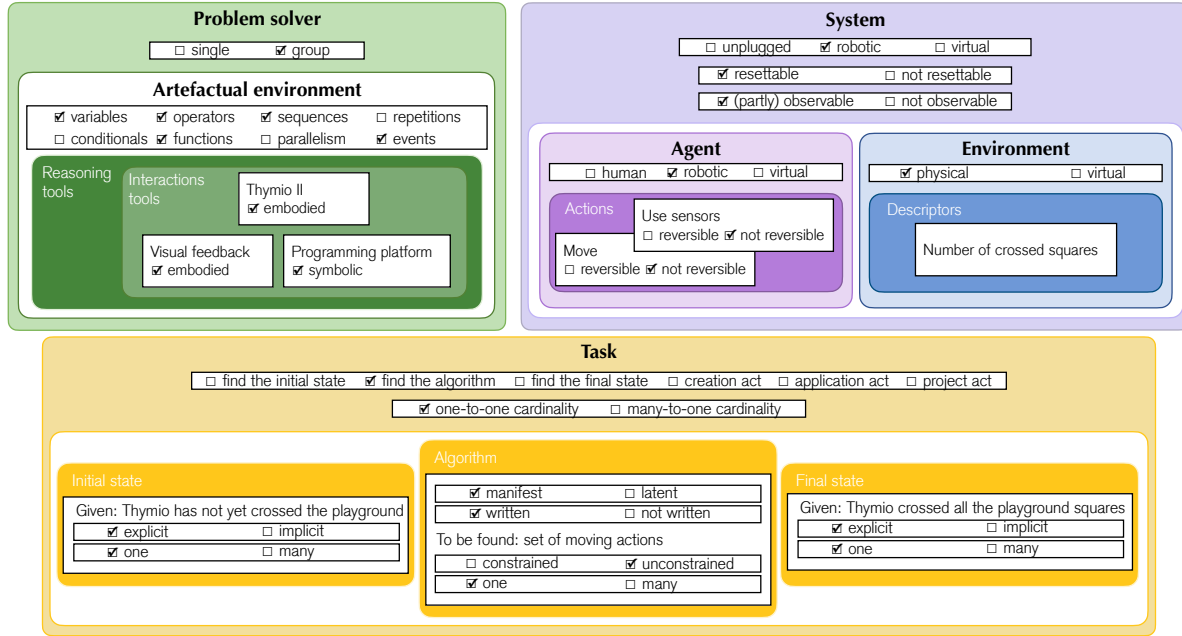


Fig. B.8 Graphical template for analysing the Thymio Lawnmower Mission (control group).

Table B.8 Relationship between the Thymio Lawnmower Mission (control group) characteristics and CT competencies.

		Tool functionalities								System				Task												
		Variables	Operators	Sequences	Repetitions	Conditionals	Functions	Parallelism	Events	System resettable	System not resettable	System observable	System not observable	State to be found	Algorithm to be found	One-to-one cardinality	Many-to-one cardinality	Explicit elements	Implicit elements	Unconstrained elements	Constrained elements	Algorithm manifest	Algorithm latent	Algorithm written	Algorithm not written	
		✓	✓	✓	□	□	✓	□	✓	✓	□	✓	□	□	✓	□	✓	□	✓	□	✓	□	✓	□	✓	□
Data collection	✓	✓						+	+	+	+	+	+			+	+		+	+	+	+	+	+	+	
Pattern recognition	✓	+		+	✓*	+	✓*		+	+	+	+	+			+	+	+	+	+	+	+	+	+	+	
Decomposition	✓	+	+	✓*	+	+		+	+			+	+			+	+	+	+	+	+	+	+	+	+	
Abstraction	✓	✓			+	+	✓		+	+	+	+	+			+	+	+	+	+	+	+	+	+	+	
Data representation	✓	✓		+	+	+	+	+		+		+	+			+		+	+	+	+	+	+	+	+	
Variables	✓	✓	+	+	+	+	+	+	+			+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+	
Operators	✓	+	✓	+	+	+	+	+	+			+	+		✓ ^F	+		+	+	+	+	+	+	+	+	
Sequences	✓	+	+	✓	+	+	+					+	+		✓ ^F			+	+	+	+	+	+	+	+	
Repetitions	✗	+	+	+	✓	+	+					+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+	
Conditionals	✗	+	+			✓		+				+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+	
Functions	✓	+	+	+	+		✓					+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+	
Parallelism	✗	+	+					✓				+	+		✓ ^F		+	+	+	+	+	+	+	+	+	
Events	✓	+	+		+			✓				+	+		✓ ^F			+		+	+	+	+	+	+	
Algorithm debugging	✓	+	+	+	+	+	+	+	✓	✗	+			✓				+			✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F	✗ ^F	
System state verification	✗								✓	✗	+		✓					+			✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F	✗ ^F	
Constraints validation	✗	+	+	+	+	+		+	✓	✗	+								+	✗	✓					
Optimisation	✓	+	+	+	+	+	+	+	✓	✗	+															
Generalisation	✓	✓		+	+	+	✓	+	✓	✗	+					+		+		+						

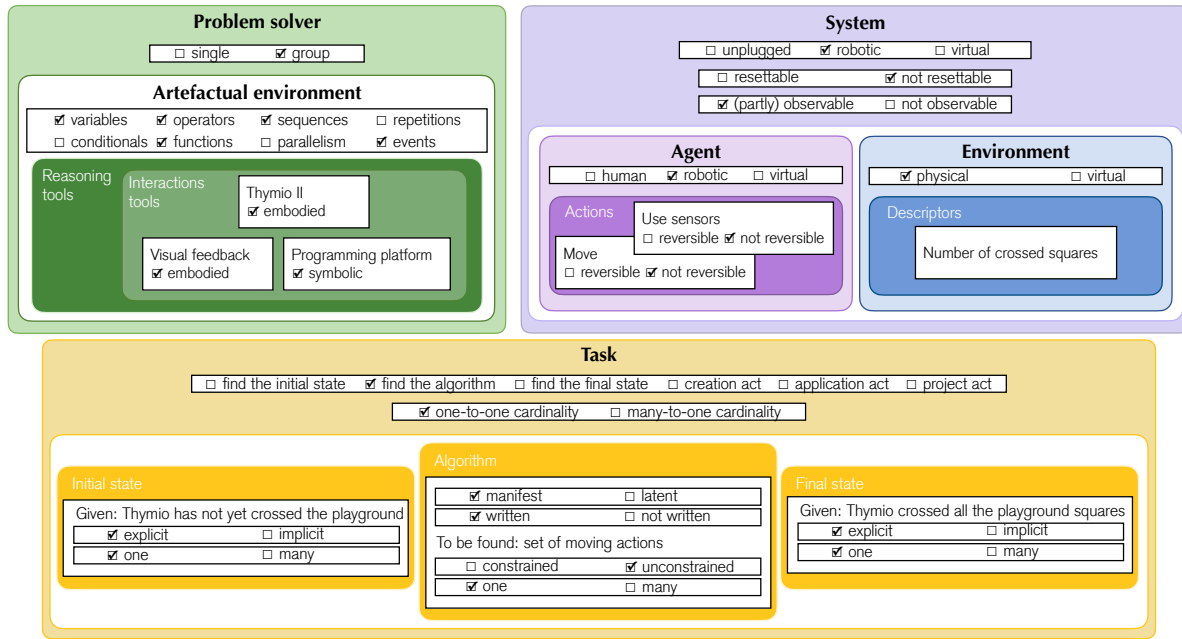


Fig. B.9 Graphical template for analysing the Thymio Lawnmower Mission (test group).

Table B.9 Relationship between the Thymio Lawnmower Mission (test group) characteristics and CT competencies.

		Tool functionalities								System				Task											
		Variables	Operators	Sequences	Repetitions	Conditionals	Functions	Parallelism	Events	System resettable	System not resettable	System observable	System not observable	State to be found	Algorithm to be found	One-to-one cardinality	Many-to-one cardinality	Explicit elements	Implicit elements	Unconstrained elements	Constrained elements	Algorithm manifest	Algorithm latent	Algorithm written	Algorithm not written
		✓	✓	✓	□	□	✓	□	✓	□	✓	□	□	□	✓	✓	□	✓	□	✓	□	✓	□	✓	□
Data collection	✓	✓						+	+	+	+	+				+	+		+	+	+	+	+	+	
Pattern recognition	✓	+		+	✓*	+	✓*		+	+	+	+				+	+	+	+	+	+	+	+	+	
Decomposition	✓	+	+	✓*	+	+	✓*	+	+	+		+				+	+	+	+	+	+	+	+	+	
Abstraction	✓	✓			+	+	✓			+	+	+				+	+	+	+	+	+	+	+	+	
Data representation	✓	✓		+	+	+	+	✓			+	+				+	+	+	+	+	+	+	+	+	
Variables	✓	✓	+	+	+	+	+	+	+		+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+	
Operators	✓	+	✓	+	+	+	+	+	+		+	+		✓ ^F	+		+	+	+	+	+	+	+	+	
Sequences	✓	+	+	✓	+	+	+				+	+		✓ ^F			+	+	+	+	+	+	+	+	
Repetitions	✗	+	+	+	✓	+	+				+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+	
Conditionals	✗	+	+			✓		+			+	+		✓ ^F		+	+	+	+	+	+	+	+	+	
Functions	✓	+	+	+	+		✓				+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+	
Parallelism	✗	+	+					✓			+	+		✓ ^F		+	+	+	+	+	+	+	+	+	
Events	✓	+	+		+			✓			+	+		✓ ^F			+		+	+	+	+	+	+	
Algorithm debugging	✗	+	+	+	+	+	+	+	✓	✗	+			✓				+			✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F	
System state verification	✗								✓	✗	+		✓					+			✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F	
Constraints validation	✗	+	+	+	+	+		+	✓	✗	+								✗	✓					
Optimisation	✗	+	+	+	+	+	+	+	✓	✗	+														
Generalisation	✗	✓		+	+	+	✓	+	✓	✗	+					+		+		+					

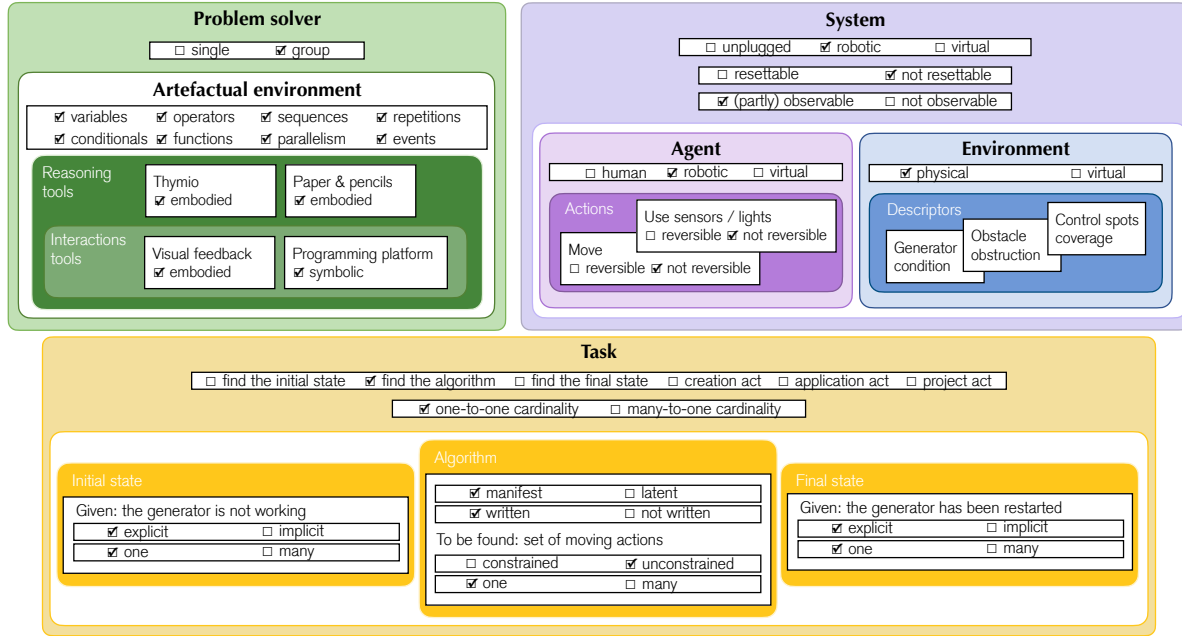


Fig. B.10 Graphical template for analysing the R2T2 mission.

Table B.10 Relationship between the R2T2 mission characteristics and CT competencies.

		Tool functionalities								System				Task											
		Variables	Operators	Sequences	Repetitions	Conditionals	Functions	Parallelism	Events	System resettable	System not resettable	System observable	System not observable	State to be found	Algorithm to be found	One-to-one cardinality	Many-to-one cardinality	Explicit elements	Implicit elements	Unconstrained elements	Constrained elements	Algorithm manifest	Algorithm latent	Algorithm written	Algorithm not written
		✓	✓	✓	✓	✓	✓	✓	✓	□	✓	✓	□	□	✓	✓	□	✓	□	✓	□	✓	□	✓	□
Data collection	✓	✓							+	+	+	+	+		✓		+	+	✓	□	+	+	+	+	+
Pattern recognition	✓	+		+	✓*	+	✓*			+	+	+	+				+	+	+	+	+	+	+	+	+
Decomposition	✓	+	+	✓*		+	✓*	+		+			+			+	+	+	+	+	+	+	+	+	+
Abstraction	✓	✓			+	+	✓*			+	+		+			+	+	+	+	+	+	+	+	+	+
Data representation	✓	✓		+	+	+	+				+		+				+		+		+	+	+	+	+
Variables	✓	✓	+	+	+	+	+	+	+			+	+		✓ ^F	+	+	+	+	+	+	+		+	+
Operators	✓	+	✓	+	+	+	+	+	+			+	+		✓ ^F	+		+	+	+	+	+		+	+
Sequences	✓	+	+	✓	+		+					+	+		✓ ^F			+	+	+	+	+	+	+	+
Repetitions	✓	+	+	+	✓		+					+	+		✓ ^F			+	+	+	+	+	+	+	+
Conditionals	✓	+	+			✓						+	+		✓ ^F	+		+	+	+	+	+	+	+	+
Functions	✓	+	+	+	+		✓					+	+		✓ ^F		+	+	+	+	+	+	+	+	+
Parallelism	✓	+	+					✓				+	+		✓ ^F		+	+	+	+	+	+	+	+	+
Events	✓	+	+			+			✓			+	+		✓ ^F			+		+		+		+	+
Algorithm debugging	✗	+	+	+	+	+	+	+	+	✓	✗	+			✓			+			✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F	
System state verification	✗									✓	✗	+		✓				+			✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F	
Constraints validation	✗	+	+	+	+	+	+	+	+	✓	✗	+							✗	✓					
Optimisation	✗	+	+	+	+	+	+	+	+	✓	✗	+													
Generalisation	✗	✓		+	+	+	✓	+		✓	✗	+					+	+		+					

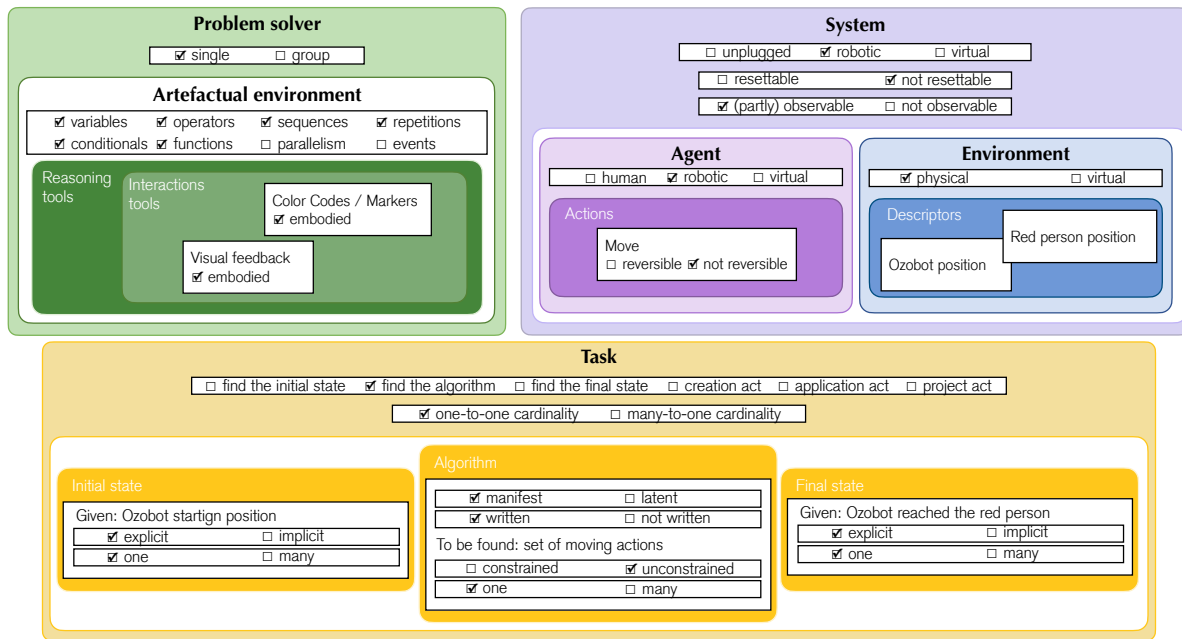


Fig. B.11 Graphical template for analysing the Ozobot maze.

Table B.11 Relationship between the Ozobot maze characteristics and CT competencies.

		Tool functionalities								System				Task											
		Variables	Operators	Sequences	Repetitions	Conditionals	Functions	Parallelism	Events	System resettable	System not resettable	System observable	System not observable	State to be found	Algorithm to be found	One-to-one cardinality	Many-to-one cardinality	Explicit elements	Implicit elements	Unconstrained elements	Constrained elements	Algorithm manifest	Algorithm latent	Algorithm written	Algorithm not written
		✓	✓	✓	✓	✓	✓	□	□	□	✓	✓	□	□	✓	✓	□	✓	□	✓	□	✓	□	✓	□
Data collection	✓	✓						+		+	+	+	+		✓		+	+		+	+	+	+	+	
Pattern recognition	✓	+		+	✓*	+	✓*			+	+	+	+			+	+	+	+	+	+	+	+	+	
Decomposition	✓	+	+	✓*	+	+	✓*	+		+			+			+	+	+	+	+	+	+	+	+	
Abstraction	✓	✓			+	+	+	✓			+	+	+				+	+	+	+	+	+	+	+	
Data representation	✓	✓			+	+	+	+			+	+	+				+	+	+	+	+	+	+	+	
Variables	✓	✓	+	+	+	+	+	+	+			+	+		✓ ^F	+	+	+	+	+	+	+	+	+	
Operators	✓	+	✓	+	+	+	+	+	+			+	+		✓ ^F	+		+	+	+	+	+	+	+	
Sequences	✓	+	+	✓	+		+					+	+		✓ ^F			+	+	+	+	+	+	+	
Repetitions	✓	+	+	+	✓		+					+	+		✓ ^F		+	+	+	+	+	+	+	+	
Conditionals	✓	+	+			✓		+				+	+		✓ ^F	+		+	+	+	+	+	+	+	
Functions	✓	+	+	+	+		✓					+	+		✓ ^F	+	+	+	+	+	+	+	+	+	
Parallelism	✗	+	+					✓				+	+		✓ ^F		+	+		+	+	+	+	+	
Events	✗	+	+			+			✓			+	+		✓ ^F			+		+		+	+	+	
Algorithm debugging	✗	+	+	+	+	+	+	+	✓	✗	+			✓				+			✓	✗	✓ ^F	✗ ^F	
System state verification	✗								✓	✗	+		✓					+		✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F		
Constraints validation	✗	+	+	+	+	+	+	+	✓	✗	+								+	✗	✓				
Optimisation	✗	+	+	+	+	+	+	+	✓	✗	+														
Generalisation	✗	✓		+	+	+	✓	+	✓	✗	+					+		+		+					

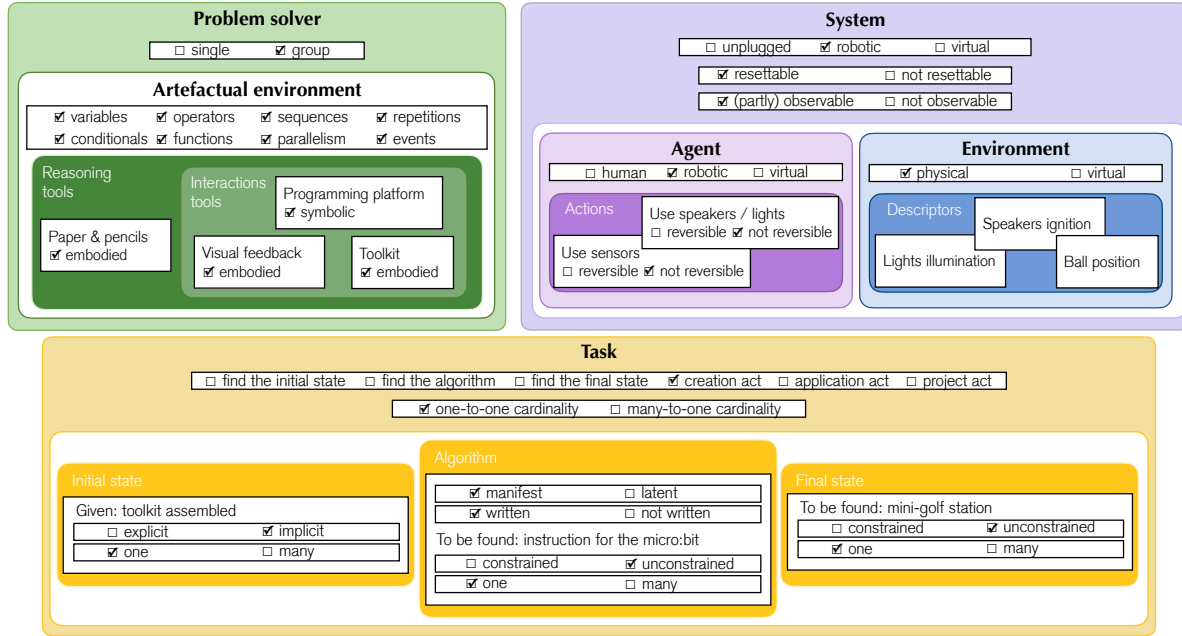


Fig. B.12 Graphical template for analysing the Mini-golf challenge.

Table B.12 Relationship between the Mini-golf challenge characteristics and CT competencies.

		Tool functionalities								System				Task											
		Variables	Operators	Sequences	Repetitions	Conditionals	Functions	Parallelism	Events	System resettable	System not resettable	System observable	System not observable	State to be found	Algorithm to be found	One-to-one cardinality	Many-to-one cardinality	Explicit elements	Implicit elements	Unconstrained elements	Constrained elements	Algorithm manifest	Algorithm latent	Algorithm written	Algorithm not written
		✓	✓	✓	✓	✓	✓	✓	✓	✓	□	✓	□	✓	✓	✓	□	□	✓	✓	□	✓	□	✓	□
Data collection	✓	✓						+	+	+	+	+	+	✓	✓	+	+		+	✓	+	+	+	+	
Pattern recognition	✓	+		+	✓*	+	✓*		+	+	+	+	+			+	+	+	+	+	+	+	+	+	
Decomposition	✓	+	+	✓*	+	+	✓*	+	+	+		+	+			+	+	+	+	+	+	+	+	+	
Abstraction	✓	✓			+	+	✓			+						+	+	+	+	+	+	+	+	+	
Data representation	✓	✓		+	+	+	+	+		+		+	+			+		+	+	+	+	+	+	+	
Variables	✓	✓	+	+	+	+	+	+	+			+	+	✓ ^F	✓ ^F	+	+	+	+	+	+	+		+	+
Operators	✓	+	✓	+	+	+	+	+	+			+	+	✓ ^F	✓ ^F	+		+	+	+	+	+		+	+
Sequences	✓	+	+	✓	+	+	+					+	+	✓ ^F	✓ ^F			+	+	+	+	+		+	+
Repetitions	✓	+	+	+	✓	+	+					+	+	✓ ^F	✓ ^F		+	+	+	+	+	+		+	+
Conditionals	✓	+	+		✓	+	+					+	+	✓ ^F	✓ ^F	+	+	+	+	+	+	+		+	+
Functions	✓	+	+	+	+	✓	+					+	+	✓ ^F	✓ ^F		+	+	+	+	+	+		+	+
Parallelism	✓	+	+				✓					+	+	✓ ^F	✓ ^F		+	+		+	+	+		+	+
Events	✓	+	+		+			✓				+	+	✓ ^F	✓ ^F			+		+	+	+		+	+
Algorithm debugging	✓	+	+	+	+	+	+	+	✓	✗	+			✓	✓			+			✓ ^{SF}	✗	✓ ^F	✗ ^F	
System state verification	✓								✓	✗	+		✓				+			✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F		
Constraints validation	✗	+	+	+	+	+	+	+	✓	✗	+							✗	✓						
Optimisation	✓	+	+	+	+	+	+	+	✓	✗	+									+	✓				
Generalisation	✓	✓		+	+	+	✓	+	✓	✗	+					+		+			+				

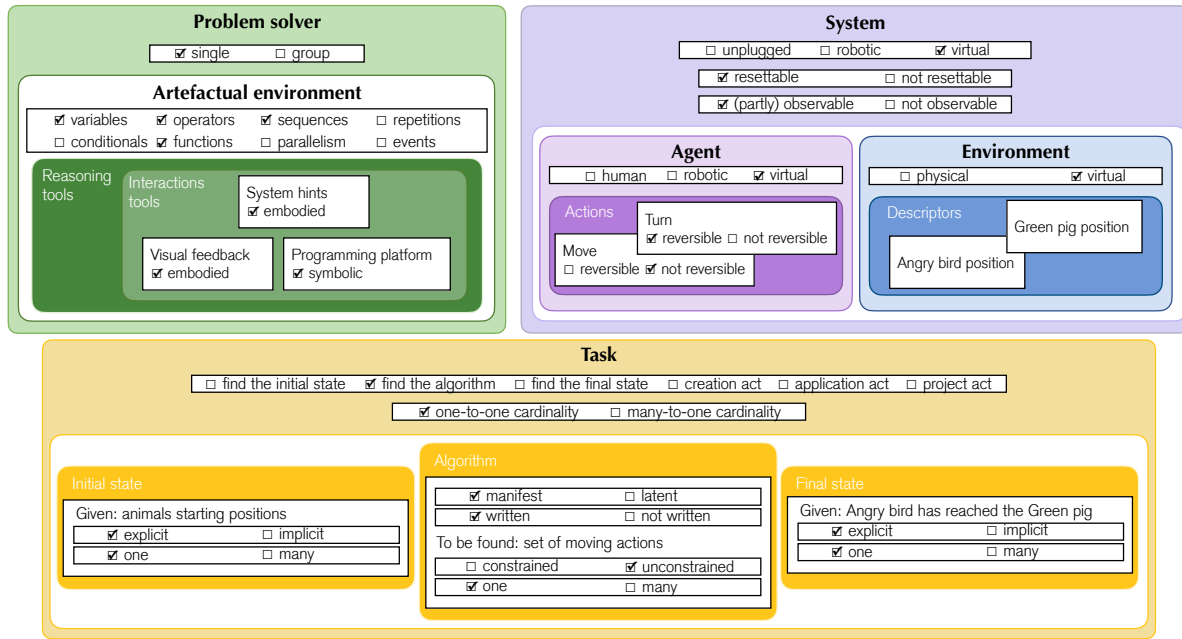


Fig. B.13 Graphical template for analysing the Angry Bird Classic Maze.

Table B.13 Relationship between the Angry Bird Classic Maze characteristics and CT competencies.

	Tool functionalities								System				Task											
	Variables	Operators	Sequences	Repetitions	Conditionals	Functions	Parallelism	Events	System resettable	System not resettable	System observable	System not observable	State to be found	Algorithm to be found	One-to-one cardinality	Many-to-one cardinality	Explicit elements	Implicit elements	Unconstrained elements	Constrained elements	Algorithm manifest	Algorithm latent	Algorithm written	Algorithm not written
	☒	☒	☒	☐	☐	☒	☐	☐	☒	☐	☒	☐	☐	☒	☒	☒	☒	☐	☒	☐	☒	☐	☒	☐
Data collection	✓	✓						+	+	+	+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+
Pattern recognition	✓	+	+	✓*	+	+	+	+	+	+	+	+		+	+	+	+	+	+	+	+	+	+	+
Decomposition	✓	+	+	✓*	+	+	+	+	+	+	+	+		+	+	+	+	+	+	+	+	+	+	+
Abstraction	✓	✓	+	+	+	+	+	+	+	+	+	+		+	+	+	+	+	+	+	+	+	+	+
Data representation	✓	✓	+	+	+	+	+	+	+	+	+	+		+	+	+	+	+	+	+	+	+	+	+
Variables	✓	✓	+	+	+	+	+	+			+	+		✓ ^F	+	+	+	+	+	+	+		+	+
Operators	✓	+	✓	+	+	+	+	+			+	+		✓ ^F	+		+	+	+	+	+		+	+
Sequences	✓	+	+	✓	+	+	+	+			+	+		✓ ^F			+	+	+	+	+		+	+
Repetitions	✗	+	+	+	✓	+		+			+	+		✓ ^F		+	+	+	+	+	+		+	+
Conditionals	✗	+	+		✓			+			+	+		✓ ^F	+		+	+	+	+	+		+	+
Functions	✓	+	+	+		✓					+	+		✓ ^F		+	+	+	+	+	+		+	+
Parallelism	✗	+	+				✓				+	+		✓ ^F		+	+		+	+	+		+	+
Events	✗	+	+		+			✓			+	+		✓ ^F			+		+		+		+	+
Algorithm debugging	✓	+	+	+	+	+	+	+	✓	✗	+			✓				+			✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F
System state verification	✗								✓	✗	+		✓					+			✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F
Constraints validation	✗	+	+	+	+	+	+	+	✓	✗	+								✗	✓				
Optimisation	✓	+	+	+	+	+	+	+	✓	✗	+													
Generalisation	✓	✓	+	+	+	✓	+	+	✓	✗	+				+		+		+					

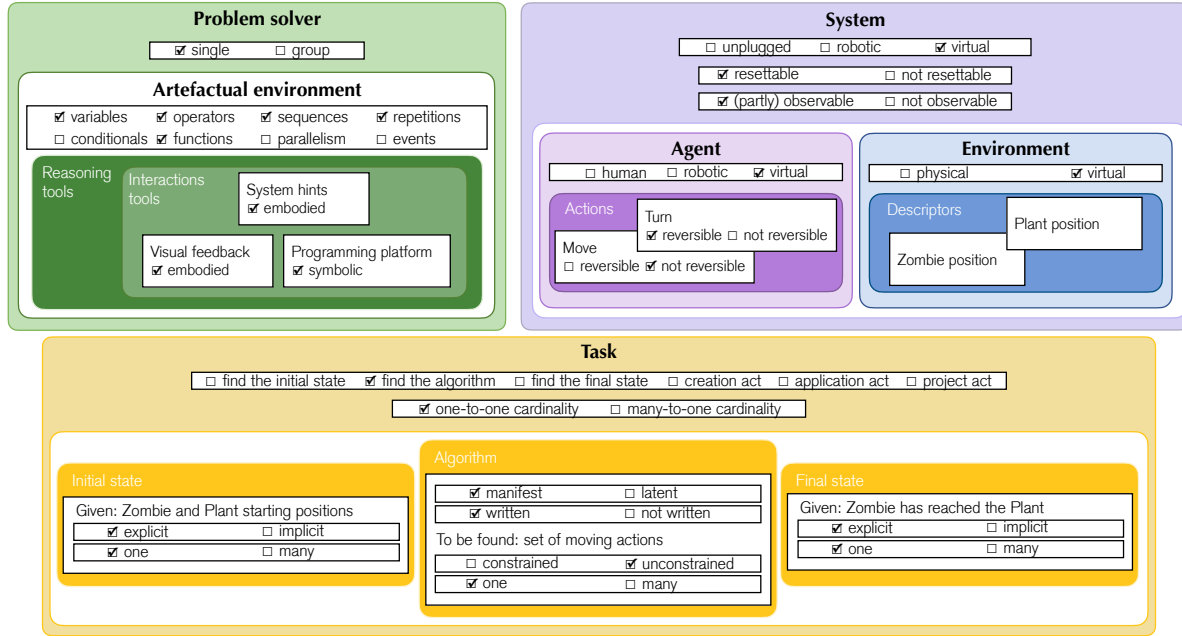


Fig. B.14 Graphical template for analysing the Plant vs Zombie Classic maze.

Table B.14 Relationship between the Plant vs Zombie Classic Maze characteristics and CT competencies.

		Tool functionalities								System				Task												
		Variables	Operators	Sequences	Repetitions	Conditionals	Functions	Parallelism	Events	System resettable	System not resettable	System observable	System not observable	State to be found	Algorithm to be found	One-to-one cardinality	Many-to-one cardinality	Explicit elements	Implicit elements	Unconstrained elements	Constrained elements	Algorithm manifest	Algorithm latent	Algorithm written	Algorithm not written	
		✓	✓	✓	✓	□	✓	□	□	✓	□	✓	□	□	✓	✓	✓	□	✓	□	✓	□	✓	□	✓	□
Data collection	✓	✓						+		+	+	+	+				+	+		+	+	+	+	+	+	
Pattern recognition	✓	+		+	✓*	+	✓*			+	+	+	+				+	+	+	+	+	+	+	+	+	
Decomposition	✓	+	+	✓*	+	+	+	+		+			+				+	+	+	+	+	+	+	+	+	
Abstraction	✓	✓			+	+	+	✓		+	+	+	+				+	+	+	+	+	+	+	+	+	
Data representation	✓	✓			+	+	+	+		+			+				+	+	+	+	+	+	+	+	+	
Variables	✓	✓	+	+	+	+	+	+	+			+	+		✓ ^F	+	+	+	+	+	+	+	+	+	+	
Operators	✓	+	✓	+	+	+	+	+	+			+	+		✓ ^F	+		+	+	+	+	+	+	+	+	
Sequences	✓	+	+	✓	+		+					+	+		✓ ^F			+	+	+	+	+	+	+	+	
Repetitions	✓	+	+	+	✓		+					+	+		✓ ^F		+	+	+	+	+	+	+	+	+	
Conditionals	✗	+	+	+		✓		+				+	+		✓ ^F	+		+	+	+	+	+	+	+	+	
Functions	✓	+	+	+	+		✓					+	+		✓ ^F		+	+	+	+	+	+	+	+	+	
Parallelism	✗	+	+					✓				+	+		✓ ^F		+	+		+	+	+	+	+	+	
Events	✗	+	+		+			✓				+	+		✓ ^F			+		+		+	+	+	+	
Algorithm debugging	✓	+	+	+	+	+	+	+	+	✓	✗	+			✓			+			✓	✗ ^{SF}	✓ ^F	✗ ^F	✗ ^F	
System state verification	✗									✓	✗	+		✓				+		✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F	✗ ^F		
Constraints validation	✗	+	+	+	+	+	+	+	+	✓	✗	+							✗	✓						
Optimisation	✓	+	+	+	+	+	+	+	+	✓	✗	+														
Generalisation	✓	✓		+	+	+	✓	+	+	✓	✗	+				+		+		+						

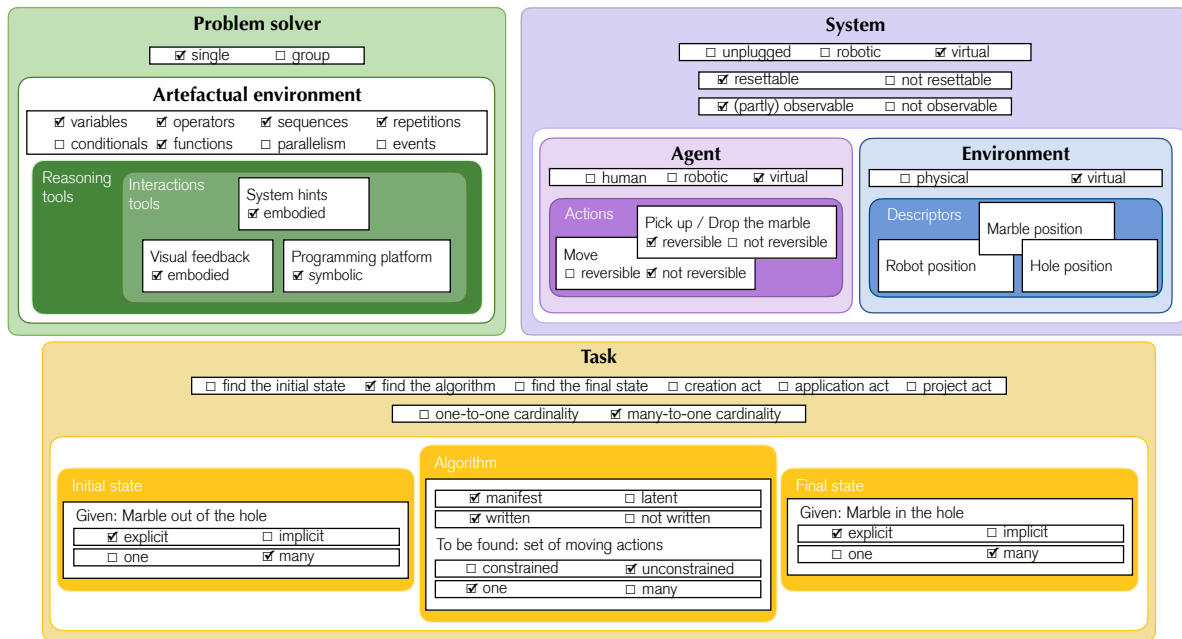


Fig. B.15 Graphical template for analysing the Store the Marbles activity.

Table B.15 Relationship between the Store the Marbles characteristics and CT competencies.

		Tool functionalities								System				Task											
		Variables	Operators	Sequences	Repetitions	Conditionals	Functions	Parallelism	Events	System resettable	System not resettable	System observable	System not observable	State to be found	Algorithm to be found	One-to-one cardinality	Many-to-one cardinality	Explicit elements	Implicit elements	Unconstrained elements	Constrained elements	Algorithm manifest	Algorithm latent	Algorithm written	Algorithm not written
		✓	✓	✓	✓	□	✓	□	□	✓	□	✓	□	□	✓	□	□	✓	□	✓	□	✓	□	✓	□
Data collection	✓	✓						+	+	+	+	+	+		✓ ^F	+	+	+	+	+	+	+	+	+	
Pattern recognition	✓	+		+	✓*	+	✓*		+	+	+	+	+		+	+	+	+	+	+	+	+	+	+	
Decomposition	✓	+	+	✓*	+	+	✓*	+		+		+	+			+	+	+	+	+	+	+	+	+	
Abstraction	✓	✓			+	+	+	✓				+	+			+	+	+	+	+	+	+	+	+	
Data representation	✓	✓			+	+	+	+		+		+	+			+	+	+	+	+	+	+	+	+	
Variables	✓	✓	+	+	+	+	+	+	+		+	+	+		✓ ^F	+	+	+	+	+	+	+	+	+	
Operators	✓	+	✓	+	+	+	+	+	+		+	+	+		✓ ^F	+		+	+	+	+	+	+	+	
Sequences	✓	+	+	✓	+		+				+	+	+		✓ ^F			+	+	+	+	+	+	+	
Repetitions	✓	+	+	+	✓		+				+	+	+		✓ ^F		+	+	+	+	+	+	+	+	
Conditionals	✗	+	+			✓		+			+	+	+		✓ ^F	+		+	+	+	+	+	+	+	
Functions	✓	+	+	+	+		✓				+	+	+		✓ ^F		+	+	+	+	+	+	+	+	
Parallelism	✗	+	+					✓			+	+	+		✓ ^F		+	+		+	+	+	+	+	
Events	✗	+	+			+			✓		+	+	+		✓ ^F			+		+	+	+	+	+	
Algorithm debugging	✓	+	+	+	+	+	+	+	+	✓	✗	+		✓				+			✓	✗ ^{SF}	✓ ^F	✗ ^F	
System state verification	✗									✓	✗	+	✓					+		✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F		
Constraints validation	✗	+	+	+	+	+	+	+	+	✓	✗	+							✗	✓					
Optimisation	✓	+	+	+	+	+	+	+	+	✓	✗	+													
Generalisation	✓	✓		+	+	+	✓	+	✓	✗	+					+	+	+		+					

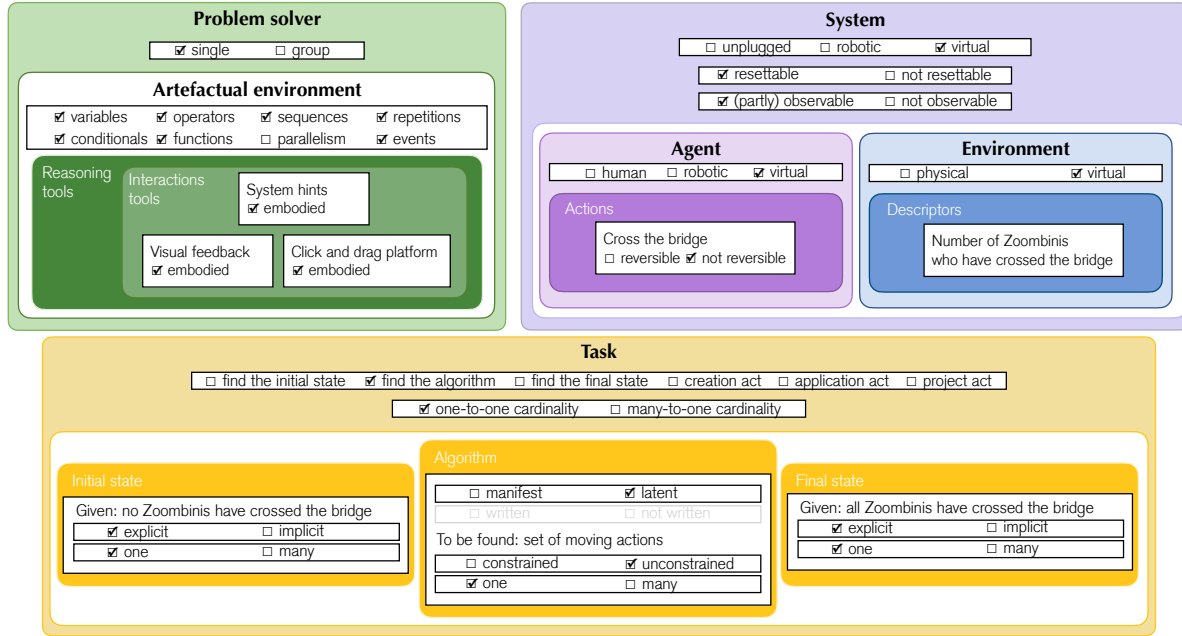


Fig. B.16 Graphical template for analysing the Zoombinis Allergic Cliffs Puzzle.

Table B.16 Relationship between the Zoombinis Allergic Cliffs Puzzle characteristics and CT competencies.

		Tool functionalities								System				Task											
		Variables	Operators	Sequences	Repetitions	Conditionals	Functions	Parallelism	Events	System resettable	System not resettable	System observable	System not observable	State to be found	Algorithm to be found	One-to-one cardinality	Many-to-one cardinality	Explicit elements	Implicit elements	Unconstrained elements	Constrained elements	Algorithm manifest	Algorithm latent	Algorithm written	Algorithm not written
		✓	✓	✓	✓	✓	✓	□	✓	✓	□	✓	□	□	✓	✓	□	✓	□	✓	□	□	✓	□	□
Data collection	✓	✓						+	+	+	+	+	+		✓	+	+	+	+	+	+	+	+	+	
Pattern recognition	✓	+		+	✓*	+	✓*		+	+	+	+	+			+	+	+	+	+	+	+	+	+	
Decomposition	✓	+	+	✓*	+	+	+	+	+	+		+	+			+	+	+	+	+	+	+	+	+	
Abstraction	✓	✓			+	+	+	✓*	✓			+				+	+	+	+	+	+	+	+	+	
Data representation	✓	✓		+	+	+	+	+				+				+	+	+	+	+	+	+	+	+	
Variables	✓	✓	+	+	+	+	+	+	+			+	+		✓ ^F	+	+	+	+	+	+	+		+	+
Operators	✓	+	✓	+	+	+	+	+	+			+	+		✓ ^F	+		+	+	+	+	+		+	+
Sequences	✓	+	+	✓	+		+					+	+		✓ ^F			+	+	+	+	+		+	+
Repetitions	✓	+	+	+	✓		+					+	+		✓ ^F		+	+	+	+	+	+		+	+
Conditionals	✓	+	+			✓		+				+	+		✓ ^F	+	+	+	+	+	+	+		+	+
Functions	✓	+	+	+	+		✓					+	+		✓ ^F	+	+	+	+	+	+	+		+	+
Parallelism	✗	+	+					✓				+	+		✓ ^F		+	+		+	+	+		+	+
Events	✓	+	+			+		✓				+	+		✓ ^F			+		+		+		+	+
Algorithm debugging	✗	+	+	+	+	+	+	+	✓	✗	+			✓				+				✓	✗ ^{SF}	✓ ^F	✗ ^F
System state verification	✗								✓	✗	+		✓					+			✓ ^{SF}	✗ ^{SF}	✓ ^F	✗ ^F	
Constraints validation	✗	+	+	+	+	+	+	+	✓	✗	+							+		✗	✓				
Optimisation	✓	+	+	+	+	+	+	+	✓	✗	+														
Generalisation	✓	✓		+	+	+	✓		✓	✗	+					+		+		+					