Issues in Informing Science + Information Technology

An Official Publication of the Informing Science Institute
InformingScience.org

IISIT.org

Volume 15, 2018

INCREASING INTRINSIC MOTIVATION OF PROGRAMMING STUDENTS: TOWARDS FIX AND PLAY EDUCATIONAL GAMES

Selvarajah Mohanarajah — University of North Carolina at Pembroke, Pembroke, USA — mohanra@uncp.edu

ABSTRACT

Aim/Purpose	The objective of this research is to investigate the effectiveness of educational games on learning computer programming. In particular, we are examining whether allowing students to manipulate the underlying code of the educational games will increase their intrinsic motivation.
Background	Young students are fond of playing digital games. Moreover, they are also interested in creating game applications. We try to make use of both of these facts.
Methodology	A prototype was created to teach the fundamentals of conditional structures. A number of errors were intentionally included in the game at different stages. Whenever an error is encountered, students have to stop the game and fix the bug before proceeding. A pilot study was conducted to evaluate this approach.
Contribution	This research investigates a novel approach to teach programming using educational games. This study is at the initial stage.
Findings	Allowing the programming students to manipulate the underlying code of the educational game they play will increase their intrinsic motivation.
Recommendations for Practitioners	Creating educational games to teach programming, and systematically allowing the players to manipulate the gaming logic, will be beneficial to the students.
Recommendation for Researchers	This research can be extended to investigate how various artificial intelligence techniques can be used to model the gamers, for example, skill level.
Impact on Society	The future generations of students should be able to use digital technologies proficiently. In addition, they should also be able to understand and modify the underlying code in the digital things (like Internet of Things).This research attempts to alleviate the disenchantment associated with learning coding.

Accepting Editor: Eli Cohen | Received: January 16, 2018 | Revised: March 12, 2018 | Accepted: March 26, 2018.
Cite as: Mohanarajah, S. (2018). Increasing intrinsic motivation of programming students: Towards fix and play educational games. *Issues in Informing Science and Information Technology, 15,* 69-77. https://doi.org/10.28945/4027

Future Research	A full scale evaluation – including objective evaluation using game scores – will be conducted. One-way MANOVA will be used to analyze the efficacy of the proposed intervention on the students' performance, and their intrinsic motivation and flow experience.
Keywords	educational games, games based learning, learning programming

INTRODUCTION

Today, digital technology plays a key role in our daily lives. Even the kids' toys are becoming more and more digital and some of them are programmable. The future generations of students should be able to use digital technologies proficiently. In addition, they should also be able to understand and modify the underlying computer programs. Organizations such as ACM and code.org are promoting fundamental computer science education at k-12 level. The former US president Barack Obama delivered a message in which he encouraged young students to learn computer programming. He said, "Now we have to make sure all our kids are equipped for the jobs of the future – which means not just being able to work with computers, but developing the analytical and coding skills to power our innovation economy. In the new economy, computer science isn't an optional skill – it's a basic skill, right along with the three 'R's (reading, writing and arithmetic)" (Obama, 2017). The U.S Bureau of Labor Statistics shows that the demand for software developers grows by 28% to 32% by 2020. The above mentioned factors could extrinsically motivate students to learn computer programming. According to the 2016 CRA-Taulbee Report (CRA - Taulbie Survey, 2016), the number of new undergraduate computing majors has been steadily increasing for the past seven years.

Nevertheless, there is a major obstacle that needs to be addressed; learning computer programming is considered challenging, and beginning students are easily frustrated and become bored (Koulouri, Lauria, & Macredie, 2014). Even the students who were initially enthusiastic about computer programing with the hope of creating cool computer games and innovative mobile applications, later reported that programming was tiresome and challenging (Beaubouef & Mason, 2005). In order to alleviate this disenchantment, we are focusing on using educational games to intrinsically motivate the beginner programing students by entertaining as well as challenging appropriately.

One of the surveys conducted by PEW Research Center revealed that, nearly 70% of the college students played video, computer or online games in 2015 (PEW, 2013). Not surprisingly, the above survey also reported that most of the student gamers had positive feelings about gaming, such as "pleasant", "exciting", and/or "challenging". Interestingly, other surveys conducted by Entertainment Software Association (ESA, 2015) and PEW (PEW, 2015) reveal that girls play equally as boys. In this research, we are trying to take advantage of the above facts in order to design a strategy that utilizes fix-and-play educational games to increase the intrinsic motivation of the students while learning computer programming.

RESEARCH QUESTION

The central hypothesis of this research is that empowering beginner programming students to manipulate the digital games they play will significantly increase the student's intrinsic motivation and performance.

PAST RESEARCH

For more than four decades, computer science educators have been extensively investigating the challenges faced by novice programmers. The seminal book edited by Soloway and Spohrer (1989) in late eighties includes a wide-ranging collection of articles that documented very early research activities on learning programming. In 2013, Robins, Rountree, and Rountree (2003) published a comprehen-

sive review of the research relating to teaching and learning programming. Kelleher and Pausch (2005) provide taxonomy of languages and/or environments that were designed to support learning programming. According to them, most of the systems either try to simplify the mechanics of programming or try to support the learners (e.g., by providing social and collaborative environments and/or entertaining/ motivating environments using Robots, Games, Videos etc.).

Programming Environments, Tools and Languages

Notable approaches that attempt to alleviate programming difficulties include ITS: Intelligent Tutoring Systems (e.g. LISP-Tutor (Anderson & Reiser, 1985)), Algorithm Animation (software visualization, e.g., JELIOT-3 (Moreno, Sutinen, & Joy, 2014)), Simplified Languages or Scaffolding (e.g., BlueJ (Kölling & Patterson, 2003)), Syntax-free Block-based drag-and-drop programming (e.g., AppInventor (Turbak, Sherman, Martin, Wolber, & Pokress, 2014)), Visual Immediate Feedback approach (e.g., Real or Simulated Robots, Animation) (McWhorter & O'Connor, 2009; Pattis, 1981).

Intelligent Tutoring Systems (e.g., LISP TUTOR)

Anderson and Reiser (1985) reported that students who received private tutoring learned LISP nearly four times faster than students who did not learn from private tutors. However, providing adequate one-on-one tutoring for all students is impractical in educational institutions. In this situation, intelligent tutoring systems (ITSs) can play a key role because they can be used for learning at any time, at any pace. However, research shows that designing ITSs is very challenging for complex disciplines such as programming (Dadic, 2011).

Software /Program/ Algorithm Visualization (e.g., Jeliot (Moreno, Sutinen, & Joy, 2014)*)*

Programming is an abstract and dynamic activity. Algorithm visualization (AV) techniques try to visualize the effects of each line of the code. This can help the student to formulate a mental model of how the program will be executed in a complex digital environment (Du Boulay, O'Shea, & Monk, 1981) . Research shows that AV alone is not enough to support learning programming (Pears, et al., 2007).

Simplified or Scaffolded Languages (e.g., Karel (Pattis, 1981), *BlueJ* (Kölling & Patterson, 2003)*)*

Many universities adopt industry level programming languages such as Java and C++ in their CS1 courses. These languages include many complex features that are valuable for professionals but dreadful for novices. A number of studies have attempted to address this problem by using scaffolding techniques to hide the undesired complexities (Kölling & Patterson, 2003), or designed simple mini languages for teaching purposes only (Brusilovsky, Calabrese, Hvorecky, Kouchnirenko, & Miller, 1997).

Syntax-free, Block-based Drag-and-Drop Environments (e.g., AppInventer (Turbak, Sherman, Martin, Wolber, & Pokress, 2014)*)*

After devising a solution for a problem, the next challenge is expressing the solution in a programming language. The popular programming languages have unusual syntax and complex semantics. Drag-and-Drop programming environments like Alice and AppInventor (Turbak, Sherman, Martin, Wolber, & Pokress, 2014) are designed to overcome these challenges.

Construct and Review- Immediate Visual Feedback Systems (e.g., Robotics e.g. LEGO (McWhorter & O'Connor, 2009)*), Games (e.g. Game2Learn* (Barnes, Richter, Powell, & Chaffin, 2007)*)*

In this approach, beginning learners are encouraged to participate in some creative activity. For example, students might be challenged to write code to control a robot (e.g., Karel the Robot, Pattis, 1981) or to create simple digital games (e.g., Barnes & Lipford, 2008).

Educational Games for Learning Programming (E.G., CEEBOT)

There are a few research reported related to educational games for learning programming or algorithm design. Shabalina and Pavel Vorobkalov (2008) discuss an industry level commercial game for learning C#, where the game engine Ogre3D is used with extended game logic. Kahn (1999) describes an interactive puzzle game to teach ToonTalk, which is intended to be a visual programming tool. We found another web based educational game for learning programming CeeBot (CeeBot4, n.d.). To the knowledge of the authors none of the above mentioned games were seriously used by novice learners.

Limitations

The limitations of the above mentioned approaches have been extensively discussed in the literature. For example, ITS utilizes AI techniques in order to provide appropriate course sequencing and feedback tailored to individual learners. A student can also study at his/her own pace. There are some ITSs for learning programming reported in the literature, but none of them are used for real learning (Robins et al., 2003). Designing an ITS for complex disciplines like programming is very challenging. Algorithm Animation techniques are used to help the students to ease the burden of abstraction- to visualize how the algorithm will be executed on a notional machine. AA is more suitable for intermediate, motivated learners than novices (Pears, et al., 2007).

Simplified languages and block based language environments avoid unnecessary complexities associate with industry level languages like Java or C++. Pappert (1980) mentioned that the programming languages should be simple and entertaining to learn ("low floor") but also should be powerful enough to challenge ("high ceilings"). The languages like Alice, Scratch and AppInventor are created based on this principle. They allow the novice learner to build problem solving skills without hindered by the complexities of syntax and semantics of an implementation environment. However, research shows that these environments do not scale-up to large, real programming problems (Ragonis & Ben-Ari, 2005).

Why Learning Programming Is Challenging

In May 2015, Robins (2015) mentioned "After several decades of research on the core topic of programming, ---, we still don't have a consensus on the reasons why so many novice programmers fail to learn,.." Nevertheless, based on the past researches discussed above, we may conclude that learning programming requires at least three key skills:

- able to abstract a problem and construct a step-by-step solution for a particular environment
- able to understand the semantic structures of a programming language and choose the appropriate structure to design the solution
- able to use the correct syntax to implement the design on the given environment

Learning programming is challenging for beginners as they have to develop the skills necessary to design a solution for a given problem so that it can be implemented in a particular computer environment. The novices struggle to create the appropriate mental model of the computer environment in which their solution need to be implemented. The only closed analogy a beginner can think of is a

person working on a set of natural language instructions. Du Boulay, O'Shea, and Monk (1981) coined a term notational machine to denote the required mental model for a programmer (an abstraction of the relevant hardware, operating system and the programming language). In addition, trial-and-error type of learning will be very frustrating for beginners as identifying the type of error and isolating the cause of the error may require all three types of skills simultaneously.

OUR APPROACH

The proposed research is built on etymological constructivism (Ben-Ari, 2001), self-determination theory (Ryan & Deci, 2000b), and flow theory (Csikszentmihályi, 2008). According to constructivism (Ben-Ari, 2001), learning environments should help the students to build their own knowledge by motivating them to actively engage in some meaningful and creative activities. In our approach, the amendable gaming environment motivates the students to build their knowledge by actively engaging them in enjoyable creative endeavors. The challenges immersed in the game will be exciting and achievable. The activities will neither be too hard (reduce anxiety) nor too easy (reduce boredom). Students should be able to tackle the challenges with an appropriate level of help from mentors.

One of the reasons for the higher student drop-out from the programming courses is reported as lack of intrinsic motivation (Bergin & Reilly, 2005) . Research shows that intrinsic motivation plays important role in student's engagement on an activity; on the other hand, self-efficacy has positive impact on intrinsic motivation (Ryan & Deci, 2000a). Intrinsic motivation refers to the engagement in an activity mainly for pleasure and satisfaction. Self-efficacy is defined as students' beliefs about their capabilities to perform certain tasks. In this study, we are investigating a unique approach that could maintain the intrinsic motivation of the students towards learning programming by keeping their self-efficacy level high.

Digital games are written in computer programs. This gives a unique advantage when designing educational games for learning programming. The game program can be exposed to the learners and they could be encouraged and empowered to examine the code, modify the logic, and see the effect of their actions immediately. In this research, we use casual games for investigation. Casual games are easy to learn and play. A number of bugs will be integrated in a game seamlessly. While playing, students will be challenged to fix the bugs that they come across. The bug-fixing action will be perceived as part of the game and not as an annoying interruption. To fix a bug, students should first inspect the code and understand the flaw. By doing this, we believe that the students will get a sense of ownership on the game they play, and in turn, their self-efficacy level will be increased. As a result, intrinsic motivation of the students will be increased and they will continuously be engaged in playing the game while learning the relevant programming concepts. We believe that this feature is stimulating and exciting.

PROTOTYPE, PILOT STUDY AND CONCLUSION

We created a prototype for a simple first-person shooting game to teach the fundamentals of conditional structures (Figure 1). The prototype was implemented in Java. There were many logical errors included in the program. For example, according to the given instructions, pressing the Left arrow should move the Gun to left- but the Gun will move to right. At this point, the players should stop the game and fix the error (see Figure 2). A pilot study was conducted with the help of seven student volunteers who had completed a first year programming course (CS1). The executable game was given to the students to play and learn. After playing, the students were asked to complete a five-point Likert style questionnaire. The questionnaire consists of eight questions and a comment section. The prototype is not even close to a simple industry level game. However, five students indicated that they prefer to play the game for revising conditional branching than using text-based revision materials. Instead of just playing the game students were also requested to fix many errors in the game. Four students stated that they felt some sort of pleasure and pride since they were able to fix the errors while playing the game. Note that, this is only a pilot study and the sample size is too small and

not randomized. The effectiveness of interventions cannot be generalized to general population. However, based on the results, it can be concluded that there is a possibility that allowing the programming students to manipulate the underlying code of the educational game they play will increase their intrinsic motivation. The game program will be improved and a full-scale study will be conducted. The next section will describe the future plan in detail.

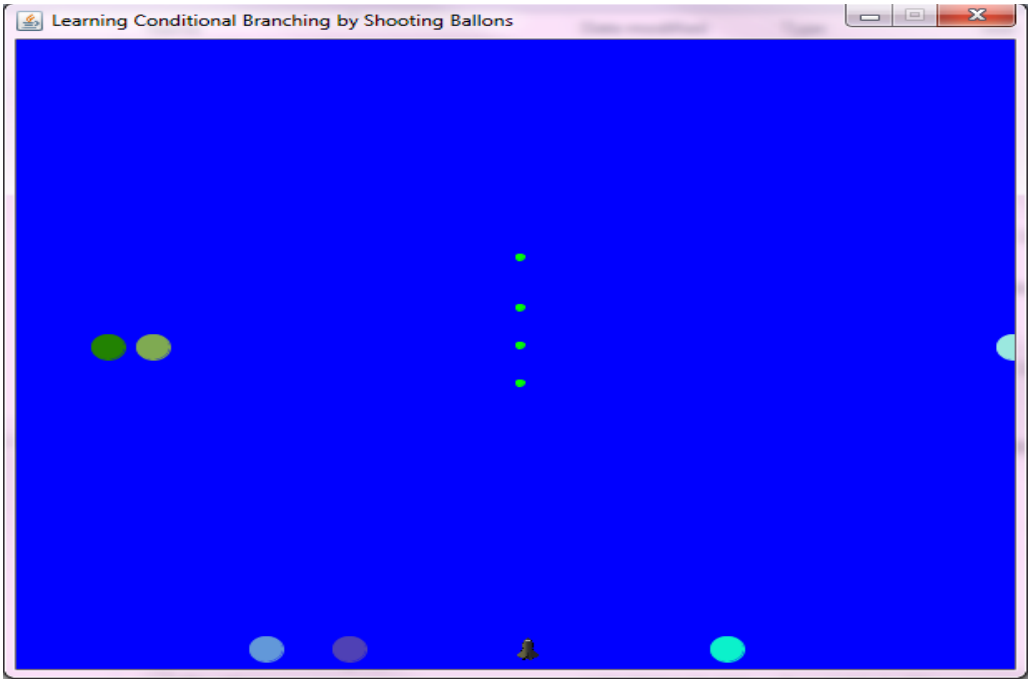

Figure 1: A Simple Shooting Game

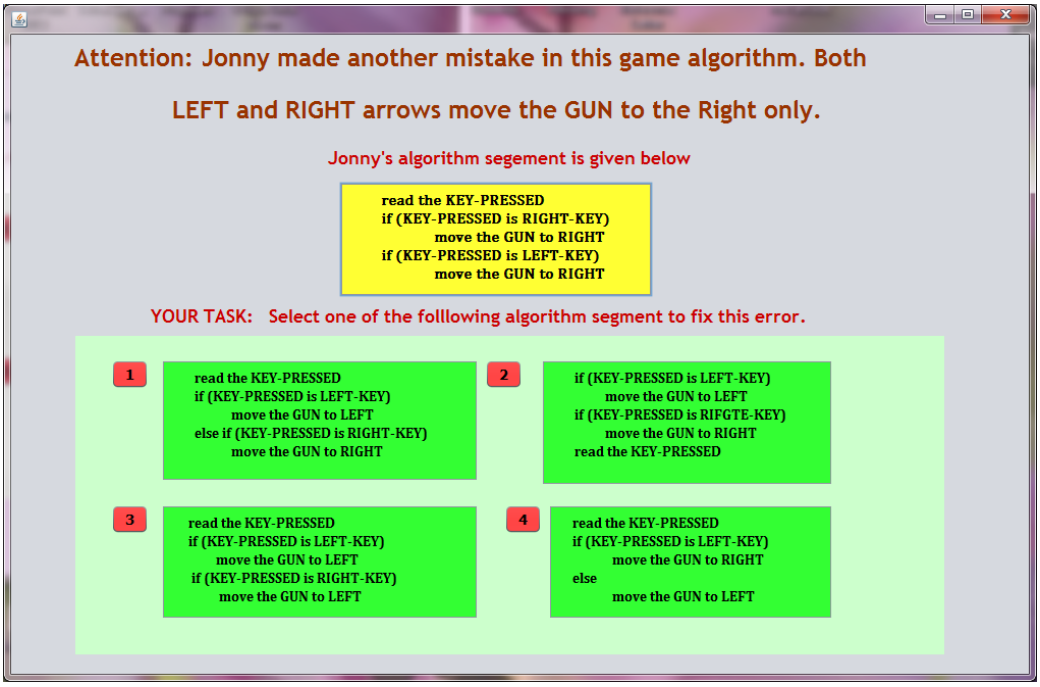

Figure 2: Fix and Play Games - Opening Game Program Logic to Gamers

FUTURE RESEARCH

In future, different types of casual games will be developed to learn different topics in computer programming, and a full scale evaluation including objective evaluation using game scores will be conducted. This research will follow the principles outlined in the US Department of Education's Common Guidelines for Education and Research (Institute of Education, 2013). Initially, two small educational games will be created as described before. The games will be used as revision material for learning two concepts; conditional branching and looping. We will also create identical, but text-based revision materials. CS1 courses are taught in fall and spring at University of Pembroke. Two sections, each semester, enrolls around 50 students in total. One section will be randomly selected to receive the game-based revision material, and the other will receive the text-based revision materials. A five-point scale Likert-style questionnaire (based on (Jackson & Eklund, 2004)) will be created to measure intrinsic motivation and flow experience of the students.

Data analysis will be conducted based on the guidelines in Kesselman et al (1998). A pre- and post-test will be given before and after the revision materials are used. At the end, all participants will be asked to complete the questionnaire. The reliability of the questionnaire will be measured using Cronbach's alpha. One-way MANOVA will be used to analyze the efficacy of the proposed intervention on the students' performance, and their intrinsic motivation and flow experience. The sample sizes may be different. A priori analysis will be conducted to verify existence of multivariate outliers, normality condition, and homogeneity of covariance. Power and Effect size analysis will be reported

Playing digital games requires some other skills such as eye-hand coordination. An experienced gamer has definite advantage over a student with poor gaming skills. To overcome this problem, we will also investigate how the players can be modelled using some AI techniques (such as Bayesian network and Fuzzy Logic).

REFERENCES

Anderson, R., & Reiser, B. (1985). LISP Tutor. *Byte*, 159-175.

Barnes, T., & Lipford, H. (2008). Game2Learn: Improving the Motivation os CS1 students. *ACM Game Development in Computer Science Education*. Miami, Florida. https://doi.org/10.1145/1463673.1463674

Barnes, T., Richter, H., Powell, E., & Chaffin, A. G. (2007). Game2Learn: Fuilding CS1 learning games for retention. *SIGCSE Bulletin, 39*(3), 121-125. https://doi.org/10.1145/1269900.1268821

Beaubouef, T., & Mason, J. (2005). Why the high attrition rate for computer science students: Some thoughts and observations. *ACM SIGCSE Bulletin, 37*(2), 103-106. https://doi.org/10.1145/1083431.1083474

Ben-Ari, M. (2001). Constructivism in computer science education. *Journal of Computers in Education, 20*(1), 45-73.

Bergin, S., & Reilly, R. (2005). The influence of motivation and comfort leval in learning programming. *Proceedings of the 17th Workshop of the Psychology of Programming Interest Group, PPIG '05*. University of Sussex, Brighton, UK.

Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A., & Miller, P. (1997). Mini-languages: A way to learn programming principles. *Education and Information Technologies, 2*(1), 65-83. https://doi.org/10.1023/A:1018636507883

CeeBot4. (n.d.). *Learn programming with Ceebot*. Retrieved 1 24, 2017, from http://www.ceebot.org/index.php?option=com_content&task=view&id=9&Itemid=52

CRA – Taulbie Survey. (2016). *CRA - Taulbie survey*. Computer Research Association. Retrieved from http://cra.org/crn/wp-content/uploads/sites/7/2017/05/2016-Taulbee-Survey.pdf

Csikszentmihályi, M. (2008). *Flow: The psychology of optimal experience*. Harper Perennial Modern Classics.

Dadic, T. (2011). Intelligent tutoring systems for programming. In S. Stankov, V. Glavinic, & M. Rosic (Eds.), *Intelligent tutoring systems in e-learning environments* (pp. 166-186). Croatia: IGI Global. https://doi.org/10.4018/978-1-61692-008-1.ch009

Du Boulay, B., O'Shea, T., & Monk, J. (1981). The black box inside the glass box: Presenting computing. *International Journal of Man-Machine Studies*, *14*(3), 237-249.

ESA. (2015). *Essential facts about the computer and video game industry*. Entertainment Software Association. Retrieved from http://www.theesa.com/wp-content/uploads/2015/04/ESA-Essential-Facts-2015.pdf

Institute of Education. (2013). *Common guidelines for education and research*. US Department of Education & NSF.

Jackson, R., & Eklund, S. A. (2004). *The flow scales manual*. Morganstown, WV: Fitness Education Technology.

Kahn, K. (1999). A computer game to teach programming. *National Educational Computing Conference*.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys, 37*(2), 83-137. https://doi.org/10.1145/1089733.1089734

Keselman, H. J., Huberty, C. J., Lix, L. M., Olejnik, S., Cribbie, R. A., Donahue, B., . . . Levin, J. R. (1998). Statistical practices of educational researchers: An analysis of their ANOVA, MANOVA, and ANCOVA analyses. *Review of Educational Research, 68*(3), 350-386. https://doi.org/10.3102/00346543068003350

Kölling, Q., & Patterson, R. (2003). The BlueJ system and its pedagogy. *Journal of Computer Science, 4*, 13.

Koulouri, T., Lauria, S., & Macredie, R. D. (2014). Teaching introductory programming: A quantitative evaluation of different approaches. *ACM Transactions on Computing Education, 14*(4), 1-28. https://doi.org/10.1145/2662412

McWhorter, W., & O'Connor, B. (2009). Do LEGO mindstorms motivate students in CS1? *ACM Technical Symposium on Computer Science Education*. TN. https://doi.org/10.1145/1508865.1509019

Moreno, A., Sutinen, E., & Joy, M. (2014). Defining and evaluating conflictive animations for programming education: The case of Jeliot ConAn. *ACM Technical Symposium on Computer Science Education (SIGCSE '14)*. , 629-634. https://doi.org/10.1145/2538862.2538888

Obama, B. (2017, January 30). *Computer science for all*. Retrieved from White House: https://www.whitehouse.gov/blog/2016/01/30/computer-science-all

Pappert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books.

Pattis, R. E. (1981). *Karel the robot: A gentle introduction to the art of programming with Pascal*. NY: John Wiley and Sons.

Pears, A., Seidman, S., Malm, L., L. M., Adams, E., Bennedsen, J., . . . Paterson, J. (2007). A survey of literature on the teaching of intro-ductory programming . *SIGCSE Bulletin, 39*(4), 204-223. https://doi.org/10.1145/1345375.1345441

PEW. (2013). *Internet & American life: Let the games begin*. Retrieved 9 22, 2017, from http://www.pewinternet.org/2003/07/06/let-the-games-begin-gaming-technology-and-college-students/

PEW. (2015). *Games and gamers*. Retrieved 2017, from http://www.pewinternet.org/2015/12/15/gaming-and-gamers/

Ragonis, N., & Ben-Ari, M. (2005). On understanding the statics and dynamics of object-oriented programs. *ACM SIGCSE Bulletin, 37*(1).

Robins, A. (2015). Editorial. *Computer Science Education, 25*(2), 115-119. https://doi.org/10.1080/08993408.2015.1034350

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming : A review and discussion. *Computer Science Education, 13*(2), 137-172. https://doi.org/10.1076/csed.13.2.137.14200

Ryan, R. M., & Deci, E. L. (2000a). Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary Educational Psychology, 25*, 54-67. https://doi.org/10.1006/ceps.1999.1020

Ryan, R. M., & Deci, E. L. (2000b). Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American Psychologist, 55*, 68-78. https://doi.org/10.1037/0003-066X.55.1.68

Shabalina, O., & Pavel Vorobkalov, A. K. (2008). *Educational games for learning programming*. International Book Series : Information Science and Computing.

Soloway, E., & Spohrer, J. (1989). *Studying the novice programmer*. Hillsdale, NJ: Lawrence Erlbaum.

Turbak, F., Sherman, M., Martin, F., Wolber, D., & Pokress, S. C. (2014). Events-first programming in App Inventor. *Journal of Computing Sciences in Colleges, 29*(6), 81-89.

BIOGRAPHY

Selvarajah (Mohan) Mohanarajah is an Associate Professor at University of North Carolina at Pembroke, NC, USA. His research interests include AI in Computer Science Education, Educational Games, Cyber Security and Big Data. Mohan can be reached at mohanara@uncp.edu