Research Article doi: 10.12973/eu-jer.9.3.1025

European Journal of Educational Research

Volume 9, Issue 3, 1025 - 1045.

ISSN: 2165-8714

http://www.eu-jer.com/

Patterns of Computational Thinking Development while Solving Unplugged Coding Activities Coupled with the 3S Approach for Self-Directed Learning

Arinchaya Threekunprapa
Mahidol University, THAILAND

Pratchayapong Yasri[*]
Mahidol University, THAILAND

Received: January 14, 2020 ▪ Revised: May 10, 2020 ▪ Accepted: June 5, 2020

Abstract: Using unplugged coding activities to promote computational thinking (CT) among secondary learners has become increasing popular. Benefits of using unplugged coding activities involve the cost-effective implementation, the ability to promote computer science concepts and self-efficacy in learning computer programming, and the engaging nature of active learning through collaboration. However, there is insufficient information regarding qualitative investigation on how learners develop their CT skills while working on unplugged coding tasks. This study therefore developed unplugged coding activities using flowcharts for high school students to learn computer science concepts, and to promote their CT skills. The activities consisted of five missions encompassing the concepts of sequence, repetition, input & variable, condition, and loop with condition. The data collection was carried out with 120 high students whose participation was video recorded and observed. A thematic analysis revealed that patterns of CT development started from initially developed, to partially developed and fully developed stages, respectively. The various stages were derived from different abilities to apply the computer science concepts to complete the missions with different expressions of CT skills. In addition, the study proposed a 3S self-directed learning approach for fostering the CT development, composing of self-check (in pairs), self-debug (in pairs), and scaffolding. It is therefore suggested to use the 3S model integrated with the unplugged coding activities for developing CT among high school learners.

Keywords: *Computational thinking, unplugged coding, flowcharts, 3S approach, computer science concepts.*

To cite this article: Threekunprapa, A., & Yasri, P. (2020). Patterns of computational thinking development while solving unplugged coding activities coupled with the 3S approach for self-directed learning. *European Journal of Educational Research*, *9*(3), 1025-1045. https://doi.org/10.12973/eu-jer.9.3.1025

Introduction

When it comes to the development of educational pedagogy in the 21st century, novel instructional approaches have been emphasised in the literature including STEM education (Changtong et al., 2020), hands-on activities (Pisanpanumas & Yasri, 2018), active learning (Setiawan et al., 2019), game-based learning (Pauline-Graf & Mandel, 2019), gamification (Zvarych et al., 2019), self-directed learning (Bourdeau et al., 2017), and unplugged coding (Threekunprapa & Yasri, 2020), to name a few. Despite differences in classroom practices, the common value of these approaches is the necessity to promote conceptual understanding of the subject matter, as well as to leverage soft skills such as creativity, critical thinking, collaboration and communication of learners simultaneously; all of which are considered crucial elements of effective classrooms in the 21st century (Toheri et al., 2020)

In addition to the novel teaching methods, new school curriculum in various countries including USA, UK, New Zealand, Germany, India, Georgia, France, Korea, Japan, Sweden, Finland, Israel, Russia, and Italy have included computer science as one of the core subjects (Grover & Pea, 2013; Hubwieser et al., 2015; Kert, 2019). Of course, Thailand is no exception (Threekunprapa & Yasri, 2020). This is due to the fact that computational thinking (CT) has been regarded as an important skill for students in the 21st century where technological advancement and robotic applications take precedence (Bulus Kirikkaya & Basaran, 2019; Ertugrul-Akyol, 2019). It is not only important for learning computer science concepts but also for solving everyday problems particularly in digital workplaces (Sondakh et al., 2020). Therefore, CT is generally referred to the thinking process involved in defining problems and proposing solutions that can be effectively implemented by a processing agent (Shute et al., 2017). In other words, Aho (2012) defines CT as the

[*] **Corresponding author:**
Pratchayapong Yasri, Institute for Innovative Learning, Mahidol University, Thailand. ✉ pratchayapong.yas@mahidol.edu

© 2020 The Author(s). **Open Access** - This article is under the CC BY license (https://creativecommons.org/licenses/by/4.0/).

cognitive process of computational procedures and algorithms to determine specific problem-solving methods, which can signify the thinking process that computer scientists use when they are confronted with a problem (Grover & Pea, 2013). Likewise, Shute et al. (2017) explain CT in the light of conceptual foundation required to solve problems effectively and efficiently with algorithmic solutions that are applicable to different contexts.

To emphasise more on the definition of CT, the literature informs that it consists of four pillars: decomposition, pattern recognition, abstraction and algorithmic thinking. First, decomposition is the ability to break down a complex task to smaller manageable tasks. Second, abstraction is the ability to choose an appropriate representation of a problem from multiple levels of abstraction to develop a solution that is applicable for varied situations. Despite the extensive use of the term abstraction in the literature, it does not explicitly capture the linguistic sense of the ability to exclude the unnecessary details (non-relevant information) of a problem. Therefore, the term is referred to throughout paper as abstraction and generalisation. Third, pattern recognition is the ability to search through a large amount of data to look for a shared pattern. Lastly, algorithmic thinking is the ability to find a promising way to solve the problem step by step (Sondakh et al., 2020; Wing, 2006). In addition to the aforementioned, the key dimensions of CT defined by Brennan and Resnick (2012) compose of computational concepts (the concepts executed for solving a particular problem based on an algorithmic manner), computational practices (the practices developed as the algorithmic designer solves the problem), and computational perspectives (the perspectives conceived by the algorithmic designer in relation to the real-world settings and their interactions with the designer himself). It is therefore considered here that any learning activities that aim to promote CT among learners should incorporate the four CT skills (Sondakh et al., 2020; Wing, 2006) into the lens of Brennan and Resnick (2012) whose focus is on educational implications.

In order to cultivate the CT skillset, much weight is given to the learning of computer programming (Hsu et al., 2018). Traditionally, it can be done either by text-based programming or by visual language programming (Lye et al., 2014). While the former includes understanding how to write codes according to commonly used computer languages like C/C++, Python, Java, PHP, and ASP, the latter adopts graphical elements and figures like Blockly to develop the sense of computer programming (Gunbatar & Karalar, 2018; Threekunprapa & Yasri, 2020). Of course, to be able to master in text-based programming is the ultimate goal if one wants to pursue a career involving computer programming. However, its passive nature and cognitive demands to remember computational codes may put novice learners off. In contrast, visual language programming is rather more active and emphasises more on algorithms, which has been proven more effective to engage and motivate novice learners to study computer programming (Mladenovic et al., 2018; Noone & Mooney, 2018). However, its simplicity can lead to misunderstanding of computer science concepts, especially advanced ones such as loops and conditional loops (Mladenovic et al., 2018). Last but not least, both depend heavily on the accessibility to computer devices which may be out of reach by many schools.

One way to make programming more conceptually accurate while maintaining the importance of visualisation is unplugged coding using flowcharts. Unplugged coding activities adopt game-based learning to promote logical thinking in which learners are given a mission to think of an algorithm in a sequential manner without using a computer. It is suggested that the use of game elements in learning can enhance cognitive engagement, motivation to learn, as well as deeper reflection upon learning activities (Zvarych et al., 2019). In addition, it is a learning environment where students can participate in active learning and collaboratively interact with others including peers and facilitators (Yuksel, 2019). Flowcharts are diagrams showing the process, system, or computer algorithms. Different shapes of diagrams convey different meanings of programming commands that allow advanced computer science concepts such as loops and conditional loops to be cultivated. This can sufficiently compensate what visual language programming lacks (Threekunprapa & Yasri, 2020). Research has shown that this approach can increase the accuracy of coding and help reduce the time for students to understand the algorithm when receiving visual expressions (Scanlan, 1989). In addition, it has shown to be more sophisticated to promote CT skillset than pseudocodes which rely on the use of informal language for human reading rather than machine reading (Threekunprapa & Yasri, 2020).

Besides, research has shown that university students learning introductory programming to develop algorithmic thinking preferred the use of flowcharts to pseudocodes as it leads to better understanding of algorithms, higher confidence in learning programming, fewer coding errors, and fewer time spent on learning (Scanlan, 1989). Furthermore, others show that using flowcharts for teaching algorithms contributes to positive learning outcomes, well-established problem-solving skills, and sufficiently developed programming skills (Giordano & Maiorana, 2015; Hooshyar et al., 2016; Noone & Mooney, 2018; Westphal et al., 2003). Among high school learners, research has shown that using unplugged coding activities integrated with flowcharts can help promote students' learning of computer programming in a variety of aspects including understanding of basic computer science concepts (Giordano & Maiorana, 2015; Hooshyar et al., 2016; Noone & Mooney, 2018; Westphal et al., 2003), advanced computer science concepts (Threekunprapa & Yasri, 2020), algorithmic designs (Giordano & Maiorana, 2015), computational and logical thinking (Bachu & Bernard, 2014; Threekunprapa & Yasri, 2020), and self-efficacy to learn computer programming (Threekunprapa & Yasri, 2020). Last but not least, the usefulness of unplugged activities is shown to be effective also among primary school pupils as those participating in the unplugged activities exhibited a greater level of enhancement in computational thinking skills compared to their counterparts who were exposed to a traditional way of learning (Brackmann et al., 2017).

However, little is known from a qualitative perspective how unplugged coding using flowcharts can help promote such positive development, especially among secondary school students, because the aforementioned studies rely purely upon quantitative research. An in-depth qualitative investigation allows researchers and educators to realise challenges that students may have when solving algorithmic tasks that require computational thinking and problem-solving skills. Therefore, this study set two main objectives. One was to explore characteristics of different patterns of CT development as students proceed with assigned unplugged coding tasks and problem-solving. The other was to explore how the 3S approach integrated in the unplugged coding activities based on flowcharts can help contribute to positive CT development. It is important to note that this present study is an extended investigation from a qualitative lens of our previous work (Threekunprapa & Yasri, 2020) which emphasises on a statistical comparison between student learning achievements on computer science concepts and CT skills before and after participating in the developed unplugged coding activities. In the previous research, no emphasis is given to patterns of CT development among learners; in addition, little is focused on the role of the 3S approach. No evidence is presented how the approach can potentially assist the development. However, all of which are important to explain what actually happens in between that may contribute to the improvement of the learning achievements after participating in the activities. The said gaps now become the central theme in this present study.

To emphasise on the 3S approach, it adopts two predominantly used educational practices: student self-assessment and scaffolding. While the former focuses on students evaluating their own work and learning process (Sharma et al., 2016), the latter on the role of teacher or facilitator to help provide support as needed on how to deal with a certain task and then step back (Van De Pol et al., 2010). Research has shown that self-assessment is a powerful tool to promote students' learning both in the cognitive and affective domains. Alaoutinen and Smolander (2010) show that it can be used to motivate learning and to follow a student's progress whose results demonstrate that students can place their knowledge along Bloom's revised taxonomy appropriately, and they perceive that this could help advance their learning. In addition, the study suggests that the conduct of self-assessment can provide instructors a more objective criterion for measuring the level of knowledge acquisition. The same is true with Brusilovsky and Sosnovsky (2005) whose work reveals students' positive perceptions towards an out-of-class self-assessment method used in the learning of the C language. It is proven also in the study that the students significantly developed their knowledge and skills related to the C language, and exhibited a great level of perceived usefulness of the computer language.

Furthermore, Kim et al. (2018) demonstrate the effectiveness of computer-based scaffolding in the context of problem-based learning whose results present a significant improvement of cognitive outcomes in the said learning environment. Linder et al. (2006) also reveal that the use of scaffolding can help lessen anxiety among learners dealing with software design. Their qualitative results show that even though the learning activities are progressively more difficult as they proceed, the learners express that scaffolding helps them become more confident in handling the tasks and see a clear direction how to improve their design skills. Therefore, in sum, the 3S approach proposed here is a form of collaborative learning in which students interact with peers through self-assessment as well as their teacher through scaffolding. Despite their extensive use in the educational arena, this approach extends the usefulness of the two principles in the learning of computer science concepts by diving self-assessment into two parts: self-check (in pairs) and self-debug (in pairs), coupled with scaffolding which is utilised throughout the activities. How to put these into practice in this particular study is discussed later in the methodology section.

Methodology

Participants and data collection

This study adopted a qualitative methodology based on on-site observations and analysis of students' work through video records. Participants were 120 secondary school students (54 males and 66 females) with an average age 16.7 years. All participants had no prior programming experience who were recruited by a convenience sampling method where subjects were convenient and readily available to participate. The access to the participants began from choosing a participating school where computer teachers were keen to let their students try out unplugged coding activities and allowed the researchers to conduct the data collection. The announcement was publicly made for secondary school students in the school who would be interested in participating in this data collection which took place as an extracurricular activity after their school time. Ethical aspects of research were taken into consideration with respect to the right and safety of participants both physically and psychologically. Their role in the data collection process was fully made known, as well as their right to withdraw from it if they wished, and thus their decision to take part was solely voluntary (a consent form was signed by each).

The entire process of data collection took approximately 2.5 hours, starting from the introduction to the activities, followed by working on five assigned missions in pairs until completion. To proceed with the five missions, students were paired up by themselves on a voluntary basis. Therefore, the whole group of 120 students consisted of 60 pairs who were facilitated by 4 well-trained facilitators throughout the period of data collection. The participants were also informed that while completing the missions, their responses in the tasks would be recorded. However, only their hands, voice and their arranged flowcharts would be recorded without a possibility of personal identification. In addition, these records would be kept confidential and only used for data analysis and nobody could gain access to these, apart from the two researchers in this study.

Research Tool: Unplugged coding using flowcharts

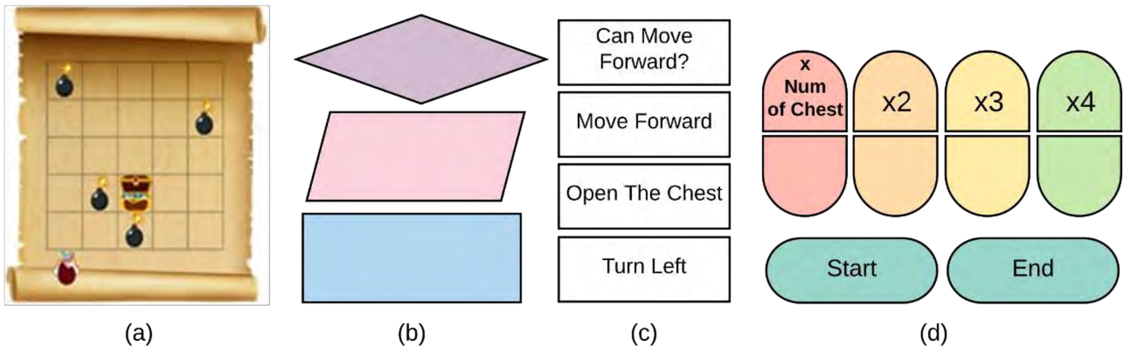


Figure 1. Game components: (a) mission board; (b) flowcharts blocks; (c) commands (d) auxiliary blocks

The unplugged coding game is called Treasure Hunter which composes of five missions, starting from simple to complex ones. The components of the game include one mission board, commands, flowchart blocks, and auxiliary blocks (see Figure 1). The number of commands in each mission will be detailed later in the result section. This particular part aims to propose the overall feature of the game. Two players are matched voluntarily in order to complete one mission at a time. Each pair is given one mission board per mission in the form of 5x5 grid table (see Figure 2). The expected response is to find a treasure chest (or chests depending on the given mission), starting from an assigned beginning point. In order to get the treasure chest, the players have to walk the treasure hunter from one square to another in a designated direction (i.e. their own designed algorithm). Obstacles (bombs) are placed throughout the mission board for them to avoid. Once they arrive at the place where the treasure chest is located, they can take it and role a 6-faced dice to get the number of diamonds as a reward.

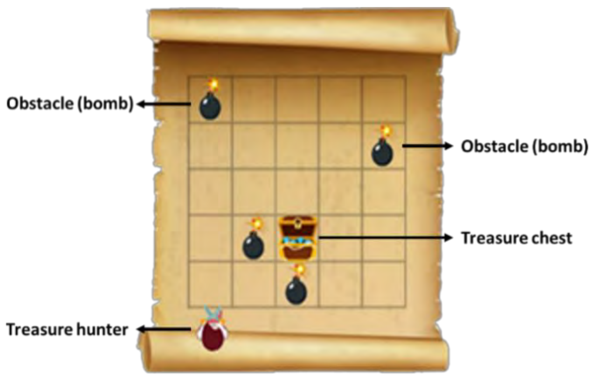


Figure 2. A mission board

In order to walk the treasure hunter along the mission board, the players have to use the flowchart blocks, commands and auxiliary blocks that are provided for each of the missions. Different shapes of the flowchart blocks convey different command meanings (see Figure 1). The parallelogram represents the input and output. The rectangle represents the process. The diamond means decision (or condition). The commands determine what certain actions to be made. However, the players have to match the commands with the right shape of flowchart block according to diagrammatic command meanings as described above.

For example, the *Can move forward?* command is used as a condition to check whether the treasure hunter can move forward to the next square or not. This condition requires binary responses (two out paths: yes or no). If the answer is yes, then do action X. If the answer is no, then do action Y. Figure 3 shows an example that this condition has to be placed onto a diamond flowchart block in order for the action to be made correctly.

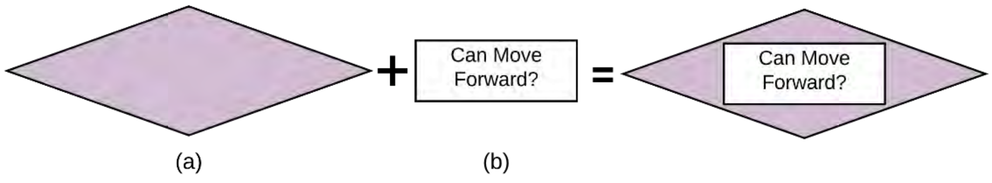


Figure 3 How to combine the flowcharts block (a) and the condition command (b)

To form a complete algorithm, the players have to start off with the auxiliary block *start* (a blue oval shape with the word *start* printed on) as shown in Figure 1(d), followed by a set of commands with flowchart blocks (interchangeably called codes), and end the entire flowchart with the *end* auxiliary block (another blue oval shape) as also shown in Figure 1(d). In later missions where the players can use the concepts of repetition, they can use other auxiliary blocks to construct their flowchart. An example of repetition is shown in Figure 4 where the *turn left* action in placed onto the rectangular shape within the loop of 2x repetition, which means that this action has to be repeated twice.

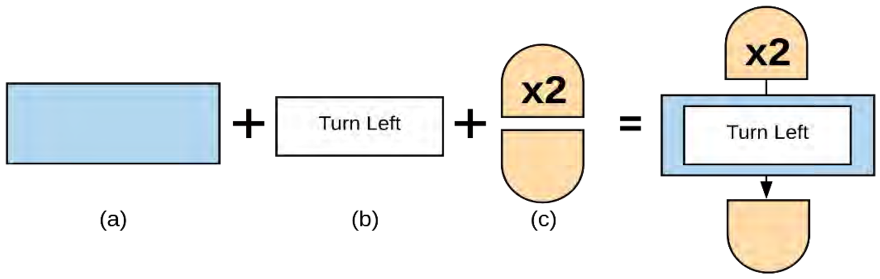


Figure 4. How to combine flowchart block (a) + command (b) + auxiliary block (c)

It is important to note that each mission has a different starting point, chest number, obstacle number, and chest position, leading to various strategies of algorithmic design which give various bonus points as rewards. Each of the missions is set to promote a different computer science concept. In addition, when proceed to a more advanced mission, the players can apply the previously learned concept to help accomplish the mission. The five missions aim to cultivate the concepts of sequence, repetition, input and variable, condition, and loop with condition, respectively. The detail of each mission is later described in the result and discussion section so that students' responses can be viewed against the assigned mission more clearly. Figure 5 shows an expected flowchart that the players can complete in the 5th mission (the most advanced). Therefore, the five computer science concepts are expected to be used altogether to accomplish this.

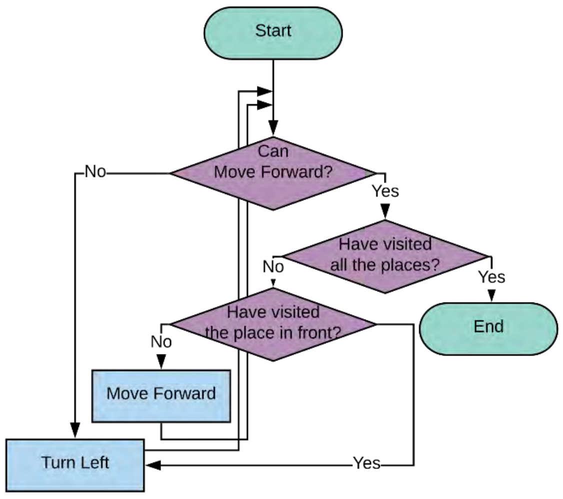


Figure 5. A complete flowchart (algorithm) to complete the 5th mission in the game

The 3S approach is an integral part of the entire game: self-check (in pairs), self-debug (in pairs), and scaffolding. Before proceeding to the next mission, the two players have to split their role into two. One is to read out the codes arranged in the algorithm and the other has to follow accordingly. This is a self-check (in pairs) process. Through this, the players can realise by themselves whether they can get the treasure chest as expected or not. If they can complete the mission, they can roll the dice to get the reward and then proceed to the next mission. However, if they cannot, they have to identify which part in the flowchart that makes the mission unsuccessful. This is a self-debug (in pairs) process or the modification of some parts of the flowchart which can help them complete the mission. Scaffolding comes into play as the players work through their tasks. This is when well-trained facilitators can help ask questions to guide the players to think what they should do to make it work, what computer science concepts may be drawn to complete the mission, or what they could have done wrong. However, the facilitators are to offer support as needed, and that they must not tell the players directly how to complete the mission.

A certain protocol for scaffolding is given to the facilitators to follow. Whenever students find a struggle, a facilitator would assist by first asking what struggle they seem to face. This can be done either by observation or called attention by students themselves. Then the facilitator would ask the students to explain the computer science concept(s) that they think may be relevant to this particular mission. After that, the facilitator would prompt the students to think whether a new computer science concept introduced to solve that particular mission can be of any use. Here the facilitator gets the students to think about the main purpose of the mission in the light of the newly introduced concept without directly telling them what to do with the codes. Subsequently, the students are encouraged to divide the main mission into subtasks where the facilitator guides them to work through each of them using self-check (in pairs) and self-debug (in pairs) once again. The whole process can be repeated once the students find another struggle.

Data analysis

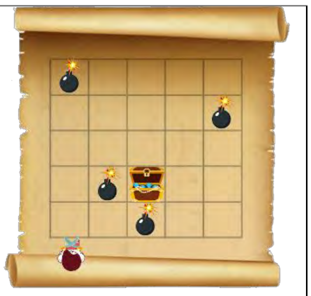
Based on 120 student participants who formed 60 different pairs, a total of 300 algorithms were retrieved from their responses to the five missions. However, in this analysis, only algorithms produced from pairs that show different forms of CT development were selected to analyse in greater depth. A thematic analysis allowed the researchers to classify three main patterns of CT development in each of the missions, composing of initially developed, partially developed and fully developed stages. The initially developed stage represents algorithms that cannot complete the mission due to misunderstanding of computer science concepts or the inability to apply appropriate computer science concepts to complete the mission. The partially developed stage is where players begin to apply the computer science concepts to handle the mission; however, they tend to take a partial advantage of the concept, or take a full advantage but cannot complete the mission. In contrast, the fully developed stage is where players can take a full advantage of the concepts and effectively complete the mission, as well as they can exhibit satisfactory characteristics of CT skills.

Results and Discussion

Mission 1: Sequence

In mission 1 which is the most fundamental, the players are given a mission board containing one treasure chest and four bombs. This mission aims to cultivate students' understanding of the concept of sequence where six *move forward*, four *turn left*, four *turn right*, and one *open the chest* commands are provided with different forms of flowchart blocks for them to complete the mission.

Table 1. Game components and the focused concept in Mission 1

Mission 1: Let's get the treasure chest and avoid the bombs. ● Move Forward x 6 ● Turn Left x 4 ● Turn Right x 4 ● Open the chest Concept: <u>sequence</u> is a basic algorithm that allows actions to be carried out in order and step by step to complete a certain task.	

The flowchart in Figure 6(a) shows that in the *initially developed* pattern, the players cannot construct any algorithm at all (the blue dotted boxes represent the missing flowchart blocks). Here, only the commands are sequenced without using any flowchart blocks to form proper codes, pointing to a lack of understanding how flowcharts work. Through the 3S approach, the players begin to develop partial understanding how to form a flowchart as they correctly put the *start* and *end* codes with certain commands combined with flowchart blocks in Figure 6(b). However, they cannot complete the mission by this algorithm due to two major flaws (indicated in the red dash boxes). One is the wrong sense of direction (left and right). The other is that they fail to include the correct number of codes that makes the treasure hunter move forward. Similar forms of response happen with other pairs of players with a variation of error positions, pointing to their incomplete development of sequential and spatial thinking. Therefore, this kind of algorithm is classified as the *partially developed* pattern. However, after using the 3S approach, the players can reach the *fully developed* pattern, shown in Figure 6(c), in which the players put the right commands with correct flowchart blocks (showing their complete understanding how flowcharts work), and the right number of codes (showing their complete development of the concept of sequence) to accomplish the mission.

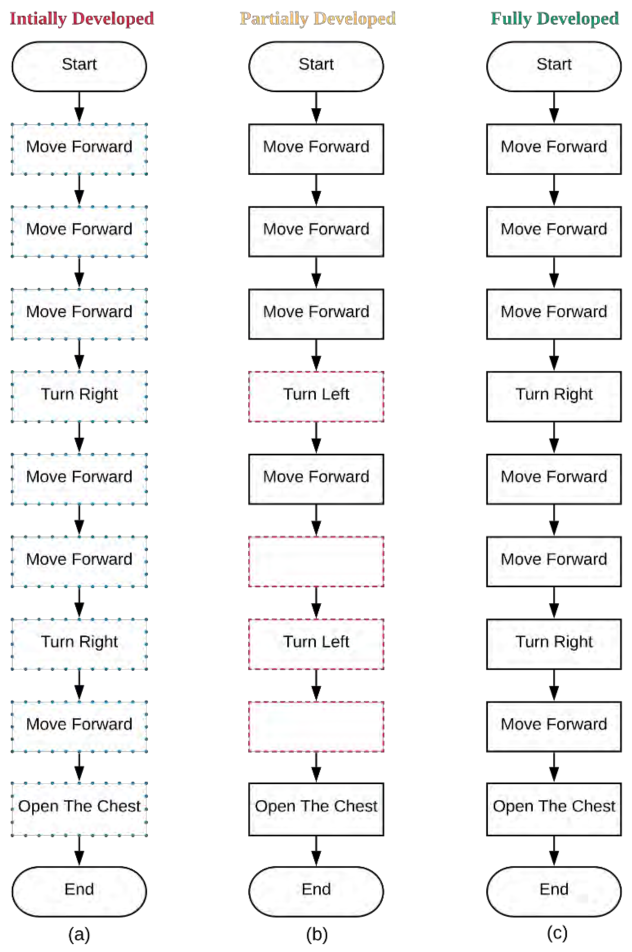
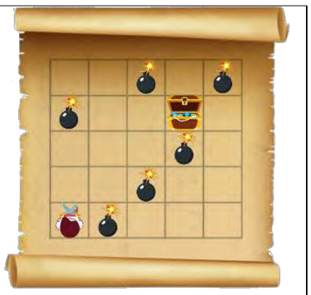


Figure 6. Patterns of CT development in mission 1 (sequence): (a) Initially Developed; (b) Partially Developed; (c) Fully Developed (remarks: the blue dotted boxes represent the missing flowchart blocks; the red dashed boxes represent the wrong codes)

Mission 2: Repetition

In mission 2, the players are given a mission board containing one treasure chest and six bombs. This mission aims to cultivate students' understanding of the concept of repetition while carrying on the use of the concept of sequence. In this mission, two *move forward*, two *turn left*, two *turn right* and one *open the chest* commands are provided with a collection of flowchart blocks. In addition, one repetition auxiliary block is provided. It is a mandatory code to use specified by the mission. It should be noted that the players would not be able to complete the mission only by using the concept of sequence as the number of codes is limited, and thus prompting them to apply the new concept of repetition to solve the task.

Table 2. Game components and the focused concept in Mission 2

Mission 2: Let's get the treasure chest and avoid the bombs through the shortest route • Move Forward x 2 • Turn Left x 2 • Turn Right x 2 • Open the chest • Repeat (x4, x3, x2) Concept: <u>repetition</u> is a function to repeat an action which helps shorten the length of coding, representing a more effective way to solve a problem	

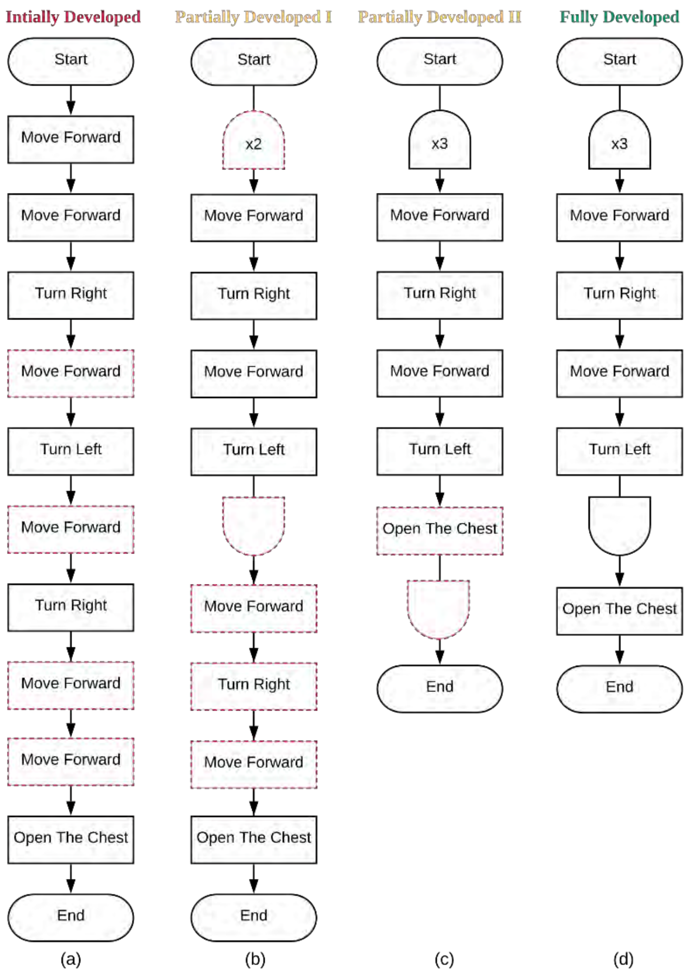


Figure 7. Patterns of CT development in mission 2 (repetition): (a) Initially Developed; (b) Partially Developed I; (c) Partially Developed II; (d) Fully Developed (remarks: the red dashed boxes represent the wrong points of algorithms)

In Figure 7(a), it is interesting to see that the players can reach the treasure chest by using the concept of sequence learned from the previous mission. However, this completion does not follow the rule of the mission as the number of codes actually exceeds what is allowed. The red dashed boxes show excess codes that the players use to complete the mission. As a result, this algorithm is categorised in the initially developed pattern where students hold insufficient understanding of the concept of repetition focused in this mission, although can fully incorporate the concept of sequence.

However, through the 3S approach, they begin to arrive at the pattern of *partially developed I* in Figure 7(b). The flowchart reveals that they carry on using the concept of sequence with a sign of attempts to integrate the concept of repetition. However, the repeated pattern is not successfully developed, making them miss out one *turn left* step which is supposed to be added in the third repetition round. Later on, they begin to show the pattern of *partially developed II* in Figure 7(c) which appears to be more advanced as three repetitions are used correctly. Nonetheless, the only flaw emerged in this algorithm is the inclusion of a code that does not need to be repeated (i.e. *open the chest*). To include this code in the loop, it means that this action has to be repeated three times which is, in fact, not necessary.

Again, through 3S, they exhibit the *fully developed* pattern in Figure 7(d) which shows their successful development of *pattern recognition* as they can see the three repetitions of the four actions. In addition, they can put these repeated actions in one repetitive loop with a correct number indicated after the *start* code using the auxiliary block. Once the repeated actions are completed, they can put the action of *open the chest* before ending the entire algorithm using the *end* code.

Mission 3: Input and variable

In mission 3, the players are given a mission board containing two or three treasure chests and two bombs. The students are required to compose an algorithm that is flexible to collect the two or three treasure chests depending on the number shown on a rolled dice which is additionally provided. It is important to note that a 6-sided dice is specifically made to serve this mission, and thus it contains three sides with number 2 and the rest with 3. So, the players have to use roll the dice as a variable to be an input for repeating the set of actions to collect each treasure chest. This mission aims to cultivate students' understanding of the concept of input and variable while carrying on the use of the concepts of sequence from mission 1 and repetition from mission 2. In this mission, the players can use only six *move forward*, four *turn left*, four *turn right*, one *open the chest*, and one *Num of Chest = roll the dice* (input) commands. In addition, an *x Num of Chest* auxiliary block (variable repeat) and other repeat blocks are provided. It should be noted that students would not be able to complete the mission by using only the concept of sequence or a fixed round of repetitions like (x2, x3, x4) as the number of treasure chests could vary depending on the number shown on the rolled dice, thus prompting them to apply the new concept of input and variable to solve the mission.

Table 3. Game components and the focused concept in Mission 3

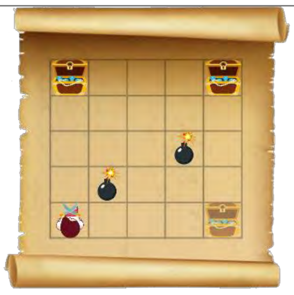
Mission 3: Let's get all treasure chests determined by the number shown on a rolled dice. ● Move Forward x 6 ● Turn Left x 4 ● Turn Right x 4 ● Open the chest ● Input (Num of Chest = Roll the dice) ● Repeat (x Num of Chest, x4, x3, x2) Concept: Input and variable: Input is the way to get the value from the user (a dice). Variable is the way to store the value to use in the program.	

Figure 8(a) reveals that the players use the concept of sequence and repetition learned from the previous missions where they take the advantage of repetition within the limited number of commands using the *x3* auxiliary block to a set of action. However, in this mission, it is different from the previous one because the number of the chest is not always 3. This algorithm would not help the players complete the mission if the rolled dice indicates that they have to collect two treasure chests. Furthermore, having five *move forward* commands as well as having the *open the chest* command outside the loop would not allow them to achieve what they desire. Therefore, this form of computational thinking is classified in the *initially developed* pattern.

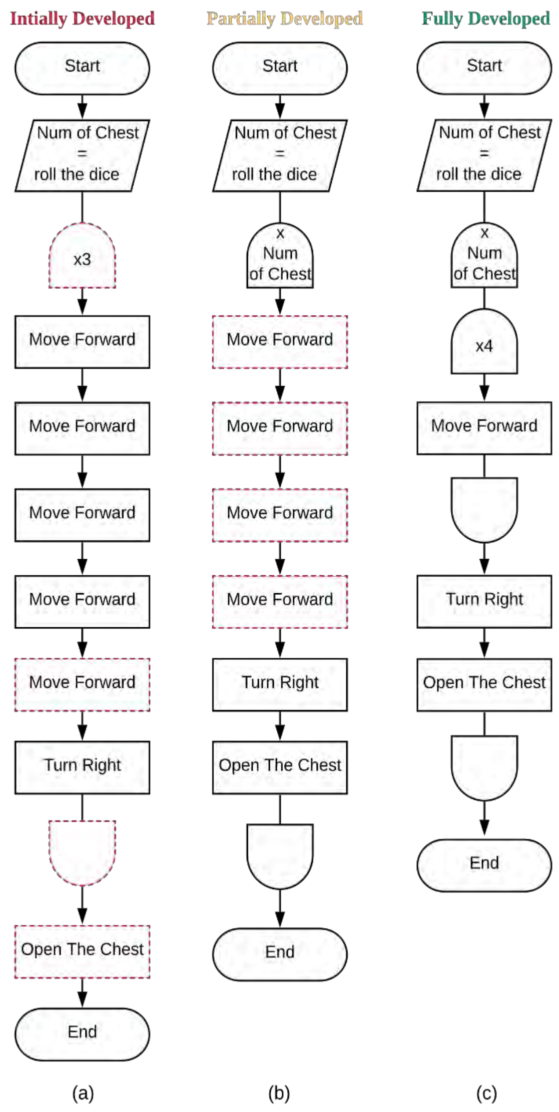


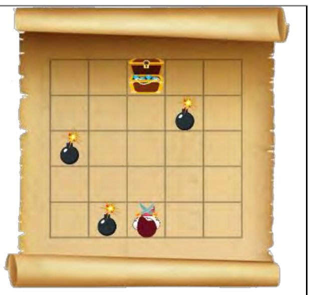
Figure 8. Patterns of CT development in mission 3 (input and variable): (a) Initially Developed; (b) Partially Developed; (c) Fully Developed (remarks: the red dashed boxes represent the wrong point of algorithms)

However, the aforementioned mistake can be easily corrected when the players do *self-check (in pairs) and self-debug (in pairs)* by themselves. It can be seen from Figure 8(b) that the *open the chest* command is put back within the loop and that the action would be repeated every time when the players reach the treasure chest. In addition, one *move forward* command is removed so that the actions are made to accomplish the mission. Through scaffolding, the players realise that they can use the *x Num of Chest* command to allow flexible repetition to input the number of the chest to repeat the actions according to the exact number of chests. However, this form of computational thinking development is still classified in the *partially developed* pattern because the players exhibit an *incomplete pattern recognition* which is essential for them to shorten their algorithm. To be more precise, the players still put four *move forward* commands while these could have been shortened by another loop of actions. Once again, through *scaffolding*, it is made possible for the players to see that they can shorten the algorithm by using the *x4* auxiliary blocks. This is the stage where the players are classified in the *fully developed* pattern as shown in Figure 8(c), meaning that they can completely incorporate the concepts of input and variable alongside sequence and repetition to complete the mission, alongside the expression of *complete pattern recognition*.

Mission 4: Condition

In mission 4, the players are given a mission board containing one treasure chest and three bombs. This mission aims to cultivate students' understanding of the concept of condition while carrying on the use of the concept of sequence and repetition. In this mission, the players are provided with two *move forward*, two *turn left*, two *turn right*, and one *open the chest* commands. Additionally, they are given one condition command asking *Can move forward?* to complete the mission. Although the players can simply complete the mission using only the concepts of sequence and repetition, it is the rule of this mission that they cannot use the repetition auxiliary block. In addition to this, there is a limited number of *move forward* commands.

Table 4. Game components and the focused concept in Mission 4

Mission 4: Let's get the treasure chest and avoid the bombs, use condition to move forward to the chest ● Move Forward x 2 ● Turn Left x 2 ● Turn Right x 2 ● Open the chest ● Can move forward? Concept: Condition is for different actions to be made which generally appear in the form of yes and no condition. Hence, actions different depending on the absence or presence of the condition.	

It is not surprising to see that the players start this off with the *initially developed* pattern where they decide to use the *move forward* commands with the *x4* repetition auxiliary block to make the shortest path to reach the chest as shown in Figure 9(a). Although the algorithm is correct, the rule of this particular mission does not allow them to do so. Therefore, this form is classified as the *initially developed I* pattern.

Through scaffolding, the players begin to use the condition command in Figure 9(b). However, this is with a critical error. In order to use the condition correctly, it is supposed that the response should be binary (having two out paths: *yes* and *no*). Thus, having one direction of response which is a *yes* in this algorithm makes it an incomplete flowchart. In addition, from the arranged codes, it is impossible to distinguish when the *move forward* response should be stopped so that the *open the chest* command can be done (shown in the red dash). This is where the computational thinking pattern is still classified as the *initially developed* pattern. Nonetheless, with the attempt to incorporate the concept of condition to solve the mission, this is classified as *initially developed II*.

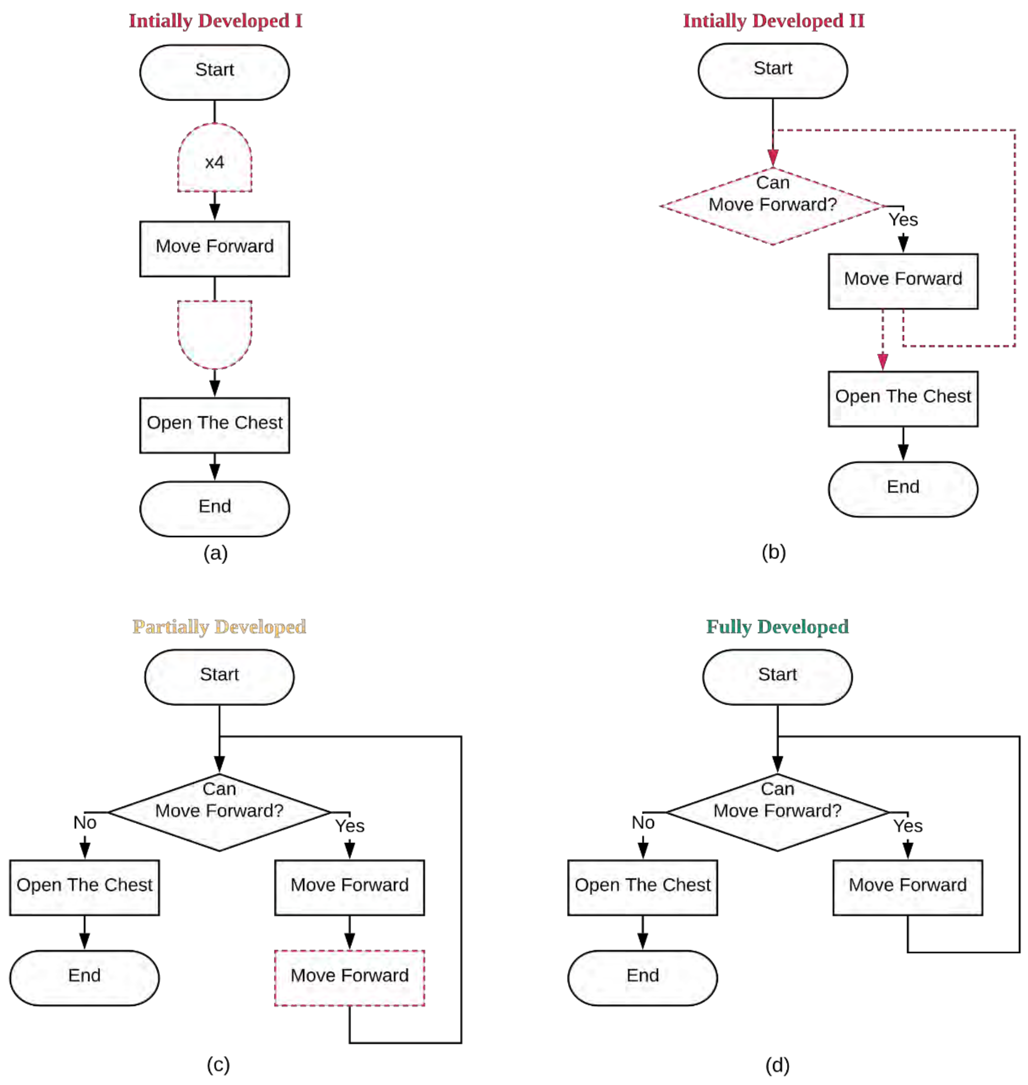


Figure 9. Patterns of CT development in mission 4 (condition): (a) Initially Developed I; (b) Initially Developed II; (c) Partially Developed; (d) Fully Developed (remarks: the red dashed boxes represent the wrong point of algorithms)

Moving on to the response in Figure 9(c), a form of *partially developed* pattern can be found. This is when the players use the condition command with two out paths. However, there is an issue when the players have the duplication of the *move forward* commands. Although this duplication is algorithmically valid, this algorithm is only successful in this particular case. This is due to the fact that whenever the players can move forward (according to the condition), they have to do it twice every time. This algorithm can be only successful when the number of moving forward is even (like the one shown in the mission board). However, when the number of moving forward is odd, this algorithm is not applicable. This mission encourages the players to design an algorithm that can be applicable to other different scenarios and this is when *abstraction and generalisation* can be fully developed.

Through scaffolding, the players can immediately realise that by removing one *move forward* command, their algorithm can be applicable to any number of the repeated actions no matter the number of moves is odd or even. This is when they are classified as holding a *fully developed* pattern of computational thinking as shown in Figure 9(d).

Mission 5: Loop with condition

In mission 5, the players are given a mission board which does not have a specific location where the treasure chest is. The board symbolises a very dark cave where no bombs are present inside, but the treasure hunters have to move along the cave to every single place (square) to make sure that they do not miss out any chest. Obviously, there is no treasure chest to be collected. Therefore, the mission aims for the players to design a complex algorithm to move to every single square from the assigned starting point. To be more specific, it aims to cultivate students' understanding of the concept of loop with condition while carrying on the use of the previous concepts. This is considered the most advanced mission in which computer science concepts, ranging from basic (i.e. sequence and repetition) to advanced ones (i.e. input & variable, condition, and loop with condition) can be exercised.

The players are provided with one *move forward*, one *turn left*, and one *turn right* commands with flowchart blocks. Three additional condition commands in the form of questions were given, composing of *Can move forward?*, *Have visited the place in front?*, and *Have visited all the places?* with flowchart blocks. It should be noted that the players would not be able to complete the mission by using only the concept of sequence or repetition or one condition due to the limited number of commands and the complexity of the mission itself, thus prompting them to apply the new concept of loop with condition with all previous concepts to complete the mission.

Table 5. Game components and the focused concept in Mission 5

Mission 5: Let's get the treasure chest. It hiding in the dark of the cave without bomb. Go and visit every single place (square) to look for it. ● Move Forward ● Turn Left ● Turn Right ● Can move forward? ● Have visited the place in front? ● Have visited all the places? Concept: Loop with condition: To repeat an action by the yes and no condition. For example, a while-loop is to execute an action while a set condition is true.	

Furthermore, mission 5 is considered to be the mission where patterns of computational thinking can be elicited through the four pillars, composing of *algorithmic thinking, pattern recognition, abstraction and generalisation,* and *decomposition*. First, the *algorithmic thinking* skill, the players have to transfer the concepts gained from all previous missions to construct a series of codes containing the right commands with correct shapes of flowchart blocks. This skill is also emphasised in missions 1, 2 and 3 where essential computer science concepts are also promoted. However, in this current mission, they have to integrate all the computer science concepts that have been previously learned. Second, the *pattern recognition* skill is used in this mission for finding the solution by looking at the "walk through the wall" pattern (similar to the *move forward* command with condition in mission 4). Third, an *abstraction and generalisation* skill is used to find a generalised algorithm that can be applicable to other similar situations, in addition to the one given in the mission. It should be noted here that the particular skill is first promoted in mission 4 where only one condition is set to challenge the players to think abstractedly. However, in mission 5, this involves multiple conditions from moving forwards to visiting all the places.

The final pillar is a *decomposition* skill which is heavily emphasised in this mission. Based on the mission given, the players have to think carefully how to move to every single square with an algorithm that is supposed to be as short as possible. Therefore, theoretically, the players are supposed to decompose the major task (i.e. move every single square) to four subtasks as shown in Figure 10. First, they should begin by visiting all the outermost square s as shown in the yellow arrow in Figure 10(a). Subsequently, when the players are about to come back to the place where they have visited, they have turn left to an inner round indicated in the green arrow in Figure 10(b). Now, the players can repeat subtasks 1 and 2 again to be certain that every single square is visited as shown in Figure 10(c). Then, the fourth subtask is the stop action when all the squares have been completely visited as shown in the red dot in Figure 10(d).

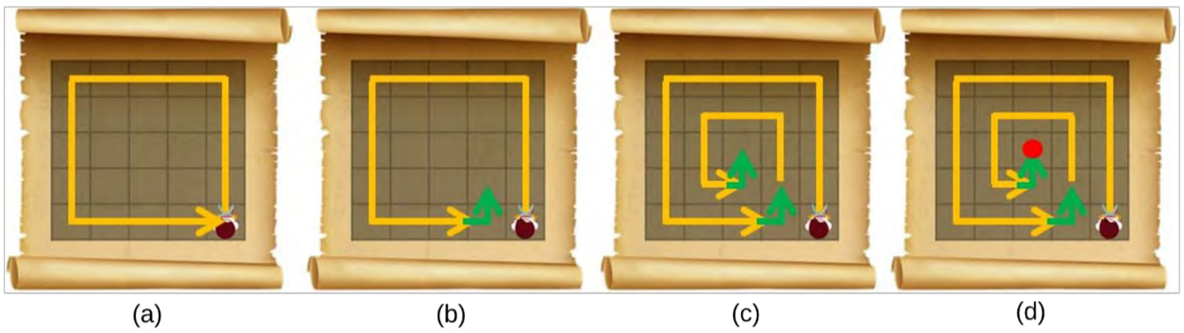


Figure 10. Decomposition in mission 5 (loop with condition): (a) 1st subtask to visit around the outermost round (yellow arrow); (b) 2nd subtask to turn left to an inside round (green arrow); (c) 3rd task to repeat 1st and 2nd subtasks to get inside the innermost round; (d) 4th subtask to stop when all the places have been visited

Overall, there are nine patterns of CT development elicited in mission 5, composing of three *initially developed*, one *partially developed*, and five *fully developed* patterns. To start off, Figure 11 shows the *initially developed* and *partially developed* patterns. It is evident that all of those *initially developed* patterns exhibit a lack of the *decomposition* skill. In Figure 11(a), the players try to complete the mission using only two conditions. The first condition is set to ask whether they *can move forward* or not with two out paths. The second concerns the condition whether they *have visited the place in front* with two out paths. However, this algorithm would make them stop only at the outermost round.

In Figure 11(b), the players modify the algorithm, but remain using only two conditions as previously done. However, errors appear including the deadlock loop and no specified response after the *turn left* code. However, in Figure 11(c), the players begin to incorporate the *have visited all the places* condition to fix the deadlock loop, but this condition has to have two out paths instead of one. In addition, the *turn left* command has no out path and no end. Therefore, the mission is not completed.

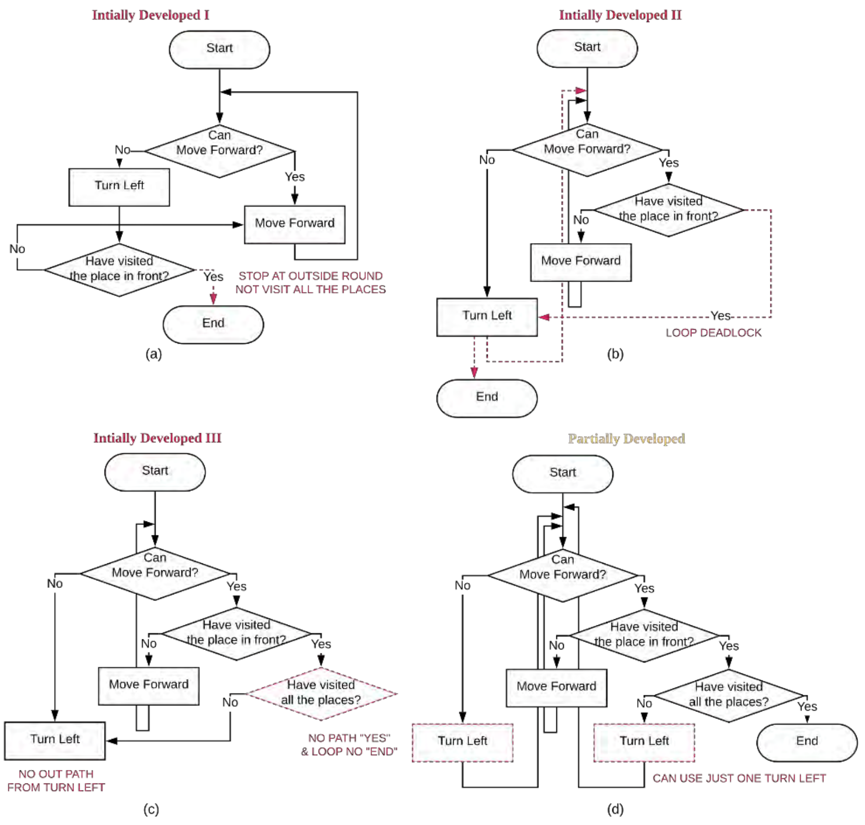


Figure 11. Patterns of CT development in mission 5 (loop with condition): (a) Initially Developed I; (b) Initially Developed II; (c) Initially Developed III; (d) Partially Developed; (remarks: the red dashed boxes represent the wrong point of algorithms)

Through 3S, the players begin to arrive at the *partially developed* pattern of computational thinking as shown in Figure 11(d). To sum up their major flaw, this algorithm exhibits *partial abstraction and generalisation* due to the excess *turn left* command which could have been one instead of two (shown in red dashed boxes). However, this algorithm shows an excellent progress in *decomposition*. To digest Figure 11(d) into subtasks, Figure 12 shows how their computational thinking in this respect is developed. Figure 12(a) shows the 1st subtask that helps move along the outermost round using the condition of *Can move forward?*. If they can *move forward*, then continue to *move forward*, or else *turn left*. Figure 12(b) shows the 2nd subtask that helps the players *turn left* before they go back to the same path through the condition of *Have visited the place in front?*. Figure 12(c) shows the 3rd subtask that adds the path from two *turn left* commands back to the 1st and 2nd subtasks. Then, Figure 12(d) shows that the use of the *Have visited all the places?* condition to end the actions. These reveal that the players can decompose the four subtasks algorithmically. However, as mentioned above, the two *turn left* commands should have been combined (as they go back to the same condition anyway) so that *complete abstraction and generalisation* could be shown (see Figure 5 for a more sophisticated algorithm).

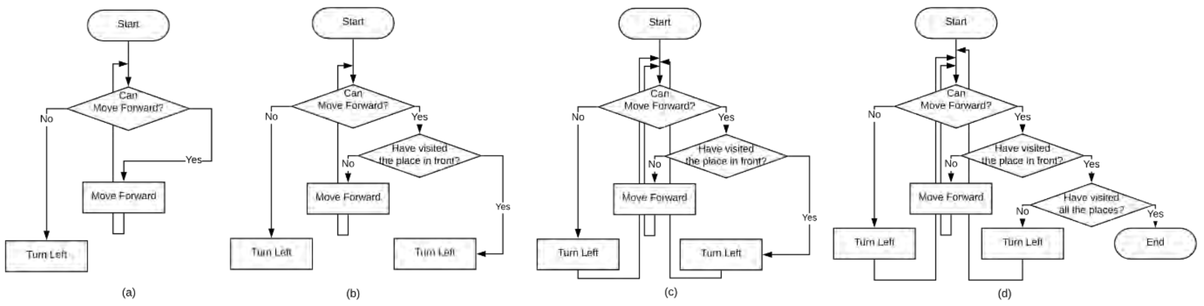


Figure 12. CT decomposition subtasks: (a) 1st subtask to visit around the outermost round (yellow arrow); (b) 2nd subtask to turn left to an inside round (green arrow); (c) 3rd task to repeat 1st and 2nd subtasks to get inside the innermost round; (d) 4th subtask to stop when all the places have been visited

Turning to the *fully developed* patterns, it is interesting to see that the players exhibit a reasonable variation of algorithms that help complete the mission. There are five different algorithms in total based on the whole group of participants. This variation comes from changes in the order of the three conditions. However, they all can help the players move to every single square as expected (shown in Figure 13). Although it is not the emphasis in the activity that the players should design an algorithm that is both successful and minimal, meaning that the number of checking steps should be as few as possible, this criterion would help determine which algorithm is the most effective.

Table 1 shows that the checking steps vary from 66 (*fully developed V*) to 94 (*fully developed I*). Therefore, the lowest number is considered the most sophisticated. This is due to the fact that among the five different algorithms that can equally complete the mission, the number of checking steps done according to the algorithm shown in Figure 13(i) is the smallest, compared to the others.

Table 6. The number of checking steps of successful algorithms

Pattern	**1st Condition**	**2nd Condition**	**3rd Condition**	**Number of Checking**
Figure 12(e): Fully developed I	33	32	29	94
Figure 12(f): Fully developed II	33	32	27	92
Figure 12(g): Fully developed III	33	30	29	92
Figure 12(h): Fully developed IV	33	30	6	69
Figure 12(i): Fully developed V	33	27	6	66

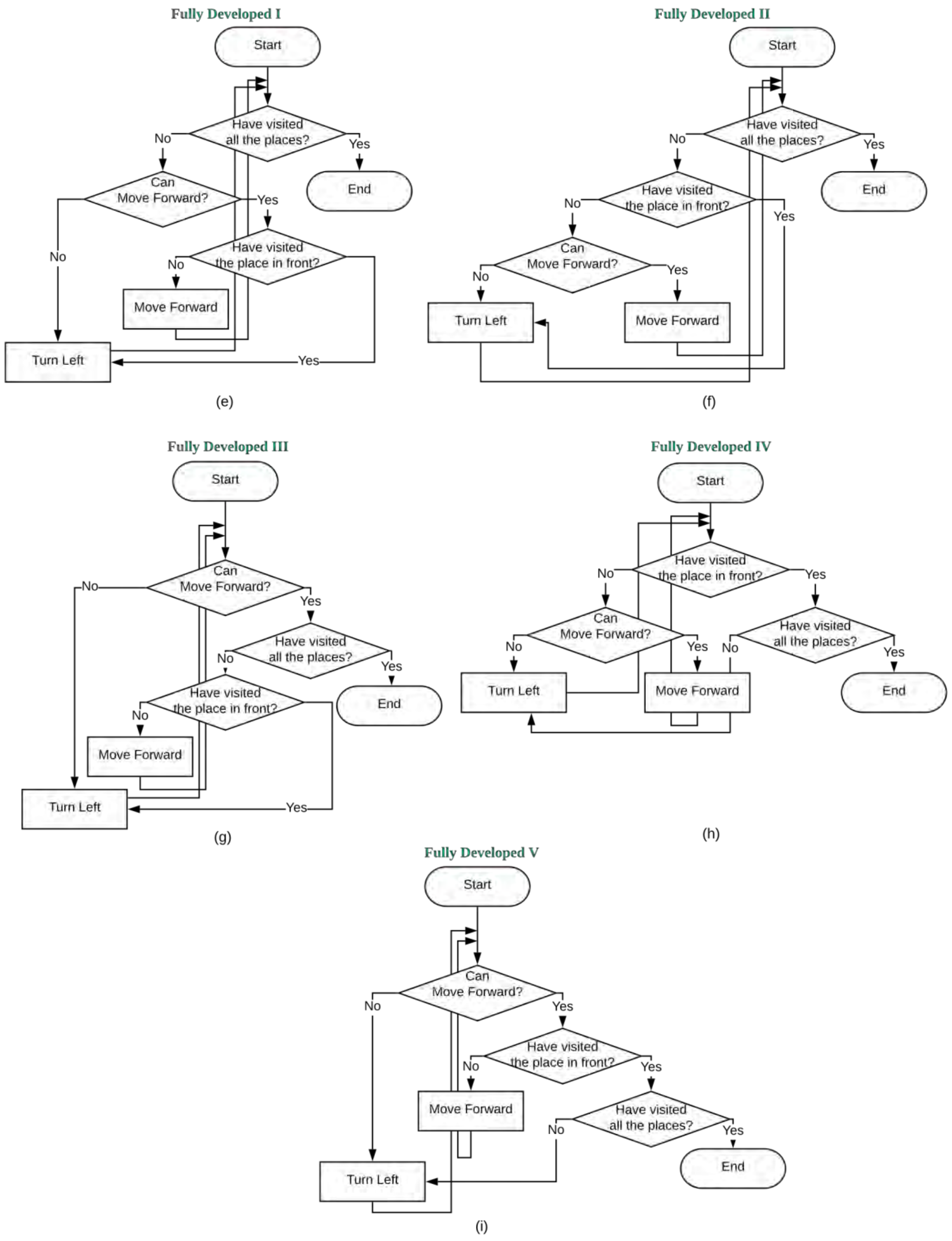


Figure 13. Patterns of CT development in mission 5 (loop with condition): (e) Fully Developed I; (f) Fully Developed II; (g) Fully Developed III; (h) Fully Developed IV; (i) Fully Developed V

Implications of the Study

Computational thinking has been shown to be an essential skill in the 21st century as learners can apply it to problem solving in daily life. It is generally defined as algorithmic design of processes to solve problems varying from simple to complicated tasks. This study contributes to the research community in three aspects. First, it offers a set of content specific unplugged coding activities using flowcharts to promote students' understanding of computer science concepts (i.e. sequence, repetition, input & variable, condition, and loop with condition) and computational thinking skills (i.e. *algorithmic thinking, pattern recognition, decomposition*, and *abstraction and generalisation*) through completing the five missions. Second, it offers an instructional approach for helping students develop the understanding and the skill by themselves through the use of 3S (i.e. *self-check (in pairs), self-debug (in pairs),* and *scaffolding*). Last but not least, it highlights three main patterns of computational thinking development (i.e. *initially developed, partially developed*, and *fully developed* patterns) that are content-specific. Now the three areas of contribution are discussed in turn.

In terms of the unplugged coding activities using flowcharts, this study shows that the whole set of activities can potentially engage high school students in the learning materials. Although no evidence for this claim is provided in the result section where the developmental patterns are primarily focused, it is important to point out that throughout the period of 2.5 hours, the whole group of participants expressed enthusiasm and interest from start to finish, even though the data collection was not at all compulsory to them, no extra scores were given to their participation, no class teachers were actually present to check their attendance, no reimbursement was incentivised, on top of that, all this took place outside their lesson time. Their presence in the data collection, participation in the activities, eagerness to complete the missions, enthusiasm to fix their codes until completion, and questions arisen along the way would be sufficiently assumed that they were effectively engaged in the activities both cognitively and emotionally.

Reported elsewhere, these unplugged coding activities can help promote students' conceptual understanding of the five computer science concepts and computational thinking based on the comparison between pretest and post-test mean scores, as well as increase the level of students' self-efficacy in learning about computer programming (Threekunprapa & Yasri, 2020). It is important to note that the five missions in the activities incorporate both computer science concepts and computational thinking skills. Mission 1 is designed to lay a good foundation of sequential and spatial thinking, alongside how flowcharts work. This is when *algorithmic* skills can be developed. Understanding and skills developed from this mission are used repeatedly in the other four missions. Mission 2 is used to promote the understanding of the concept of repetition where the skill of *pattern recognition* is cultivated. Mission 3 is designed to promote the understanding of input and variable where basic *decomposition* skills can be learned. Despite the emphasis on the previously learned concepts and skills, mission 4 aims to help students understand the concept of condition, as well as develop *abstraction and generalisation.* The skill of *abstraction and generalisation* here is specific to the ability to design an algorithm that can complete the given mission and is applicable to other similar situations. Finally, mission 5 is the most advanced one where the five computer science concepts have to be used to complete the mission, alongside the greater emphasis on *abstraction and generalisation,* and *decomposition* (of course *pattern recognition* and *algorithmic thinking* are fundamental here). It is therefore highly recommended for other researchers and computer science teachers to use this form of activities with their students. Its usefulness has been evident in the literature and its cost-effective implementation has been captivating.

Moving to the approach assisting students to learn about the computer science concepts and to develop their computational thinking, namely 3S (see Figure 14). The developed activities require students to work on the missions in pairs. Despite collaborative learning, the processes of *self-check (in pairs) and self-debug (in pairs)* can help students take an active role in developing their own understanding – self-directed learning. Once they finish their algorithm for a particular mission, one has to read the codes out loud step by step, and the other has to act accordingly. This is called *self-check*. By doing so, the two students can realise by themselves immediately and they are allowed to modify their algorithm as they think would help correct the actions. This is called *self-debug (in pairs).* Whenever the students can understand the computer science concepts effectively, only these two processes are sufficient for them to complete the mission. However, while a new mission is given, another computer science concept comes along, it is therefore important to have the final process which is *scaffolding* in which capable facilitators are required. The facilitators would help guide students to think from multiple perspectives so that they can spot errors by themselves. On top of that, *scaffolding* can be more helpful when the facilitators can help students think of more effective algorithms that lead to completion of the given mission while containing the smallest number of codes (the shortest algorithm). Reported elsewhere are our findings that instead of using human facilitators, it is also possible to adopt the augmented reality technology to replace this human-based scaffolding part (Threekunprapa & Yasri, In Press). As shown in Figure 14, the 3S approach is in fact not sequential, but rather it is interwoven. One of these can come at any time as the players proceed with the missions.

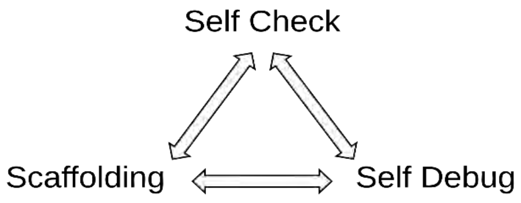


Figure 14. The 3S approach for shifting computational thinking stages

Turning to the three patterns of computational thinking development presented in this study, this study proposes that high school students start their computational thinking from an *initially developed* stage, to *partially developed* and *fully* *developed* stages, respectively. The *initially developed* stage is defined as a lack of ability to apply computer science concepts to design an effective algorithm, and thus the mission cannot be completed. The *partially developed* stage is where students begin to use computer science concepts to solve a particular problem. However, they appear to hold some minor misunderstanding of the concepts and exhibit some forms of premature computational thinking, and thus the mission cannot be perfectly accomplished. Finally, the *fully developed* stage is where computer science concepts are applied correctly and sophisticated computational thinking skills are exercised appropriately. The study shows that the development along these three stages can be assisted by the 3S approach, especially *scaffolding*. On-site observations convey that when appropriate guidance through *scaffolding* is offered, students show a significant move from one stage to another. However, it should be noted that the *scaffolding* process in this study is limited to only useful guidance that helps students think further by themselves. Any forms of direction instruction are strictly prohibited.

While the *initially developed* and *fully developed* stages can be defined with simplicity, much can be learned from diverse responses from students whose algorithms are classified in the group of *partially developed* patterns. First, the issue concerns with *incomplete algorithmic thinking*. As simple as the use of flowcharts, one misconception that is evident is the use of condition with only one out path instead of two. This simple mistake is also found in other studies like Rahimi et al. (2017), Mladenovic et al. (2018), and Žanko et al. (2019). Second, the issue is termed *incomplete pattern recognition*. Evident in mission 2 of this study (Figure 7b and 7c in particular) is the fact that the students fail to recognise repeated patterns, making them have a long algorithm that even though can complete the mission, it does not follow the rule of the mission that limits the number of commands to be used. Third, this issue is called *incomplete decomposition*. It is a subtle failure to digest the main mission into subtasks. This can be seen obviously in mission 3 (Figure 8b), and subtly in mission 5 (Figure 11d). Finally, the issue concerns with *partial abstraction and generalisation* where students can actually complete the mission, however, within limited situations. The designed algorithms cannot be well applied to other similar scenarios. In fact, the skill of *abstraction and generalisation* is essential to develop sophisticated computational thinking as one algorithmic solution can be used in any situations. In this study, *partial abstraction and generalisation* can be seen throughout missions 3, 4 and 5. Obviously, they are not wrong in a particular mission. However, their algorithm is specific to this only. When situations change, for example, the place where the treasure chest is, this algorithm is not applicable.

To reemphasise, this spectrum of CT development stages helps researchers see how students develop their skill from one to another based on a specific content that is of interest. A *fully developed* pattern in one basic mission can be an *initially developed* pattern in the more advanced missions in this study. Therefore, there is no attempt to make any general claim on this. However, this study intends to provide generic definitions which other researchers have to define by themselves in which contexts to be used. It is important to have an expected outcome such as the skills required and the content focused so that it can be much clearer what the border line between each of the development stages is. More importantly, this study highlights that attention should be carefully paid to the *partially develop* patterns as these shows incomplete development of CT skillset which is more difficult to detect, compared to the *initially developed* patterns which heavily rely on the understanding of computer science concepts.

Recommendations for further research

Although this study points out a conceptual pathway (from *initially developed, partially developed to fully developed* stages) where students progress to comprehend the understanding and application of computer science concepts to solve unplugged coding tasks using computational thinking through the aids of the 3S approach (*self-check in pairs, self-debug in pairs, and* scaffolding), it leaves three prominent questions to further explore. First, it is considered important to investigate more deeply on what kind of mistakes and misunderstandings that are more reluctant to correct while developing computational thinking among secondary learners. It remains ambiguous whether such difficulties arise from the complexity of computer science concepts themselves, or from the insufficient ability to decompose the main task to smaller ones, to recognise repetitive patterns, to exercise algorithmic thinking, and/or to ignore unnecessary information in order to generalise a solution that is applicable to other situations.

In addition, although this study points out the possible direction of shifting from *initially developed*, to *partially developed*, and eventually to *fully developed* stages, this existing patter has to be validated by further studies. No certain claims are made that this is sequential. There could potentially exist "jumping patterns" that require others to explore in greater depth. Statistical investigation based on an experimental study can certainly help verify the findings of this current study. A pretest assessing computational thinking and computer science concepts is essential to demonstrate that at which stage in particular that students start from. Then a "checkpoint" test can be administered before moving from one mission to another to verify that each of the students arrives a fully developed stage relevant to that particular mission, in addition assessing from their algorithmic design, which is done in pairs. This current study wishes to initiate research interest in the area by emphasising on the qualitative aspects, and thus such quantitative studies can help strengthen or perhaps disprove this present claim.

Second, it is important to try out the 3S approach in other settings in order to assess its usefulness and establish a standard practice. While *self-check (in pairs) and self-debugging (in pairs)* can be quite straightforward and naturalistic as they are student-led processes, *scaffolding* is perhaps more systematic, and thus requires further investigation. More specifically, it is crucial to come up with a certain guideline for facilitators to consult what should and should not be done while facilitating learners in unplugged activities. In addition, it is interesting to see specific strategies which are effective to deal with certain forms of mistake and misunderstandings. Even more importantly, this scaffolding approach could be automated and digitised (within a digital system of automatics hints or similar) to alleviate the issue of the shortage of the number of teachers as well as to promote self-directed learning.

Finally, this study provides a theoretical background about characteristics of each of the developmental stages of computational thinking. Other researchers may be interested to develop rubrics for classroom purposes. This will be helpful for teachers to do both formative and summative assessments. Also, the research community can be benefited from this development as it allows a more statistically sophisticated study to investigate statistical shifts from one stage to another, determined by a certain significance level. Also, it is believed that the *partially developed* solutions found in this study provide relevant information to help extend the usefulness of the computational thinking test proposed by (Roman-Gonzalez et al., 2017). The *partially developed* responses based on each of the computer science concepts can potentially be developed to design item distractors to capture both misconceptions and premature development of computational thinking. Furthermore, the *fully developed* solutions provide a fantastic opportunity to introduce the concept of "algorithm efficiency" which is a measure of the average execution time necessary for an algorithm to complete work on a set of data as shown in Table 1.

Conclusion

This study explores students' computational thinking patterns that emerge from completing the five missions of unplugged coding activities using flowcharts which incorporate various computer science concepts from basic to advanced ones. A range of patterns have been detected, consisting of *initially developed, partially developed* and *fully developed* stages of computational thinking development. The *initially developed* stage represents algorithms that cannot complete the mission due to misunderstanding of computer science concepts or the inability to apply appropriate computer science concepts to complete the mission. The *partially developed* stage is where players begin to apply the computer science concepts to handle the mission; however, they tend to take a partial advantage of the concept, or take a full advantage but cannot complete the mission. In contrast, the *fully developed* stage is where players can take a full advantage of the concepts and effectively complete the mission, as well as they can exhibit satisfactory characteristics of CT skills.

In addition, this study proposes the 3S approach to help improve students' computational thinking consisting of *self-check (in pairs), self-debug (in pairs), and scaffolding. Self-check (in pairs)* is a process in which a pair of students divide their tasks into two. One reads out the code and the other acts out accordingly. Whenever they learn that the action does not go as expected, then they learn how to do self-debug. After *self-debug (in pairs),* they can proceed with the same process of self-check, until they accomplish the mission. Finally, scaffolding is the process in which a facilitator assists the learning process through minimal guidance and thought-provoking questions. This process has to be done by a trained facilitator who knows how to deliver appropriate support that fosters learning effectively.

Acknowledgements

Much appreciation is given to the scholarship of the 60th Year Supreme Reign of His Majesty King Bhumibol Adulyadej Scholarship from Institute for Innovative Learning, Mahidol University. Also, special thanks are given to Dr. Parames Laosinchi from the Institute for Innovative Learning, Mahidol University, Professor Tim Bell, College of Engineering, University of Canterbury, New Zealand, and Dr. Caitlin Duncan from University of Canterbury, New Zealand for their continual support.

References

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal*, *55*(7), 832–835. https://doi.org/10.1093/comjnl/bxs074

Alaoutinen, S., & Smolander, K. (2010). Student self-assessment in a programming course using bloom's revised taxonomy. In R. Ayfer & J. Impagliazzo (Eds.), *Proceedings in the 15th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 155–159). Association for Computing Machinery. https://doi.org/10.1145/1822090.1822135

Bachu, E., & Bernard, M. (2014). Visualizing problem solving in a strategy game for teaching programming. In H. R. Arabnia, A. Bahrami, L. Deligiannidis, G. Jandieri, A. M. G. Solo & F. G. Tinetti (Eds.), *Proceedings in the 2014 International Conference on Frontiers in Education: Computer Science and Computer Engineering* (pp. 1–7). CSREA Press.

Bourdeau, D., Roberts, D., Wood, B., & Korioth, J. (2017). A study of video-mediated opportunities for self-directed learning in required core curriculum. *International Journal of Educational Methodology*, *3*(2), 85–91. https://doi.org/10.12973/ijem.3.2.85

Brackmann, C. P., Roman-gonzalez, M., Robles, G., Moreno-leon, J., Casali, A., Barone, D., Federal, I., Iffar, F., Brackmann, C. P., Roman-gonzalez, M., Rey, U., Carlos, J., Rey, U., Carlos, J., Moreno-leon, J., Casali, A., & Barone, D. (2017). Development of computational thinking skills through unplugged activities in primary school. In M. Knobelsdorf & R. Romeike (Eds.), *Proceedings in the 12th Workshop on Primary and Secondary Computing Education* (pp. 65–72). Association for Computing Machinery. https://doi.org/10.1145/3137065.3137069

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In A. F. Ball (Ed.), *Proceedings in the 2012 Annual Meeting of the American Educational Research Association* (pp. 25–50). American Educational Research Association. http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf

Brusilovsky, P., & Sosnovsky, S. (2005). Individualized exercises for self-assessment of programming knowledge: an evaluation of quizpack. *Journal on Educational Resources in Computing*, *5*(3), 1–22. https://doi.org/10.1145/1163405.1163411

Bulus Kirikkaya, E., & Basaran, B. (2019). Investigation of the effect of the integration of arduino to electrical experiments on students' attitudes towards technology and ict by the mixed method. *European Journal of Educational Research*, *8*(1), 31–48. https://doi.org/10.12973/eu-jer.8.1.31

Changtong, N., Maneejak, N., & Yasri, P. (2020). Approaches for implementing stem (science, technology, engineering & mathematics) activities among middle school students in thailand. *International Journal of Educational Methodology*, *6*(1), 185–198. https://doi.org/10.12973/ijem.6.1.185

Ertugrul-Akyol, B. (2019). Development of computational thinking scale : Validity and reliability study. *International Journal of Educational Methodology*, *5*(3), 421–432. https://doi.org/10.12973/ijem.5.3.421

Giordano, D., & Maiorana, F. (2015). Teaching algorithms: visual language vs flowchart vs textual language. In O. Kaynak, M. E. Auer & M. Llamas (Eds.), *Proceedings in 2015 IEEE Global Engineering Education Conference (EDUCON)* (pp. 499–504). Institute of Electrical and Electronics Engineers. https://doi.org/10.1109/EDUCON.2015.7096016

Grover, S., & Pea, R. (2013). Computational thinking in k-12: a review of the state of the field. *Educational Researcher*, *42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Gunbatar, M. S., & Karalar, H. (2018). Gender differences in middle school students' attitudes and self-efficacy perceptions towards mblock programming. *European Journal of Educational Research*, *7*(4), 925. https://doi.org/10.12973/eu-jer.7.4.923

Hooshyar, D., Ahmad, R. B., Yousefi, M., Fathi, M., Horng, S.-J., & Lim, H. (2016). Applying an online game-based formative assessment in a flowchart-based intelligent tutoring system for improving problem-solving skills. *Computers & Education*, *94*, 18–36. https://doi.org/10.1016/j.compedu.2015.10.013

Hsu, T. C., Chang, S. C., & Hung, Y. T. (2018). How to learn and how to teach computational thinking: suggestions based on a review of the literature. *Computers & Education*, *126*, 296–310. https://doi.org/10.1016/j.compedu.2018.07.004

Hubwieser, P., Giannakos, M., Berges, M., Brinda, T., Diethelm, I., Magenheim, J., Pal, Y., Jackova, J., & Jasute, E. (2015). A global snapshot of computer science education in k-12 schools. In N. Ragonis & P. Kinnunen (Eds.), *Proceedings in the 2015 ITiCSE on Working Group Reports* (pp. 65–83). Association for Computing Machinery. https://doi.org/10.1145/2858796.2858799

Kert, S. B. (2019). A proposal of in-service teacher training approach for computer science teachers. *European Journal of Educational Research*, *8*(2), 477–489. https://doi.org/10.12973/eu-jer.8.2.477

Kim, N. J., Belland, B. R., & Walker, A. E. (2018). Effectiveness of computer-based scaffolding in the context of problem-based learning for stem education: bayesian meta-analysis. *Educational Psychology Review*, *30*(2), 397–429. https://doi.org/10.1007/s10648-017-9419-1

Linder, S. P., Abbott, D., & Fromberger, M. J. (2006). An instructional scaffolding approach to teaching software design. *Journal of Computing Sciences in Colleges*, *21*(6), 238–250. https://doi.org/10.5555/1127442.1127472

Lye, S. Y., Koh, J. H. L., Yee Lye, S., & Hwee Ling Koh, J. (2014). Review on teaching and learning of computational thinking through programming: What is next for k-12? *Computers in Human Behavior*, *41*, 51–61. https://doi.org/10.1016/j.chb.2014.09.012

Mladenovic, M., Boljat, I., & Zanko, Z. (2018). Comparing loops misconceptions in block-based and text-based programming languages at the k-12 level. *Education and Information Technologies*, *23*(4), 1483–1500. https://doi.org/10.1007/s10639-017-9673-3

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. https://doi.org/10.1007/s40692-018-0101-5

Pauline-Graf, D., & Mandel, S. E. (2019). Defining preliminary research for digital game-based learning evaluation: Best practices. *International Journal of Educational Methodology*, *5*(4), 623–635. https://doi.org/10.12973/ijem.5.4.623

Pisanpanumas, P., & Yasri, P. (2018). SOLO taxonomy: increased complexity of conceptual understanding about the interconnection between convection and natural disasters using hands-on activities. *SSRN Electronic Journal*, *7*(2), 91–103. https://doi.org/10.2139/ssrn.3262589

Rahimi, E., Barendsen, E., & Henze, I. (2017). Identifying students' misconceptions on basic algorithmic concepts through flowchart analysis. In V. Dagien & A. Hellas (Eds.), *Informatics in Schools: Focus on Learning Programming* (pp. 155–168). Springer International Publishing.

Roman-Gonzalez, M., Perez-Gonzalez, J.-C., & Jimenez-Fernandez, C. (2017). Which cognitive abilities underlie computational thinking? criterion validity of the computational thinking test. *Computers in Human Behavior*, *72*, 678–691. https://doi.org/10.1016/j.chb.2016.08.047

Scanlan, D. A. (1989). Structured flowcharts outperform pseudocode: an experimental comparison. *IEEE Software*, *6*(5), 28–36. https://doi.org/10.1109/52.35587

Setiawan, D. W., Suharno, & Triyanto. (2019). The influence of active learning on the concept of mastery of sains learning by fifth grade students at primary school. *International Journal of Educational Methodology*, *5*(1), 177–181. https://doi.org/10.12973/ijem.5.1.189

Sharma, R., Jain, A., Gupta, N., Garg, S., Batta, M., & Dhir, S. K. (2016). Impact of self-assessment by students on their learning. *International Journal of Applied & Basic Medical Research*, *6*(3), 226–229. https://doi.org/10.4103/2229-516X.186961

Shute, V. J., Sun, C., & Asbell-clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142–158. https://doi.org/10.1016/j.edurev.2017.09.003

Sondakh, D. E., Osman, K., & Zainudin, S. (2020). A proposal for holistic assessment of computational thinking for undergraduate: content validity. *European Journal of Educational Research*, *9*(1), 33–50. https://doi.org/10.12973/eu-jer.9.1.33

Threekunprapa, A., & Yasri, P. (n.d.). The role of augmented reality based unplugged computer programming approach in the effectiveness of computational thinking. *International Journal of Mobile Learning and Organisation*.

Threekunprapa, A., & Yasri, P. (2020). Unplugged coding using flowblocks for promoting computational thinking and programming among secondary school students. *International Journal of Instruction*, *13*(3). Advance online publication. https://doi.org/10.29333/

Toheri, Winarso, W., & Haqq, A. A. (2020). Where exactly for enhance critical and creative thinking: the use of problem posing or contextual learning. *European Journal of Educational Research*, *9*(2), 877–887. https://doi.org/10.12973/eu-jer.9.2.877

Van De Pol, J., Volman, M., & Beishuizen, J. (2010). Scaffolding in teacher–student interaction: a decade of research. *Educational Psychology Review*, *22*, 271–296. https://doi.org/10.1007/s10648-010-9127-6

Westphal, B. T., Harris, F. C., & Fadali, M. S. (2003). Graphical programming: a vehicle for teaching computer problem solving. In E. Innovations (Ed.), *Proceedings in 33rd Annual Frontiers in Education (FIE2003)* (pp. 19–23). Stipes Publishing L.L.C. https://doi.org/10.1109/FIE.2003.1264759

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35. https://doi.org/10.1145/1118178.1118215

Yuksel, H. (2019). Experiences of prospective physical education teachers on active gaming within the context of school-based physical activity. *European Journal of Educational Research*, *8*(1), 199–211. https://doi.org/10.12973/eu-jer.8.1.199

Zanko, Z., Mladenovic, M., & Boljat, I. (2019). Misconceptions about variables at the k-12 level. *Education and Information Technologies*, *24*(2), 1251–1268. https://doi.org/10.1007/s10639-018-9824-1

Zvarych, I., Kalaur, S., Prymachenko, N. M., Romashchenko, I. V., & Romanyshyna, O. I. (2019). Gamification as a tool for stimulating the educational activity of students of higher educational institutions of ukraine and the united states. *European Journal of Educational Research*, *8*(2), 875–891. https://doi.org/10.12973/eu-jer.8.3.875