Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia - Social and Behavioral Sciences 47 (2012) 1991 – 1999

Procedia
Social and Behavioral Sciences

CY-ICER 2012

A serious game for developing computational thinking and learning introductory computer programming

Cagin Kazimoglu, Mary Kiernan, Liz Bacon, Lachlan Mackinnon

University Of Greenwich, London, UK

Abstract

Owing to their ease of engagement and motivational nature, especially for younger age groups, games have been omnipresent in education since earliest times. More recently, computer video games have become widely used, particularly in secondary and tertiary education, to impart core knowledge in some subject areas and as an aid to attracting and retaining students. Academics have proposed a number of approaches, using games-based learning (GBL), to impart theoretical and applied knowledge, especially in the computer science discipline. Our research is concerned with the design of an innovative educational game framework focused on the development of Computational Thinking (CT) skills, and herein we introduce a serious game, based on our framework, which encourages the development of CT skills to facilitate learning introductory computer programming. We describe how a limited number of key introductory computer programming concepts have been mapped onto the game-play, and how an equivalent set of skills characterising CT can be acquired through playing the game. A survey response group of 25 students, following computer science and related degree programmes but with very diverse backgrounds and experience, provided initial usability feedback on the game. Their feedback confirmed that they found the game enjoyable, and also universally believed that this approach would be beneficial in helping students learn problem-solving skills for introductory computer programming. Feedback from this group will be incorporated in a revised version of the game, which will now be subject to rigorous experimental evaluation and analysis, to provide structured empirical evidence in support of our approach.

Keywords: Serious games, game based learning, computational thinking, introductory programming, learning programming with games.

1. Introduction

Existing research has led to many discussions and ideas on how best to teach introductory computer programming as students suffer from a wide range of difficulties in computer programming (CP) courses (Bonar & Soloway, 1983; Lahtinen, Mutka & Jarvinen, 2005; Coull & Duncan, 2011). Numerous studies argue that students view computer programming as a purely technical activity rather than a set of combined problem solving skills (Bennedsen, & Carpersen, 2008; Kazimoglu et al., 2010; Liu, Cheng & Huang, 2011). Therefore, the majority of students who are learning introductory computer programming tend to develop superficial knowledge and fail to create problem solving strategies through using programming constructs. Additionally, recent work in this field reports that enrolment in computer science (CS) programmes has been facing a steady decline despite steps taken to counter this and to bring more students into CS (Ali & Shubra, 2010).

One strategy proposed to facilitate the teaching and learning of introductory computer programming is the use of video game technologies in an educational game context (also referred to as "serious games"). The rationale for this is that because games are engaging and motivational, students will be encouraged to learn programming constructs

1877-0428 © 2012 Published by Elsevier Ltd. Selection and/or peer review under responsibility of Prof. Dr. Hüseyin Uzunboylu
doi:10.1016/j.sbspro.2012.06.938

in an entertaining and potentially familiar environment, and will then be able to transfer their learning outcomes from that environment into learning introductory computer programming with a programming language. Moreover, curricula that used serious games to specialise in learning programming have found positive effects on students as well as on learning outcomes (Ater-Kranov et al., 2010). Despite these efforts, few studies evaluated serious games as learning environments and how game-play can be associated to support the education of computer programming (Sung et al., 2010). The empirical evidence that verifies games are educationally effective tools for learning introductory computer programming is still absent from the literature (Guzdial, 2011). Furthermore, the existing work in this field tends to focus on how to adapt and assess serious games in classroom environments rather than proposing concrete methods to improve game-play. Therefore, there is a significant need for clear instructions and analysis on how games can be developed specifically for acquiring problem solving skills to support the education of introductory computer programming. To address these issues, we discuss and present the following: 1) A definition of Computational Thinking (CT) based on the current research literature, and a consideration of how this can be developed through playing games. 2) An analysis of currently available serious games designed to support teaching and learning in introductory computer programming. 3) A description of the game we have developed, based on our innovative educational game framework, and the potential benefits for students in acquiring CT skills to support learning introductory computer programming. 4) Initial feedback on the game, and the value of the approach in teaching students, from 25 survey respondents, all studying computer science or related degree programmes, with a wide diversity of backgrounds and experience.

2. Related Work

2.1. Defining computational thinking

Computational thinking (CT) has been the focus of several studies and reports in recent years (Guzdial 2008, Qualls & Sherrell, 2011).Wing (2006) defines CT as a set of intellectual and reasoning skills that states how people interact and learn to think through the language of computation. In other words, thinking computationally involves using methods, language and systems of computer science (CS) in order to solve problems in any discipline regardless of where the problem lies. Many authors state that CT is vaguely defined and a clear definition is necessary in order to use this construct to gain insight into problems (Guzdial, 2008; Dennings, 2009). Additionally, recent research in this field attempted to identify CT independently and thus several definitions exist in the literature (Perkovic et al., 2010). Despite these efforts, little work has successfully demonstrated how CT can be integrated into the curriculum and classroom environment. Because CT has multiple definitions and involves a broad range of skills, it is arguable which cognitive skills characterise CT and which real interactions can be identified as CT. For example, Wing (2006) argues that CT incorporates all critical skills and that involves problem solving with mathematical and engineering thinking. However, a recent study investigating the importance of skills characterising CT reveals that mathematical and engineering thinking is not necessarily a main characteristic of CT because complex CT can also happen spontaneously (Ater-Kranov et al., 2010). Furthermore, recent work in this field examined the categories of CT by summarising the rationale derived from the literature and according to this research none of these categories include mathematical and engineering thinking (Berland & Lee, 2011). Although empirical evidence is currently absent from the literature, the CT categories academics ubiquitously agree on are: *conditional logic*, *building algorithms*, *debugging*, *simulation* and *distributed computation* (Wing 2006; Wing 2008; Ater-Kranov et al., 2010; Berland & Lee, 2011). *Conditional logic* is the building block of CT and refers to local consequences of true/false value of a given statement. *Building algorithms* contain set of conditional logic and presents instructions to solve a complex problem in a step-by-step approach. While *debugging* refers to the act of determining problems in an algorithm, *simulation* states modelling or implementing an algorithm as test beds in order to identify which circumstances and abstractions to consider. Finally, *distributed computation* refers to the social aspect of CT and involves multiple parties when developing abstractions.

Many authors draw the attention that CT is not a synonym for programming (Wing, 2006; Guzdial, 2008; Repenning, Webb & Ioannidou, 2010). However, a survey revealed that the majority of high school teachers believe

that CT is identical to programming (Blum & Cortina, 2007). Therefore, at this point it is crucial to differentiate a programming tool from a CT tool. A programming tool should support students in writing programs by providing specific feedback on syntax errors, method implementation and programming logic as these issues have been identified to be most common mistakes made by students (Haden & Mann, 2003). Equally, a CT tool should offer a simple mapping between a problem and its alternative solutions by using relevant feedback and a context familiar to students. On one hand, we have programming tools where activities often involve writing excessive programming code in order to learn the structure of programming and produce efficient outcomes whereas on the other hand, a CT tool may allow development of simple solutions to CS challenges with little or no programming background. Recently, researchers in this field stressed that CT should be accessible to everyone and further ask "what kind of tools can make CT most accessible to everyone?" (Qualls & Sherrell 2010; Kazimoglu et al., 2011). In conjunct with this discussion, we have successfully developed a game framework which allows students to acquire the following skills through playing the game:

- to create and apply algorithms for a particular problem;
- to evaluate an algorithm by specifying appropriate criteria used;
- to apply computational thinking methods to problems;
- debugging algorithms and detecting logical errors;
- simulating algorithms and observing which consequences to consider when completing abstractions.

2.2. Serious games supporting the education of introductory computer programming

A number of studies used serious games as learning environments to support the education of introductory programming. Robocode (2001) is one of the first environments developed as an open source educational game in order to support java programming. The game objective is to develop an artificial intelligence (AI) for a tank to fight against other tanks programmed by other players. Students simply develop their war strategy using java programming and the battles run interactively when all players complete programming their own AI. Colobot (2007) is known to be the only commercial game that is specifically developed to teach computer programming. Players command different vehicles by writing pseudo codes in an in-game specific programming language (which is similar to C++) in order to complete various tasks. Despite presenting an interactive game-play, Colobot (2007) is not free and cannot be modified according to a specific curriculum. Catacombs (Barnes et al., 2007), Saving Serra (Barnes et al., 2007) and Elemental (Chaffin, Doran, Hicks & Barnes, 2009) are other examples of games that are specifically developed to teach about programming. More recently, Muratet et al. (2011) created "prog&play", a multiplayer real time strategy (RTS) game and asked both students and teachers to evaluate their game as a learning environment. Their initial results indicate that majority of students found their game motivational although some teachers who participated in their study reported negative perceptions because they thought the game might misrepresent CS as only being made of video games. Furthermore, recent studies also started to evaluate the learning behaviours of students in addition to their motivation in learning programming. For example, Liu, Cheng & Huang (2011) created a simulation game and analysed the feedback and problem solving behaviours of 110 students during their game-play. It was found that students motivated with the game used problem solving strategies in order to discover available solutions and also explored ways to apply them. In contrast to this, students who felt bored with the game only solved problems at a superficial level.

Additionally, there are many studies that run assessments on the existing visual programming tools (such as in Scratch, Alice) depending on game design principles (Resnick et al., 2009; Wu et al., 2010). These tools cleverly remove the syntax of a programming language and present a simple interface through drag and drop interactions. However, it is crucial to underline that programming tools are not games and cannot be considered as game based learning (GBL) environments because they lack some of the crucial features that exist in all good games such as timely feedback and a rewarding mechanism to drive students to discover more. The majority of programming tools deliver feedback only during the run-time of the designed projects, in the form of demonstrations of actions predetermined by the students (Kazimoglu et al., 2011).While using these tools, students might develop a good

programming practice or acquire a CT strategy, however 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 reusability and other good programming practices. In this case, students might create output that works by designing a bad 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.

Although current studies in serious games for learning programming are encouraging, it is crucial to underline that the majority of approaches do not consider the acquisition of CT skills but rather promote abstract and conceptual knowledge while encouraging student motivation in computer programming subjects (Kazimoglu et al., 2011). Supporting the learning of conceptual knowledge through a serious game can be an effective way but it does not allow opportunities for students to develop their skills in CT. Therefore, a clear definition should be made here between games that support the learning and reinforcement of conceptual knowledge, and games that support the learning of procedural and applied knowledge, and through this, skills acquisition and development. In the first case the contextual relationship between the focus of the game and the knowledge being acquired is less important and may be completely abstract, whereas in the latter case the contextual relationship between the game and the knowledge is paramount, hence our concern to see the utilisation of game-play.

3. The Game

Program your robot is a serious game designed to enable students to practice working with introductory programming constructs, within an environment that explicitly supports the acquisition of CT skills (such as algorithm building, debugging and simulation). The game is developed in Adobe Flash CS5 (2010) using actionscript 3 as the default programming language. The goal of the game is to assist a robot and help him to escape from a series of platforms by constructing an escape plan called a *solution algorithm*. Players construct their solution algorithm by giving various commands to the robot to perform. These commands are divided into two as action commands and programming commands. Action commands are those that have a direct effect on the robot (such as go forward, turn left), while programming commands indirectly affect these actions by supporting the solution developed by the player (such as repetition of a series of commands or making a decision on a condition). The current version of the game contains three programming commands which are functions, decision making and loops respectively. Functions are used to create repeatable patterns; loops are basically used to repeat a series of action commands and decision making is used to evaluate a condition such as whether or not the robot faces an enemy. All command can be dragged from their associated toolbars and dropped into specific areas called *slots*. Additionally, players can use any number of commands in any sequence, for as long as they have empty slots. For example, in early levels players often build their solution algorithms simply by dragging and dropping action commands into the *main method* which is the robot's default controlling function.

The problems are represented as levels and currently there are six levels in the game. Each game presents a different challenge and aims to teach a different programming construct. In order to pass a level, players need to reach a destination point called the *teleporter* within that particular level by developing their own solutions. As players progress through the levels, the platforms expand and the game increases in complexity. In each level, players encounter items that can be captured by the robot. These items reward players in the game and are randomly scattered every time players start to play a level. The random distribution of items is controlled in order to ensure that the complexity of each level remains broadly consistent. Therefore, we intend to deliver a game-play where, although the level of difficulty remains similar, a problem presented to one player can be significantly different to that given to another player to solve, who is playing the same level or indeed the same player repeating the level to consolidate their learning.

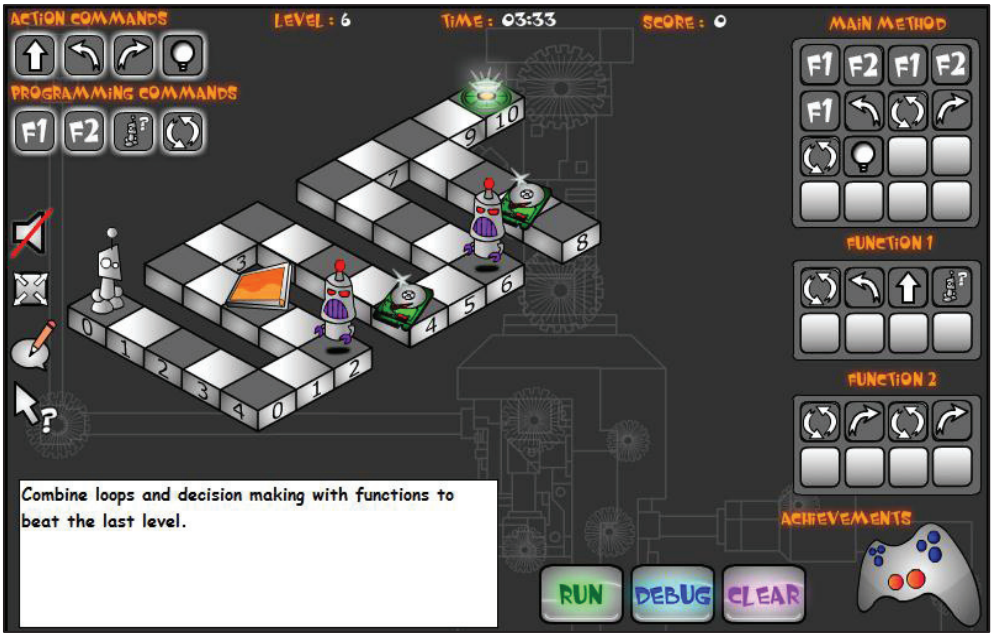

Figure 1. Current version of our game, showing level 6.

As shown in figure 1, players can perform a few actions after they finish arranging their sequences of commands inside the slots. The first of these actions is to execute the commands by pressing the *run* button. During runtime only the commands set inside the Main method are performed by default, in the initial sequence determined by the players. Alternatively, players can use programming commands to design more advanced solutions rather than simply dropping all action commands inside the Main method. While players can ignore programming commands (such as loops, decision making) early in the game, designing repeatable patterns becomes essential in the more advanced levels due to the lack of slots in the Main method. Moreover, we have built a scoring system into the game to measure players' understanding of game rules and their ability to devise strategies for optimising the behaviour of the robot according to these rules. The score calculation is based on how players use programming commands in the game. As an example, players can complete early levels without using a single function. However doing so creates an inelegant solution and thus produces a low score. Therefore, the game is aimed to motivate players to design reusable algorithms through using functions, loops and conditionals rather than placing all action commands inside the main method. In this way the game ensures that players discover for themselves the necessity of using functions and loops, and thus eventually recognise how crucial it is to separate the logic inside the main method into repeatable patterns in order to develop reusable solutions. Hence, students can learn by constructing their own knowledge, as well as developing their skills in CT by designing and building algorithms.

Another important action is to *debug* a solution designed by the players. At any time in the game, players can use the debug mode to detect potential errors in their solutions. After a debug process, the error/warning results are shown in the equivalent logic area within the game as messages (see Figure 1). Similar to an integrated development environment (IDE), the debug mode in our game allows players to develop the good practice of constantly debugging a solution before running it. We also avoided using programming jargon or technical terms in our game as players might have a non-technical/non-programming background.

In many ways *Program your robot* is similar to other games such as Light-Bot (2008) and Robozzle (2010) that deliver a similar game experience. However, because these games are created for fun and not for learning purposes, they do not consider a curriculum or developing skills in CT. In addition to this, these games do not sufficiently make a difference between different programming constructs (such as the difference between a recursive function and a loop). Therefore, although our prototype is similar to these games, there are differences that guide our efforts

such as the necessity to consider good programming practices, and the intention to encourage players to think computationally through motivation to achieve a high score.

Although we currently lack empirical evidence, we argue that our game encompasses main cognitive skills in CT because of the following: 1) players build algorithms during the game-play by designing their own solutions 2) players often use condition logic in order to achieve a high score in the game and also when they want to create reusable solutions 3) players track a simulation when they press the run button and observe the actions of the robot 4) players can debug their solution to detect errors in their logic.

4. Initial Evaluation

As an initial evaluation, we designed an exercise to get feedback from students who are studying degrees within the computer science discipline at University of Greenwich. The purpose of the exercise was to identify positive and negative issues of our game before we move to the structured empirical part of our research. Because participating students are studying on 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 diverse knowledge backgrounds and experiences. Twenty five students completed the exercise and some of them wrote reports up to four pages. The feedback showed that the majority of participants found the game well-suited to helping students to understand introductory computer programming constructs and develop their problem solving skills in this regard. The following quotes from student reports support this point:

Student 1: "As I am a programmer, I didn't find the game complicated or hard. The game felt straightforward and I found that the game puts across the idea of structuring a program. The functions could be considered as classes and the decision making is a Boolean value. Those are the basics of programming, a way to show how to simplify the way of coding a program."

Student 2: "In my point of view, this game was really good to introduce the fun of programming to students who want to study programming."

Student 3: "I have completed all levels in the game. I didn't have any problems as I found the commands easy to understand. As the game went on it became quite complex but I managed to understand the concept behind it."

Student 4: "In the robot game, I managed to play up to level 5 with a score of 38000. I found the game interesting to play as it was easy to follow the instructions. I think the interface is quite simple and not overly done. I had no major issues with the game."

It is encouraging for us that none of the participants reported an error or a crash in the game. However, almost all of them provided their suggestions regarding the game mechanics and user interface. We found some of these comments very valuable and decided to deliver a better game experience in the light of the following student suggestions:

Student 5: "Having no achievements in the game was quite a let-down as games like this require some sort of reward for how you coded the robot."

Student 6: "It isn't clear that you need to activate the lights at the end of the run, if you run debug mode it doesn't find an error or tell you that you have missed the lights"

Student 7: "There is no option to return back to the main screen of the game if the players decide to stop playing half way through the game. The game has an auto save system which is impressive but it doesn't notify users of such a system exist."

Student 8: "I had no major issues in the game but at times when the application is running the game/robot seems to slow down."

Student 9: "I felt that some dialogue boxes were unnecessary as it gave information players didn't need in order to complete a level."

Student 10: "The game needs a high score page to reward the people who use guile and don't rush through completing the levels."

5. Conclusion and Future Work

This study examines the relationship between computational thinking and learning programming within a serious game context. The paper argues that current serious games specifically developed for learning programming purposes do not consider a deep game-play for developing computational thinking skills. To address this problem, the paper describes *program your robot*, a serious game aimed at integrating core computational thinking skills and various programming constructs as an integral part of the game-play. Twenty five students participated in an exercise to evaluate *program your robot* and it was found that participants enjoyed playing the game. Furthermore, participants reported that this type of approach can enhance the problem solving abilities of students who are learning introductory computer programming.

Our future work involves improving the game by addressing all of the above suggestions raised by the students. An *achievements* section will be available in the game to reward players after they discover good practice in programming. Additionally, a high score chart is being designed where players can submit their scores 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 are also planning to make minor changes to the user interface in order to make the game more accessible to students.

A set of rigorous experiments are currently being designed to provide a systematic and structured evaluation of both the framework and the game. These will provide analytic data to determine whether or not the framework is successful in encouraging the development of CT skills and, as a result, whether or not the game helps students to learn and use key concepts in introductory computer programming. Both of these aspects will be analysed separately and in combination, to ensure we can accurately determine the impact of our approach and any benefits that can be derived from it. The statistical data generated from these experiments, and subsequent analysis, will also be made available to the research community, to provide a further contribution to the body of knowledge in this area. Finally, although *program your robot* is a prototype, the game is free and accessible at: http://www.programyourrobot.com.

Acknowledgements

Game and main menu music are composed by Dan O'Connor (dano@danosongs.com).

References

Adobe Flash CS5, (2010). Industry-leading authoring environment for producing expressive interactive content. Retrieved 10 December 2011, from http://www.adobe.com/products/flash.html

Ali, A., & Shubra, C. (2010). Efforts to Reverse the Trend of Enrollment Decline in Computer Science Programs. *Issues in Informing Science and Information Technology*, 7, 16.

Ater-Kranov, A., Bryant, R., Orr, G., Wallace, S., & Zhang, M. (2010). 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*.

Barnes, T., Richter, H., Powell, E., Chaffin, A., & Godwin, A. (2007). Game2Learn: building CS1 learning games for retention. *SIGCSE Bull.*, 39(3), 121-125.

Bennedsen, J., & Carpersen, M. E. (2008). Exposing the Programming Process. In Reflection on the Theory of Programming: Methods and Implementation, Bennedsen, J., Carpersen, M. E., Kolling, M. Eds. Springer, Verlag 6-16.

Blum, L., & Cortina, T. J. (2007). CS4HS: an outreach program for high school CS teachers. *SIGCSE Bull.*, 39(1), 19-23.

Bonar, J., & Soloway, E. (1983). Uncovering the principles of novice programming. Paper presented at the *Tenth ACM SIGACTSIGPLAN Symposium on Principles of Programming Languages*.

Chaffin, A., Doran, K., Hicks, D., & Barnes, T. (2009). Experimental evaluation of teaching recursion in a video game. Paper presented at the *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*.

Colobot, (2007). 3D real time game of strategy and adventure for learning programming. Retrieved 10 December 2011, from http://www.ceebot.com/colobot/index-e.php

Coull, N.J. and Duncan, I.M.M. (2011). Emergent requirements for supporting introductory programming. *ITALICS*, 10(1), 78-85. Retrieved 10 December 2011, from http://www.ics.heacademy.ac.uk/italics/vol10iss1.htm

Denning, P. J. (2009). The profession of IT Beyond computational thinking. *Commun. ACM*, 52(6), 28-30.

Guzdial, M., (2011). Any cognitive benefit of video games? Video-game studies have serious flaws. Retrieved 10 December 2011, from
Guzdial, M. (2008). Education: Paving the way for computational thinking. *Commun. ACM*, 51(8), 25-27.

Kazimoglu, C., Kiernan, M., Bacon, L., & Mackinnon, L. (2010). Developing a game model for computational thinking and learning traditional programming through game-play. Paper presented at the *World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2010*.

Kazimoglu, C., Kiernan, M., Bacon, L., & MacKinnon, L. (2011). 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), 30-52.

Lahtinen, E., Mutka, K. A. & Jarvinen, H. M. (2005). A study of the difficulties of novice programmers. In Proceedings of the 1*0th Annual SIGSCE Conference on Innovation and Technology in Computer Science Education (ITICSE 2005)*, 14–18.

Light-Bot, (2008) Control a robot by giving commands to it. Retrieved 10 December 2011, from http://armorgames.com/play/2205/light-bot.
Liu, C.-C., Cheng, Y.-B., & Huang, C.-W. (2011). The effect of simulation games on the learning of computational problem solving. *Computers & Education*, 57(3), 1907-1918.

Muratet, M., Torguet, P., Viallet, F., & Jessel, J. P. (2011). Experimental Feedback on Prog&Play: A Serious Game for Programming Practice. *Computer Graphics Forum,* 30(1), 61-73.

Qualls, J. A., & Sherrell, L. B. (2010). Why computational thinking should be integrated into the curriculum. *J. Comput. Small Coll.*, 25(5), 66-71.

Perkovic, L., Settle, A., Hwang, S., & Jones, J. (2010). A framework for computational thinking across the curriculum. Paper presented at the *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*.

Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable game design and the development of a checklist for getting computational thinking into public schools. Paper presented at the *Proceedings of the 41st ACM technical symposium on Computer science education*.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., et al. (2009). Scratch: programming for all. *Commun. ACM*, 52(11), 60-67.

Robocode, (2001). Open source educational game. Retrieved 10 December 2011, from http://robocode.sourceforge.net.

Robozzle, (2010). An addictive robot-programming puzzle game. Retrieved 10 December 2011, from http://www.robozzle.com.

Sung, K., Hillyard, C., Angotti, R.L., Panitz, M.W., Goldstein, D.S., & Nordlinger, J. (2010). Game-Themed Programming Assignment Modules: A Pathway for Gradual Integration of Gaming Context into Existing Introductory Programming Courses, *IEEE Education Society*, 54(3), 416 -427.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(2), 33-35.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881), 3717-3725.

Wu, W., Chang, C. & He, Y. (2010). Using Scratch as game-based learning tool to reduce learning anxiety in programming course. In Z. Abas et al. (Eds.), *Proceedings of Global Learn Asia Pacific 2010*, 1845-1852. AACE.