International Journal of Instruction *July 2020 ● Vol.13, No.3*
e-ISSN: 1308-1470 ● www.e-iji.net *p-ISSN: 1694-609X*
pp. 207-222

Received: 20/09/2019
Revision: 12/01/2020
Accepted: 17/01/2020
OnlineFirst:12/04/2020

Unplugged Coding Using Flowblocks for Promoting Computational Thinking and Programming among Secondary School Students

Arinchaya Threekunprapa
Ph.D. candidate, Institute for Innovative Learning, Mahidol University, Thailand, *arinchaya.thr@gmail.com*

Pratchayapong Yasri
Dr., Institute for Innovative Learning, Mahidol University, Thailand, *pratchayapong.yas@mahidol.edu*

Computational thinking (CT) has become a necessary skill of students in the 21st century. Various learning approaches have been developed to foster CT among school students. However, these approaches predominantly rely on computer devices and internet connection and fail to promote advanced computer concepts necessary for programming. Therefore, this study developed an unplugged coding activity using flowblocks, the term is coined to represent modified Blockly based on flowcharts with user-friendly syntaxes, as a visual and programming tool, delivered in the form of game-based learning. The activity included a series of game missions to develop five programming concepts. The unplugged coding activity was implemented based on a pre and post intervention design with 160 secondary students who had no prior experience about programming. Statistical analyses showed that students' conceptual understanding of coding and CT increased significantly after participation. In addition, the perceptions of their ability to learn programming, namely self-efficacy, statistically grew in the posttest. It is therefore recommended for school teachers teaching basic programming and CT to consider using this offline, engaging and cost-effective approach as an alternative to computer-based methods of programming.

Keywords: computational thinking, unplugged coding, programming, flowcharts, flowblocks, self-efficacy

INTRODUCTION

Computational thinking is considered to be an essential skill of 21st century learners. Not only is it important for learning computer science conceptions, but also for solving problems on a daily basis. It is generally defined as a thought process involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by computational steps and algorithms (Wing, 2006).

Citation: 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), 207-222. https://doi.org/10.29333/iji.2020.13314a

Educational organisations have implemented learning activities to promote computational thinking among students. One method intensively used for this is the study of computer programming which can be done by either text-based programming or visual programming language (VPL). However, limitations exist. First and foremost, they rely on computer devices and internet connection which may be out of reach by schools in less privileged areas. Second, text-based programming is rather passive, leading to a lack of motivation and unsatisfactory learning achievement. Furthermore, VPL in its currently available form fails to incorporate advanced computer science concepts such as inputs, variables, loops and loops with conditions which are of importance for learning about computer programming.

Therefore, this study develops an unplugged coding using flowblocks activity to fill the gaps mentioned. The term flowblock used throughout this paper is defined to represent modified Blockly in the way that makes the shape of visual blocks more conceptually accurate according to the concept of flowchart with user-friendly syntaxes. Its usage requires no digital devices and can be completely used in a game-based learning environment. In addition, the use of flowblocks that consist of diagrammatic shapes of flowcharts and syntaxes lies at the heart of this development so that basic to more advanced programming concepts can be cultivated. In addition, it aims to improve school students' interest in applying computational thinking to daily life rather than to make them future programmers, its main mission is emphasise on algorithms rather syntaxes and structures of computer programming. The effectiveness of this activity is assessed by a conceptual test of programming concepts and a survey of self-efficacy.

RELEVANT RESEARCH

Computational Thinking: A Crucial Skill of the Current Century

Computational thinking (CT) has gained its popularity within the research community (Lye et al., 2014) with over 1000 research articles related to it published in SCOPUS journals during 2006 and 2017 (Hsu et al., 2018). In the educational arena, education ministries in various countries have incorporated computer science in their national curricula including USA, UK, New Zealand, Germany, India, Georgia, France, Korea, Japan, Sweden, Finland, Israel, Russia, and Italy (Hubwieser et al., 2015). The most recent one is Thailand where the subject has been integrated since 2018 (Ministry of Education, 2018).

Computer Programming: A Way to Develop Computational Thinking

Although there exist different ways in which CT can be developed among school learners, two main approaches are predominantly used: text-based programming and visual language programming (Hsu et al., 2018; Lye et al., 2014).

Traditionally, a heavy emphasis rests on text-based programming which relies on the syntax of a programming language. However, this is perceived difficult to understand and master. Therefore, it is not a surprise that many students face difficulties at the beginning of learning, leading to an increase in a dropping rate (Mladenović et al., 2018). In addition, this form of learning is passive, leading to a lower level of intrinsic motivation to learn (Benware & Deci, 1984). Furthermore, research shows that there is no statistical correlation between the number of hours that students learned how to run text-based programming and their improved skills in programming (Amoako et al., 2013). This may be due to the fact that time is mostly used for correcting the syntax and structure, instead of focusing on learning algorithm which is more useful to cultivate computational thinking (Ma et al., 2011; Oddie et al., 2010). In addition, this form of programming requires a high cognitive demand (Bati et al., 2014; Papavlasopoulou et al., 2019).

A more recent development is the use of visual programming languages (VPL) or Blockly such as Scratch (Burke, 2012; Lee, 2010) and Alice (Noone & Mooney, 2018; Pellas & Vosinakis, 2018). In this way, learners are not required to remember the syntax and structure, but to choose a block of a graphic syntax to complete a certain mission. Using VPL, coupled with a game-based learning approach, has shown to potentially improve learners' programming skills, motivation to learn, self-efficacy, positive attitude for programming, and reflective thinking skills (Adler & Kim, 2018; Ching et al., 2018; Durak, 2018; Kalelioğlu, 2015; Noone & Mooney, 2018; Pellas & Vosinakis, 2018; Topalli & Cagiltay, 2018). Importantly, it helps K–12 novice programmers focus more on algorithms (Mladenović et al., 2018; Noone & Mooney, 2018). However, misconception can be arisen in relation to the relationship between the shape of blocks and functions as current available programmes contain only one shape for different functions (Mladenović et al., 2018).

One way to make the relationship between shapes and functions more conceptually accurate while maintaining the essence of visualisation is through the use of flowcharts, a diagram that depicts a process, system or computer algorithm. Learners are required to connect different shapes to represent the sequence of coded instructions fed into a computer, enabling it to perform specified logical and arithmetical operations. Research has shown that it is an effective tool to improve computer science conceptions leading to improved CT among learners (Giordano & Maiorana, 2015; Hooshyar et al., 2016; Noone & Mooney, 2018). It is also found to be more effective, more engaging, and less time consuming compared to using pseudocodes (Scanlan, 1989). Last but not least, learning with flowcharts helps learners develop problem-solving skills (Giordano & Maiorana, 2015; Hooshyar et al., 2016; Noone & Mooney, 2018).

Unplugged Coding: Coding without Computers

A pedagogical approach recently developed to engage students to learn about computer science concepts without using digital devices and internet connection is known as unplugged coding (Bell & Vahrenhold, 2018). It is shown to actively engage learners and produce positive attitude towards learning computer science (Nishida et al., 2008). It is considered an effective starting point for setting students ready for further programming as its emphasis is on algorithm which is key to computational thinking and prerequisite to more advanced computer programming (Bell & Vahrenhold, 2018). In addition, as the term implies, it can be done off-line, requiring no computer devices, making computer science more accessible to less privileged schools where digital means and internet connection are out of reach.

Despite their advantages over traditional approaches for learning computer science, currently available unplugged coding activities (Bell et al., 2009, 2012; Bell & Vahrenhold, 2018; Thies & Vahrenhold, 2013) embrace some challenges. First of all, they are primarily designed for primary school students involving only basic computer science concepts (Bell & Vahrenhold, 2018; Thies & Vahrenhold, 2013) which may lack potential to develop more sophisticated computational thinking skills such as problem solving and programming. In addition, although the game-based learning environment has been integrated in various unplugged coding activities, the purpose is mainly for entertainment, while more collaborative learning can in fact be leveraged by this playful setting. In addition, little attempt has been made to convey to effectiveness of unplugged activities in terms of learning achievement, students' perceptions towards their ability to learn about computer programming and computer science, and their perceptions towards unplugged coding activities. Such integrative frameworks would be appropriate evidence for one to be certain about the usefulness and fruitfulness of unplugged computer science.

Game-Based Learning: Environmentally Friendly Classroom

Game-based learning is a form of gameplay with specifically defined learning outcomes (Plass et al., 2015). It is an environment in which games are used to enhance knowledge and skills, and where game activities involve problem-solving spaces and challenges that provide players/learners with a sense of achievement (Qian & Clark, 2016). Attempts have been made to use game-based learning as an approach of teaching programming among students in various age groups. Research has shown that this learning environment can help promote students' motivation to learn, self-confidence and efficacy, as well as positive attitudes towards learning computer science in general and programming in particular (Adler & Kim, 2018; Ching et al., 2018; Kalelioğlu, 2015). Furthermore, recent research has revealed that it is an effective learning strategy to teach complex computational skills (Czerkawski & Lyman, 2015).

Self-Efficacy: A Psychological Framework of Learning

Self-efficacy is a social-cognitive theory that explains self-confidence of ability to perform a certain task and how it influences what one does (Bandura, 1977). There are four sources of efficacy expectations: mastery experience (doing), vicarious experience (seeing), verbal persuasion (hearing) and psychological states (feeling). Self-efficacy has been an important research area in education. In education, both learners' and teachers' perceived self-efficacy levels have a direct impact on the effectiveness of instruction (Kadirhan et al., 2018). It is also used as a theoretical framework for conveying the effectiveness of another learning innovation for teaching C programming with a mobile game-based environment among university students (Daungcharone et al., 2019). Also, it is found to increase when students are exposed to an environment of game-based learning (Tapingkae et al., 2018). Of course, the purpose of learning is not only for gaining better conceptual understanding, but also the level of confidence; thus this framework is chosen as a psychological len for assessing the effectiveness of learning innovation.

METHOD

Unplugged Coding Using Flowblocks Activity

The principle underlying the design of this unplugged coding with flowblocks activity is that it has to be a paper-based game that has an objective to deliver computer science concepts (including sequence, repetition, input and variable, condition and loop). To make the game more vivid, a scenario is important to allow players to think of the situation and solve the problem. It has to involve a number of missions representing various computer science concepts. As players proceed, they have to be challenged by the mission which has more advanced computer science concepts. It is believed that while completing each mission, players should be allowed to check the correctness of their syntaxes by themselves so that self-directed learning can be promoted.

Based on the aforementioned, an activity is developed called Treasure Hunter. The main mission is to find a treasure chest, walking grid by grid from an assigned starting point. Two players are paired up on a voluntary basis to complete each mission. In order to move to the treasure chest in their own designed direction, the players have to connect flowblocks provided which include a set of ready-to-use syntaxes consisting of a start, an end, and the number of repetitions, as well as a set of shapes and individual syntaxes for players to connect by themselves. In this latter set, different shapes convey different meanings. A parallelogram represents input/output. A rectangle represents a process. A diamond shape represents a decision. These flowblocks are used together with a list of actions of movements as well as questions (see Figure 1).

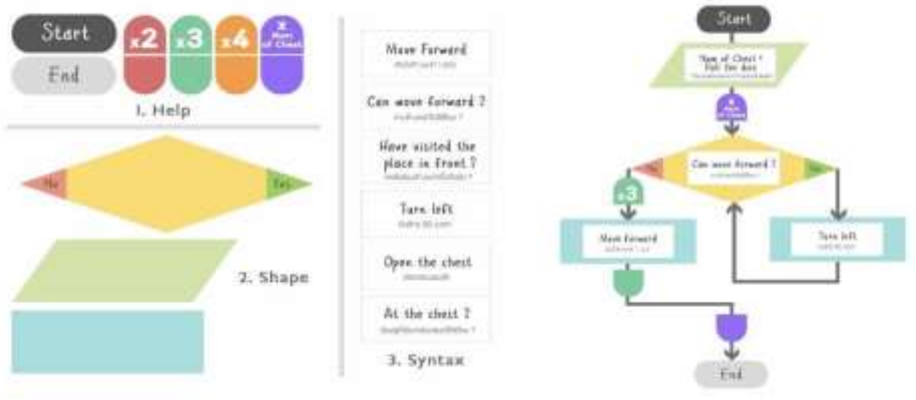
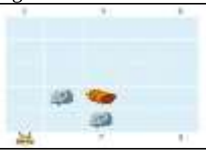
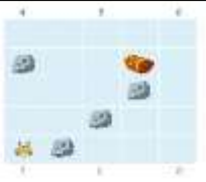
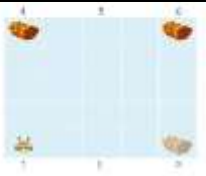
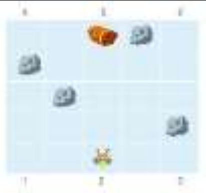
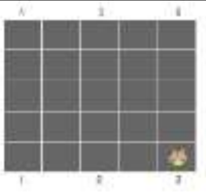


Figure 1
Flowblock Components and How to Connect Them

Therefore, to use these flowblocks in the missions, the players have to match a desired syntax with a correct shape. Once they finish their arranged flowblocks. They have to do a self-check where one group member reads out the syntax one by one, whereas the other acts accordingly. If the process of self-check informs some errors, they have to debug. Once they reach the position where the treasure chest is placed, they have to open it and get the number of diamonds in the chest determined by the number of a 6-face dice being rolled.

It is important to note that each mission has a different starting place, the number of chests, the position of the chest, a variety of bonus points, different flowblock syntaxes with different commands. Each pair of players have to move towards the treasure chest in their designated direction from the assigned starting point. Barriers (rocks) are put there for them to avoid. In this game, there are five missions in total (summarised in Table 1). Each mission aims to promote at least one fundamental concept of computer programming. As players proceed to the next mission, they have to apply the concept that they previously learned. Therefore, to complete a mission, they have to use the previously learned concepts as well as a recently explored concept. The concepts focused in this game consist of sequence, repetition, input and variable, condition, loop, as well as loop with condition.

Table 1
Mission Description

	Aim Concept	Stage	Short Description	Syntax provided
1	Sequence: A basic algorithm that allows actions to be carried out in order and step by step to complete a certain task.		Let's get the treasure chest and avoid the rocks.	Move Forward x 6 Turn Left x 4 Turn Right x 4 Open the chest
2	Repetition: A function to repeat an action which helps shorten the length of coding, representing a more effective way to solve a problem.		Let's get the treasure chest and avoid the rocks through the shortest route	Move Forward x 2 Turn Left x 2 Turn Right x 2 Open the chest Repeat
3	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.		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 Num of Chest = Roll the dice Repeat
4	Condition: 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.		Let's get a treasure chest, move while can move forward	Move Forward x 2 Turn Left x 2 Turn Right x 2 Open the chest Can move forward?
5	Loop with condition: To repeat an action by the yes and no condition. For example, while loop is to execute an action while a set condition is true.		The journey turns dark and the hunters have to visit every single place (grid) to look for the treasure.	Move Forward Turn Left Turn Right Can move forward? Have visited the place in front? Have visited all the places?

Data Collection and Analysis

This study adopts a quantitative methodology using a pre-post intervention research design. Participants in this study were 160 secondary school students recruited by convenience sampling who had no experience in programming. The process of data collection took approximately 3 hours. It started with a pretest in which each of the participants voluntarily took for 15 minutes. After that, they were introduced to the activity in which the participants were paired up and completed all the missions within 2.5 hours. In the meantime, the participants were assisted in the learning process through scaffolding by facilitators. Once they accomplished the given tasks, they were asked to complete a posttest which took up to 15 minutes. Ethical considerations are taken into consideration in this study with respect to the right and safety of participant both physically and psychologically.

Classroom materials include a set of flowblock components, five mission cards, a dice and a character representing a treasure hunter. All of these were given all at once since start. Therefore, each pair spent different time per mission as they went through. Two research tools were used as the pretest and posttest. First, a computational thinking test is to assess skills to use computer science concepts to solve problems which contains 3 items, each of which is worth 5 points (15 in total). CT1 assess the concept of sequence, while CT2 and CT3 assess repetition, inputs as well as variables, and conditions as well as loops the conditions, respectively. Second, the self-efficacy questionnaire, containing 12 statements describing the four main sources of self-efficacy and beliefs (3 statements for each), assessed by a 5-Likert scale. Statistical analyses were used to examine the difference between the mean scores.

FINDINGS

Computational Thinking Test

There was a statistically significant increase in the posttest mean score (11.23) compared to the pretest mean score (1.98), based on a Wilcoxon-signed rank test of the total score (Z = -10.875, p = 0.000). The same test is done for specific items (CT1, CT2 and CT3) which also shows a statistically significant increase in the posttest mean score. Specifically, the mean score of CT1 statistically increased from 1.406 to 4.344 (Z = -9.567, p = 0.000), which is the highest learning gain. The mean score of CT2 statistically increased from 0.519 to 3.800 (Z = -10.203, p = 0.000). Likewise, the mean score of CT3 statistically rose from zero to 3.081 (Z = -10.012, p = 0.000).

Table 2
Computational Thinking Test: Pretest and Post-test Mean Scores

	Pretest				Posttest			
	Mean	N	SD	SE	Mean	N	SD	SE
CT1	1.406	160	2.046	0.162	4.344	160	1.334	0.106
CT2	0.519	160	1.197	0.095	3.800	160	1.528	0.121
CT3	0.056	160	0.257	0.020	3.081	160	1.785	0.141
Total	1.981	160	2.608	0.206	11.230	160	3.209	0.254

Self-Efficacy Test

There was a statistically significant increase in the posttest mean score (14.79 out of 20) compared to the pretest mean score (13.45 out of 20), based on a Wilcoxon-signed rank test of the summation of self-efficacy (Z = -6.902, p = 0.000). Inferential statistics were performed to assess a significant difference in each of the four constructs which showed a statistically significant increase in the mean scores of mastery experience (Z = -6.009, p = 0.000), vicarious experience (Z = -5.076, p = 0.000), as well as psychological state (Z = -6.086, p = 0.000). However, this statistical difference in mean scores between the pretest and the posttest was not present in verbal persuasion, despite the fact that the mean score of verbal persuasion was the highest among the other four aspects in both tests.

Table 3
Self-Efficacy Test

Mean	Pretest	Posttest
Mastery Experience	3.24	3.81
Vicarious Experience	3.49	3.76
Verbal Persuasion	3.76	3.83
Psychological State	2.96	3.39
Total Self-Efficacy	13.45	14.79

DISCUSSION

This study integrates the usefulness of visual programming language (VPL) as well as flowcharts to make an unplugged coding activity more conceptually appropriate. In its current form, VPL such as Blockly and Scratch, does not emphasise on the meaning of block shapes, while this is a focus on flowchart in which different shapes signify different algorithmic tasks. Therefore, we bring in the user-friendly nature of VPL and integrate it with the concepts of flowchart, and coin a new term as flowblocks. In addition, this study incorporates human friendly language to make syntaxes accessible to younger age groups of learners and novice learners of programming. On top of this, this study adopts game-based learning into the activity in order to engage students in the learning process and make the activity more engaging and challenging.

It is believed that a careful design of the unplugged coding with flowblocks activity that adopts diagrammatic representation of flowcharts, the five computer science concepts integrated missions, the repetition of particular concepts throughout the missions, and the self-check process, together contributes to the improvement of students' computational thinking score. To be more specific, in the pretest as shown in Table 4, this chosen example which in fact represents the majority exhibits that the student was not aware of using flowcharts, but rather used a form of pseudocodes to complete the pretest. In addition, although the sense of sequence can be slightly detectable, it is not fully developed. However, through the aid of the unplugged coding using flowblocks activity, the student learned how the syntax is supposed to start and how the following syntaxes should be connected using provided flowblocks in a correct sequence (see Appendix for selected flowblocks from students). Before getting the mission completed, many students had to go through a series of debugs which, of course, help them learn and grow a sense of computational thinking as they proceeded. Therefore, the student could respond to the posttest correctly with accurate understanding of sequences. Since this fundamental concept can be easily comprehended and is repeatedly used in every single task of the further missions, the mean score of CT 1 (assess the concept of sequence) reached the highest compared to the other two tests, undoubtedly.

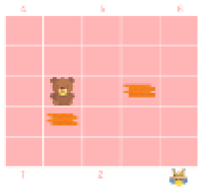
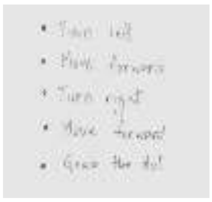
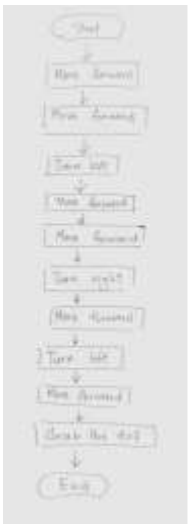
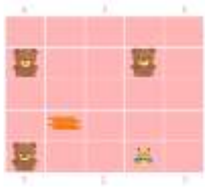
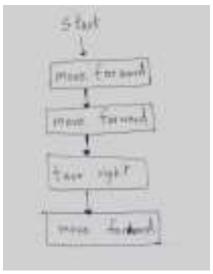
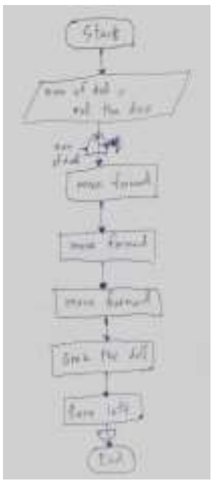
A similar scenario is found in the test result of CT 2 which assesses students' concepts of repetition, input and variable. A student whose response is representative is chosen who appears to have some understanding of the shape of flowcharts. However, the sense of using repetition, input and variable is absent (see also Table 4). Missions 2 and 3 in the developed activity are designed to cultivate such concepts in which students can learn more complex and advanced algorithm. Many had to debug their coding repeatedly with appropriate scaffolding. Assumingly, the fact that students repeatedly took the input from an indicated number by rolling a dice and store it in the variable and use that number for repetition helps them to understand more deeply about these concepts. Because of this, it is believed that students then were able to complete the posttest successfully. However, due to the complication of these concepts and a lack of repeated actions in this current level, the mean score of CT 2 is slightly lower than that of CT 1 which is not surprising.

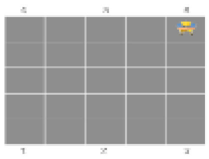
Improved understanding is evidently presented in the result of CT 3 which assesses students' concepts of loops and loops with condition. A representative sample is chosen here who did not provide an answer to the pretest. A process of reaffirmation is done to ensure that this response is not due to the limited time, but merely a lack of background knowledge. However, once participating in missions 4 and 5, the student started to learn about loops and loops with condition, alongside the repeated use of the previously learned concepts of sequence, repetition, input, and variable. With a series of debugs and scaffolding, the participant completed these final two missions successfully. However, it is important to admit that the time allocated for these two missions, although the longest, is not adequate to ensure that all students fully comprehended these most advanced computer science concepts chosen to teach in this unplugged activity. Having said that, a number of students could gain a fair level of understanding as their mean score of the whole sample is around 3 out of 5 which is considered a very good starting point for the first exposure to such concepts, not to mention the improvement of understanding compared to the pretest.

The aforementioned results are in line with several studies adopting unplugged coding to cultivate students' learning basic concepts of computer programming (Bell & Vahrenhold, 2018). However, this study offers a new set of evidence that unplugged coding using flowblocks can be used to teach advanced concepts such as input, variable, loops and loops with condition (missions 3, 4 and 5) which are found missing in early studies that mainly focus on sequence and repetition. In addition, the use of a series of debugs and self-compiling by students themselves with some scaffolding is novel in this study. Last but not least, this is the first unplugged coding activity using game-based learning to layout the missions in order, each of which advances understanding.

Despite showing a variety of advantages, there are areas of improvement to be done in further study. First, extra time with multiple exposures are required for school students to fully develop conceptual understanding of loops with conditions. There may be an additional mission for this concept alone to be exercised. Furthermore, since students have to choose the right shape of flowblocks and syntaxes by themselves, there is a possibility that, by the process of self-check, some may choose the right syntax but a wrong shape, yet they can proceed the mission without realising that this is not accurate. Of course, the facilitator can help observe and minimise this potential problems. However, when a group of students is larger, this could be problematic. It is suggested here that if the use of mobile devices and internet connection are not limited, a development of an AR application to validate the correctness of shape and syntax may be crucial. Some may be interested to just produce videos or even worksheets for students to consult when needed.

Table 4
Student Responses and Questions used in CT 1, CT 2 and CT 3

CT 1 Question & Pretest	CT 1 Posttest	CT 2 Question & Pretest	CT 2 Posttest
 CT 1 Question  CT 1 Pretest answer	 CT 1 Posttest answer	 CT 2 Question  CT 2 Pretest answer	 CT 2 Posttest answer

CT 3 Question	CT 3 Posttest
	

Turning to the increased self-efficacy, it is believed that the game-based environment where students work in pairs to self-check so as to complete the missions is key to this.

Hermans and Aivaloglou (2017) whose study focuses on using a 4-week unplugged lesson to introduce basic concepts of computer science including loops, conditions and variables to elementary students, without applying these concepts to coding, also result reveals that after the implementation of the lesson the level of self-efficacy of student participants increased statistically in comparison to their counterparts (those learning through a traditional way of learning computer science concepts using computer devices). To be more specific, theoretical framework by Bandura (1977) explains that there are four major sources that contribute to changes in levels of self-efficacy, composing of mastery experience, vicarious experience, verbal persuasion and psychological state which are then discussed in turn.

First and foremost, before proceeding to the next mission, students in this study have to self-check their algorithm. When errors exist, they have to debug their algorithm until they complete the mission (successfully reach and open the treasure box). Throughout this practice which demands a great deal of trial-and-error effort, students are able to develop mastery experience as shown in the statistical increase in the score given to this aspect of self-efficacy in the posttest against the pretest. A study conducted by Kudo & Mori (2015) reveals that mastery experience that comes with the completion of a task is key to enhance self-efficacy. The study shows the statistically greatest learning gains among students who have direct and successful experience, compared to those without direct experience and those with direct experience but without success.

Furthermore, working in pairs helps students see how the partner is doing and hear what he or she is talking about simultaneously. Also, the setting of this activity is an entire classroom where each pair of students can also see and hear what other pairs are doing. Therefore, being able to see what others are achieving is believed to be a factor that allows students to gain vicarious experience. Likewise, being able to hear what others are completing could promote verbal persuasion. It can be repeatedly observed in the class that whenever one pair can complete a mission, a voice of success can be joyfully heard which makes other pairs feel more motivated and become more confident that they could also do the same. This somewhat explains why the level of vicarious experience and verbal persuasion increased statistically in the posttest.

However, it is important to note that verbal persuasion also plays a great role as encouragement when missions get harder and students encounter a number of failures. It is evident that each pair always encourage each other when they feel that they are behind the other pairs. Also, whenever a pair is left behind, the presence of a facilitator is found to be encouraging when words of encouragement and proper guidance are given appropriately. This is something that instructors and facilitators have to be concerned when a setting where teaching is conducted is a naturalistic one. It is unavoidable that students compare themselves to others. There are times when comparison puts them down. Therefore, instructors and facilitators are to offer constructive verbal persuasion that helps students proceed their tasks more effectively. It is shown that verbal persuasion also has negative effects on individuals' efficacy beliefs because when delivered inappropriately, it can diminish self-efficacy beliefs of individuals than to enhance them (Kiran & Sungur, 2012). In terms of psychological state, the final source of self-efficacy, it is believed that collaboration within pairs and mild competition among others are a medium that excites students to learn joyfully and actively. This may contribute to the positive increase in the level of motivation displayed in the form of psychological state in the posttest in this study. This result is also in line with other studies where the adoption of unplugged coding activities can help promote psychological state and motivation to learn of learners (Daungcharone et al., 2019; Dorji et al., 2015), especially secondary school students (Nishida et al., 2008).

CONCLUSION

Computational thinking has become a topic of interest among educators recently. Learning approaches have been developed to assist learners to cultivate the skill and put it into practice. So far, two main approaches have been used. One is text-based programming which is a traditional way to type various characters from a syntax. However, it is rather passive and inaccessible to general leaners. The other is visual programming languages which is the use of drag and drop provided blocks to construct an algorithm. However, it misses some computational concepts such as the meaning of block shapes. Therefore, this study aims to overcome the challenges by developing an unplugged coding with flowblocks activity that is adopted in an environment of game-based learning. This activity consists of a set of missions that intend to cultivate concepts related to computer programming and computational thinking with no aids of computer devices. Adopting a quantitative methodology, data collection is done among secondary school students who have no prior knowledge about computer programming based on a pre-post intervention approach. A set of flowblock components are made ready for them to proceed with five missions that incorporate different computer science concepts, with the aid of facilitators who help scaffold. Data analysis shows that by participating in the learning innovation, students exhibit statistically improved learning achievement and significantly increased self-efficacy. It is therefore recommended for other instructors teaching basic concepts in computational science to consider using this form of unplugged coding with flowblocks activity to promote conceptual understanding and emotional engagement among their own groups of learners. Also, it is suggested to other researchers that this strand of research is emerging, yet knowledge about it is far from saturation. Aspects related to the development of unplugged coding are to be uncovered.

ACKNOWLEDGEMENT

This work has been supported by the 60th Year Supreme Reign of His Majesty King Bhumibol Adulyadej Scholarship from Institute for Innovative Learning, Mahidol University. Furthermore, special thanks are given to Dr.Parames Laosinchi from the Institute for Innovative learning, Mahidol University for his contribution on my work regarding logic in the missions and statistical inference.

REFERENCES

Adler, R. F., & Kim, H. (2018). Enhancing future k-8 teachers' computational thinking skills through modeling and simulations. *Education and Information Technologies, 23*(4), 1501–1514.

Amoako, P. Y. O., Adu-Manu, K., Arthur, J., & Adjetey, C. (2013). Performance of students in computer programming: Background, field of study and learning approach paradigm. *International Journal of Computer Applications*, *77*, 17–21.

Bandura, A. (1977). Self-efficacy: Toward a unifying theory of behavioral change. *Advances in Behaviour Research and Therapy*, *1*(4), 139–161.

Bati, T. B., Gelderblom, H., & van Biljon, J. (2014). A blended learning approach for teaching computer programming: Design for large classes in sub-saharan africa. *Computer Science Education*, *24*(1), 71–99.

Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer science unplugged: School students doing real computing without computers. *New Zealand Journal of Applied Computing and Information Technology*, *13*(1), 20–29.

Bell, T., Rosamond, F., & Casey, N. (2012). Computer science unplugged and related projects in math and computer science popularization. In *The Multivariate Algorithmic Revolution and Beyond* (pp. 398–456). Springer, Berlin, Heidelberg.

Bell, T., & Vahrenhold, J. (2018). *CS unplugged—how is it used, and does it work?* Springer, Cham.

Benware, C. A., & Deci, E. L. (1984). Quality of learning with an active versus passive motivational set. *American Educational Research Journal*, *21*(4), 755–765.

Burke, Q. (2012). The markings of a new pencil: Introducing programming-as-writing in the middle school classroom. *Journal of Media Literacy Education*, *4*(2), 121–135.

Ching, Y.-H., Hsu, Y.-C., & Baldwin, S. (2018). Developing computational thinking with educational technologies for young learners. *TechTrends*, *62*(6), 563–573.

Czerkawski, B. C., & Lyman, E. W. (2015). Exploring issues about computational thinking in higher education. *TechTrends*, *59*(2), 57–65.

Daungcharone, K., Panjaburee, P., & Thongkoo, K. (2019). A mobile game-based c programming language learning: Results of university students' achievement and motivations. *International J of Mobile Learning and Organisation*, *13*(2), 171–192.

Dorji, U., Panjaburee, P., & Srisawasdi, N. (2015). A learning cycle approach to developing educational computer game for improving students' learning and awareness in electric energy consumption and conservation. *Educational Technology & Society*, *18*(1), 91–105.

Durak, H. Y. (2018). The effects of using different tools in programming teaching of secondary school students on engagement, computational thinking and reflective thinking skills for problem solving. *Technology, Knowledge and Learning*. https://doi.org/10.1007/s10758-018-9391-y

Giordano, D., & Maiorana, F. (2015). Teaching algorithms: Visual language vs flowchart vs textual language. *2015 IEEE Global Engineering Education Conference (EDUCON)*, 499–504.

Hermans, F., & Aivaloglou, E. (2017). To scratch or not to scratch?: A controlled experiment comparing plugged first and unplugged first programming lessons. *Proceedings of the 12th Workshop on Primary and Secondary Computing Education - WiPSCE '17*, 49–56.

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 & Edu*, *94*, 18–36.

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.

Hubwieser, P., Giannakos, M., Berges, M., Brinda, T., Diethelm, I., Magenheim, J., … Jasute, E. (2015). A global snapshot of computer science education in k-12 schools. *ITICSE-WGR '15 Proceedings of the 2015 ITiCSE on Working Group Reports*, 65–83.

Kadirhan, Z., Gül, A., & Battal, A. (2018). Self-efficacy to teach coding in k-12 education. In C. B. Hodges (Ed.), *Self-Efficacy in instructional technology contexts* (pp. 205–226). Cham: Springer International Publishing.

Kalelioğlu, F. (2015). A new way of teaching programming skills to k-12 students: Code.org. *Computers in Human Behavior*, *52*, 200–210.

Kiran, D., & Sungur, S. (2012). Middle school students' science self-efficacy and its sources: Examination of gender difference. *Journal of Science Education and Technology*, *21*(5), 619–630.

Kudo, H., & Mori, K. (2015). A preliminary study of increasing self-efficacy in junior high school students: Induced success and a vicarious experience. *Psychological Reports*, *117*(2), 631–642.

Lee, Y.-J. (2010). Developing computer programming concepts and skills via technology-enriched language-art projects: A case study. *Journal of Educational Multimedia and Hypermedia*, *19*(3), 307–326.

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.

Ma, L., Ferguson, J., Roper, M., & Wood, M. (2011). Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education*, *21*(1), 57–80.

Ministry of Education. (2018). *Computing science teacher guide*. The Institute for the Promotion of Teaching Science and Technology.

Mladenović, M., Boljat, I., & Žanko, Ž. (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.

Nishida, T., Idosaka, Y., Hofuku, Y., Kanemune, S., & Kuno, Y. (2008). New methodology of information education with “computer science unplugged.” In R. T. Mittermeir & M. M. Sysło (Eds.), *Informatics education - supporting computational thinking*: *Vol. 5090 LNCS* (pp. 241–252). Berlin, Heidelberg: Springer Berlin Heidelberg.

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.

Oddie, A., Hazlewood, P., Blakeway, S., & Whitfield, A. (2010). Introductory problem solving and programming: Robotics versus traditional approaches. *Innovation in Teaching and Learning in Information and Computer Sciences*, *9*(2), 1–11.

Papavlasopoulou, S., Giannakos, M. N., & Jaccheri, L. (2019). Exploring children’s learning experience in constructionism-based coding activities through design-based research. *Computers in Human Behavior*, *99*, 415–427.

Pellas, N., & Vosinakis, S. (2018). The effect of simulation games on learning computer programming: A comparative study on high school students’ learning performance by assessing computational problem-solving strategies. *Education and Information Technologies*, *23*(6), 1–30.

Plass, J. L., Homer, B. D., & Kinzer, C. K. (2015). Foundations of game-based learning. *Educational Psychologist*, *50*(4), 258–283.

Qian, M., & Clark, K. R. (2016). Game-based learning and 21st century skills: A review of recent research. *Computers in Human Behavior*, *63*, 50–58.

Scanlan, D. A. (1989). Structured flowcharts outperform pseudocode: An experimental comparison. *IEEE Software*, *6*(5), 28–36.

Tapingkae, P., Panjaburee, P., & Srisawasdi, N. (2018). Development of a digital citizenship computer game with a contextual decision-making-oriented approach. *2018 International Symposium on Educational Technology (ISET)*, 230–234.

Thies, R., & Vahrenhold, J. (2013). On plugging unplugged into cs classes. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education - SIGCSE ’13*, 365. New York, New York, USA: ACM Press.

Topalli, D., & Cagiltay, N. (2018). Improving programming skills in engineering education through problem-based game projects with scratch. *Computers & Education*, *120*, 64–74.

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

APPENDIX

Selected flowblocks from students

Mission 1	Mission 2 & 3	Mission 4 & 5