Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 9 (2012) 522 – 531

Procedia Computer Science

International Conference on Computational Science, ICCS 2012

Learning Programming at the Computational Thinking Level via Digital Game-Play

Cagin Kazimoglu*, Mary Kiernan, Liz Bacon and Lachlan MacKinnon

Smart Systems Technologies Department, University Of Greenwich, SE10 9LS, London, UK

Abstract

This paper outlines an innovative game model for learning computational thinking (CT) skills through digital game-play. We have designed a game framework where students can practice and develop their skills in CT with little or no programming knowledge. We analyze how this game supports various CT concepts and how these concepts can be mapped to programming constructs to facilitate learning introductory computer programming. Moreover, we discuss the potential benefits of our approach as a support tool to foster student motivation and abilities in problem solving. As initial evaluation, we provide some analysis of feedback from a survey response group of 25 students who have played our game as a voluntary exercise. Structured empirical evaluation will follow, and the plan for that is briefly described.

Keywords: computational thinking; game based learning; serious games; introductory programming, games and learning

1. Introduction

In computer science (CS), it is essential to identity concepts with precise definitions and functionalities. Understanding the precise terminology used in CS is essential for understanding problems and to enable effective solutions to be developed. This is perhaps more significant when considering computational thinking (CT) as many studies reveal that this concept is defined abstractly at best and covers a wide variety of skills [1, 2]. The widely referenced article of Jeannette Wing defines computational thinking as a problem solving approach concerned with conceptualizing, developing abstractions and designing systems which overlaps with logical thinking and requires concepts fundamental to computing [3, 4]. Several studies defend the idea of making CT accessible to everyone and also stress that it is crucial for students to develop skills in CT before they are introduced to formal programming [5, 6, 7]. However, because CT has multiple definitions in the literature, it is arguable what ubiquitous skills and abilities encompass the development of CT [8] and how these should be integrated to the education of CS [9]. To address this problem, many studies have been undertaken to define what skills involve CT and what tools and techniques can be used to support students in the education of CS [10]. Since digital games are attractive and engaging for all groups of people, game based learning (GBL) has been proposed as one pedagogical framework for developing CT skills in an innovative way. Additionally, curricula that use a GBL approach in teaching CT for

* Corresponding author. Tel.: +44 (0)20 8331 8550; fax: +44 (0)20 8331 8665.
E-mail address: C.Kazimoglu@gre.ac.uk

doi:10.1016/j.procs.2012.04.056

learning programming have found positive effects on students [8]. Currently, the literature in GBL focuses on two popular approaches to facilitate the development of CT skills and learning of introductory programming. These methods are "learning through the exercise of designing games" and "learning through game-play" respectively. "Learning through the exercise of designing games" has been explored in a broad variety of ways which includes, but is not limited to, scalable design of strategic board games [11], game programming modules specifically focusing on development of CT [12, 9], and the use of visual programming tools (such as Scratch, Alice, Agentsheets) to support the education of introductory programming [13]. On the other hand, the existing literature provides less empirical research in developing CT for the purpose of learning programming through playing digital games [7]. Furthermore, studies in this field tend to focus on the motivational aspect of games where the main goal is often student engagement with various learning outcomes. Despite these efforts, few studies demonstrate how game-play can be associated with CT and how the education of introductory programming can be supported by playing games [14, 15]. Therefore, there is an urgent need to have a better understanding of the impact of games in the education of introductory programming and the development of CT skills though game-play.

To address these issues this research explores an ongoing GBL framework and its benefits in acquiring CT skills to support learning introductory programming. We demonstrate how our game supports thinking computationally, and how in-game elements can be mapped to various programming constructs and cognitive skills that are integral parts of learning computer programming. Subsequently, we present an initial analysis based on feedback from a volunteer group of students studying computing courses at University of Greenwich, and our plans for more detailed and structured evaluation. Finally, we conclude with a discussion of how our approach can be developed further and used as a precursor for learning formal programming in conjunction with developing skills in CT.

2. Related Work and Discussions

The task of learning to program is often recognized as a frustrating and demanding activity by students [16, 10] and numerous studies argue that poor teaching methods, low levels of interaction with students and a lack of interest are the major problems in learning programming [17, 18]. It is widely accepted that students need to demonstrate an understanding of the patterns evident in programming rather than focusing only on syntax and semantics of computer programming [19]. To achieve this, computational thinking (CT) has been the focal point of recent studies especially within the computer science (CS) discipline in order to integrate CT into the basic curriculum [5, 6, 9]. Researchers have also attempted to identify this concept independently, and thus various definitions are constructed in the literature [1, 8, 2]. However, very little of this work has successfully delivered guidance on what cognitive skills demonstrate CT and how these skills can be taught [14]. In other words, which specific skills comprise CT and how to scaffold these are still controversial, because few studies have empirically evaluated CT [8].

In her original work, Wing [3] identified five core aspects of CT which are conditional logic, distributed processing, debugging, simulation and algorithm building. She argued that CT incorporates all critical skills that involve problem solving with mathematical and engineering thinking and also with systematic and logical thinking. Guzdial [20] reported that CT is vaguely defined and precise definition in the literature is needed to understand what specific skills/abilities encompass the concept. Denning [1] argues that CT should not be seen as what CS is all about or a way to decrease the high dropout rates and poor retention of students in CS. He further explains the principles of computing in seven categories referring each of these as particular perspectives or classifications to view CT: computing, coordination, communication, recollection, automation, design and evaluation. Ater-Kranov et al [8] investigated the magnitude of importance of skills and abilities characterizing CT by evaluating the perspective of academics and students. They compared their perspectives and concluded that critical and algorithmic thinking alongside the application of abstractions to solve problems are the top skills that encompass CT. Furthermore, their findings propose that mathematical and engineering thinking is not necessarily a main characteristic of CT because complex CT can also happen spontaneously. Recently, Guzdial [2] argues the research in this field should move away from trying to define what is or what isn't CT and instead focus on implications of currently identified cognitive skills. Guzdial further discusses how these skills and abilities can be taught and what ways can be used to measure them.

To this end, the use of storytelling visual programming tools and digital game creation have been proposed as frameworks for teaching CT and computer programming because games are attractive and motivational in nature [7]. Visual programming tools are often perceived as ideal because they allow students to create various abstractions

quickly without the need for excessive program code. To create a working scenario in these tools, students select different characters and behaviors from a repository of choices and then build scenes where each scene contains these characters and behaviors. Complex abstractions can be created through combining character attributes and behaviors, which inevitably requires an understanding of programming sequence, conditionals, iteration and methods. Furthermore, these environments remove the syntax rules of genuine programming languages and allow students to think through programmatic representations. Despite these advantages, recent studies argue that CT is not a synonym for programming and that although visual programming tools can be useful for creating enthusiasm in programming, it is arguable whether or not students with little or no programming background can develop skills in CT through this mechanism. For example, while using these tools, students might develop good programming practice or spontaneously acquire a CT strategy, but there is little feedback available to alert them to this. The corollary to this might be when students create a working linear scenario without considering reusable patterns and other good programming practices. In this case, students might create output that works by designing an inefficient programming strategy, such as a statement repeated lots of times without using a loop, because they do not possess the requisite level of knowledge to develop a better solution and the tool provides no feedback to address this. In other words, visual programming tools require a debug mechanism to support students in developing their abstractions as well as good programming practices. Moreover, it is crucial to underline that visual programming tools are not games and should not be considered as game based learning (GBL) environments because they lack some of the crucial features that exist in all good games, such as a rewarding mechanism and clear goals to drive students to discover more. We argue that an efficient CT tool should support students in developing good programming practices with clear guidance through relevant feedback that would make sense to them.

Another popular approach being followed for teaching introductory programming is the use of digital game creation via game development stage(s). Existing work in this context can be categorized as assignment-based modules [14], strategic board games [11] and mobile application development [22]. The objective of this approach is to understand abstract programming concepts and not necessarily to learn about building games. Prior studies in this field reported considerable success with a significant increase in student motivation [23, 24]. However, it is also reported that game development needs a substantial effort because students are not excited by creating simple games [22], while advanced game creation requires the use of complex game-specific concepts such as collision detection, gravity calculation and path-finding. Although these concepts are valuable to learn, it is crucial to underpin these with CT as CT patterns are context and application independent, meaning that once students understand how to conceptually present a pattern they should be able to transfer and use it in a context they choose. Moreover, Sung et al. [14] argue that learning introductory programming through game creation is significantly challenging without considerable knowledge in computer graphics and a background in playing/designing games. They further indicate that extensive game development approaches require considerable preparation time and material content, in addition to the need to rework the existing curriculum in order to make it relevant to all students. More importantly, the majority of studies using this approach follow an instructivist style, where students are given instructions by an expert tutor and knowledge acquisition is governed by that tutor in a module based teaching model. However, in a recent study we discuss and conclude that one key aspect of GBL is an inherent constructivist structure [10].

An alternative and certainly less explored way to scaffold development of CT is through digital game-play. To-date a limited amount of work has been undertaken to investigate the relationship between learning programming and developing skills in CT through playing games. Chaffin et al. [25] studied student ability to write algorithms to generate data structures (e.g. binary tree) as part of game-play. Although lacking empirical evidence, their initial feedback suggests that students who played their game were better able to visualize how data structures work than the students who did not play the game. Muratet et al. [26] created a multiplayer Real-Time Strategy (RTS) game specifically for introductory programming students to foster their motivation. They encouraged students to write pseudo code in their game in order to give commands to different units interactively. Although they observed a significant increase in student interest in the subject, they highlighted the need to support students who learned at a slower pace than the majority of their colleagues, in developing their strategies during game-play. Similarly, Piteira & Hadded [27] created an educational game in order to increase the interest of students in learning programming. They compared their application with various programming tools and argued that the benefits of their approach were in personalization of the game experience, and in tracking player progress. There are also studies that evaluated the learning behaviors of students in addition to their motivation in learning programming. For example, Liu, Cheng & Huang [19] created a simulation game and analyzed the feedback and problem solving behaviors of 110 students

during game-play. It was found that students motivated by the game frequently used analytical strategies such as critical thinking in order to discover available solutions, and they also explored ways to apply them. On the other hand, it is reported that students who felt bored with the game only solved problems at a superficial level.

In a previous study [7], we developed guidelines for designing educational games specifically for learning introductory programming and developing problem-solving skills. To support the extension of this research, the next sections of this paper describe which cognitive skills are identified as most relevant to CT within CS, and how these can be incorporated into digital game-play.

3. Skills that encompass Computational Thinking

Berland & Lee [11] summarized the categories of CT according to computational activities as they are described in the literature: conditional logic, algorithm building, debugging, simulation and distributed computation. Lee et al. [9] undertook comparable research and examined CT in three aspects: analysis, abstractions and automation. While Ater-Kranov et al. [8] report that "critical thinking" and "problem solving" are the two common skills ubiquitously agreed in the literature. Perkovic et al. [6] discuss various skills (such as executing algorithms, coordination, communication and experimental analysis) according to the fundamental principles of computing stated by Dennings [1]. Furthermore, Dierbach et al. [28] defined the most common set of CT skills as: identifying and applying problem decomposition, evaluating, building algorithms and developing computation models to problems.

As can be seen from this there are various definitions and there is a lack of empirical evidence in defining the explicit boundaries of CT. However, from our analysis of the research and our own work we would argue that "problem solving", "building algorithms", "debugging", "simulation" and "socializing" are the core five skills that characterize CT within Computer Science discipline.

- **Problem solving** in CT refers to solving problems with logical thinking through using various computational models. This includes applying problem decomposition to identify problems and/or generating alternative representations of them. At this level students distinguish between problems and decide whether these problems can or cannot be solved computationally. Furthermore, students are able to evaluate a problem and specify appropriate criteria in order to develop applicable abstractions.
- **Building algorithms** involves the construction of step-by-step procedures for solving a particular problem. Selection of appropriate algorithmic techniques is a crucial part of thinking computationally as this develops abstractions robust enough that they can be reused to solve similar problems.
- **Debugging** is analyzing problems and errors in logic or in activities. At this stage, students receive feedback on their algorithms and evaluate them accordingly, which also includes reviewing current rules and/or strategies used. Debugging is central to both programming and CT because it involves critical and procedural thinking [3, 11].
- **Simulation**, also called "model building", is the demonstration of algorithms and involves designing and implementing models on the computer, based on the built algorithm(s). In simulation, students design or run models as test beds to make decision about which circumstances to consider when completing their abstraction.
- **Socializing** refers to the social aspect of CT, which involves coordination, cooperation and/or competition during the stages of problem solving, algorithm building, debugging and simulation. This characteristic of CT allows brainstorming and encourages assessment of incidents as well as strategy development among multiple parties. It is reported that socializing is one distinct feature of CT that distinguishes it from traditional computer programming [11].

We have successfully developed a game framework that allows the development of the skills related to CT using a limited number of programming constructs. The implementation is currently ongoing and is mapped onto part of the introductory programming curriculum taught within the School of Computing & Mathematical Sciences at the University of Greenwich.

4. Programming at the Computational Thinking Level

4.1 The Game Framework

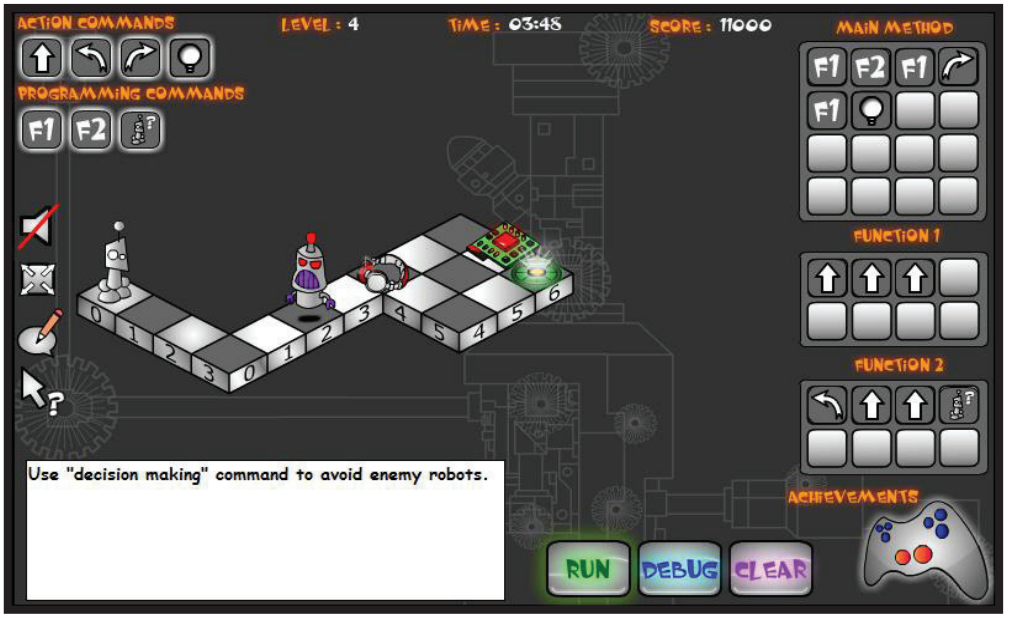

Fig. 1. Current version of prototype, showing level 4

Program Your Robot is the name of the game prototype we have been developing to support players in practicing various CT skills, as well as learning and using introductory programming constructs. The game, as illustrated in Figure 1, simulates a puzzle solving game where players control a robot by giving various commands to it. The goal of the game is to assist the robot to reach the *teleporter* where each level contains only one teleporter. During the game-play, players need to design a solution algorithm through using programming and symbolic representations in order to find pathways to help the robot to reach the teleporter. The commands players can give to the robot are divided into two as "action commands" and "programming commands". While action commands are used to move the robot in an environment consisting of a grid of streets, programming commands indirectly affect these actions and facilitate designing algorithms. Both types of commands are dragged from their associated toolbars and can only be dropped into specific areas called *slots*. Players play the game by dragging and dropping any number of commands into these slots in any sequence they choose, for as long as they have empty slots in their game-play. To complete a level, players need to move their robot on the teleporter, activate the robot's lights, and this will then allow them to proceed to the next level. Furthermore, players need to complete each level within the time that is available to them. As players progress through the levels, the grid environment expands and the game increases in complexity. In each level, players also encounter items that can be captured by the robot. These "collectible items" are randomly scattered every time players start to play a level, and thus this kind of approach ensures that the problem presented to a player at one level is significantly different from a problem presented to another player playing at the same level, or indeed the same player repeating the level to consolidate their learning. The randomness of collectible items is also controlled in order to guarantee the complexity of levels remains consistent. The game rewards players with new features (such as new collectible items, new slots and enemy robots to avoid) as players advance through the game.

There are three types of actions that players can perform when they finish designing their algorithms. The first of these is to execute the commands by pressing the "run button". During run-time, commands inside the slots are locked (so that they cannot be changed) and then executed by the robot in the order the player arranged them. The second action is to clear all commands dropped into the slots by pressing the "clear button". This function simply allows players to clear and re-design their solution. The final type of action is to debug a solution. Whenever players

make a mistake they cannot find or when they observe unexpected behavior performed by the robot, they can use the debug mode to identify potential errors in their solutions. The debug mode supports players during the development process and offers assistance with relevant messages rather than programming jargon or technical terms. By having this feature, the game not only encourages the players to develop the good practice of debugging solutions, but it also encourages them to think critically about their solutions (i.e. to ask themselves, is there a better or more efficient solution that I could have designed?). While players analyze the problem and build an algorithm to solve it, the game mechanism provides various forms of automated feedback as reactions to player actions. Players use this feedback to abstract game rules, debug, and develop winning strategies. Within these design-debug-run stages, three key aspects of computational thinking, analysis, abstraction and automation, come into the game-play.

The programming commands integrated into the game-play are symbolic representations of introductory programming constructs, namely programming sequence, selection (decision making), iteration loops and functions (methods). To this end, we have designed five levels in the game where, in each level, players are introduced to new challenges as well as new programming commands, which they can use to beat these challenges. In level 1, players discover how programming sequence works and they build their algorithms simply by dragging and dropping action commands into the *main method*, which is the controlling function for the robot. In levels 2 and 3, players learn how to change programming sequences and how to create programming patterns by designing and calling user-defined functions. While players can ignore using functions early in the game, because the same type of solution patterns are repeated frequently, they eventually find themselves designing reusable functions rather than placing all the commands inside the main method in order to get high scores in the game.

In level 4, players learn how to use conditionals and discover selection in programming. As shown in Figure 1, the player's robot needs to pass an enemy robot in order to complete level 4. The enemy robot cannot leave its location but it blocks the path of the player's robot by randomly disappearing and appearing again on the grid environment. Because it is uncertain when the enemy is, players need to overcome this situation by defining a condition to wait until the enemy disappears before passing. Therefore, players learn to develop their algorithmic thinking to overcome a certain problem and also discover how to make a selection when there is a level of uncertainty. In level 5, players practice how to combine iteration loops with functions in order to avoid a series of walls. As these walls are designed in similar formations, players can combine iteration loops with functions to create repeatable patterns in order to achieve a winning strategy.

At the basic level, our game is a system of rules in which players need to adjust the existing behavior of a robot using combinations of programming sequence, conditionals, iteration loops and methods. The game structure consists of two types of rules: operational and consecutive. Operational rules are guidelines players require in order to play the game. These are delivered through tutorial screens as dialogue boxes at the beginning of each level. The tutorial screens predominantly explain the features of the game and how programming commands work. Further to this, the consecutive rules are designed to be the underlying logical structure of the game. These are the unwritten procedures about developing efficient strategies to win the game. Current literature defines abstracting game rules as a great way of demonstrating computational thinking skills [9, 11]. Consequently, we enhanced our framework to continuously drive players to find the underlying consecutive rules through a game scoring system. We associated this scoring system with player understanding of game rules, and devising of strategies to optimize the behavior of the robot according to these rules. To achieve this, the scoring system calculates player scores based on three criteria: collectible items, slots and programming commands. Accumulating collectible items is an extra challenge in the game and often requires contriving repeatable patterns as the number of slots in the game are limited. The more collectible items players accumulate, the higher the points score they get. Additionally, the fewer number of slots players use to build an algorithm, the higher they score. Thus, the desired solution lies in creating repeatable patterns with as few slots as possible, which can only be accomplished by accurate use of programming commands. The last criterion is based on the use of programming commands and it evaluates the efficiency of player solutions at the end of each level. This score calculation is specific to each level and measures how well players understand how programming constructs work. As an example, players can complete level 2 and level 3 without using a single function. However doing so creates an inefficient solution and thus produces a low score. Furthermore, because the main method slots are limited it is not always possible to accumulate all collectible items by only using the main method slots. On the other hand, players can achieve a high score when they demonstrate deep understanding of the programming concepts used in the game, such as when they create recursive functions or loops combined with functions. Therefore, building efficient algorithms illustrates good game-play as well as promoting the acquisition

and development of the CT skills discussed above. Finally, the intention of the scoring system is to motivate players to use programming constructs in order to create winning strategies, as well as encouraging them to think which strategy is the most efficient. As a result, players can analyze whether or not their solutions are sufficient to create a winning strategy in the game.

4.2 Developing Computational Thinking Skills and Learning Introductory Programming

From the set of game activities described above we have shown how a student can develop their skills in game play, and in Table 1 we associate these with the previously defined skills that characterize CT. This illustrates how cognitive skills can be developed through game-play and validates the rationale of these skills outlined in the literature. Moreover, because each programming construct has a corresponding action in the game, we argue that this type of game-play allows players to visualize how programming constructs work.

Table 1. Examples of game activities associated with various characteristics of CT

Task	Associated CT skill category	Game activity	Rationale of the skill category
Problem identification and decomposition	Problem Solving	Help the robot to reach the teleporter. Activate robot's light when robot stands on the teleporter.	CT is described as a problem solving approach in various studies (Wing, 2006; Guzdial 2008). In conjunct to this, Schell (2008) explains the idea of what a game is as "a problem solving activity, approached with a playful attitude."
Creating efficient and repeatable patterns	Building Algorithms	Create a solution algorithm to complete all levels with as few slots as possible. Use functions to create repeatable patterns.	Perković *et al.* (2010) describe computation as "the execution of algorithms that go through a series of stages until a final state is reached."
Practicing debug-mode	Debugging	Press the debug button to monitor your solution algorithm to detect any potential errors in your logic.	Wing (2006) describes "debugging" as an essential component of both CT and programming.
Practicing run-time mode	Simulation	Observe the movements of your robot during the run-time. Can you follow your solution algorithm? Do you observe the expected behaviours?	Moursund (2009) reports that "the underlying idea in computational thinking is developing models and simulations of problems."
Brainstorming	Socialising	Examine the winning strategies of other players. Compare their solutions with yours. What advice would you give yourself and to them for scoring better in the game? Discuss.	Berland & Lee (2011) refers social perspective of computational thinking as "distributed computation in which different pieces of information or logic are contributed by different players during the process of debugging, simulation or algorithm building."

5. Student Feedback

The prototype game has now reached the stage where a detailed and structured evaluation can be carried out, and we will be evaluating with several groups of high school pupils and first year University students in the coming months. However, as a precursor to that activity, we wanted to obtain some initial feedback and impressions of the game, to ensure we were ready for the evaluation phase. To this end, we sought voluntary feedback from a group of students who were all studying degrees within the computer science discipline and had all studied at least one computer programming course. We intended that any key issues identified through this exercise would be addressed before we moved to the structured empirical part of our research. Since the participating students were volunteers across a range of different degree programmes, their programming knowledge and skills were considerably different. This proved beneficial in terms of evaluation as we got feedback from participants with a diverse range of knowledge, backgrounds and experiences. Twenty-five students provided feedback and some of these provided reports in remarkable detail.

The feedback shows that the majority of participants found the game well suited to help students to understand introductory programming constructs. We have cited several quotes from students that verify this point and we linked these quotes to five stages outlined in Table 1. Particular student quotes are cited below to show a flow of game activities relating to the CT stages from the game description:

Associated CT skill category: problem solving

Student 1: "I tried all sort of tricks using decision making instruction but I failed going any further than level 4 probably because of my poor problem solving skills☺. Nonetheless, it was good fun crossing the first 3 levels. I liked the fact that the further I was going the more sense it was making."

Student 2: "I enjoyed playing the game and it enhanced my knowledge towards methods and how to call declared functions. Overall, I thought the game was really interesting and entertaining at the same time."

Associated CT skill category: building algorithms

Student 3: "The game is very well designed and it is one of the games which need a lot of thinking. I got total score of 30750. I didn't experience any errors while finishing this game and it was very easy. In my point of view this game was really good to introduce the fun of programming to students who want to study programming."

Associated CT skill category: debugging

Student 4: "I found debug button useful because it provides messages when you forget to call a function. However, when you run debug mode it doesn't find an error or tell you that you have missed the lights or you cannot progress until you have done it."

Associated CT skill category: simulation

Student 5: "The game is very well thought out, for example, the demonstration of decision making logic through 'IF' statement was a well thought out example, and the graphical demonstration of this concept is quite creative."

Student 6: "The game is not difficult as you have to pre-plan what steps and where to turn in order to collect key items and land on a teleporter to complete the stages. However, whilst playing on level 4, I planned my predicted movements and as I began to run simulation I was confronted with a confusing message about degraded performance. Overall, the game has some issues that need to be addressed but I believe it is a fun way in order to beginners to understand the concept of programming."

Associated CT skill category: socialising

Student 7: "The game needs a high score page to reward people who use guile and don't rush through the screen. Nonetheless, I enjoyed playing it because I competed against a friend of mine."

Although none of the participants reported an error or a crash in the game, almost all of them put forward their suggestions regarding the game mechanics and user interface. We found some of these comments very useful and cited them below:

Student 8: "Achievements operation is under construction and there is no point of adding an operation into the game that does not work."

Student 9: "The game doesn't keep a high score chart so there is no advantage of using the loops (you do score better for using loops and functions). There isn't much of a celebration for getting things right and I think game needs a prize to invoke the rat-response/interest."

The Majority of these criticisms will be addressed in our future work which is described in the next section.

6. Future Work

Our future work involves the completion of the game prototype incorporating initial feedback from students, followed by an empirical study using first year computer science students within the Computing and Mathematical Sciences School at University of Greenwich, and at least one group of high school students.

We plan to enhance the game experience by making the "achievements" section available to reward players after they demonstrate good practice in programming. Although we are planning to add an achievement-based system, the feedback received from students also emphasised the importance of this in order to increase their motivation through behavioural conditioning. It is proposed that the achievements system will deliver conceptual knowledge in computer programming while the game-play offers an opportunity for practice and application of that conceptual knowledge. Furthermore, a high score chart is being designed where players can submit their score and share it with other players. The participation in the high score chart is going to be optional because we do not want players to stop playing if they are not doing very well. We hope the chart will encourage players to perform better each time they play the game so that they can show their high score when they do really well.

One distinct feature of our research is that the game framework we have been developing is open to the public and is online at http://www.programyourrobot.com. We would like to extend this by designing a community website and transferring the additional information currently given in dialogue screens in the game into this website.

Finally, an empirical study is being designed as a series of rigorous tests to examine whether or not this game develops and supports the abilities of students to think computationally in order to facilitate the education of introductory programming. We also aim to measure through a pre and post study questionnaire whether or not participants learn how programming constructs work from the game after they played it. The statistical data generated from these tests will be used to support our research, and to provide a contribution to body of knowledge in this area.

7. Conclusion

This paper has illustrated an innovative approach to "learning through digital games" in an effort to integrate computational thinking with learning of introductory programming constructs. We summarized different cognitive skills, characterizing computational thinking, into five categories and successfully incorporated the development of these skills into the context of a digital game. We have undertaken an initial freeform evaluation of the game with a group of volunteer students, who had already studied programming. The results show that the majority of students found our game interesting and relevant, and provided positive feedback that they thought this approach could develop the problem solving abilities of students who are learning introductory programming. We are currently addressing the issues raised by those students to improve the prototype by designing an achievements and high score system. Once these modifications are complete, we will run a significant structured evaluation exercise, with first year University students and high school students, and the empirical evidence from that exercise will be analysed and used to validate our research.

References

1. P. J. Denning, “The profession of IT Beyond computational thinking,” Commun. ACM, 52, pp. 28-30, 2009.

2. M. Guzdial. (2011). “A Definition of Computational Thinking from Jeannette Wing.” Available : http://computinged.wordpress.com/2011/03/22/a-definition-of-computational-thinking-from-jeanette-wing/

3. J. M. Wing, “Computational thinking,” Communications of the ACM, 49(2), pp. 33-35, 2006.

4. J. M. Wing, “Computational thinking and thinking about computing,” Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, vol. 366, pp. 3717-3725, 2008.

5. J. A. Qualls and L. B. Sherrell, “Why computational thinking should be integrated into the curriculum,” J. Comput. Small Coll., 25, pp. 66-71, 2010.

6. L. Perkovic, et al., “A framework for computational thinking across the curriculum,” presented at the Proceedings of the fifteenth annual conference on Innovation and technology in computer science education, Bilkent, Ankara, Turkey, 2010.

7. 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.”, International Journal of Game-Based Learning (IJGBL), 1(3), pp. 30-52, 2011.

8. A. Ater-Kranov, et al., “Developing a community definition and teaching modules for computational thinking: accomplishments and challenges,” Paper presented at the Proceedings of the 2010 ACM conference on Information technology education, pp. 143-148, 2010.

9. I. Lee, et al., “Computational thinking for youth in practice,” ACM Inroads, vol. 2, pp. 32-37, 2011.

10. C. Kazimoglu, M. Kiernan, L. Bacon and L. Mackinnon, “Developing a game model for computational thinking and learning traditional programming through game-play”, J. Sanchez and K. Zhang, (eds.), World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education, pp. 1378-1386, 2010.

11. M. Berland & V. R. Lee, “Collaborative Strategic Board Games as a Site for Distributed Computational Thinking,” International Journal of Game-Based Learning (IJGBL), 1(2), pp. 65-81, 2011.

12. P. Kuruvada, et al., (2010). “The Use of Rapid Digital Game Creating to Learn Computational Thinking.” Available: http://arxiv.org/pdf/1011.4093.pdf

13. A. R. Basawapatna, et al., “Using scalable game design to teach computer science from middle school to graduate school,” presented at the Proceedings of the fifteenth annual conference on Innovation and technology in computer science education, Bilkent, Ankara, Turkey, 2010.

14. K., Sung, et al., “Game-Themed Programming Assignment Modules: A Pathway for Gradual Integration of Gaming Context into Existing Introductory Programming Courses”, IEEE Education Society, 54(3), pp. 416 -427, 2010.

15. R., Ibrahim, et al.,“Students Perceptions of Using Educational Games to Learn Introductory Programming. Computer and Information Science”, 4(1), pp. 205 – 216, 2011.

16. J. Bennedsen, et al., Reflections on the Teaching of Programming: Methods and Implementations: Springer Publishing Company, Incorporated, 2008.

17. L. J. Barker, et al., "Exploring factors that influence computer science introductory course students to persist in the major," SIGCSE Bull., vol. 41, pp. 153-157, 2009.

18. N.J. Coull, and I.M.M. Duncan, “Emergent requirements for supporting introductory programming.,” ITALICS. 10 (1), pp.78-85. Available : http://www.ics.heacademy.ac.uk/italics/vol10iss1.htm , 2010.

19. C.C. Liu, Y. Cheng, & C. Huang, “The effect of simulation games on the learning of computational problem solving”, Computers & Education, 57 (3), pp. 1907-1918, 2011.

20. M. Guzdial, "Education: Paving the way for computational thinking," Commun. ACM, vol. 51, pp. 25-27, 2008.

21. A. Repenning, et al., “Scalable game design and the development of a checklist for getting computational thinking into public schools,” presented at the Proceedings of the 41st ACM technical symposium on Computer science education, Milwaukee, Wisconsin, USA, 2010.

22. Q.H. Mahmoud, and P. Popowicz, “A mobile application development approach to teaching introductory programming”, Frontiers in Education Conference (FIE), IEEE, pp. 4F-1 - T4F-6, 2010.

23. J.D. Bayliss, and S. Strout, “Games as a "flavor" of CS1,” SIGCSE Bull., 38(1), pp. 500-504, 2006.

24. S. Leutenegger, and J. Edgington, “A games first approach to teaching introductory programming,” Proceedings of the 38th SIGCSE technical symposium on Computer science education. pp. 115-118, 2007.

25. A. Chaffin, K. Doran, D. Hicks and T.Barnes, “Experimental evaluation of teaching recursion in a video game,” Paper presented at the Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games, 2009.

26. M. Muratet, P. Torguet, J.-P. Jessel, and F. Viallet, “Towards a serious game to help students learn computer programming,” Int. J. Comput. Games Technol., 1-12, 2009.

27. M. Piteira, and S.R. Haddad, "Innovate in your program computer class: an approach based on a serious game," Proceedings of the 2011 Workshop on Open Source and Design of Communication., pp. 49-54, 2011.

28. C. Dierbach, et al. “A model for piloting pathways for computational thinking in a general education curriculum,” Paper presented at the Proceedings of the 42nd ACM technical symposium on Computer science education, 2011.