IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

Enhancing confidence in using computational thinking skills via playing a serious game: A case study to increase motivation in learning computer programming

Cagin Kazimoglu

[1]Faculty of Engineering, Cyprus International University, Nicosia, Cyprus

Corresponding author: Cagin Kazimoglu (e-mail: cagin.kazimoglu@gmail.com).

This work greatly appreciates the support of Eur Ing Dr Mary Kiernan, Prof. Liz Bacon, and Prof. Lachlan Mackinnon.

ABSTRACT Computer Science (CS) is a profession that positively impacts every single area of society without which today's world would come to a complete halt. Yet, there is a consensus that CS has serious conundrums particularly in attracting students, low motivation for learning computer programming and developing computational thinking (CT) skills. New motivational methods are needed to take the attention of students and adapt to their learning patterns as how people learn have changed drastically over the last two decades. To address these issues, video games and video game-based tools are proposed as a primary approach for motivating and supporting students in developing their skills in CT and support their learning of introductory programming. This research is concerned with the capture of statistical evidence on the educational effectiveness of playing a serious game specifically designed to enhance the development of CT skills to facilitate learning introductory computer programming. A total of 190 students were invited to participate in a *quasi-experimental pre-post study* for the purpose of analyzing the impact of an *adhoc* game to students' confidence in learning programming constructs and using their skills in CT. All students were studying a CS degree at the time and they were all registered to a first-year computer programming course. 151 out of 190 students successfully completed the study and the findings suggest that a) the intrinsic motivation to learn programming; b) students' perception of their knowledge and their tangible knowledge in key programming constructs (i.e. programming sequence, methods, decision making and loops); and c) students' confidence in using their CT skills were all statistically and significantly improved after playing the game. Additionally, students perceived computer programming significantly less difficult in their post study responses when compared to their pre study responses.

INDEX TERMS computational thinking, serious games, computer programming, video game design, game-based learning, gamification

I. INTRODUCTION

While there is an increasing demand in Computer Science (CS) profession as a mainstream discipline, CS degrees have the highest number of students dropping out in recent years according to the latest figures from the Higher Education Statistics Agency (HESA) [1]. The difficulty of learning computer programming is cited as one of the major factors for the high attrition rates within the CS discipline [2] [3] [4]. Recent studies in this field argue that the task of learning to program is often recognized as a frustrating and demanding activity by introductory programming student because of multiple factors such as poor teaching methods, low levels of interaction with students and a lack of interest in the subject [5], [6]. Moreover, introductory programming students often get confused with multiple levels of branching and locations in the programming logic and they have major difficulties in visualizing programming constructs from given problems [7]–[10]. Thinking within the syntax of a programming language is not *natural* to many introductory programming students and this creates numerous challenges for them as when they get error messages, they tend to edit programs unrelated to these error messages [11]. It is often discussed that students need to work at an operational level of abstraction before producing code in a specific programming

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3043278, IEEE Access

language so that they can develop their abilities in solving problems before they start programming [12].

The difficulties students experience in learning computer programming are not novel, and the reasons behind these challenges have been discussed by the field experts for many decades. In their seminal work, Bonar & Soloway [13] stated that the real problems introductory programming students have lie in "*putting the pieces together*". They highlighted the problem of recalling domain specific plans in order to encode pieces of information into meaningful units which means that the crux of the problem is the lack of critical thinking skills. Additionally, previous studies emphasized that failure rates worldwide of 30-50% in the introductory programming courses have been reported for decades [14], [15]. This has led to many discussions and ideas on how best to teach computer programming, because students tend to view of computer programming as a purely technical activity, rather than as a set of fundamental design and problem solving skills [16], [17]. Studies show that there are strong reasons to believe that the majority of students who are learning introductory computer programming tend to develop superficial knowledge on programming constructs, and thus fail to create problem-solving strategies through using programming constructs [16], [18], [19].

To address these problems several studies proposed visual programming tools, game design classes, video games and video game-like environments (henceforth both referred to as games) as ways to attract students into computer programming activities and as methods to teach the fundamental concepts of CS [20]–[25]. The first of these approaches are interactive syntax-free visual programming environments, such as Scratch [26] and Alice [27], where students often use graphical programming commands to build their programs in order to gain a visual perspective to abstract concepts fundamental to computer programming. This approach has taken massive attention by researchers as the mechanism behind these tools is closely related to fundamental design principles of programming and perceived to be ideal to quickly create solutions without the need for excessive program code [10], [28]–[30]. Another approach being followed to support learning introductory programming is via game development classes where although the objective is to design a new game as the product, the rationale is the realization of basic programming constructs in addition to planning algorithms [6], [31], [32]. This approach can cover an entire curriculum based on custom libraries, different game engines, visual programming tools or new programming languages. Therefore, students can learn introductory programming constructs in an engaging environment alongside developing the fundamental skills necessary to be a computer programmer (such as problem solving and team-working abilities). The final and certainly the least common way is to facilitate the teaching and learning of introductory computer programming through the use of video game technologies in an educational game context, also referred to as *serious games*, due to several exhibiting features of games such as learner-centricity, interactivity and immediate feedback. The rationale behind this educational gaming context, which is the fundamental argument for *serious games*, is that the immersive and engaging nature of game-play encourages students to greater levels of investigation, experimentation and re-use [33], [34]. Such games can thus support the realization of basic programming constructs in addition to planning algorithms and enhancing problem-solving skills [17].

This research is solely focused on this third approach mainly for the following two reasons: a) there is an overwhelming number of work regarding the first two approaches in the literature, but there are few examples of *serious games* that specifically focused on learning introductory computer programming and developing skills in CT via game-play [32], [35]–[37]. In other words, only a limited amount of work has been undertaken to scaffold the development of CT skills and learning how computer programming constructs work through serious game-play; b) there is a clear need for more empirical work in game-based learning (GBL) that investigates the tension between teaching CT and computer programming via game-play. It is crucial to provide more structured studies in this area which can combine a design framework and an experimental statistical analysis in order to strengthen the evaluations of learning in GBL for CS [38]–[41].

To address the above issues, this paper investigates the correlations between game-play and developing skills in CT, through learning a limited number of introductory programming constructs via serious game-play. The paper reports a detailed analysis of the current literature and presents an experimental evaluation through a *quasi pre – post study* experimental design in the premise to answer the following research questions:

"Can a serious game be designed to support the development of computational thinking through the medium of learning computer programming? If so, can this game enhance students' intrinsic motivation to learn computer programming and their perception of computational thinking skills?"

II. RESEARCH BACKGROUND

Many studies argue that there is a link between dropout rates in Computer Science (CS) degrees and the low motivation for learning computer programming, since often the mechanisms for learning computer programming are seen by students as neither interesting nor relevant [2], [14], [16], [42]. According to HESA figures [1], undergraduate enrollment numbers to CS degrees increased by 4% in recent years. However, the dropout rates from CS degrees equally increased as 72% of students considered dropping out of the university at some point in between 2016 and 2017, with 49% totally leaving their programme.

Further to above, a seminal work investigating the introductory programming failure rates undertaken a systematic review of the introductory programming

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3043278, IEEE Access

literature, and conducted a statistical analysis on the pass rates extracted from the relevant articles [3]. Their research showed the mean worldwide failure rate in introductory programming classes as 32.3%. A more recent study supported these findings and reported the worldwide failure rates in introductory programming courses as 28% [43]. While the failure rates in introductory programming classes are not alarmingly high, almost one in every three students fails in introductory programming courses. Furthermore, recent research in this field states that even students who have completed introductory programming courses still do not know how to program and/or may not have the ability to use programming codes to solve problems within the CS discipline [44].

It is widely accepted that learning to program requires comprehending abstract concepts about CS and arranging these concepts in a rational order in order to solve real life problems successfully. In other words, introductory programming students need to demonstrate an understanding of the patterns evident in programming, rather than focusing only on the syntax and semantics of computer programming languages [17]. However, the majority of introductory programming students perceive computer programming as a technical activity rather than a chain of cognitive skills [15]. Students often find the process of learning computer programming difficult because they need to find a solution to a problem by acquiring a new way of thinking in addition to the need to practice a new syntax and grammar in order to communicate their solution to real life problems [45], [46]. Many students are not conscious of this and, despite their training, when they undertake a computing project the reaction of the majority of them is to start coding immediately, skipping the crucial steps of analysis and design and the need to develop abstractions and algorithmic thinking [47]. Thus, learning computer programming becomes a demanding task and requires abstraction of CS concepts to describe a problem and propose a solution, followed by the need to design and code in order to convert the solution into the syntax of a programming language.

To overcome some of these challenges, Computational Thinking (CT) has been the focal point of a number of studies within the CS discipline in an attempt to integrate it into the basic curriculum [12], [48]. The concept, CT, was first used by Papert [49] and later deeply investigated by Wing [50]. In her seminal work, Wing described CT as a problem solving approach that combines logical thinking with CS programming constructs, and that it can be used to solve a problem in any discipline regardless of where the problem lies [50], [51]. In other words, CT is described as a set of intellectual and reasoning skills that state how people interact and learn to think through the language of computation that involves using methods, language and systems of CS. This does not mean that CT proposes problems that need to be solved in the same way a computer tackles them, but rather it encourages the use of critical thinking using concepts fundamental to the CS discipline. Many researchers support this and draw attention to the fact that CT is not a synonym for computer programming [52], [53]. Instead they define it as the process of making abstractions [21], [24] whereby the data and various CS concepts (i.e. computer programming constructs) are presented without clarifying any implementation details, in a form similar to the description of semantics [52]. Thus, if CT is defined as the ability to develop high-level conceptual design skills, then students with CT abilities can find computer programming much easier than others as learning computer programming requires the ability to select the appropriate symbolic and numerical data to produce generic solutions. Therefore, CT is related to conceptualizing and modelling solutions at an abstract, context-free level, whereas computer programming is directly related to the context rather than the concept of a solution and consequently focuses on the logical and physical realization of a solution within that context.

Owing to their easy engagement and motivational nature, games and game-based tools have been proposed by many studies both to increase the motivation in computer programming classes and also as a method to practice skills in CT [54]–[57]. As discussed above, there is an overwhelming number of works in this area which can be classified into three main categories as *game development with visual tools* [58], [59], *extensive game development* [60], [61] and *learning through game-play* [17], [62]–[64]. All three approaches reported success with a radical increase in students' motivation to learn programming, hence this provided some evidence that integrating games into the education of introductory computer programming is a promising strategy.

In *game development with visual tools*, students are exposed to create visual abstractions without the need to write excessive programming code or have a background in games programming via visual programming tools such as Scratch [26] or Alice [27]. Complex scenarios can be created in these environments by combining character behaviors which inevitably requires an understanding of programming sequence, conditionals, iterations and methods. Furthermore, visual programming tools remove the syntax rules of actual programming languages and present programmatic representations as *blocks* through a simple drag and drop interface. This cleverly separates the programming logic from programming grammar and syntax, allowing students to focus on developing programming strategies with little or no programming background. Despite all these positive traits, research in this field points out that visual programming environments are merely tools and without well-organized teaching methods and learning materials to support them, all they can provide is a "*short burst of enthusiasm*" [65]. An extensive study identified that visual programming environments influence not only the learning of introductory programming but also the habits of programing that students develop during their learning process [66]. According to this research, when

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3043278, IEEE Access

students are asked to perform a programming task they do not approach it by thinking at the algorithmic level but instead, they attempt to solve the problem by using all the blocks that seemed to be relevant for solving the task and randomly combine these blocks into a script in order to try to solve the problem. Additionally, it has been observed that students tend to produce unstructured programming solutions through using various blocks (such as with a repeat-until loop) where the body of blocks are logically coherent and easy to understand, but the outcomes produced by students are no longer coherent and well-organized. One could argue that this is not related to the characteristics of visual programming environments but might be the poor software development skills and weak programming abilities of students. However, scripts and graphical objects are often executed concurrently in visual programming environments. As the scripts are written in the graphical objects, it is difficult for students to develop the skills necessary for building logically coherent solutions as the execution of objects always happen simultaneously. Studies argue that concurrent programming exists as an integral part of the visual programming environments and although debugging concurrent programs can be seen as a viable concept to support learning, students' tendency to develop unstructured programming solutions (such as an incorrect use of a loop construct), leads to outcomes that contain lots of repetitions in different scripts which is a bad programming practice [10], [66]. At this point, it is important to highlight that the intention here is not to alienate visual programming environments from learning introductory programming or to blame these environments in any way, but rather to emphasize that these environments are simply design tools which do not necessarily consider good programming practices as this was not their purpose. They simply lack a mechanism to support students in their quest to understand fundamental ideas in CS such as to algorithmic thinking, debugging programs and the correct use of programming constructs. Although visual programming environments are very valuable tools and can generate well-structured programs with hundreds of concurrent scripts, one must avoid thinking of these environments as a substitute for pedagogy in learning introductory programming because their characteristics might allow students to incorrectly use programming constructs. More importantly, a student can transfer their bad habits in programming gained from these environments into their further studies in CS. As suggested by many researchers, visual programming environments remove programming syntax problems when learning introductory programming, but the need to write algorithms before programming remains essentially a cognitively demanding task [8], [65], [66].

The second approach is the *extensive game development* which aims to develop new games or linear scenarios as an end product. This approach is often supported with game-engines, and/or programming languages with graphical libraries. The idea here is not to complete a polished end-product but the learning experience and visualization of how programming constructs work. While this approach was proven to be useful, students need to consider all aspects of producing a product including, but not limited to, game graphics, sound, game play, physics and narratives [31], [67]. This can sometimes be overwhelming for introductory programming students and therefore, the approach requires game development experience and careful planning [63].

The final category is *learning through game-play* which has taken less attention from researchers compared to the first two approaches. It is discussed in the literature that CT patterns (e.g. decomposition, abstraction, pattern generalization, algorithm design) are context and application independent and therefore, they can easily be reflected and developed through game-play [52] (Basawapatna et al., 2011). Research in scalable game design highlights that once students understand conceptually how to present a CT pattern, they should be able to transfer and use it in the context they choose [65].

In recent years, studies investigating the relationship between digital game-play and learning computer programming have increased [63]. Survey research in this field indicate that there are over 40 games that were reported in the literature within the last two decades [63]. Some of the most cited of these are Robocode [68], Wu's Castle [69], Colobot [70], Prog & Play [62], LightBot [71], RoboBuilder [72], Code Combat [73] and CodeSpells [74]. While many studies showed these games are powerful motivators [62], [69], [71], majority of them use a text-based programming language that requires the player to pay considerable attention to the details of syntax, which is not very well aligned with CT [75]. Additionally, some of these games were evaluated without well-defined research design and the results were often preliminary with small sample sizes.

Although limited in number, there are very good structured evaluations of learning through game-play for CS. Lee and Ko [44], [76] introduced their game *Gidget,* where a robot protagonist is expected to write code to complete a series of missions. They managed to run randomized controlled trials with a control and an experimental group where the control group played a generic version of the game and the experimental group played a personalized version of the same game with player selected images and sound effects. Based on a total number of 116 responses, the motivation level of the participants in the experimental group was higher than the motivation level of the participants in the control group in terms of learning computer programming [76]. More recently, the same researchers undertook another study with 60 participants where they divided these participants into multiple experimental groups and measured their knowledge before and after they a) played their game and b) completed an online Python class. The findings of this study suggest that introductory programming concepts can be taught within a few hours with their game when this is

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3043278, IEEE Access

explicitly guided by a curriculum. While this research is very valuable for GBL, it is important to note that the evaluation was conducted at the computer programming level rather than the level of CT because participants were expected to write code in a programming language to show their knowledge/skills. Similar to this, another study investigated the learning effectiveness of 120 students who were learning the database Structured Query Language (SQL) at the time [77]. In this experimental study, participants were asked to play an *adhoc* game in different modes and their performance was compared to a paper-based learning. It was found that the game modes they developed, produced better learning outcomes than paper-based learning. However, like many others, this research also evaluated the learning performance of participants at the level of coding. In a very recent study, Zhao and Shute [64] introduced their game *Penguin Go* and asked a total of 69 participants to play their game and provide feedback. Unlike the other studies however, they focused on the main components of CT rather than programming which included but not limited to problem decomposition, algorithmic thinking, conditional logic, recursive thinking, and debugging. Their findings indicate that participants felt their skills significantly improved in CT after playing the game. Yet, this did not impact on their learning of programming which means that neither their attitudes toward CS nor their knowledge/skills in programming were significantly enhanced.

Based on the forementioned studies, there are promising research findings regarding the attitude and cognitive development of participants in playing video games specifically designed to teach computer programming and practicing skills in CT. However, it is also clear that this area needs further structured assessments and more evidence particularly in learning through playing games for CT. In other words, more studies need to emphasis on cognitive skills in CT rather than focusing only on the programming knowledge through the syntax of code and good programming practices.

For the reasons defined above, this study investigates the relationship between game-play and participants' confidence in learning computer programming and using CT skills. Having done a detailed literature review on the existing work in GBL for CT, the research questions presented in the introduction part were refined and fit into an experimental study structure.

As shown from Table 1, a null and an alternative hypothesis were added to each refined research question in order to statistically analyze the collected data with the aim of answering these new research questions. Several studies were reviewed before categorizing these questions [34], [44], [64], [71] and the focus of the research was kept on the confidence level of participants as well as their potential knowledge gain and skill development from this study.

Table 1. Showing refined research questions, null and alternative hypotheses in this research

Research Question	Null Hypothesis (Ho1)	Alternative Hypothesis (Ha1)
RQ1: Is there a difference in students' attitude to dropping their degree programmes between the pre and the post study?	There is no significant difference in students' attitude to dropping their degree programmes between the pre and the post study	Students' attitude to dropping their degree programmes is significantly changed between the pre and the post study
RQ2: Is there a difference in students' motivation to learn computer programming between the pre and the post study?	There is no significant difference in students' motivation to learn programming between the pre and the post study	Students' motivation to learn programming is significantly changed between the pre and the post study
RQ3: Is there a difference in students' perception of difficulty of computer programming between the pre and the post study?	There is no significant difference in students' perception of difficulty of computer programming between the pre and the post study	Students' perception of difficulty of computer programming is significantly changed between the pre and the post study
RQ4: Is there a difference in students' understanding of programming constructs* used in the game between the pre and the post study?	There is no significant difference in students' understanding of programming constructs between the pre and the post study	Students' understanding of programming constructs is significantly changed between the pre and the post study
RQ5: Is there a difference in students' perception of their CT skills** between the pre and the post study?	There is no significant difference in students' perception of their CT skills between the pre and the post study	Students' perception of their CT skills significantly changed between the pre and the post study

* Programming constructs used in the game are *programming sequence*, *methods*, *decision making* and l*oops*.
** CT Skills refer to *problem solving*, *constructing algorithms*, *debugging* and *simulation*.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3043278, IEEE Access

III. DEVELOPING A RESEARCH TEST BED

An *adhoc* video game named *Program Your Robot* was developed from scratch as a test-bed in order to assess the refined research questions in this study. As discussed previously, there are several games available to assess motivation to learning programming. However, it was decided to develop a new *adhoc* game for this research mainly for two reasons:

Firstly, to develop a model that would allow students to practice a series of cognitive abilities that characterize CT, regardless of their programming background. These cognitive abilities are defined as *problem solving*, *constructing algorithms*, *debugging*, *simulation* and *socialization* [75]. Majority of the existing games in the literature are played via coding where players need to consider syntax of coding in order to efficiently play them. This is clearly closer to the level of computer programming rather than the cognitive abilities of CT. On the other hand, this research is primarily focused on practicing skills in CT, even if the participants have little or no programming background.

Secondly, to support the learning process of introductory programming by demonstrating how a limited number of key introductory computer programming constructs can be used as vividly explicit game elements which introductory programming students often find challenging and/or difficult to understand. Previous guidelines prepared for CT and serious games suggest that when programming constructs are used as integral elements of game-play, this tends to motivate and engage players more than when it is not [78]. Based on these guidelines, the game in this research needed to be a platform to practice fundamental programming constructs students often find difficult through familiarizing these constructs as integral parts of the game-play. Moreover, the programming constructs used in the game had to be well aligned with the curriculum of the introductory programming course participants were taking at the time because this was needed for the ethical approval of the research. As a result, this research is concerned with the educational effectiveness of *Program Your Robot* which is specifically designed to practice CT skills through the medium of learning introductory programming constructs.

Program Your Robot was developed in Adobe AIR [79] with HTML 5, JavaScript and ActionScript support. Additionally, it was designed as a single player isometric puzzle solving experience, and the core idea behind this was to encourage the development of individual cognitive skills which would expectantly support students to learn how computer programming constructs work. In other words, it was aimed to practice core skills of CT particularly *problem solving*, *constructing algorithms*, *debugging* and *simulation* via game-play. Despite this, the *socialization* aspect of CT was not implemented in this version of the game because this needed a multi-player game design and careful planning on the top of the other four core skills.

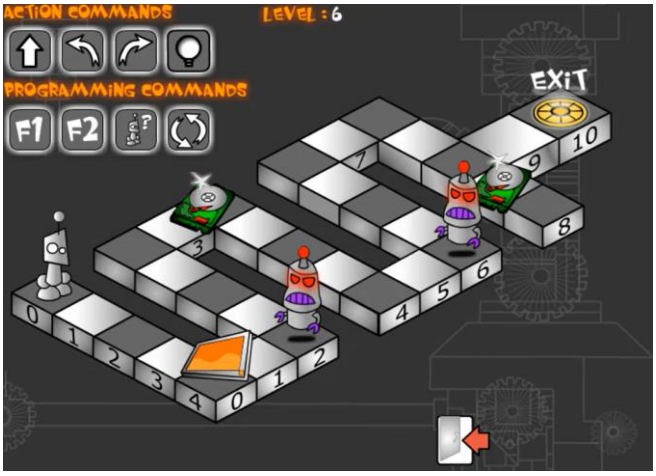


Figure 1. Program Your Robot Game-Play

As shown in Figure 1, the aim of *Program Your Robot* is to steer a character to its target via the most viable route through using a series of commands that plays a key aspect in constructing efficient solutions. The game is designed to be a puzzle solving action game where players control a robot and help it to reach specific destination(s) by giving commands. To play the game, players need to drag and drop commands from a selection interface into a limited number of areas, which leads them to construct their own algorithms. There are two types of commands players can instruct to the robot: *action commands* and *programming commands*. *Action commands* have a direct effect on what the robot does (such as go forward, turn right), while *programming commands* affect robot's actions by supporting the solution developed by the player through a programming construct. The current version of the game contains four fundamental programming commands which are the *programming sequence*, *methods*, *decision making* and *loops*. When designing the game, the *programming sequence* was taught as the initial construct students need to instruct to the robot and discover how these are performed. Having done so, the m*ethods* were implemented into game-play which can support the players to break down their solutions and create repeatable patterns. The *decision making* is designed to evaluate certain conditions such as to detect whether the robot faces an enemy or an obstacle. Finally, the *loops* in the game allows players to repeat a series of commands for a number of times they predefine. All commands, whether action or programming, can be dragged from their associated toolbars into specific areas called *slots*. Each slot can contain only one command and players can use any number of commands in any sequence, for as long as they have empty slots. Once the player is happy about the solution they created within these slots, they can either *debug* their solution or *run* it. If the player debugs their solution, they can observe a quick trace of the solution they built in and can find any mistakes they might have made in their solution. This is particularly useful for players when the robot does not behave the way that they expect or when they misuse a programming command.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3043278, IEEE Access

Alternatively, the players can run their solution which then the robot is animated and follows the instructions given in the slots one by one. This simple mechanism was applied to encourage players to debug their solutions and find out mistakes whenever they get stuck in the game. In other words, the learning material in the game was designed to be an integral part of the game-play without any need for coding.

In addition to these, the game-play is mapped onto part of the computer programming curriculum, more specifically on four key areas (i.e. programming sequence, methods, decision making and loops). The constructs are introduced as the players progress through the game. Due to the puzzle solving structure of the game, players are required to use their *problem solving* to find the most effective pathway for their robot to beat each level. Further to this, when players design their solutions, they need to *construct algorithms* in order to complete levels. *Debugging* in the game allows monitoring of solution algorithms and detecting potential errors which is an integral component of both CT and programming [52]. Correspondingly, s*imulation* is the run-time mode where one can observe the behavior of the robot and analyze whether or not a winning strategy has been created in the game.

As the players progress through the levels in the game, the levels become more complex and the available number of empty slots decreases. Consequently, two different reward systems were applied in the game in order to stimulate players to discover efficient solutions during their game-play. These are the *score system* and the *achievements*.

The *score system* was designed to give points to players according to how efficiently they designed their solutions to beat that level. The score of the players significantly increased when they used repeatable patterns with smaller number of slots to construct solutions. As an example, a player that beats a level with using 8 slots and without considering any reusability scores lower than another player that considered reusability of commands and completed the same level with using 4 slots. In other words, if players want to achieve a high score in the game, they need to demonstrate deep understanding of the programming concepts used in the game, such as when they create repeatable patterns with methods or loops. Therefore, building efficient algorithms illustrates good game-play as well as promoting the acquisition and development of the CT skills discussed above.

The second reward systems applied was the *achievements* which are the trophies players can unlock when they demonstrate certain programming patterns in the game. As an example, earlier levels in the game do not require the use of nested loops to overcome them. However, should the players successfully demonstrate the use of nested loops to beat these levels, they can unlock the *nested loop*er achievement which is a trophy they earn and can show the other players in addition to extra score points they obtain from unlocking this achievement. Therefore, when players discover good programming practices, they are rewarded in their game-play. This mechanism is popular among the current generation of games and emphasizes the importance of increasing the motivation through behavioral conditioning [75]. Finally, there is a high score chart in the game where players can submit their score or share the trophies they collected with other players. The participation in the high score chart is optional as when players do not do well in their game-play, they do not need to share this with others.

In many ways, *Program Your Robot* is similar to other games such as to Light-Bot [80], Microsoft's Tinker [81] and Robozzle [82]. However, there are considerable differences between *Program Your Robot* and these games which are discussed below:

Firstly, the learning material in *Program Your Robot* is represented in game elements and mapped onto part of the computer programming curriculum taught within the Computer Science department of University of Greenwich. Secondly, four out of five main categories of CT skills (i.e. *problem solving*, *constructing algorithms*, *debugging* and *simulation*) are explicitly integrated as patterns into the game mechanics. In other words, the game is built on the top of the cognitive structure of CT rather than for fun only whereas the games listed above are created for fun and not for learning purposes. Additionally, none of the above-mentioned games sufficiently focused on the accurate use of programming constructs or that map to an introductory programming curriculum as this was not their aim. Therefore, although the game-play of *Program Your Robot* was inspired from other games, crucial differences guided the development, such as the necessity to consider accurate use of programming constructs and the intention to practice cognitive CT skills during game-play.

Last but not least, *Program Your Robot* was designed to be gender and expertise neutral as the theme of the game (i.e. a robot trying to escape from a maze) is not male or female oriented; and players do not need to have prior computer programming knowledge to play the game.

In a previous study, a free form pilot study was conducted with a wide variety of students as an initial qualitative evaluation of the game before moving to the experimental study stage of the research [75]. Having improved the game based on qualitative feedback obtained in the pilot study such as by adding *achievements* and *high score* features, an experimental study was conducted at the University of Greenwich on a first-year computer programming class of 190 students, who had diverse backgrounds and variable prior knowledge in computer programming. The details about the research design of this experimental study are discussed below.

IV. RESEARCH DESIGN

This study uses *one group pre – post study quasi experimental design* to measure participants' perception and confidence levels in the refined research questions before and after they played the game. This design is often

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3043278, IEEE Access

referred to as *one-shot case study* and it is particularly useful in observing before and after effects of a treatment or a practical approach (in this case, the game) [83]. However, the major limitation of this design is the lack of a comparison or a control group. Additionally, participants cannot be randomly assigned to groups in this research design which is a threat to internal validity of the research.

Although these limitations are well known, and a Randomized Controlled Trial (RCT) or a Clustered Randomized Control Trial (cRCT) would have been a more effective approach in this research, this quasi experimental design was selected because of three main reasons.

Firstly, this study was conducted at the University of Greenwich, London (UoG) on first year Computer Science students and hence, an ethical approval from the UoG ethics committee was needed. In order to obtain the ethical approval, the ethical committee insisted on ensuring that students receive the same experience throughout the study. Despite the fact that this research aims to measure the effect of an educational tool (i.e. *Program Your Robot*), if some students were advantaged from it, this would be considered unfair and it was simply not possible to treat students differently. In other words, the ethical restrictions prevented this research to divide students into two random groups and apply different educational interventions.

Secondly, when the study was conducted, the participants had only just registered for their computer programming course. As the study was conducted on the fourth week that the computer programming course officially started at UoG, the students were just learning the programming constructs when they were asked to undertake this study. More importantly, students' background in computer programming and in video games were not formally evaluated before they agreed to participate in the study. As it was the beginning of the term, this was considered as a critical time for them to learn and practice with equal opportunity.

Finally, there is no universally agreed way of teaching CT skills in higher education to students as this is an abstract concept and how to teach CT is an active research area [12], [84], [85]. As there is no standard or a traditional way of teaching CT in higher education, how to teach CT skills to students in the most convenient way is an experimental area. In addition to this, *Program Your Robot* is not intended to replace or compete against any experimental approach for practicing CT but rather developed to practice a specific way of thinking. Given these circumstances, one group pre-post study design was perceived to be safest design to follow in order to conduct the experimental study on schedule.

As discussed above, the participants of this study were first year CS students recruited from the UoG who were all adults (i.e. 18+ years old) and registered to the introduction to computer programming course taught at the CS department at the time. All participants were aware of computer algorithms particularly of pseudo codes and flowcharts. Additionally, all participants were introduced all of the programming constructs used in the game (i.e. programming sequence, methods, decision making and loops) during their computer programming classes.

The study was conducted within the university during the computer programming tutorial hours and the participation in the study was voluntary work. The structure of the study was clearly explained to the students (i.e. pre-post study) and they were aware that they all had the option to dropout from the study at any time without providing a reason. The research was confidential, and the participants were not asked to provide any information that would reveal their identity. All participants were asked to use a randomly generated unique number in the pre and the post questionnaires so that it was possible to match their responses. Each participant received a consent form at the beginning of the study so that they were able to decide whether or not to participate. The environment was familiar to the participants and they were all informed that their honest answers were vital and valuable for the research. Additionally, it was clearly explained to them that their decision on participating in this study would not have any effect on their grades.

Having signed the consent forms, the participants were asked to complete two online questionnaires both before and after they played *Program Your Robot*. When filling the questionnaires, participants were specifically asked not to contact to one another as this might bias their answers. On the other hand, the participants were free to contact their peers during their game-play. Each of the CT skills also explained to participants such as *simulation* was explained as the ability to visualize how programming constructs work and *debugging* as the skill of detecting and removing existing and potential errors.

At the beginning of the study, the participants were invited to complete the pre study which involved undertaking an online questionnaire to rank their perception on a) giving up their degree programmes; b) motivation for learning computer programming; c) difficulty of learning computer programming; d) knowledge in programming constructs and e) their skills in CT.

Participants' confidence and their perception were assessed using a 5-point rating Likert scale in each question. The main reason for using a close ended questionnaire was to make the participants' perception of abilities quantifiable. The Likert scale ranged from one extreme attitude to another from 1 (*strongly disagree*) to 5 (*strongly agree*), 3 being the moderate (neutral) point. A *not applicable* option added to some questions in case these questions were not relevant to them. Each rating indicated more satisfaction than the rating before it, but the distinction between the ratings were not measurable. As an example, the difference between a *strongly agree* and an *agree* response can be less than the difference between a *strongly disagree* and *disagree* response. Therefore, all observations gathered from the participants exist on an ordinal scale.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3043278, IEEE Access

In addition to these perspective questions, the participants were asked to answer four knowledge test questions regarding each programming construct presented in the game (i.e. programming sequence, methods, decision making and loops). These questions were prepared by the tutors of the computer programming course taught at the UoG and revised several times before being put in the questionnaires. Within these questions, the participants were provided a problem in the format of pseudo code and asked to draw a flowchart or provide an algorithmic solution to solve the problem. Each of these questions were multiple choice and only one of the responses were correct. The participants were asked to provide their solution before selecting an answer, and if they did not know the answer to the questions, they were asked not to provide an answer at all since this would have had no effect on their studies. This approach was specifically used to observe how students think, approach, and solve a problem, rather than merely measuring their ability to code in a specific programming language.

The post study online questionnaire given to the participants was identical to the pre study questionnaire provided to them at the beginning of the study. The only major difference between the pre and the post study was the knowledge test asked to the participants regarding the programming constructs. While these questions were kept at the same level of difficulty, the questions were different in order to accurately measure their knowledge in the relevant programming constructs both before and after they played the game.

Table 2 shows the *independent* and the *dependent* variables sought through the online questionnaires (i.e. pre and post). While the independent variables were only asked in the pre study, the dependent variables were asked both in the pre and the post study.

Table 2. Independent and Dependent Variables of the Study

Variable Type	Variable
Independent	Age, Sex, Ethnicity, Mathematical Qualification
Dependent	Attitude to giving up the degree Programme Perception on the difficulty of computer programming Intrinsic motivation to learn computer programming Perception of knowledge in programming constructs (i.e. sequence, methods, decision making and loops) Knowledge test on programming constructs (i.e. sequence, methods, decision making and loops) Perception of CT skills (i.e. problem solving, constructing algorithms, debugging and simulation)

Lastly, the total time allocated for the study was about three hours, but it took longer than this as majority of the participants insisted on playing the game. When participants finally completed the study, the results of the post study were matched with the results obtained from the pre study by using the unique number generated for each participant. Hence, it was possible to investigate the participants' confidence and perception in the measured categories as well as any potential knowledge/skill they gained from the study.

V. DEMOGRAPHICS AND THE STATISTICAL APPROACHES

A total of 190 participants were invited to participate in this study and all participants were registered to the Computer Programming course taught at University of Greenwich. The participants were randomly invited to participate and they came from a wide variety of backgrounds and ethnicities.

While the study was conducted with 190 participants, 39 (20.5%) of these dropped out or did not complete the study adequately such as they did not complete the questionnaires or did not play the game. Overall, 151 out of 190 (79.47%) valid responses were successfully collected and interpreted by matching pre study responses to post study responses. While the exact reasons for the dropout rates are not known, majority of these happened within the first half an hour of the study and the responses were left in a state that it was simply not possible to compare these with the adequate responses collected.

Among the valid responses obtained, 127 (84.1%) out of 151 were from *male* participants and 24 (15.9%) out of 151 were from *female* participants. Whilst 131 (86.8%) participants were in the 18-24 age range, 15 (9.9%) were in between 25 and 29. Additionally, 4 (2.6%) participants were in the 30-39 age range and 1 (0.7%) participant was over 40.

In terms of ethnicity, the UK government standard ethnicity classification was used in this study. According to the data obtained, 62 (41.1%) participants defined themselves as *White*. Consecutively, 36 (23.8%) out of 151 participants defined themselves as *Asian or Asian British* and 34 (22.5%) out of 151 participants defined themselves as *Black or Black British*. While 8 (5.3%) participants declared themselves as *Mixed/Dual Background*, 11 (7.3%) participants indicated that they have *other ethnic background* that did not fit into the UK government standard ethnic classification.

In addition to age, sex and ethnicity, the mathematical qualifications of participants were collected in this study to analyze whether or not participants' mathematical qualifications had any effect on their experience in the study. According to the data obtained, 36 (23.8%) out of 151 participants had *A-Level Maths or equivalent* qualification. While 4 (2.6%) participants had *AS-Level Maths or equivalevent* qualification, 50 (33.1%) participants *GCSE Maths grade A or B or equivalent*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3043278, IEEE Access

degree. Moreover, 47 (31.1%) participants stated that they had *GCSE Maths grade C or equivalent* degree, and 9 (6%) more participants had *lower than GCSE Maths grade C*. Finally, 5 (3.3%) participants indicated that they had *other mathematical qualifications*.

Two statistical methods were considered to evaluate the data obtained from this research which are common statistical approaches used when comparing groups in a before–after design: the paired t-test (post-pre score) and the analysis of covariance (ANCOVA).

Upon careful investigation of these methods, it was decided to use only the paired t-tests in this research because the initial and final scores are related to some ability (i.e. CT) and the effect of intervention (i.e. the game) is the same across all levels of this ability [86]. While the ANCOVA test is a more flexible model than paired t-test, this was not used due to the lack of a control group [87] . Regardless, research in this field discuss that both paired t-tests and ANCOVA generally produce good estimates when random allocation is used in studies [86] .

Having decided to use the paired t-tests to analyze the collected data, a test of normality was undertaken to investigate the distribution of data as this is an assumption for using paired t-tests.

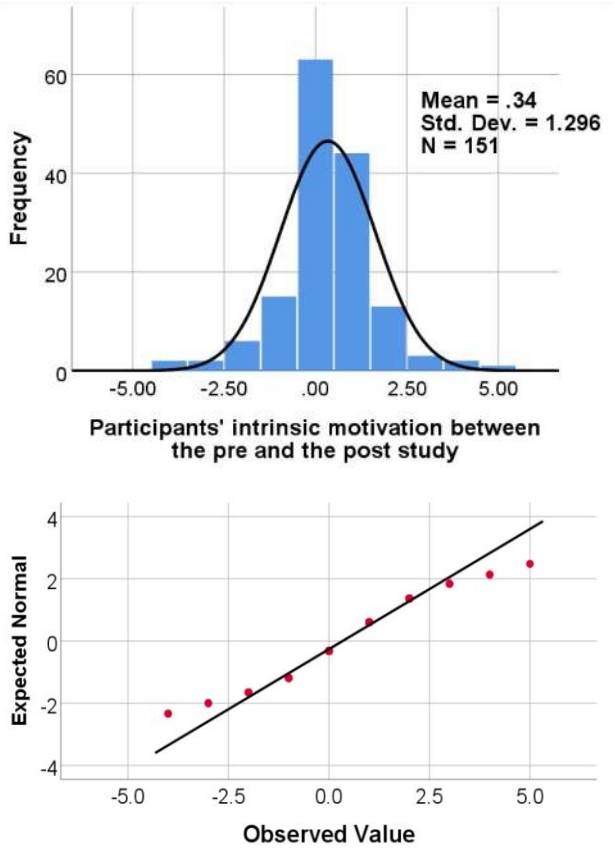


Figure 2. Showing the histogram and the Quantile-Quantile (Q-Q) Plots of participants' intrinsic motivation between the pre and the post study.

According to the Central Limit Theorem (CLT), when random variables are added, their means tend toward a normal distribution even if the original variables themselves are not normally distributed [88]. Therefore, a computer simulation was run to estimate the average number on the pre-post differences of collected data in the repeated samples. Having done so, a histogram and quantile-quantile (Q-Q) plots for each measured category were investigated in detail to ensure that the data came from a normally distributed population. As the normality check is an iterative process, only the histogram and the Q-Q plots regarding the intrinsic motivation of participants are presented in this paper.

As shown in Figure 2, the histogram generated from the data demonstrates a close resemblance to a normal distribution as the distribution is neither too flat nor too peaked. The population mean value is close to 0 ($\mu = 0.34$, N=151) and the standard deviation is just over 1 ($\sigma = 1.29$, N=151). The Q-Q plots also support the data distribution on the histogram as the observations embrace the linear line of Q-Q plots with a slight positive slope. Overall, a linear distribution pattern was observed and therefore, it is possible to assume that the data came from a normally distributed population. This procedure was repeated for each research question, and in all cases, it was found that the distribution of data was close to normal distribution.

In addition to the paired t-tests, a multiple linear regression (MLR) was used to investigate the potential effect of independent variables to a possible change that might happened in dependent variables in between the pre and the post study. The MLR analysis consisted of three main statistical stages which are an Analysis of Variance (ANOVA) test, a model summary and correlation coefficients. The first stage focused on the ANOVA test results regarding the overall impact of independent variables (also called predictors) on the dependent variable. To interpret the ANOVA test results correctly, a null and an alternative hypothesis was created. The null hypothesis (Ho1) indicates that there is no significant linear relationship between predictors and dependent variables. Correspondingly, the alternative hypothesis (Ha1) indicates that there is a strong and significant linear relationship between predictors and dependent variables. Accepting the null hypothesis means that the outcome of dependent variable has no significant correlation with the independent variables which provide evidence that regression threat does not have a major impact on the outcome of the study. The opposite of this is rejecting the null hypothesis which means that the outcome of dependent variable is somehow correlated with the independent variables and therefore, predictors impact on the outcome of the study. The second stage is a model summary that shows how strong the correlations are in between the predictors and the dependent variables. The final stage is the correlation coefficients which provide evidence on whether or not the correlation between each independent variable and the dependent variable is significant and strong. Each independent

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3043278, IEEE Access

variable is matched with the dependent variables individually to identify whether or not the predictors have a significant impact on observations.

To summarize, the paired t-tests were used to investigate the data collected in the pre and in the post study and a MLR was used to examine any potential associations in between the independent variables and the dependent variables.

VI. RESULTS

The results of this study are investigated in two sections: Raw Data and Statistical Analysis. The raw data section reports the responses of participants in the study, whereas the statistical analysis investigates whether the collected data are statistically significant or not.

A. RAW DATA

Having collected the demographic data, one of the first questions directed to participants in the study was whether or not they considered giving up their degree programmes and whether the difficulty of programming was a key reason for this or not. There were two aspects to this question: a) to define how many students have thought giving up their degree programmes since they started, and b) how many of these thought that the difficulty of learning computer programming was a key reason for this.

As shown in Figure 3, 24 (15.9%) participants strongly disagreed and 43 (28.5%) more disagreed on giving up their degree programmes when filling the pre-questionnaire. While 53 (35.1%) participants were neutral, 27 (17.9%) participants agreed and 4 (2.6%) more strongly agreed on giving up their degree programmes in the pre study. After playing the game, the number of participants that strongly disagreed stayed the same (i.e. 24). On the other hand, the number of disagrees increases from 43 (28.5%) to 48 (31.8%) and consequently, the number of participants who were neutral increased from 53 (35.1%) to 59 (39.1%). Furthermore, those participants who agreed on giving up their degree programmes reduced from 27 (17.9%) to 16 (10.6%). Finally, the number of strongly agrees stayed the same (i.e. 4).

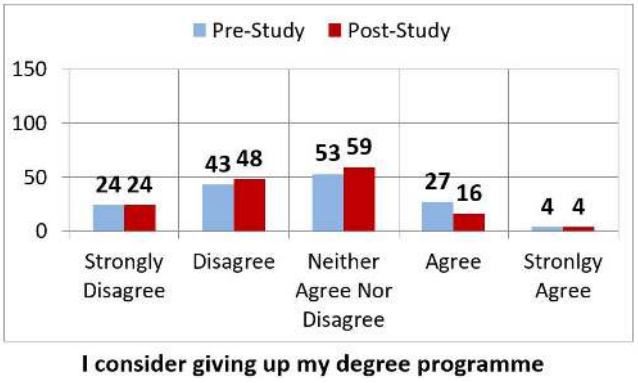


Figure 3. Distribution of participants who consider giving up their degree programmes.

In addition to investigating how likely participants were to give up their degree programme, the potential impact of the difficulty of computer programming to this was also investigated. As shown in the figure 4, both in the pre and in the post study, 24 (15.9%) participants never considered giving up their degree programme. While 13 (8.6%) participants strongly disagreed on giving up their degree programme because of the difficulty of computer programming, a total of 32 (21.2%) participants disagreed in the pre study which was then raised to 35 (23.2%) in the post study. Similarly, the number of neutral participants raised from 44 (29.1%) to 49 (32.5%); and the number of agreed participants reduced from 32 (21.2%) to 24 (15.9%) after playing the game. Lastly, the number of strongly disagrees did not change in between the pre and the post study.

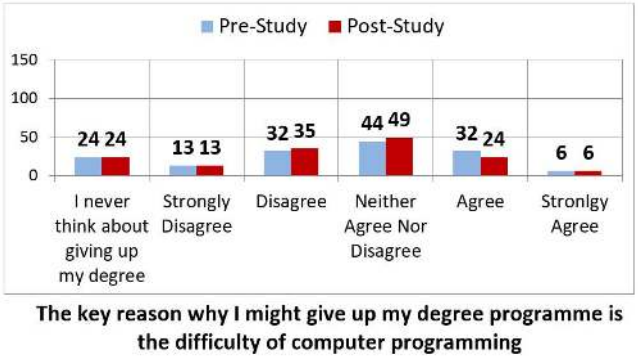


Figure 4. Distribution of participants who consider giving up their degree programmes because of the difficulty of computer programming.

This data presented some evidence that participants' attitude to studying their degree programme was affected in a positive way. To investigate this further, participants' perception on the difficulty of programming was explored both before and after they played the game.

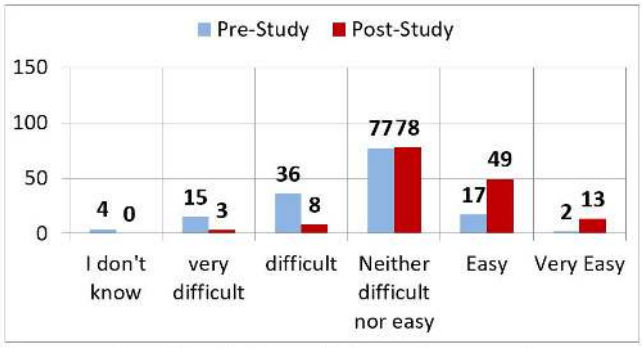


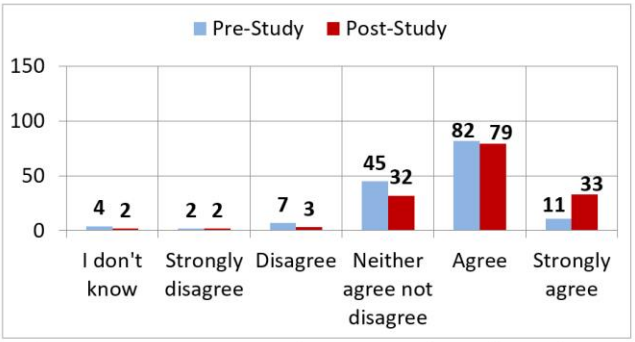
Figure 5. Distribution of participants' perception on the difficulty of computer programming.

As shown in Figure 5, majority of participants changed their perspective on the difficulty of learning computer

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3043278, IEEE Access

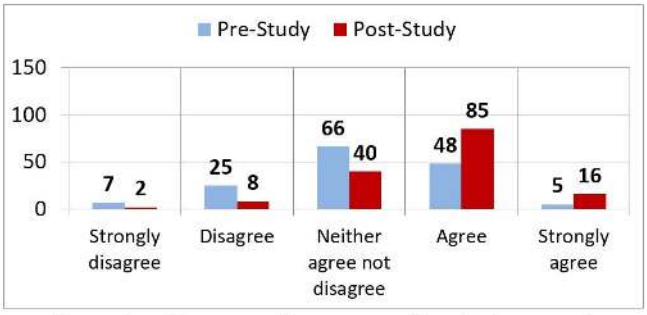
programming after playing the game. Although 4 (%2.6) participants indicated that they do not know the answer in the pre study, none of the participants chose this option in the post study. More importantly, 15 (9.9%) participants found learning computer programming *very difficult* in the pre study whereas this was reduced to 3 (2%) in the post study. Similarly, 36 (23.8%) participants selected the *difficult* choice in the pre study, but this was reduced to 8 (5.3%) in the post study. While the *neutral* option raised from 77 (51%) to 78 (51.7%), those who found learning computer programming *easy* raised from 17 (11.3%) to 49 (32.5%) and those who found *very easy* increased from 2 (1.3%) to 13 (8.6%). These results put forward some evidence that participants' confidence in learning computer programming increased in between the pre and the post study as many of them changed their opinion positively and none of them marked the *I don't know* choice in the post study.

When participants were asked whether they have intrinsic motivation to learning programming or not, majority of them provided more positive answers in the post study than in the pre study. As shown in Figure 6, those who did not know what to answer to this question reduced from 4 (2.6%) to 2 (1.3%) in between the pre and the post study. While the participants who strongly disagreed were the same (2, 1.3%), the disagrees and neutral responses reduced from 7 (4.6%) to 3 (2%) and 45 (29.8%) to 32 (21.2%) respectively. The agree responses also reduced from 82 (54.3%) to 79 (52.3%). On the other hand, the strongly agrees increased from 11 (7.3%) to 33 (21.9%).

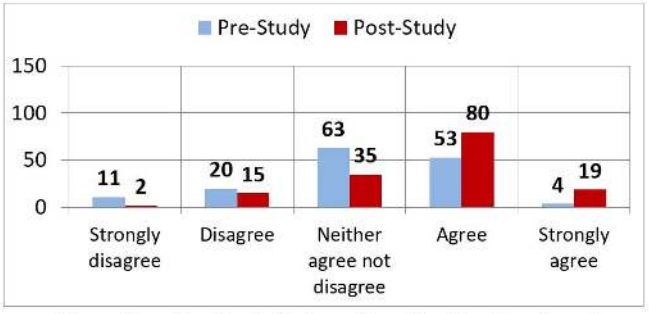


I have intrinsic motivation (motivation that is driven by an interest or enjoyment) to learn computer programming.

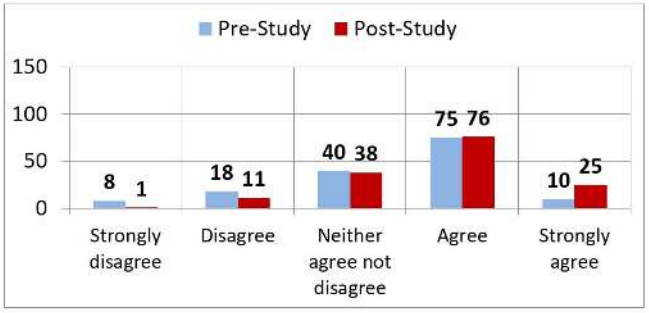
Figure 6. Distribution of participants' intrinsic motivation to learn computer programming.



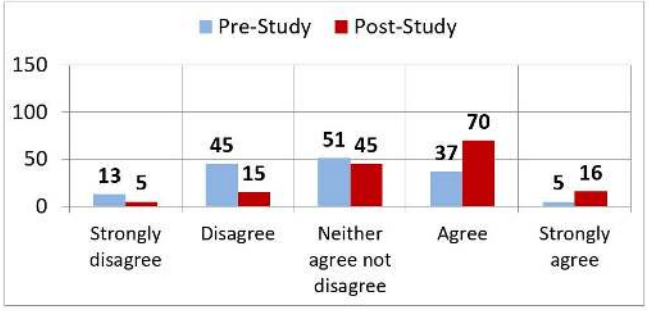
I know how "programming sequence" works in computer programming.



I know how "methods" (also referred as functions) work in computer programming.



I know how "decision making" (also referred as selection, conditionals) work in computer programming.



I know how "loops" (also referred as iteration) work in computer programming.

Figure 7. Distribution of participants' perception of their knowledge in programming constructs used in the game.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3043278, IEEE Access

Figure 7 displays participants' ranking of their own knowledge in four programming constructs implemented in the game both before and after they played the game. As shown in the figure, participants' confidence in their own knowledge increased in between the pre and the post study.

When participants were asked to rank how *programming sequence* works, 7 (4.6%) participants strongly disagreed and 25 (16.6%) more disagreed that they know how programming sequence works in the pre study. While 66 (43.7%) participants were neutral, 48 (31.8%) of them agreed and 5 (3.3%) more strongly agreed that they know how *programming* sequence works. After their game-play, the number of strongly disagree and disagree responses reduced from 7 (4.6%) to 2 (1.3%) and 25 (16.6%) to 8 (5.3%) respectively. Additionally, the neutral responses also dropped from 66 (43.7%) to 40 (26.5%). On the other hand, the strongly agree and agree responses increased from 48 (31.8%) to 85 (56.3%) and 5 (3.3%) to 16 (10.6%) correspondingly.

A similar increase also happened when participants were asked to rank their knowledge regarding how *methods* work. While 11 (7.3%) participants strongly disagreed and 20 (13.2%) more disagreed, 63 (41.7%) out of 151 were neutral before playing the game. Additionally, 53 (35.1%) participants agreed and 4 (2.6%) more strongly agreed that they know how *methods* work in the pre-study. Having played the game, the number of strongly disagrees and disagrees reduced from 11 (7.3%) to 2 (1.3%), and 20 (13.2%) to 15 (9.9%). The neutral responses also dropped from 63 (41.7%) to 35 (23.2%). Contrarily, the number of agree and strongly agree responses increased from 53 (35.1%) to 80 (53%) and from 4 (2.6%) to 19 (12.6%) correspondingly meaning that participants felt more confident in their knowledge regarding how *methods* work in the post-study.

Among the four assessed programming constructs, the lowest increase happened in *decision making*. In the pre study, 8 (5.3%) participants strongly disagreed and 18 (11.9%) more disagreed that they know how *decision making* construct works. 40 (26.5%) participants neither agreed nor disagreed, 75 (49.7%) participants agreed and 10 (6.6%) more strongly agreed with this statement in the pre-study. In the post study, the number of strongly disagrees reduced from 8 (5.3%) to 1 (0.7%), and the disagrees reduced from 18 (11.9%) to 11 (7.3%). The neutral and agreed responses slightly changed in the study as the neutral responses reduced from 40 (26.5%) to 38 (25.2%) and the agree responses increased from 75 (49.7%) to 76 (50.3%). Lastly, an increase happened in strongly agree responses from 10 (6.6%) to 25 (16.6%).

Loops were the final programming construct asked to participants to rank their own knowledge, and as shown from Figure 7, 13 (8.6%) participants strongly disagreed and 45 (29.8%) more disagreed that they know how *loops* work. 51 (33.8%) of the total responses were neutral whereas 37 (24.5%) participants agreed and 5 (3.3%) more agreed that they know how *loops* work before playing the game. After the game-play, the strongly disagree responses reduced from 13 (8.6%) to 5 (3.3%) and the disagree responses decreased from 45 (29.8%) to 15 (9.9%). Additionally, the neutral answers reduced from 51 (33.8%) to 45 (29.8%). At the same time, the strongly agree and agree responses increased from 37 (24.5%) to 70 (46.4%), and 5 (3.3%) to 16 (10.6%) respectively.

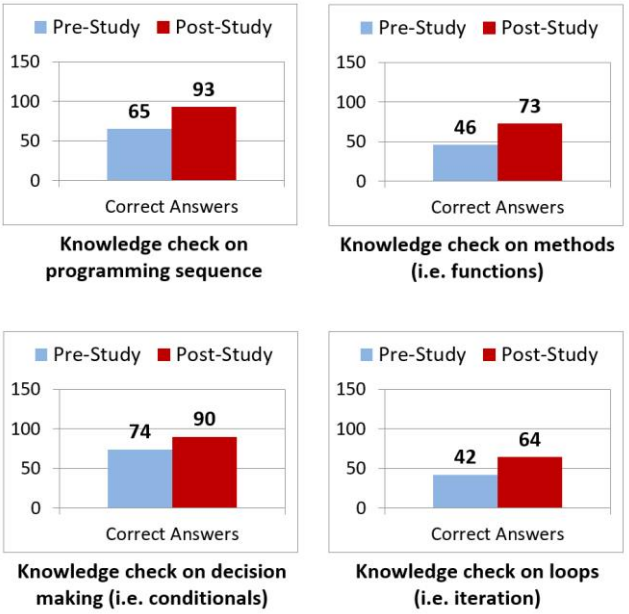
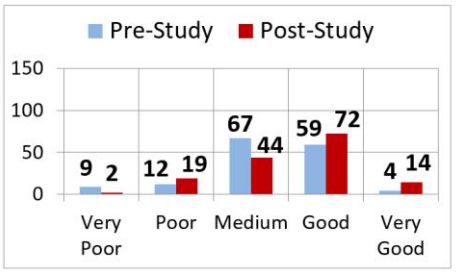


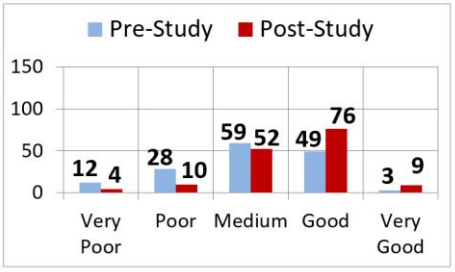
Figure 8. Correct answers given to questions asked in the knowledge check in the pre and in the post study

In addition to participants' raking of their knowledge, each participant was asked a series of knowledge check questions both before and after they played the game. As shown from figure 8, the correct answers given to these knowledge check question increased in every category after participants played the game. In the pre study, the correct answers given to questions were 65 (43%) for *programming sequence*, 46 (30.5%) for *methods*, 74 (49%) for *decision making* and 42 (27.8%) for *loops*. In the post study, these were increased to 93 (61.6%) for *programming sequence*, 73 (48.3%) for *methods*, 90 (59.6%) for *decision making* and 64 (42.4%) for *loops*. The lowest increase in correct answers between the pre and the post study was happened in *decision making*. These results check out with the previous raw data obtained that is participants' perception of their knowledge of programming constructs implemented in the game. Among the four programming constructs, participants' perception of their knowledge in *decision making* was the least one that was improved. Similarly, the knowledge check results show that the lowest increase in the number of correct answers happened in the *decision making* question.

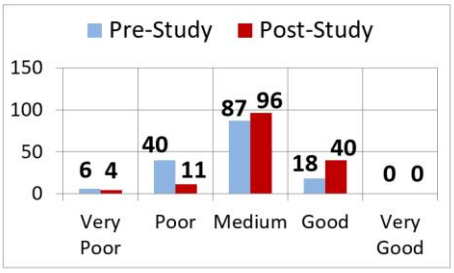
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3043278, IEEE Access



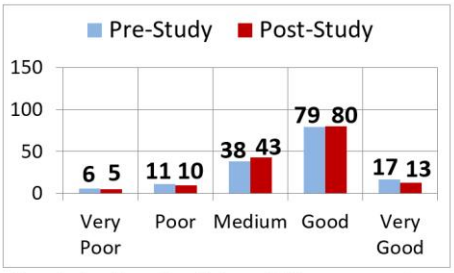
Problem Solving



Constructing Algorithms



Debugging (i.e. identifying and removing errors)



Simulation (i.e. visualising which programming constructs to use and work on from given problems)

Figure 9. Distribution of participants' perception of their CT skills

The final assessment was measuring participants' perception of their CT skills in the pre and in the post study where participants themselves ranked their own CT skills both before and after they played the game.

As displayed in figure 9, 9 (6%) participants ranked their *problem solving* skill as very poor and 12 (7.9%) more ranked this as poor in the pre study. While 67 (44.4%) participants ranked this skill as medium, 59 (39.1%) more ranked as good and 4 (2.6%) more ranked it as very good.

In the post study, the number of very poor responses reduced from 9 (6%) to 2 (1.3%). While the poor responses increased from 12 (7.9%) to 19 (12.6%), the medium responses reduced from 67 (44.4%) to 44 (29.1%). Finally there was an escalation in good and very good responses as these increased from 59 (39.1%) to 72 (47.7%) and from 4 (2.6%) to 14 (9.3%) respectively.

A similar increase happened in *constructing algorithms*. In the pre study, 12 (7.9%) participants ranked this skill as very poor and 28 (18.5%) more ranked it as poor. 59 (39.1%) participants ranked their skill in *constructing algorithm* as medium, 49 (32.5%) more ranked as good, and 3 (2%) more participants ranked it as very good. In the post study, the number of very poor, poor and medium responses were all reduced. The very poor responses decreased from 12 (7.9%) to 4 (2.6%), the poor responses decreased from 28 (18.5%) to 10 (6.6%), and the medium responses decreased from 59 (39.1%) to 52 (34.4%). On the other hand, the good and the very good responses increased from 49 (32.5) to 76 (50.3%) and from 3 (2%) to 9 (6%) correspondingly.

When participants were asked to rank their *debugging* skill, none of them ranked this as very good both in the pre and in the post study. While 6 (4%) participants ranked this skill as very poor, 40 (26.5%) more ranked it as poor in the pre study. Moreover, 87 (57.6%) participants ranked their debugging skill as medium and 18 (11.9%) more ranked this as good in the pre study. In the post study, the very poor and poor responses reduced from 6 (4%) to 4 (2.6%), and from 40 (26.5%) to 11 (7.3%) respectively. At the same time, the medium responses increased from 87 (57.6%) to 96 (63.6%) and the good responses also increased from 18 (11.9%) to 40 (26.5%).

Compared to the other three CT skills (i.e. problem solving, constructing algorithms and debugging), the *simulation* skill did not change very positively in between the pre and the post study. While 6 (4%) participants ranked their simulation skill as very poor, 11 (7.3%) participants ranked this as poor and 38 (25.2%) more ranked this as medium in the pre study. Furthermore, 79 (52.3%) participants ranked their skill in *simulation* as good and 17 (11.3%) more ranked this as very good in the pre study. In the post study, the very poor responses reduced from 6 (4%) to 5 (3.3%), and the poor responses also reduced from 11 (7.3%) to 10 (6.6%). As the medium and good ranks increased from 38 (25.2%) to 43 (28.5%) and 79 (52.3%) to 80 (53%), the very good responses decreased from 17 (11.3%) to 13 (8.6%) in the post study.

The results above suggest that participants' perception on their *problem solving*, *constructing algorithms* and *debugging* improved, but their perception on the *simulation* skill did not improve after their game-play. As the raw data obtained from the study only provided a generic picture about the distribution of the responses, a statistical analysis was conducted to investigate whether a statistically significant change happened in between the pre and the post study.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3043278, IEEE Access

B. STATISTICAL ANALYSIS

A series of paired-samples t-test was conducted to compare the responses collected from the participants in the pre and in the post study in order to answer each research question defined in this study.

Table 3. Paired sample t-test results of participants' attitude to giving up their degree programme, perception of difficulty of computer programming and intrinsic motivation to learn computer programming

Pair	Mean	Std. Deviation	Std. Error Mean	t	df	Sig.
Pair1	0.106	0.403	0.033	3.235	150	**0.001**
Pair2	0.073	0.285	0.023	3.139	150	**0.002**
Pair3	-0.781	1.083	0.088	-8.870	150	**0.0001**
Pair4	-0.338	1.296	0.105	-3.203	150	**0.002**

Pair 1: Pre giving up degree programme – Post giving up degree programme.

Pair 2: Pre giving up degree programme because of the difficulty of comp. programming – Post giving up degree programme because of the difficulty of comp. programming.

Pair 3: Pre difficulty of comp. programming – Post difficulty of comp. programming.

Pair 4: Pre comp. programming intrinsic motivation – Post comp. programming intrinsic motivation.

Table 3 shows the paired t-tests conducted to compare the pre and the post study responses of participants' a) attitude to give up their degree programme (i.e. pair 1); b) opinion on the difficulty of computer programming being a key reason to give up their degree programme (i.e. pair 2); c) perception on the difficulty of computer programming (i.e. pair 3) and d) perception on their intrinsic motivation to learn computer programming (i.e. pair 4). In other words, the paired t-tests were conducted to answer research questions 1,2 and 3. As shown from the above table, there was a statistically significant difference in all measured pairs, namely *the attitude to give up a degree programme* (M=0.106; SD=0.403); *the difficulty of computer programming being a key reason to give up a degree programme* (M=0.073; SD=0.285); *the difficulty of computer programming* (M=-0.781; SD=1.083) and *the intrinsic motivation to learn computer programming* (M=-0.338; SD=1.296) conditions; t(150) = 3.235, p = 0.001 for *the attitude to give up a degree programme*, t(150) = 3.139, p = 0.002 for *the difficulty of computer programming being a key reason to give up a degree programme*, t(150) = -8.870; p = 0.0001 for *the difficulty of computer programming* and t(150)=-3.203; p=0.002 for *the intrinsic motivation to learn computer programming*. These results suggest that there was a statistically significant difference between the pre and the post study responses in all measured categories. Considering the number of positive responses was greater in the post study than in the pre study, it is possible to suggest that participants felt more confident about themselves after playing the game. The paired sample t-test results suggest that participants significantly felt less likely to give up their degree programme in the post study; they perceived computer programming significantly easier; and felt significantly more motivated to learn computer programming than in the pre study. As the results of the paired t-tests were all significant, the alternative hypotheses for research questions 1, 2 and 3 can be accepted which are a) students' attitude to dropping their degree programmes is significantly changed between the pre and the post study; b) students' motivation to learn programming is significantly changed between the pre and the post study and c) students' perception of difficulty of computer programming is significantly changed between the pre and the post study.

Table 4. Paired sample t-test results of participants' perception of their knowledge in programming constructs used in the game

Pair	Mean	Std. Deviation	Std. Error Mean	t	df	Sig.
Pair1	-0.57	1.092	0.089	-6.406	150	**0.0001**
Pair2	-0.53	1.1	0.09	-5.917	150	**0.0001**
Pair3	-0.338	1.095	0.089	-3.791	150	**0.0001**
Pair4	-0.669	1.165	0.095	-7.058	150	**0.0001**

Pair 1: Pre perception of programming sequence – Post perception of programming sequence.

Pair 2: Pre perception of methods – Post perception of methods.

Pair 3: Pre perception of decision making – Post perception of decision making.

Pair 4: Pre perception of loops – Post perception of loops.

Table 4 shows the results of the paired sample t-tests of participants' perception of their knowledge in four programming constructs used in the game. As shown in the table, there was a statistically significant change in *programming sequence* (M=-0.57, SD=1.092); *methods* (M=-0.53; SD=1.1); *decision making* (M=-0.338; SD=1.095); and *loops* (M=-0.669; SD=1.165) conditions; t(150)=-6.406, p=0.0001 for *programming sequence*, t(150)=-5.917, p=0.0001 for *methods*, t(150)=-3.791, p=0.0001 for *decision making* and t(150)=-7.058, p=0.0001 for *loops*. These findings show that participants felt that their knowledge of programming constructs significantly changed in the post study when compared to the pre study. As the responses in the post study were more positive than the responses in the pre study, it is possible to suggest that participants' perception of their knowledge increased during the study.

In addition to their perception of knowledge, participants' tangible knowledge of programming constructs was also assessed in the study. As displayed in Table 5, the answers given to the knowledge check questions were found to be statistically significant for all programming constructs. According to the paired sample t-tests results, there was a statistically significant change in terms of knowledge in *programming sequence* (M=-0.185, SD=0.559); *methods* (M=-0.179; SD=0.623); *decision making* (M=-0.106;

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3043278, IEEE Access

SD=0.623); and *loops* (M=-0.146; SD=0.57); t(150)=-4.079, p=0.0001 for *programming sequence*; t(150)=-3.528, p=0.001 for *methods*; t(150)=-2.088, p=0.038 for *decision making* and t(150)=-3.139, p=0.002 for *loops*.

As participants' perception of their knowledge on the programming constructs and their knowledge check results were found to be statistically significant, it is possible to suggest the alternative hypothesis for research question 4 that is students' understanding of programming constructs is significantly changed between the pre and the post study. Additionally, in all significant cases the post study mean was higher than the pre study mean which is an indication that participants' perception of their knowledge and their tangible knowledge regarding how programming constructs work were enhanced in the study.

Table 5. Paired sample t-test results of participants' knowledge check of programming constructs used in the game

Pair	Mean	Std. Deviation	Std. Error Mean	t	df	Sig.
Pair1	-0.185	0.559	0.045	-4.079	150	**0.0001**
Pair2	-0.179	0.623	0.051	-3.528	150	**0.001**
Pair3	-0.106	0.623	0.051	-2.088	150	**0.038**
Pair4	-0.146	0.570	0.046	-3.139	150	**0.002**

Pair 1: Pre knowledge of programming sequence – Post knowledge of programming sequence.

Pair 2: Pre knowledge of methods – Post knowledge of methods.

Pair 3: Pre knowledge of decision making – Post knowledge of decision making.

Pair 4: Pre knowledge of loops – Post knowledge of loops.

The very last paired t-test samples were conducted on participants' perception of their CT skills. As shown in Table 6, there was a statistically significant difference in *problem solving* (M=-0.265, SD=1.063); *constructing algorithms* (M=-0.483; SD=0.958) and *debugging* (M=-0.364; SD=0.583) conditions; t(150)=-3.063, p=0.003 for *problem solving*, t(150)=-6.2, p=0.0001 for *constructing algorithms,* and t(150)=-7.678, p=0.0001 for *debugging*. On the other hand, there was no statistically significant change in *simulation* (M=0.026, SD=0.887), t(150)=0.367, p=0.714. The mean scores show that participants ranked their *simulation* skills higher in the pre study than in the post study which suggests that they did not feel any positive change in terms of their *simulation* skill during the study. As the statistical outcome of the *simulation* skill (i.e. pair 4) was not statistically significant, the fifth research question can only be partially answered. As a result, participants felt a significant change in between the pre and the post study regarding their *problem solving*, *constructing algorithms,* and *debugging* abilities. However, the same statistically significant change was not observed in *simulation*.

Table 6. Paired sample t-test results of participants' perception of their CT skills

Pair	Mean	Std. Deviation	Std. Error Mean	t	df	Sig.
Pair1	-0.265	1.063	0.086	-3.063	150	**0.003**
Pair2	-0.483	0.958	0.078	-6.200	150	**0.0001**
Pair3	-0.364	0.583	0.047	-7.678	150	**0.0001**
Pair4	0.026	0.887	0.072	0.367	150	0.714

Pair 1: Pre problem solving – Post problem solving.

Pair 2: Pre constructing algorithms – Post constructing algorithms.

Pair 3: Pre debugging – Post debugging.

Pair 4: Pre simulation – Post simulation.

Finally, a series of multiple linear regression (MLR) analysis was conducted to investigate the effect of age, sex, ethnicity, and mathematical qualifications of participants on their perception and knowledge scores in the study. The core reason behind this was to measure the effect of these independent variables on the mean scores of dependent variables so that it can be detected whether the characteristics of the participants impacted the results. The difference between the pre and post study results were individually investigated for each research question.

Table 7. Multiple Linear Regression Analysis on participants' attitude to giving up their degree programme

Model	Sum of Squares	df	Mean Square	F	Sig.
Regression	0.413	4	0.103	0.631	0.641
Residual	23.892	146	0.164		
Total	24.305	150			

Constant	Coefficient Std. Error	Standardized Coefficient Beta	t	Sig.
(Constant)	0.175		-0.297	0.767
Sex	0.091	0.066	0.794	0.429
Age Range	0.070	-0.040	-0.477	0.634
Ethnicity	0.024	-0.106	-1.275	0.204
Mathematical Qualification	0.025	-0.011	-0.132	0.895

As displayed in Table 7, when the effects of age, sex, ethnicity, and mathematical qualifications (referred to as predictors) were investigated in the study, it was found that none of the predictors had any significant effect on the attitude of the participants for giving up their degree programme, $F(4,146)=0.631$, $p=0.641$, $R^2=0.017$. In other words, age ($\beta=-0.04$, $p=0.634$), sex ($\beta=0.066$, $p=0.429$), ethnicity ($\beta=-0.106$, $p=0.204$) and mathematical qualifications ($\beta=-0.011$, $p=0.895$) did not have any significant impact on the participants' decision to give up on their degree programme.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3043278, IEEE Access

Table 8. Multiple Linear Regression Analysis on participants' intrinsic motivation to learn computer programming

Model	Sum of Squares	df	Mean Square	F	Sig.
Regression	14.605	4	3.651	2.248	0.067
Residual	237.170	146	1.624		
Total	251.775	150			

Constant	Coefficient Std. Error	Standardized Coefficient Beta	t	Sig.
(Constant)	0.550		-0.193	0.847
Sex	0.286	0.202	2.493	**0.014**
Age Range	0.220	0.055	0.661	0.510
Ethnicity	0.077	-0.116	-1.428	0.155
Mathematical Qualification	0.079	-0.084	-1.018	0.310

Further to above, a MLR was run to investigate whether the predictors impacted the change in intrinsic motivation to learn computer programming. As shown in Table 8, the results revealed that sex (β=0.202, p=0.014) was an independent variable affecting the outcome. However, the ANOVA test results did not show a statistically significant outcome F(4,146)= 2.248, p=0.067, R^2=0.058 and thus, the linear model did not work. In this case, it is accepted that the effect of sex to the participants' intrinsic motivation to learn programming was negligible.

Table 9. Multiple Linear Regression Analysis on participants' perception on the difficulty of computer programming

Model	Sum of Squares	df	Mean Square	F	Sig.
Regression	15.531	4	3.883	3.537	**0.009**
Residual	160.257	146	1.098		
Total	175.788	150			

Constant	Coefficient Std. Error	Standardized Coefficient Beta	t	Sig.
(Constant)	0.452		-0.569	0.570
Sex	0.235	0.169	2.119	**0.036**
Age Range	0.180	0.256	3.154	**0.002**
Ethnicity	0.065	-0.024	-0.293	0.770
Mathematical Qualification	0.063	-0.066	-0.826	0.410

Contrary to the previous findings, the predictors had a statistically significant impact on how difficult the participants found computer programming, F(4,146)=3.537, p=0.009, R^2=0.088. Table 9 shows that age range (β=0.256, p=0.002), and sex (β=0.235, p=0.036) were the primary predictors that added statistically significantly to the prediction. When the impact of the age range and sex was individually investigated with a linear regression, it was found that sex did not have a significant impact (β=0.138, p=0.099), but the age range had a significant impact (β=0.236, p=0.03). In other words, the linear regression results show that the older the participants were the more difficult they found computer programming in this study.

Table 10. Multiple Linear Regression Analysis on participants' perception of their knowledge in programming constructs

Model	Sum of Squares	df	Mean Square	F	Sig.
Regression	3.961	4	0.990	1.240	0.296
Residual	116.559	146	0.798		
Total	120.519	150			

Constant	Coefficient Std. Error	Standardized Coefficient Beta	t	Sig.
(Constant)	0.386		0.255	0.799
Sex	0.200	0.174	2.128	0.035
Age Range	0.154	0.040	0.483	0.630
Ethnicity	0.054	-0.043	-0.528	0.599
Mathematical Qualification	0.055	-0.037	-0.438	0.662

Table 11. Multiple Linear Regression Analysis on participants' knowledge check of programming constructs used in the game

Model	Sum of Squares	df	Mean Square	F	Sig.
Regression	1.983	4	0.496	4.288	**0.003**
Residual	16.875	146	0.116		
Total	18.858	150			

Constant	Coefficient Std. Error	Standardized Coefficient Beta	t	Sig.
(Constant)	0.147		1.263	0.208
Sex	0.076	0.176	2.235	**0.027**
Age Range	0.059	0.047	0.586	0.559
Ethnicity	0.020	-0.269	-3.400	**0.001**
Mathematical Qualification	0.021	-0.090	-1.124	0.263

The MLR analysis conducted on the participants' perception of knowledge and their tangible knowledge in programming constructs used in the game presented some interesting results. As displayed in Table 10, none of the predictors impacted to the participants' perception of their knowledge, F(4,146)=1.24, p=0.296, R^2=0.033. However, a MLR analysis conducted on the knowledge check answers revealed that sex (β=0.176, p=0.027) and ethnicity (β=-0.269, p=0.001) were statistically significant predictors added to the outcome, F(4,146)=4.288, p=0.003, R^2=0.105. When independent linear regressions were conducted, it was revealed that both sex (β=0.171, p=0.03) and ethnicity (β=-0.26, p=0.01) added statistically significantly to the prediction. The findings of the linear regression analysis suggest that male participants provided more correct answers to the knowledge check test in the study than female participants. Additionally, participants that had a white background provided more correct answers than any other ethnicity. As the majority of the participants in the study defined themselves as *white* (41.1%) and *male* (84.1%), these results were not unexpected.

A final MLR analysis was run on the mean scores of the CT skills of the participants and as displayed in Table 12,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3043278, IEEE Access

the findings show that none of the predictors had statistically significant impact on the outcome regarding the CT skills, $F(4,146)=0.888$, $p=0.473$, $R^2=0.024$. In other words, sex ($\beta=0.059$, $p=0.475$), age range ($\beta=-0.032$, $p=0.707$), ethnicity ($\beta=0.107$, $p=0.196$) and mathematical qualifications ($\beta=-0.097$, $p=0.249$) did not have any statistically significant impact on how the participants perceived their CT skills in this study.

Table 12. Multiple Linear Regression Analysis on participants' perception of their CT skills

Model	Sum of Squares	df	Mean Square	F	Sig.
Regression	0.909	4	0.227	0.888	0.473
Residual	37.334	146	0.256		
Total	38.243	150			

Constant	Coefficient Std. Error	Standardized Coefficient Beta	t	Sig.
(Constant)	0.218		2.000	0.047
Sex	0.113	0.059	0.716	0.475
Age Range	0.087	-0.032	-0.377	0.707
Ethnicity	0.030	-0.107	-1.300	0.196
Mathematical Qualification	0.031	-0.097	-1.157	0.249

To summarize the findings, the followings were statistically and significantly changed in between the pre and the post study: a) the participants were less likely to give up their degree programme; b) the participants' intrinsic motivation to learn computer programming increased; c) the participants perceived computer programming to be less difficult than before and d) the participants' perception of their knowledge and their tangible knowledge on computer programming constructs used in the game were improved. Additionally, it was found that the older participants found learning computer programming significantly more difficult than the younger ones. It was also found that the participants' self-confidence was positively correlated to their knowledge check scores which is complementary to previous research done in this area [89]. Finally, sex and ethnicity of the participants seemingly impacted to their success in knowledge check questions as the results showed that *white* and *male* participants were more successful than the others. However, this might be true because majority of the participants who completed the study were *male* and *white*.

VII. DISCUSSION

The findings obtained from this study are complementary to the results of the previous work [34], [62], [76] that is a) game-play can be used to enhance students' motivation for learning programming [41], [90] and b) game-play can inspire students' confidence and learning experience of CT skills which then can support their learning of computer programming [64].

Despite the statistically significant outcomes obtained from this study, there were several limitations in this research the lack of a control group being the major drawback. As explained before, the University Ethics Committee (UREC) insisted on conducting this study where all students received the same experience and none of them would be able to advance through a specific type of intervention. Due to this, it was simply not suitable to separate students into two equal random groups and conduct the study in proportionately divided environments. Additionally, when the study was scheduled to be conducted, the students were just registered to the computer programming course and their programming skill as well as their gaming background were not precisely known. Therefore, there was no control group in this study and because of this, the research became particularly vulnerable to internal threats. In other words, while the results of this research show that there is a statistically significant improvement in between the pre and the post study, it is not possible to tie these results directly to the game-play of participants because of the weaknesses of the quasi experimental structure.

Another challenge for this research was the dearth of a universally agreed conventional way of teaching CT skills to students as this is an experimental research area. Although many researchers proposed that teaching CT should be a part of the curriculum [91]–[93], this idea has not yet embraced ubiquitously, and thus this impacts on how research studies are designed because it is debatable which kind of intervention can or cannot be used in a control group.

As the lack of the control group was a known issue, further investigation on the responses was conducted by interviewing several participants and asking their opinion on whether or not a) the game was genuinely helpful to them; b) using *Program Your Robot* is a good idea to support their tutorials; c) they would like to see improvements in the game and if so, what type of improvements. Many participants provided constructive and positive feedback to these questions with an optimistic attitude. Majority of them indicated that the game provided a simplified structure of how the programming constructs work and if the game is improved, it could be a fun way of learning programming. Some quotes from these students are cited below to provide qualitative evidence that they found the game well suited to help them to understand how introductory programming constructs works:

"I think this game is going in a good path. I would like it to have more levels that would develop my ideas of computer programming even better. You could also include on the side how the code would look like if it was run in Java (or any other language), this would give insight to the player of how that "program" would look like as the real thing. Another idea would be a save button so that users could track their progress in learning, maybe even have multiplayer option."

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3043278, IEEE Access

"I think this game is useful for practicing computer algorithms and learning basic programming. I especially found loops useful and efficiently applied in the game."

"This game is good but needs improvement. When a specific function is executed, this should be highlighted in all modes. This is only done in debug mode but not in run mode. Also, at the end of each level there could be an example of a real programming code showing up."

"I think beating this game requires logical thinking. We should be given a coursework or exercises to play similar games. This can make learning programming more interesting and fun."

"I thought it was a simple and relatable way of teaching us how sequences and methods work. I want to use this game at home to practice."

The above responses clearly show that the participants critically analyzed the game and found it useful.

Moreover, several precautions were taken to minimize the possible threats to internal validity before this study was conducted. Firstly, all participants were made aware that they always had the option to withdraw at any time if they felt tired and their decision of participating in this research or not would have no adverse effect on their studies. Secondly, the participants were randomly invited to this research and although this was not well-known before conducting the study, the population had diverse background and knowledge in computer programming and in playing games. Lastly, the participants completed the study at their own pace and were allowed to play the game as much as they wanted.

These precautions undermined the *maturation threat* to some extent which is a threat that happens when subjects become tired, bored or in any other state that they can no longer pay attention to the study. Another threat to consider was the *mortality threat* which impacts the outcome of a study when too many participants drop out as those who drop out usually provide negative feedback. Hence, if too many participants drop out from a study, this often means losing a considerable number of negative responses. As discussed in the earlier sections, only 39 (20.5%) participants dropped out from this study which is not an exceedingly high number when considering this type of research. While vast majority of these participants left their questionnaires empty, it was observed that those who provided some responses did not respond with negative feedback. As a matter of fact, many participants indicated that they had fun participating in this study. Finally, the *regression threat* is deeply considered in this research which is a statistical phenomenon that occurs whenever a randomly selected population for a study is discovered to be a non-random sample with extreme scores. In other words, a *regression threat* endangers a study when subjects are selected because of extreme scores (either high or low) that might impact the outcome of a study. As an example, if participants were selected based on their extremely low knowledge in computer programming in this study, the improvements at the end of the study might be due to regression toward the mean rather than the game's effectiveness as in reality participants cannot know any lower than they already know in computer programming. This is to say when a sample is selected just because of *low performance,* any corrective measures applied will be very likely to get the scores up simply because of regression toward the mean and not because of any real improvement due to game intervention. However, this was not the case in this research because 19 out of 151 (12.5%) participants answered the pre study knowledge check questions all correctly, and 68 (45%) more answered at least half of the knowledge check questions correctly in the pre study. Moreover, 7 (4.6%) participants answered all knowledge check questions correctly both in the pre and in the post study. These results show that the participants came from a wide variety of background in computer programming and thus, the regression towards the mean argument is invalid. As presented in section VI, a series of multiple linear regression analysis investigated the potential effects of independent variables to the results, and it was found that in most cases, these independent variables did not have a statistically significant impact.

An important outcome of this research is that the mathematical qualifications of participants did not have a statistically significant prediction on any of the results. Contrary to some of the previous work conducted in this area [94]–[97], the participants' mathematical qualifications was not a statistically significant predictor on their a) perspective on the difficulty of computer programming; b) intrinsic motivation in learning computer programming; c) perception of knowledge in programming constructs used in the game and d) knowledge check scores.

Despite the lack of a control group, participants' quantitative responses, qualitative feedback, and the analysis on the internal validity provided strong reasons to consider that the game indeed affected their motivation and confidence positively in learning computer programming. Additionally, it was found that majority of the participants showed confidence in using their CT skills in the study particularly in *problem solving*, *constructing algorithms*, and *debugging*.

VIII. CONCLUSION AND FUTURE WORK

This paper discusses an adhoc game called *Program Your Robot* that was specifically designed to support CT skills of its player and thus, facilitate their learning of computer programming. A quasi pre-post experimental study was conducted to assess the potential effect of this game by collecting 151 responses from the participants who were all first-year computer programming students registered at University of Greenwich. The data generated from this

study was deeply analyzed and the statistically significant findings were highlighted.

Based on the outcomes of this study, two important contributions can be considered. The first one is *the statistically significant evidence that video games can underpin the development of major CT skills and facilitate learning computer programming at the CT level*. The second one is the *game design* contribution which is presented in the form of a serious game specifically focused to practice a) CT skills for puzzle solving and b) programming constructs as an integral part of the game-play mechanism.

While the missing control group was a big drawback of this research, the findings from the quantitative data analysis and the feedback received from the participants provided strong reasons to believe that *Program Your Robot* significantly encouraged participants' confidence and motivation both in using their major CT skills and in learning computer programming. Therefore, certain game mechanisms of the game such as those implemented to reinforce *problem solving*, *constructing algorithms,* and *debugging* seemingly helped the participants' confidence and their learning process of computer programming.

As future work, this research needs further improvement in several areas.

Firstly, a new version of the game is being planned to be developed in Unity game engine with a new mobile friendly interface. The current version of the game is not mobile friendly and needs to be easily accessible. For this reason, it is planned to redesign the game as a mobile application with modern features such as with user profiles that save online data about players' progress, an in game high score system and with challenging levels. This new version of the game will also provide an improved visualization for programming constructs. As an example, players will be able to use the *decision making* construct in a variety of ways depending on how they want to overcome challenges (e.g. bypass an enemy or incapacitate it). Some of the suggestions proposed by the participants of this study will also be applied into the game-play such as the new version of the game will support multiple players to interact with one another during their game-play. Therefore, the social aspect of CT, and how this can impact to other skills of the players can be investigated.

Secondly, the current version of the game was designed to operate at an operational level of abstraction to practice how programming constructs work and therefore, the game does not produce code in a specific programming language. It is important to indicate that the game's operational stepwise refinement approach could be described in pseudo-code, which could then be utilized with a code generator to produce programming language-specific code. This was not within the current scope of this research but will be a future development in the game.

Thirdly, only the short-term effects of the game-play was investigated in this study. An enhanced study in the future will aim to investigate both short- and long-term effects of the game-play and how this would potentially reinforce players' CT skills.

Finally, the most important future work includes obtaining a strong ethical approve for this ongoing research in the premise that this would open pathways to conduct a double-blind study with a control and an experimental group. To achieve this, a new ethical application would be undertaken to create a gold standard experimental design. Therefore, a strong experimental structure could be established to provide even more strong statistical evidence.

REFERENCES

[1] "Higher Education Statistics Agency (HESA)," 2018. [Online]. Available: https://www.hesa.ac.uk/news/11-01-2018/sfr247-higher-education-student-statistics/subjects.

[2] T. Beaubouef and J. Mason, "Why the high attrition rate for computer science students: some thoughts and observations," *ACM SIGCSE Bull.*, vol. 37, no. 2, pp. 103–106, 2005.

[3] C. Watson and F. W. B. Li, "Failure rates in introductory programming revisited," in *Proceedings of the 2014 conference on Innovation & technology in computer science education*, 2014, pp. 39–44.

[4] E. B. Costa, B. Fonseca, M. A. Santana, F. F. de Araújo, and J. Rego, "Evaluating the effectiveness of educational data mining techniques for early prediction of students' academic failure in introductory programming courses," *Comput. Human Behav.*, vol. 73, pp. 247–256, 2017.

[5] M. N. Giannakos, I. O. Pappas, L. Jaccheri, and D. G. Sampson, "Understanding student retention in computer science education: The role of environment, gains, barriers and usefulness," *Educ. Inf. Technol.*, vol. 22, no. 5, pp. 2365–2382, 2017.

[6] R. P. Medeiros, G. L. Ramalho, and T. P. Falcão, "A systematic literature review on teaching and learning introductory programming in higher education," *IEEE Trans. Educ.*, vol. 62, no. 2, pp. 77–90, 2018.

[7] M. McCracken *et al.*, "A multi-national, multi-institutional study of assessment of programming skills of first-year CS students," in *Working group reports from ITiCSE on Innovation and technology in computer science education*, 2001, pp. 125–180.

[8] R. Lister *et al.*, "A multi-national study of reading and tracing skills in novice programmers," *ACM SIGCSE Bull.*, vol. 36, no. 4, pp. 119–150, 2004.

[9] P. Kinnunen and L. Malmi, "Why students drop out CS1 course?," in *Proceedings of the second international workshop on Computing education research*, 2006, pp. 97–108.

[10] M. Armoni, O. Meerbaum-Salant, and M. Ben-Ari, "From scratch to 'real' programming," *ACM Trans. Comput. Educ.*, vol. 14, no. 4, pp. 1–15, 2015.

[11] A. Altadmri and N. C. C. Brown, "37 million compilations: Investigating novice programming mistakes in large-scale student data," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 2015, pp. 522–527.

[12] J. Voogt, P. Fisser, J. Good, P. Mishra, and A. Yadav, "Computational thinking in compulsory education: Towards an agenda for research and practice," *Educ. Inf. Technol.*, vol. 20, no. 4, pp. 715–728, 2015.

[13] J. Bonar and E. Soloway, "Uncovering principles of novice programming," in *Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, 1983, pp. 10–13.

[14] J. Bennedsen and M. E. Caspersen, "Failure rates in introductory programming," *AcM SIGcSE Bull.*, vol. 39, no. 2, pp. 32–36, 2007.

[15] L. Porter, M. Guzdial, C. McDowell, and B. Simon, "Success in introductory programming: What works?," *Commun. ACM*, vol. 56, no. 8, pp. 34–36, 2013.

[16] N. J. Coull and I. M. M. Duncan, "Emergent requirements for supporting introductory programming," *Innov. Teach. Learn. Inf. Comput. Sci.*, vol. 10, no. 1, pp. 78–85, 2011.

[17] C.-C. Liu, Y.-B. Cheng, and C.-W. Huang, "The effect of simulation games on the learning of computational problem solving," *Comput. Educ.*, vol. 57, no. 3, pp. 1907–1918, 2011.

[18] L. J. Barker, C. McDowell, and K. Kalahar, "Exploring factors that influence computer science introductory course students to persist in the major," *ACM Sigcse Bull.*, vol. 41, no. 1, pp. 153–157, 2009.

[19] A. Ali and C. Shubra, "Efforts to reverse the trend of enrollment decline in computer science programs," *Issues Informing Sci. Inf. Technol.*, vol. 7, pp. 209–224, 2010.

[20] I. Ouahbi, F. Kaddari, H. Darhmaoui, A. Elachqar, and S. Lahmine, "Learning basic programming concepts by creating games with scratch programming environment," *Procedia-Social Behav. Sci.*, vol. 191, pp. 1479–1482, 2015.

[21] S. Combéfis, G. Beresnevičius, and V. Dagienė, "Learning programming through games and contests: overview, characterisation and discussion," *Olympiads in Informatics*, vol. 10, no. 1, pp. 39–60, 2016.

[22] Y. B. Kafai and Q. Burke, "Constructionist gaming: Understanding the benefits of making games for learning," *Educ. Psychol.*, vol. 50, no. 4, pp. 313–334, 2015.

[23] T. Y. Lee, M. L. Mauriello, J. Ahn, and B. B. Bederson, "CTArcade: Computational thinking with games in school age children," *Int. J. Child-Computer Interact.*, vol. 2, no. 1, pp. 26–33, 2014.

[24] Y.-H. Ching, Y.-C. Hsu, and S. Baldwin, "Developing computational thinking with educational technologies for young learners," *TechTrends*, vol. 62, no. 6, pp. 563–573, 2018.

[25] P.-Y. Chao, "Exploring students' computational practice, design and performance of problem-solving through a visual programming environment," *Comput. Educ.*, vol. 95, pp. 202–215, 2016.

[26] "Scratch," 2020. [Online]. Available: https://scratch.mit.edu/. [Accessed: 13-Aug-2020].

[27] "Alice," 2020. [Online]. Available: https://www.alice.org/. [Accessed: 13-Aug-2020].

[28] O. Erol and A. A. Kurt, "The effects of teaching programming with scratch on pre-service information technology teachers' motivation and achievement," *Comput. Human Behav.*, vol. 77, pp. 11–18, 2017.

[29] L. A. Calao, J. Moreno-León, H. E. Correa, and G. Robles, "Developing mathematical thinking with scratch," in *Design for teaching and learning in a networked world*, Springer, 2015, pp. 17–27.

[30] D. Topalli and N. E. Cagiltay, "Improving programming skills in engineering education through problem-based game projects with Scratch," *Comput. Educ.*, vol. 120, pp. 64–74, 2018.

[31] K. Sung, C. Hillyard, R. L. Angotti, M. W. Panitz, D. S. Goldstein, and J. Nordlinger, "Game-themed programming assignment modules: A pathway for gradual integration of gaming context into existing introductory programming courses," *IEEE Trans. Educ.*, vol. 54, no. 3, pp. 416–427, 2010.

[32] B. Wu and A. I. Wang, "A guideline for game development-based learning: a literature review," *Int. J. Comput. Games Technol.*, vol. 2012, 2012.

[33] N. Pellas and S. Vosinakis, "How can a simulation game support the development of computational problem-solving strategies?," in *2017 IEEE Global Engineering Education Conference (EDUCON)*, 2017, pp. 1129–1136.

[34] S. I. Ch'ng, Y. C. Low, Y. L. Lee, W. C. Chia, and L. S. Yeong, "Video Games: A Potential Vehicle for Teaching Computational Thinking," in *Computational Thinking Education*, Springer, Singapore, 2019, pp. 247–260.

[35] C. Kazimoglu, M. Kiernan, L. Bacon, and L. Mackinnon, "A serious game for developing computational thinking and learning introductory computer programming," *Procedia-Social Behav. Sci.*, vol. 47, pp. 1991–1999, 2012.

[36] F. Bellotti, B. Kapralos, K. Lee, P. Moreno-Ger, and R. Berta, "Assessment in and of serious games: an overview," *Adv. human-computer Interact.*, vol. 2013, 2013.

[37] X. Jiang, C. Harteveld, X. Huang, and A. Y. H. Fung, "The computational puzzle design framework: a design guide for games teaching computational thinking," in *Proceedings of the 14th International Conference on the Foundations of Digital Games*, 2019, pp. 1–11.

[38] P. Felicia, *Handbook of research on improving learning and motivation through educational games: Multidisciplinary approaches: Multidisciplinary approaches*. iGi Global, 2011.

[39] T. Mitamura, Y. Suzuki, and T. Oohori, "Serious games for learning programming languages," in *2012 IEEE international conference on systems, man, and cybernetics (SMC)*, 2012, pp. 1812–1817.

[40] T. Hainey *et al.*, "Students' attitudes toward playing games and using games in education: Comparing Scotland and the Netherlands," *Comput. Educ.*, vol. 69, pp. 474–484, 2013.

[41] P. Molins-Ruano, C. Sevilla, S. Santini, P. A. Haya, P. Rodríguez, and G. M. Sacha, "Designing videogames to improve students' motivation," *Comput. Human Behav.*, vol. 31, pp. 571–579, 2014.

[42] J. Sinclair, M. Butler, M. Morgan, and S. Kalvala, "Measures of student engagement in computer science," in *Proceedings of the 2015 ACM conference on innovation and technology in computer science education*, 2015, pp. 242–247.

[43] J. Bennedsen and M. E. Caspersen, "Failure rates in introductory programming: 12 years later," *ACM Inroads*, vol. 10, no. 2, pp. 30–36, 2019.

[44] M. J. Lee and A. J. Ko, "Comparing the effectiveness of online learning approaches on CS1 learning outcomes," in *Proceedings of the eleventh annual international conference on international computing education research*, 2015, pp. 237–246.

[45] N. Dalal, P. Dalal, S. Kak, P. Antonenko, and S. Stansberry, "Rapid digital game creation for broadening participation in computing and fostering crucial thinking skills," *Int. J. Soc. Humanist. Comput.*, vol. 1, no. 2, pp. 123–137, 2009.

[46] C. Kazimoglu, "Empirical evidence that proves a serious game is an educationally effective tool for learning computer programming constructs at the computational thinking level." University of Greenwich, 2013.

[47] R. Rajaravivarma, "A games-based approach for teaching the introductory programming course," *ACM SIGCSE Bull.*, vol. 37, no. 4, pp. 98–102, 2005.

[48] I. Lee, F. Martin, and K. Apone, "Integrating computational thinking across the K--8 curriculum," *Acm Inroads*, vol. 5, no. 4, pp. 64–71, 2014.

[49] S. Papert, "An exploration in the space of mathematics educations," *Int. J. Comput. Math. Learn.*, vol. 1, no. 1, pp. 95–123, 1996.

[50] J. M. Wing, "Computational thinking," *Commun. ACM*, vol. 49, no. 3, pp. 33–35, 2006.

[51] J. M. Wing, "Computational thinking and thinking about computing," *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.*, vol. 366, no. 1881, pp. 3717–3725, 2008.

[52] A. Basawapatna, K. H. Koh, A. Repenning, D. C. Webb, and K. S. Marshall, "Recognizing computational thinking patterns," in *Proceedings of the 42nd ACM technical symposium on Computer science education*, 2011, pp. 245–250.

[53] G. Michaelson, "Teaching programming with computational and informational thinking," *J. Pedagog. Dev.*, 2015.

[54] P. Curzon and P. W. McOwan, *The power of computational thinking: Games, magic and puzzles to help you become a computational thinker*. World Scientific, 2017.

[55] M. L. Wu and K. Richards, "Facilitating computational thinking through game design," in *International Conference on Technologies for E-Learning and Digital Entertainment*, 2011, pp. 220–227.

[56] J. F. Roscoe, S. Fearn, and E. Posey, "Teaching computational thinking by playing games and building robots," in *2014 International Conference on Interactive Technologies and Games*, 2014, pp. 9–12.

[57] D. Weintrop, N. Holbert, M. S. Horn, and U. Wilensky, "Computational thinking in constructionist video games," *Int. J. Game-Based Learn.*, vol. 6, no. 1, pp. 1–17, 2016.

[58] J. Moreno-León, G. Robles, and M. Román-González, "Dr. Scratch: Automatic analysis of scratch projects to assess and

foster computational thinking," *RED. Rev. Educ. a Distancia*, no. 46, pp. 1–23, 2015.

[59] M. J. Marcelino, T. Pessoa, C. Vieira, T. Salvador, and A. J. Mendes, "Learning computational thinking and scratch at distance," *Comput. Human Behav.*, vol. 80, pp. 470–477, 2018.

[60] A. Settle, "Computational thinking in a game design course," in *Proceedings of the 2011 conference on Information technology education*, 2011, pp. 61–66.

[61] P. Boechler, C. Artym, E. Dejong, M. Carbonaro, and E. Stroulia, "Computational thinking, code complexity, and prior experience in a videogame-building assignment," in *2014 IEEE 14th International Conference on Advanced Learning Technologies*, 2014, pp. 396–398.

[62] M. Muratet, P. Torguet, F. Viallet, and J.-P. Jessel, "Experimental feedback on Prog&Play: a serious game for programming practice," in *Computer Graphics Forum*, 2011, vol. 30, no. 1, pp. 61–73.

[63] A. Vahldick, A. J. Mendes, and M. J. Marcelino, "A review of games designed to improve introductory computer programming competencies," in *2014 IEEE frontiers in education conference (FIE) proceedings*, 2014, pp. 1–7.

[64] W. Zhao and V. J. Shute, "Can playing a video game foster computational thinking skills?," *Comput. Educ.*, vol. 141, p. 103633, 2019.

[65] A. Repenning *et al.*, "Scalable game design: A strategy to bring systemic computer science education to schools through game design and simulation creation," *ACM Trans. Comput. Educ.*, vol. 15, no. 2, p. 11, 2015.

[66] O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari, "Habits of programming in scratch," in *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, 2011, pp. 168–172.

[67] S. Leutenegger and J. Edgington, "A games first approach to teaching introductory programming," in *Proceedings of the 38th SIGCSE technical symposium on Computer science education*, 2007, pp. 115–118.

[68] "RoboCode," 2001. [Online]. Available: https://robocode.sourceforge.io/. [Accessed: 13-Aug-2020].

[69] M. Eagle and T. Barnes, "Experimental evaluation of an educational game for improved learning in introductory computing," *ACM SIGCSE Bull.*, vol. 41, no. 1, pp. 321–325, 2009.

[70] "Colobot," 2007. [Online]. Available: http://www.ceebot.com/colobot/index-e.php. [Accessed: 13-Aug-2020].

[71] L. A. Gouws, K. Bradshaw, and P. Wentworth, "Computational thinking in educational activities: an evaluation of the educational game light-bot," in *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, 2013, pp. 10–15.

[72] D. Weintrop and U. Wilensky, "Robobuilder: a computational thinking game," in *SIGCSE*, 2013, p. 736.

[73] "Code Combat," 2013. [Online]. Available: https://codecombat.com/. [Accessed: 19-Aug-2020].

[74] "Code Spells," 2014. [Online]. Available: https://codespells.org/. [Accessed: 13-Aug-2020].

[75] C. Kazimoglu, M. Kiernan, L. Bacon, and L. MacKinnon, "Learning programming at the computational thinking level via digital game-play," *Procedia Comput. Sci.*, vol. 9, pp. 522–531, 2012.

[76] M. J. Lee and A. J. Ko, "Personifying programming tool feedback improves novice programmers' learning," in *Proceedings of the seventh international workshop on Computing education research*, 2011, pp. 109–116.

[77] M. Soflano, T. M. Connolly, and T. Hainey, "An application of adaptive games-based learning based on learning style to teach SQL," *Comput. Educ.*, vol. 86, pp. 192–211, 2015.

[78] C. Kazimoglu, M. Kiernan, L. Bacon, and L. MacKinnon, "Understanding computational thinking before programming: developing guidelines for the design of games to learn introductory programming through game-play," *Int. J. Game-Based Learn.*, vol. 1, no. 3, pp. 30–52, 2011.

[79] "Adoba AIR," 2008. [Online]. Available: https://get.adobe.com/air/. [Accessed: 27-Aug-2020].

[80] "LightBot," 2008. [Online]. Available: https://lightbot.com/flash.html. [Accessed: 23-Aug-2020].

[81] "Tinker," *Microsoft*, 2009. [Online]. Available: https://softfamous.com/tinker/. [Accessed: 23-Aug-2020].

[82] "Robozzle," 2015. [Online]. Available: http://robozzle.com/. [Accessed: 23-Aug-2020].

[83] D. T. Campbell and J. C. Stanley, "Experimental and Quasi-Experimental Designs for Research, Rand Mc," *Na1ly Coll. Publ. Chicago*, vol. 47, p. 1, 1966.

[84] A. Ater-Kranov, R. Bryant, G. Orr, S. Wallace, and M. Zhang, "Developing a community definition and teaching modules for computational thinking: accomplishments and challenges," in *Proceedings of the 2010 ACM conference on Information technology education*, 2010, pp. 143–148.

[85] T.-C. Hsu, S.-C. Chang, and Y.-T. Hung, "How to learn and how to teach computational thinking: Suggestions based on a review of the literature," *Comput. Educ.*, vol. 126, pp. 296–310, 2018.

[86] D. B. Wright, "Comparing groups in a before–after design: When t test and ANCOVA produce different results," *Br. J. Educ. Psychol.*, vol. 76, no. 3, pp. 663–675, 2006.

[87] L. Yang and A. A. Tsiatis, "Efficiency study of estimators for a treatment effect in a pretest–posttest trial," *Am. Stat.*, vol. 55, no. 4, pp. 314–321, 2001.

[88] M. Rosenblatt, "A central limit theorem and a strong mixing condition," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 42, no. 1, p. 43, 1956.

[89] G. Chen and J. Shen, "Student Learning of Computational Thinking in A Robotics Curriculum: Transferrable Skills and Relevant Factors," International Society of the Learning Sciences, Inc.[ISLS]., 2018.

[90] M. J. Lee, "Gidget: An online debugging game for learning and engagement in computing education," in *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2014, pp. 193–194.

[91] J. A. Qualls and L. B. Sherrell, "Why computational thinking should be integrated into the curriculum," *J. Comput. Sci. Coll.*, vol. 25, no. 5, pp. 66–71, 2010.

[92] L. Perković, A. Settle, S. Hwang, and J. Jones, "A framework for computational thinking across the curriculum," in *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, 2010, pp. 123–127.

[93] A. Yadav, J. Good, J. Voogt, and P. Fisser, "Computational thinking as an emerging competence domain," in *Competence-based vocational and professional education*, Springer, 2017, pp. 1051–1067.

[94] B. C. Wilson and S. Shrock, "Contributing to success in an introductory computer science course: a study of twelve factors," *Acm sigcse Bull.*, vol. 33, no. 1, pp. 184–188, 2001.

[95] G. White and M. Sivitanides, "An empirical investigation of the relationship between success in mathematics and visual programming courses," *J. Inf. Syst. Educ.*, vol. 14, no. 4, p. 409, 2003.

[96] I. L. Balmes, "Correlation of mathematical ability and programming ability of the computer science students," *Asia Pacific J. Educ. Arts Sci.*, vol. 4, no. 3, pp. 85–88, 2017.

[97] L. M. de Souza, B. M. Ferreira, I. M. Félix, L. de Oliveira Brandão, A. A. F. Brandão, and P. A. Pereira, "Mathematics and programming: marriage or divorce?," in *2019 IEEE World Conference on Engineering Education (EDUNINE)*, 2019, pp. 1–5.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3043278, IEEE Access

CAGIN KAZIMOGLU received the B.Sc., M.Sc., Ph.D. degrees. He is a User Experience (UX) Researcher and an Assistant Professor with the Department of Computer Engineering, Cyprus International University (CIU). Previously, he was a Senior Lecturer with the University of Greenwich, London, and a Researcher at the Center for Advanced Studies (CAS) Research Group. He has over 13 years of professional research experience with various institutions and research centers. His research interests include usability testing, user experience design for digital media, serious game design, gamification and disruptive technologies. He is a Fellow of the Higher Education Academy (FHEA). His research interests began in childhood and later developed into a passion as he sought to effectively find a way to integrate interactive media tools to the fields of education, psychology, and health in order to emphasize the acquisition of transferable skills that would support everyday life.