Journal of Information Technology Education: Research

An Official Publication of the Informing Science Institute
InformingScience.org

JITEResearch.org

Volume 21, 2022

SHOOT2LEARN: FIX-AND-PLAY EDUCATIONAL GAME FOR LEARNING PROGRAMMING; ENHANCING STUDENT ENGAGEMENT BY MIXING GAME PLAYING AND GAME PROGRAMMING

Selvarajah Mohanarajah *	University of North Carolina at Pembroke, Pembroke, USA	mohanara@uncp.edu
Thambithurai Sritharan	University of Colombo School of Computing, Colombo, Sri Lanka	rts@ucsc.cmb.ac.lk

ABSTRACT

Aim/Purpose	The key objective of this research is to examine whether fix-and-play educational games improve students' performance in learning programming languages. We also quantified the flow experiences of the students and analyzed how the flow contributes to their academic performances.
Background	Traditionally, learning the first computer programming language is considered challenging, In this study, we propose the fix-and-play gaming approach that utilizes the following three facts to alleviate certain difficulties associated with learning programming: 1. digital games are computer programs, 2. young students are fond of playing digital games, and 3. students are interested in creating their own games.
Methodology	A simple casual game Shoot2Learn was created for learning the fundamentals of branching. A number of errors were intentionally implanted in the game at different levels, and the students were challenged to fix the bugs before continuing the game. During the play, the program keeps records of the student's academic progress and the time logs at different stages to measure the flow experience of the students. The proposed approach was systematically evaluated using a quasi-experimental design in real classroom settings in two countries, Sri Lanka, and USA.
Contribution	The results derived from this research provide empirical evidence that the fix-and-play educational games ease some challenges in learning programming and motivate the students to play and learn.

Accepting Editor Peter Blakey | Received: July 5, 2022 | Revised: October 30, November 23, 2022 | Accepted: November 24, 2022.
Cite as: Mohanarajah, S., & Sritharan, T. (2022). Shoot2Learn: Fix-and-play educational game for learning programming; Enhancing student engagement by mixing game playing and game programming. *Journal of Information Technology Education: Research, 21,* 639-661. https://doi.org/10.28945/5041

Findings	The results show that the first-year programming students who play the fix-and-play game gain statistically significant improvement in their academic performance. However, the result fails to suggest a significant positive correlation between the flow experience and academic performance.
Recommendations for Practitioners	Empowering the students to fix the bugs in the educational games they play will motivate them to stay in the game and learn continuously. However, we have to make sure that the types and timing of bugs do not hinder the flow experience of the players,
Recommendations for Researchers	Students normally play industry-level high-quality games. Experience and interest in game-playing differ significantly between students. Gender difference also plays an important role in selecting game genres. We need to identify how to address these issues when resources are not sufficient to provide an individualized gaming experience.
Impact on Society	Programming is an essential skill for computer science students. The outcome of this research shows that the proposed approach helps to reduce the disenchantment associated with learning the first programming language.
Future Research	Further investigation is necessary to verify whether the AI techniques such as user modeling can be used in educational games to reduce the effects of uncertainty associated with the variations in students' gaming skills and other factors.
Keywords	CS1, novice programming, game-based learning; gamification, serious games, educational games for learning programming

INTRODUCTION

The United States Bureau of Labor Statistics (2022) shows that the demand for software developers is projected to grow 25% from 2021 to 2031 – this is much faster than the average for all occupations. Obviously, this fact could extrinsically motivate students to pursue a major in computer science. Based on the 2021 Taulbee survey, CS enrollment grows at all degree levels, with increased gender diversity, and the average undergraduate enrollment per U. S. CS department has increased to more than five times its level in fall 2006 (Zweben & Bizot, 2022). Nevertheless, there is a major obstacle that needs to be addressed. One of the key skills required for CS majors is programming; however, learning the first programming language is considered challenging. Beginning students are easily frustrated and become bored (Koulouri et al., 2015). Failure and dropout rates are traditionally high in CS1 courses (Bennedsen & Caspersen, 2007; Smith, 2022; Yadin, 2011). Even the students who were initially enthusiastic about learning programming with the hope of creating cool computer games and innovative mobile applications, later complained that learning programming was tiresome and hard, and demotivating (Beaubouef & Mason, 2005).

Several studies have been conducted to investigate the difficulties associated with learning to program and suggested various techniques and tools that could alleviate the problem. Nevertheless, learning programming is still considered challenging (Smith, 2022). In this research, we propose a novel approach to alleviate the disenchantment associated with diminishing motivation using fix-and-play educational games. These games will appropriately challenge novice programming students to fix the errors in the games they play and will continuously engage them in playing and learning.

Using digital games for learning programming is not a new idea (Johnson et al., 2016). Research shows that the game-based learning approach helps to nurture CS1 students' intrinsic motivation (Sharmin, 2022). There are many coding games available on the web (e.g. [Robocode, n.d.]). Similar to visual programming languages like Alice and Scratch (Turbak et al., 2014), they allow users to build

graphical animations using block or text-based programming snippets. Section 2.1.6 discusses three types of Game-Based Learning (GBL) approaches for learning programming; educational games, game design environments, and gamification tools. In this study, we combine both game-playing and game-designing features in a single tool. Young students are not only interested in playing digital games, but most of them are also excited about creating their own games. Since the digital games themselves are computer programs, we propose an approach where both interests are seamlessly integrated into creating scaffolded fix-and-play educational games.

The key objective of this research is to examine whether fix-and-play educational games improve students' performance in learning programming languages. We created a simple fix-and-play shooting game Shoot2Learn to evaluate the effectiveness of the proposed approach. A pilot study was conducted earlier, and the results were published (Mohanarajah, 2018). Recently, we conducted a quasi-experimental study in real classroom settings at two universities, one each from the countries USA (say US-U) and Sri Lanka (say SL-U). The next section provides a brief overview of different categories of software tools for learning programming reported in the mainstream literature. It also outlines the previous research related to flow experience and motivation pertinent to educational games. The rest of this paper discusses the research objectives, the proposed approach, evaluation methodologies, and outlines and analyses the results, and includes the future trajectory of this research.

RELATED RESEARCH

For nearly half a century, numerous studies have attempted to explain the reasons for the challenges in learning the first computer programming language and proposed various solutions to alienate these challenges. The first book on this area "Psychology of Programming" was published in the early seventies by Weinberg (1971). Later in 1989, a book edited by Soloway and Spohrer (1989) documented a wide-ranging collection of research activities on novice programmers' learning challenges. An oft-cited systematic review by Robins et al. (2003) provides a detailed analysis of the research related to various pedagogical approaches for teaching CS1 between 1970 and 2003. Another notable survey on teaching CS1 was published by Pears et al. (2007). Recently, Luxton-Reilly et al. (2018) published a comprehensive overview that covers the introductory programming education research between 2004 and 2018. Nevertheless, while revisiting this topic, Robins (2019, p. 353) stated; "*The field [teaching and learning CS1] has attracted considerable interest from researchers, teachers, practitioners, industry, and governments alike. There is now a significant body of relevant literature, but many important questions remain open.*" The next section focuses only on the studies related to the educational tools for learning introductory programming.

CS1 Teaching/Learning Tools

Becker and Quille (2019) show that nearly 10% of all the publications in ACMSIGCSE-TS proceedings in the last 4 decades were about teaching tools for programming. Kelleher and Pausch (2005) provide a detailed taxonomy of categorization of nearly 200 broad range of learning tools which are divided into two large groups: Teaching Systems (simplify-typing-code, alternative-to-typing-code, structured-languages, enable-to-track-execution, micro-worlds, social-learning, include-motivating-factors, etc.) and Empowering Systems (simplified-language, interactive languages/environments, includes-entertainment, supports-other-education, etc.). Some high-level programming languages such as BASIC, Pascal, and COBOL were also included in this taxonomy. Pears et al. (2007) reviewed more than 60 learning tools and organized them into 3 broad categories: assessment, visualization, and programming tools. A 2017 follow-up review by Saito et al. (2017) compared nearly 40 games and visual programming environments. The next section outlines some key categories of visualization and programming tools for learning. These categories have no strict boundaries, and some may overlap.

Intelligent tutoring systems / Interactive learning environments

Anderson's and Reiser's (1985) study found that students who received private tutoring learned LISP nearly four times faster than students who did not learn from private tutors. Robins (2019) postulates that this observation is true for all programming languages. However, providing adequate one-on-one tutoring for all students is impractical in a typical educational institution. In this context, intelligent tutoring systems (ITSs) can play a significant role- they will be cost-effective, and they can be used for learning at any time and any pace. ITSs include a variety of AI techniques to model students to provide adaptive course sequencing and individualized feedback. Crow et al. (2018) provide a systematic review of fourteen ITSs for programming- one of them is also included in the seven examples given in Luxton-Reilly et al. (2018).

Another type of Computer-Based Learning (CBL) system is Interactive Learning Environment (ILE). ILEs are ITSs without integrated AI features. They are usually designed based on behavioral learning theory using punish-and-reward strategies. Nearly nine examples of ILEs are given in Luxton-Reilly et al.'s survey (2018).

Research shows that creating efficient ITS for programming is very challenging (Dadic, 2011). However, it can be speculated that, at the rate the technology advances, in near future, it may be possible to create ITSs that include not only student modeling, but also intelligent teaching units supported by machine (reinforcement) learning, software visualization features, and natural language interfaces. Robins (2019) expressed a similar optimism: "In an ideal world we would like to provide individual and personally designed tuition and support to every student. It may be that breakthroughs in intelligent tutoring systems will one day achieve this ideal."

Program visualization (PV) / Algorithm animation (AA)

Program visualization tools are designed to engage the students by visualizing the effects of each line of the code using graphics and animations (Fouh et al., 2012). This will help the student to formulate his own mental model of how a program is being executed in a notional machine. The term "notional machine" was coined by du Boulay et al. (1981) to denote the high-level abstraction of the hardware and software features of a computing agent, which includes, the compiler, OS, RAM, CPU, and I/O systems. This type of mental model is required for a programmer to comprehend the structure and dynamics of the underlying execution agent of their program. Jeliot's family of tools is considered one of the most-studied PV tools (Jeliot 3, n.d.).

Algorithm animation tools are used to visualize data movements in complex algorithms and are generally used in CS2 courses. Comprehensive reviews of visualization tools are given in (Hundhausen et al., 2002; Price et al., 1993; Saito et al., 2017). A few past research indicated that PV alone is not sufficient to support learning programming (Naps et al., 2003; Pears & Rogalli, 2011).

Syntax-free, block-based, drag-and-drop microworlds

Papert (1980) argues that the programming languages should be not only simple and entertaining for the students to learn ("low-floor"), but also powerful enough for the practitioners to build complex useful systems ("high-ceilings"). However, in general, popular programming languages have unusual syntax and complex semantics. One of the challenges in learning programming is understanding the semantics of different language constructs and their syntaxes. Without this comprehensive knowledge, devising and implementing a solution to a considerably complex problem will be challenging. Drag-and-Drop visual programming environments like Alice, Scratch, and App-Inventor (Turbak et al., 2014) are designed to address this difficulty- they allow a novice learner to develop problem-solving skills without being hindered by the complexities of syntax and semantics of the programming language. Students need not worry about the properties of the underlying notional machine. Several studies show that these environments yield positive results with k-12, non-major, or

under-performing students (Meerbaum-Salant et al., 2013). A few research shows that these environments do not scale up or carry over: that is, the students still lack the skills to design algorithms for non-trivial problems (Franklin et al., 2020).

Simplified or scaffolded languages

ACM Curriculum 1978 (Austing et al., 1979) described CS1 as "the emphasis of the course is on the techniques of algorithm development and programming with style", and further it stressed that "neither the esoteric features of a programming language nor other aspects of computers should interfere with that goal (p. 151)". In general, universities adopt industry-level programming languages such as Java, C++, and Python in their CS1 courses (Farooq et al., 2014). These languages include many complex features that are valuable for professionals but annoying for novices. Educators try different methods to limit the complexity of these languages in order not to overwhelm the introductory students. Many studies have attempted to address this issue by either (1) providing various forms of scaffolding to hide the undesired complexities of the language environments to reduce the cognitive load of the students, e.g. BlueJ (Kölling et al., 2003), or (2) designing simple mini languages for teaching purposes only (Brusilovsky et al., 1997). Nearly hundreds of simplified or scaffolded languages are listed on Wikipedia (n.d.). Like Drag-and-drop environments, simplified languages are also easy to learn, but the skills learned do not easily transferable to serious programming tasks.

Construct-and-review, immediate feedback systems

This category of tools is designed based on the constructionist learning theory - it is essentially a constructivist learning theory based on Jean Piaget's experiential learning ideas (Harel & Papert, 1991). In this approach, beginner learners are encouraged to write computer programs to construct or control some tangible and/or shareable artifacts. This process could intrinsically motivate the students to get engaged in learning the tool they are using (programming language) to create/control those artifacts. For example, students might be challenged to write code snippets to control a physical or simulated robot in a microworld [e.g., Karel the Robot (Pattis, 1981), Robot Virtual World (Liu et al., 2013)], or to create simple interactive digital games [e.g. Game2Learn framework (Barnes et al., 2008; Game2Learn, 2012)], or to design multimedia animations (Code.org, n.d.), etc. Similar to drag-and-drop visual environments, these tools also provide instant visual feedback; that is, the students can see the impact of their code in the physical world or micro-world simulations immediately. Research shows inconsistent outcomes for the effectiveness of these approaches in learning programming (Major et al., 2012; McWhorter & O'Connor, 2009).

Game-based learning systems (educational games, game design tools, and gamification)

Several research projects have been reported in the CS education literature that focuses on educational games for learning programming (Barnes et al., 2007; Malliarakis et al., 2014; Shabalina et al., 2008; Villareale et al., 2020). The working group for Game Development for Computer Science Education (ITiCSE-2016) examined nearly 120 games related to CS education and found more than half of the games focus on some aspect of programming (Johnson et al., 2016). Many games challenge the students to write code to solve some problems to progress towards winning (Vahldick et al., 2014). Miljanovic and Bradbury (2018) survey provides a comprehensive review of nearly fifty game-based environments for learning programming. In general, previous research and surveys related to Game-Based Learning (GBL) tools for learning programming did not distinguish between, educational games, game design tools, and gamification tools.

Educational Games: Most educational games for learning programming were designed based on behavioral learning theory and utilize, a punish-or-award strategy, proactive supports, and various stimuli that were disposed of during the play to engage students (CodeMonkey.org, n.d.). Whereas, some

strategic and role-playing games were designed based on cognitive learning theory and situated cognition, and they excite and challenge the learners to explore and discover any non-explicit knowledge systematically hidden in the games. Opportunities for trial-and-error type learning are usually built into the games (Barnes et al., 2007).

Game Design/Creation Tools: Since digital games themselves are computer programs, the teaching tools based on game design are almost exclusive to the CS discipline (mainly for programming, algorithm design, and software engineering). Many young CS students are interested in creating their own games. Several studies explored the effectiveness of using game-designing environments for learning programming (Al-Bow et al., 2009; Code.org, n.d.). Many of these tools are based on constructionist learning theory and provide scaffolded game development environments where students can design and code fun games (Kafai & Burke, 2015). Shabalina et al. (2017) discuss a slightly different approach in which educational games and game design are combined. In this approach, students do not develop regular games, but they design and code educational games for learning programming.

Gamification: Gamification may not be associated with playing or creating any real game, but it is about incorporating game-like elements in the instruction strategy to motivate and engage students; examples include, challenges, points, rewards, control, immediate feedback, incremental levels, etc. To the best of the author's knowledge, there are not many pure gamification tools for teaching programming. Shorn (2018) discusses a gamification approach to teaching programming. Some text-based drill-and-practice tools on the web-based on the idea of "proglets" (Edmondson, 2009) may be included in this category [for example, see CodeWrite (Denny et al., 2011)]. In section 2.1, we outlined various categories of tools discussed in the CS Education research literature.

WHY LEARNING PROGRAMMING IS CHALLENGING

Based on past research (Becker & Quille, 2019; du Boulay et al., 1981; Kelleher & Pausch, 2005; Koulouri et al., 2015; Luxton-Reilly et al., 2018; Robins, 2019), we may conclude that learning programming requires at least three basic skills (a) ability to understand the problem and then construct a step-by-step solution using an appropriate level of abstraction (b) ability to understand the semantic structures of a programming language and choose suitable structures to design a solution (c) ability to use the correct syntax to implement the design. Moreover, all of these skills require a clear understanding of the limitations and capabilities of the underlying notional machine. Novices struggle to create the appropriate mental model of the structure and dynamics of the computer environment in which their solution will be implemented. The only close analogy a beginner student can think of is a person performing certain tasks based on a sequence of natural language instructions (like a recipe). This analogy is too shallow and may cause many misconceptions. For example, see the code segment in Figure 1. Novice programmers struggle to understand the flow of this conditional statement when executed by a computing agent working in fetch-execute cycles. They simply expect that the grade will be 'C' after the code is executed.

What will be the grade after the following code segment is executed?

```
int score = 75;
if (score > 60)
    grade = 'D';
else if (score > 70)
    grade = 'C';
```

Figure 1: Example: A common misconception in branching

In addition to the above-mentioned difficulties, the trial-and-error type of learning approach will be very frustrating for beginners since identifying the errors and isolating their causes still require all three types of skills discussed above.

Intrinsic Motivation and Self-Efficacy

Self-efficacy refers to an individual's belief in their own competence to carry out a specific task (Bandura, 1997). High self-efficacy increases the student's persistence at relevant tasks and helps them to be resilient and recover their motivation even after absolute failures. A student can be motivated by internal or external means. Intrinsic motivation is associated with learning something for personal pleasure or interest (e.g. playing games or creating games for fun), whereas extrinsic motivation is derived from external factors such as promised job opportunities and/or high salary scales (Ryan & Deci, 2000). Research makes it clear that learning programming is challenging, and a learner needs high determination, dedication, and persistence. Students need to be intrinsically motivated to stay focused on learning. Self-efficacy plays a key role in keeping one's intrinsic motivation stable.

Flow Experience in Educational Games

In the context of an educational game, Flow describes a state of mind experienced by the students who are completely immersed or engaged in learning by playing the game (Csikszentmihályi, 2018). An optimal flow can be described as an intrinsically enjoyable experience, where students will be intensively focused on their tasks, and nothing else seems to matter. As Kiili (2006) put it "the most important final result of flow in educational gaming: Students undertake studying activities not necessarily with the expectation of some external future benefit, but simply because playing the game is enjoyable, a reward in itself". Research shows that the optimal flow experience has a positive impact on learning (ibid).

As mentioned before, CS education researchers have been studying the difficulties of learning programming and suggested various remedies. However, in 2015, Robins stated, "*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* ---" (Robins, 2015, Editorial). Some recent studies also support Robins' observation and suggest further investigation in this area is continuously required (Loksa et al., 2022; Smith, 2022). Various studies show that designing games, as well as playing games, do help novices focus on learning programming (Sharmin, 2022). In this research, we propose an even better-enhanced tool that integrates both game-playing and game-designing challenges to help students to stay motivated by both types of adventures.

METHODOLOGY

The key objective of this research is to examine whether fix-and-play educational games could improve students' academic performances while learning their first programming languages. In particular, we investigate whether empowering the players to manipulate the underlying code of an educational game will positively impact their academic performance. We also examine how the flow experience of the students while playing educational games will contribute to their academic performances.

As stated before, this research utilizes two noticeable facts: young students are fond of playing digital games, and they are also interested in creating their own games. Since the digital games are themselves computer programs, we seamlessly mingled both excitements in one scaffolded educational game. We decided to use casual games in this study. Casual games are easy to learn and play and will not alienate girls from playing (Cote, 2020). A simple first-person shooting game called Shoot2Learn was created using swing and JavaFX. There are eight levels in the game, and it covers the basic syntax and semantics of one-way, two-way, multi-way, and nested conditional branching statements (including switch, and ternary operators). At the basic level, a plane will bomb a village periodically for a fixed number of times, and a player on the ground could use a gun to shoot and destroy the bombs

in the air. The bullet supply is limited. At this level, the gun cannot be moved but can be rotated to the left or right. If a bomb hits the gun, the gun will be destroyed. If a certain number of bombs reach the floor, the village will be destroyed, and the player lose the game.

One of the authors of this paper has been teaching CS1 for more than 30 years. Based on his and other educators' experiences (see below), eight (8) common syntax and semantic errors (java) related to branching were identified and systematically implanted in the game. Several studies examined the student's misconceptions about introductory programming (Qian & Lehman, 2017). Recently, Chiodini et al. (2021) presented an inventory of misconceptions in general programming and in specific programming languages such as java (the current list is available at https://progmiscon.org/). Alzahrani and Vahid (2021) surveyed over 47 publications related to CS1 education between 1985 and 2018 and listed 166 common logical errors in novices' programs, and among them, fourteen errors accounted for branching. For example, while playing the game, a student would notice an apparent glitch in the game; pressing the Left arrow should rotate the Gun to the right- but instead, the Gun would turn to the left. At this point, the player will be challenged to fix the bug. To debug, they should first inspect the code, understand the logic, and then select the appropriate code to fix the bug (see Figures 2 & 3). The bugs and distractors are designed to address the cognitive skills up to the fifth level in Bloom's taxonomy. Distracters may include code segments with common syntax errors in branching. Students will get appropriate feedback (see Figures 4 & 5). Students could choose to execute their choices, and if the code compiles, they will get relevant visual feedback. Visual feedback helps them to check the effects of their choices instantly and makes them feel like game programmers rather than test-takers. By fixing a bug, the player will get not only some academic points but also some more bullets. Moreover, the students can examine why their choices are wrong (or right), and they can also analyze the nature of the bug and the misconceptions associated with that bug. The game program will keep track of the academic as well as gaming performances of the students. Students will relate the academic performance score to their competence in fixing the bugs in the game.

An average freshman can complete the Shoot2Learn game in 15-20 minutes. By empowering the students to inspect the code and fix the game, the students will get some sense of ownership of the game they play. This feature will nurture their self-efficacy, and as a result, their intrinsic motivation will be increased. The inherent nature of gaming fantasy combined with high self-efficacy will keep the students motivated in playing and learning even amidst failures.

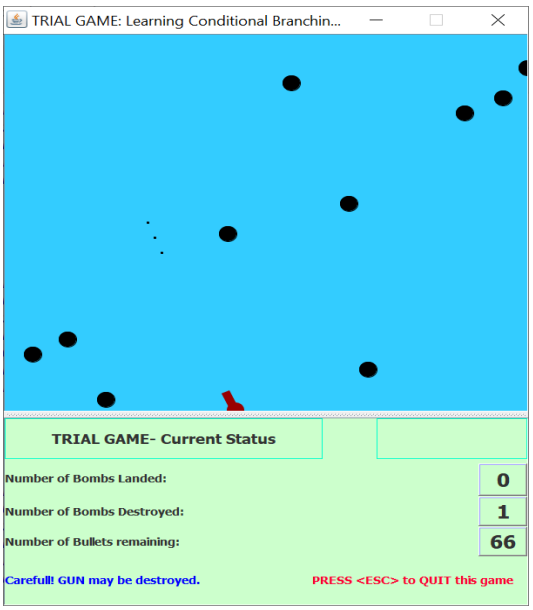
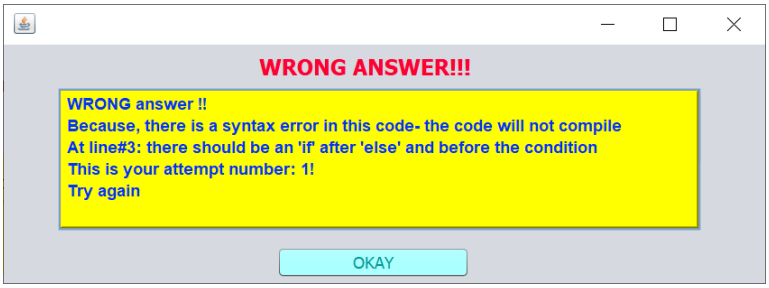
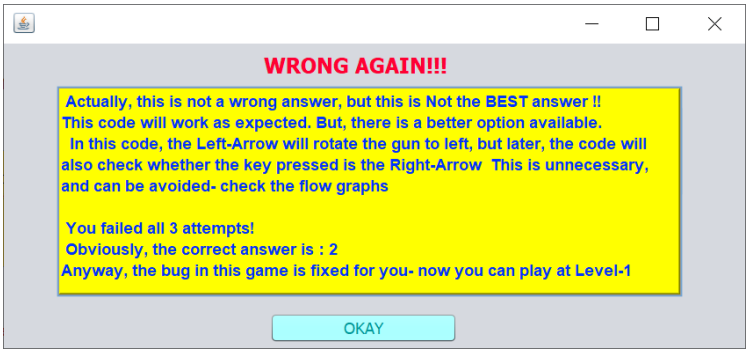


Figure 2: Shoot2Learn: Trial Game

Figure 3: Fix-and-Play: Gamers can debug the game they play





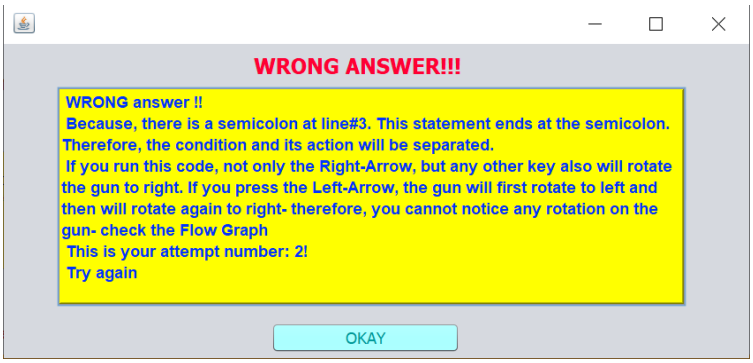


Figure 4: Text-based Feedbacks

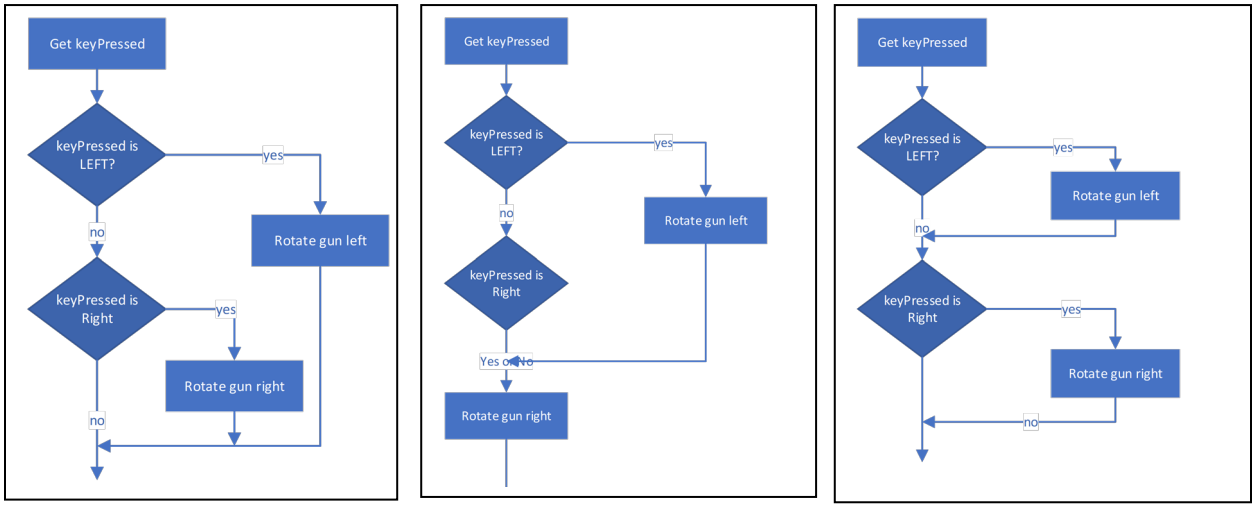


Figure 5: Flow graphs for the options 2, 3 & 4 in Figure-3 (option-1 will not compile)

In past research, the flow experience and motivation of the users of the interactive systems are usually estimated using questionnaires (Csikszentmihályi, 2018). But in this study, flow experiences are quantified, and a metric for the degree of flow experience (dfe) for a player is proposed as follows. The time logs will be recorded at sixteen (16) points while playing the game. This log file will be used to calculate the time taken to complete sixteen tasks (eight games and eight quizzes) for each student. Then the average time taken for each task by all the participants will be calculated. Next, for each participant, the Z-scores for each task will be calculated, and finally, the average of all his/her $|Z|$-scores of all tasks will be calculated. The degree of flow experience (dfe) for a particular player is defined based on the following rules (1) The $|z|$ values more than 2 will be approximated to 2 (2) $|z| = 2$ is considered as an outlier due to distraction or inattentiveness. (3) if a participant has more than 20% outliers, his/her degree-of-flow experience(dfe) is negligible (i.e. dfe=0). 3). Otherwise, the degree-of-flow experience (dfe) of a participant is defined as, $dfe = 2 - avg\ (|z|)_{\text{all-tasks}}$. Therefore, dfe can take a value between 0 and 2 inclusively, where two represents high and zero represents negligible flow experiences. A high flow experience can yield peak performances (Csikszentmihályi, 2018).

This research is designed based on the guidelines given for the Effectiveness-Research (type #5) in the Common Guidelines for Education Research and Development – a joint report published by the National Science Foundation (NSF) and the U.S. Department of Education's Institute of Education Sciences (IES) (2013). The underlying pedagogic strategy in the fix-and-play game is built on behavioral and constructionist learning theories (Ben-Ari, 2001; Kafai & Burke, 2015), and flow theory

(Csikszentmihályi, 2018). The amendable gaming environment will motivate the students to build their knowledge by actively engaging them in enjoyable creative endeavors. The challenges immersed in the game are exciting and achievable. The activities are neither too hard (reduce anxiety) nor too easy (reduce boredom). Students should be able to tackle the challenges with an appropriate level of help from the game itself (Csikszentmihályi, 2018). Anyway, we also take note of Nelson's and Ko's (2018) observation – focusing too much on general theories of learning may inhibit our search for better designs in CS Education research.

Evaluation Procedure, Participants & Instruments

As mentioned in the introduction, subjective as well as objective evaluations were conducted in real classroom settings at two universities US-U and SL-U. There were forty-nine (49) students participated in the study, including a class of nineteen (19) and thirty (30) students from the US-U and SL-U, respectively. All the students were in the CS-1 courses and a few of them might have some limited exposure to programming. Both institutions use Java as the first programming language, and there are no significant differences between the courses at both institutions. Appropriate IRB approval was obtained (Expedited Review-Protocol Number: 17-10-006), and the number was included in all consent forms.

Keselman et al.'s (1998) survey describes different types of statistical analyses used by educational researchers. In this study, we used a simple repeated measures design, and our data collection methods include, questionnaires, pre and post-tests, academic performance scores (based on bug-fixing), and automated time logs. First, the purpose and scope of the study, the evaluation procedure, and all other relevant information were clearly explained. A pre-test was given after obtaining voluntary informed consent from all the participants. The pre-test consists of eight (8) multiple-choice questions and they were all related to some common syntax and semantic errors in conditional branching (see Appendix A). After the pre-test, the students were asked to play the Shoot2Learn game. As mentioned before, during the play the students were confronted with purposefully implanted bugs, and they were empowered to fix the bugs by selecting the correct code from multiple options. The game kept the academic as well as gaming performance records of the participants. The game also kept time logs at certain stages in each game. After the game, the students were asked to take a post-test. The post-test was the same as the pre-test (due to an unexpected circumstance the post-test was not given in Sri Lanka). Finally, the students were asked to complete a five-point Likert-style questionnaire which contained ten (10) questions and an optional comment section (see Appendix B). The first part of the questionnaire includes four (4) questions related to gender identity, programming experience, and gaming experience (general and casual). The second part posed questions associated with the flow experience, self-efficacy, and learning experience of the participants.

Results

As mentioned before, forty-nine (49) CS1 students participated in this study including fourteen (14) boys and five (5) girls from the US-U and fifteen (15) boys and fifteen (15) girls from the SL-U. Except for two students, all other students from US-U (89%) identified themselves as beginner programmers, whereas eighteen students from SL-U (60%) identified themselves as beginner programmers. A large portion of the US-U students [seventy-one percent (71%) of boys and eighty percent (80%) of girls] considered themselves experienced gamers. Three boys and a girl indicated that they hadn't played games much, and two male gamers mentioned that they didn't play casual games. But among the SL-U students, only forty percent (40%) of boys and twenty-seven (27%) of the girls considered themselves experienced gamers. Two boys and five girls indicated that they didn't play games much. In addition, three female students mentioned that they had experience in playing casual games.

We used SPSS (ver-26) for statistical analysis. The pre and post-test scores of the participants were all independent and did not affect each other's scores. The average post-test scores of US-U students were 11.4% higher than their average pre-test scores. The Paired Samples T-test is used to analyze the

efficacy of the proposed intervention on the students' academic performance, and the results show that the fix-and-play educational game strategy produced statistically significant improvements in learning performance in the participants from US-U after playing Shoot2Learn (see Table 1: p = 0.0055 [one-tail] < 0.05). The effect size is moderate (Cohen's d = 0.65).

Table 1: USA-U: Pre and Post-tests: Paired Sample T-test

	Mean	N	Std. Deviation	Std. Error Mean
Post	60.5263	19	22.37794	5.13156
Pre	49.1228	19	26.33690	6.04210

	Mean	Std. Dev	T	Df	Sig. (2-tailed)
Post-Pre	11.40	17.61	2.822	18	.011

Nevertheless, we noticed that twelve students at US-U (more than 50%) do not show any improvements after playing the game (including two students who scored 100% on both pre and post-tests). This extreme frequency at the lower part of the distribution caused a negative skewness in the histogram. Therefore, we used the Kolmogorov-Smirnov test to check the normality, and the result was negative. Since the distribution is neither normal nor symmetrical under null (H_0: post-test mean - pre-test mean = 0), an Exact Sign Test was used to test the significance of the differences between pre and post-test scores (instead of the Wilcoxon signed-rank test). Results show a statistically significant median increase in the learning performance of the US-U participants after playing the game Shoot2Learn (see Table 2: p = 0.008 [one-tail] < 0.05).

Table 2: US-U Pre and Post-tests: Sign Test

Sign Test – Frequencies		N
Post - Pre	Negative Differences	0
	Positive Differences	7
	Ties	12
	Total	19

Test Statistics	Post - Pre
Exact Sig. (2-tailed)	.016

We conducted a similar analysis on the results obtained at SL-U. For this analysis, the final academic scores in the game (thirteen final scores were recorded during the evaluations) were used as the post-test scores. Once again, Kolmogorov-Smirnov statistics is used to test normality of SL-U's distribution of the difference (post-score - pre-score), and the result was positive (see Table-3: p = 0.1 [one-tail] > 0,05).

Table 3: SL-U (post – pre) score: Normality Test

	Kolmogorov-Smirnova			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	Df	Sig.
Diff	.142	13	.200	.920	13	.252

Next, we used Paired Sample T-test to test the significance of the difference, and the result shows that there are statistically significant improvements in academic learning performance in the SL-U participants after playing Shoot2Learn (see Table-4: $p = 0.00 < 0.05$). The effect size is large (Cohen's $d = 1.52$).

Table 4: USA-U: Pre and Post-tests: Paired Sample T-test

	Mean	N	Std. Deviation	Std. Error Mean
Post	71.5385	13	15.86198	4.39932
Pre	48.6538	13	14.27466	3.95908

	Mean	Std. Dev	t	df	Sig. (2-tailed)
Post-Pre	22.88	15.03	5.49	18	.000

The internal consistency of the questionnaire (for selected items related to flow experience) is measured using Cronbach's alpha, and the obtained value (0.64) is slightly lower than the acceptable values (> 6.5). One of the questions requests the respondents to indicate their experience in using the game. More than half of the US-U students and nearly one-third of the SL-U students indicated that the game was neither boring nor interesting. More than half of the SL-U students and four US-U students indicated that they like the game. This outcome is understandable since nearly 80% of the participants had played industry-level games regularly, but this game was created only for the research purpose. Another question asks the students if they were given a chance would they ever play a game of this type again for learning programming. All the students except one from the US-U mentioned that they would use this type of game again for learning. One of the questions asks whether the students felt like more studying than playing. Only 3 students at US-U and 6 students at SL-U indicate that they felt like more studying than playing.

One of the key questions asks the respondents to indicate how they felt when fixing the bugs while playing the game (in other words, whether they were proud of themselves since they were able to fix the bugs in the game). Seventy-four (74%) percent of the US-U students and sixty percent (60%) of the SL-U students mentioned that they liked this feature, Anyway, two US-U students and eight SL-U students indicated that this feature interrupted their game and they did not like this feature. Finally, respondents were asked to write comments (optional), and twelve SL-U students and four US-U students wrote comments (see Table 5). Most of the comments were encouraging. As mentioned in students' comments, we found that the game stuck sometimes in less powerful computers. From a usability point of view, the game did not include options that allow a player to modify any environment variables such as sound levels. These drawbacks could negatively impact the evaluation process.

Table 5: Participant's Comments

No.	**Student**	**Comment**
1	SL-1	Game stuck after 10 minutes
2	SL-6	I met some troubles while playing this game.
3	SL-7	That was not bad
4	SL-8	Good game but has many bugs
5	SL-10	Excellent work
6	SL-11	Excellent work

No.	Student	Comment
7	SL-19	I want to know the developer of this game
8	SL-20	It is very funny
9	SL-23	This game is very nice and interesting
10	SL-24	How to create a game like this?
11	SL-25	Simply good
12	SL-30	I am very proud of myself, but there are errors
13	USA-1	Computer Science is hard!
14	USA-16	Loved the game
15	USA-24	I quite enjoyed this
16	USA-21	Noise juxtaposed with the questions irritating

The game is divided into sixteen tasks. The game program records the time logs in certain states during the play (the start time and end time of each task). For each participant, the time taken to complete all these tasks is tabulated, and then the averages and standard deviations (SD) of the durations for each task are calculated (see Table 6).

Due to curiosity, we checked whether the means and SDs of time spent on each task for the US-U and SL-U participants were correlated and found that the correlation was significant (The Pearson Correlation coefficients are 0.66 for means and 0.92 for the SDs - at 0.01 level 2 tailed). Next, we calculate the degree of flow experience (dfe) as described in the methodology section. Table 7 gives the time taken for all the sixteen tasks by two students from both universities. The corresponding z-scores are also calculated and tabulated. For example, the US-U student-1 (Table 7.1) spent 896-time units on game-1, and based on table-6.2, the average for game-1 for all US-U participants is only 553 time units (SD is 281). Therefore, the z-score for this student for this task is (896 – 469) / 263 = 1.6; that is, this student spent 1.6 * SD more time than the average for this task. A close look at Table 7.1 reveals that this student took more time on almost all the tasks than the averages (except one).

Table 6: Means and SDs of the time taken by participants for each task

	Game1	Game2	Game3	Game4	Game5	Game6	Game7	Game8	Quiz1	Quiz2	Quiz3	Quiz4	Quiz5	Quiz6	Quiz7	Quiz8
Mean	553	549	454	582	581	655	542.62	543	514	439	518	464	452	546	608	474
S.D	281	335	290	297	253	238	348.88	341	307	322	241	274	266	321	251	304
Table: 6.1: SSL-U participants mean and SD of the time taken for each task																
Mean	469	582	559	572	605	613	527.73	638	507	496	510	499	504	556	658	415
SD	263	345	304	266	215	239	350.78	350	328	321	241	298	233	314	240	313
Table: 6.2: USA-U participants mean and SD of the time taken for each task																

Now, consider US-U-Student-1's z scores in table 7.1. The average of the absolute values of all these z-scores is 0.75 (that is, for this student the avg $(|z|)_{\text{all-tasks}} = 0.75$.). Therefore, the degree of flow experience (dfe) of the US-U-Student-1 is (2 - 0.75) = 1.25. Similarly, based on tables 7.2, 7.3, and 7.4, the dfe's for these students are 1.1, 1.13, and 1.2 respectively. The dfe explains how close the time

spent by a participant on each task is to the corresponding averages. If the deviations are small the flow experience is high. That is, the higher the dfe, the better the flow experience.

Table 7: Example: Time spent on each task by two students (arbitrarily selected) from each of the universities US-U & SL-U, and the corresponding z-scores

	Game1	Game2	Game3	Game4	Game5	Game6	Game7	Game8	Quiz1	Quiz2	Quiz3	Quiz4	Quiz5	Quiz6	Quiz7	Quiz8
U-1	896	925	462	929	587	502	543	465	297	998	419	757	334	967	860	509
Z-Sc	1.6	1	-0	1.3	-0	-0	0	-0	-0.6	1.6	-0	0.9	-1	1.3	0.8	0.3
Table 7.1: USA-U:Student#1 time taken for each task and z-score																
U-2	685	55	928	553	516	673	897	881	379	715	937	806	939	309	417	132
Z-Sc	0.8	-2	1.2	-0	-0	0.3	1.1	0.7	-0.4	0.7	1.8	1	1.9	-1	-1	-0.9
Table 7.2: USA-U:Student#2 time taken for each task and z-score																
S-1	750	970	91	282	836	796	479	142	550	906	352	413	796	736	882	877
Z-Sc	0.7	1.3	-1	-1	1	0.6	-0	-1	0.1	1.4	-1	-0	1.3	0.6	1.1	1.3
Table 7.3: SL-U:Student#1 time taken for each task and z-score																
S-2	266	121	338	343	489	408	817	962	131	730	723	399	524	372	422	796
Z-Sc	-1	-1	-0	-1	-0	-1	0.8	1.2	-1.2	0.9	0.8	-0	0.3	-1	-1	1.1
Table 7.4: SL-U :Student#2 time taken for each task and z-score																

Finally, we test the correlation between the participants' degree of flow experiences and their learning gain (post-scores – pre-scores) of all the participants. Results show that there are no statistically significant positive correlations between the dfe and gain (Pearson coefficients are 0.19 and 0. 21 for US-U and SL-U respectively).

DISCUSSION & CONCLUSIONS

Learning the first programming language is considered challenging. Nearly five decades of research have proposed numerous tools and pedagogical approaches to ease this problem. In this research, we proposed the fix-and-play educational game approach to reduce the disenchantment associated with learning programming. We created a casual game for learning conditional branching (called Shoot2Learn) and used it to evaluate our proposed approach in real classroom settings in two countries. The game includes eight stages, and a number of bugs are intentionally planted in the game at different points. The bugs were related to the basic syntax and semantics of one-way, two-way, multi-way, and nested conditional branching statements. The players are empowered to fix the bugs while they play the game.

Results show that the CS1 students who played Shoot2Learn gained statistically significant improvements in learning conditional branching. Although our sample sizes are too small to generalize, these findings do indicate that the proposed fix-and-play games have positive impact on learning first programming languages. The student responses to the questionnaire provided additional evidence that the proposed approach could be successful in improving student engagement and learning. Almost all the participants (except one) indicated that they would use this type of game again for learning. Sixty-seven percent (67%) of the students mentioned that they liked the feature that allowed them to fix the bugs in the game while playing. The participants' comments were also enthusiastically encouraging.

Result shows that the correlations between the flow experiences and learning gains were not statistically significant. In this research, we try to quantify the flow experience using the time lapse between the key events. A metric is introduced to quantify the flow experience. This outcome may be due to a

flaw in the proposed metric (dfe), and further research is required to validate the effectiveness of the proposed metric.

In Shoot2Learn, all the players will confront the same bugs during the play, and the system gives the same options to select the fix from. Fixing the errors is a kind of debugging, and debugging is harder than coding (though, in this tool, the correct option is also given along with some other distractors). In the future, we are planning to include a free-form or a drag-and-drop editing tool that will allow the learners to construct their own code segment. We also plan to incorporate a user modeling system in the games to provide individualized challenges to the users. Moreover, an experienced gamer may have a different flow experience compared to a non-gamer. The user model should also be able to provide adaptable interfaces based on the user's level of game-playing experience. We are also planning to scale up the evaluation process. We will create one industry-level casual game for at least three key knowledge areas in high-level programming (structured aspects first), and evaluate the games in real classroom settings in different countries. We will refine the questionnaire and include suitable questions to measure the flow experiences and motivation of the participants.

REFERENCES

Al-Bow, M., Austin, D., Edgington, J., Fajardo, R., Fishburn, J., Lara, C., Leutenegger, S., & Meyer, S. (2009). Using game creation for teaching computer programming to high school students and teachers. *ACM SIGCSE Bulletin*, *41*(3), 104–108. https://doi.org/10.1145/1595496.1562913

Alzahrani, N., & Vahid, F. (2021, July). Common logic errors for programming learners: A three-decade literature. In *ASEE Virtual Annual Conference* (Paper ID #32801). Virtual Event: American Society of Engineering Education (ASEE). https://doi.org/10.18260/1-2--36814

Anderson, J. R., & Reiser, B. J. (1985). The LISP tutor. *Byte*, *10*(4), 159-175. https://www.academia.edu/download/3240480/TheLISPTutor.pdf

Austing, R. H., Barnes, B. H., Bonnette, D. T., Engel, G. L., & Stokes, G. (1979). Curriculum '78: Recommendations for the undergraduate program in computer science – A report of the ACM curriculum committee on computer science. *Communications of the ACM*, *22*(3), 147-166. https://doi.org/10.1145/359080.359083

Bandura, A. (1997). *Self-efficacy: The exercise of control*. W H Freeman/Times Books/Henry Holt & Co. https://psycnet.apa.org/record/1997-08589-000

Barnes, T., Powell, E., Chaffin, A., & Lipford, H. (2008, February). Game2Learn: Improving the motivation of CS1 students. *Proceedings of the 3rd International Conference on Game Development in Computer Science Education (GDCSE '08)* (pp. 1–5). Miami, Florida: ACM. https://doi.org/10.1145/1463673.1463674

Barnes, T., Richter, H., Powell, E., & Chaffin, A., & Godwin, A. (2007). Game2Learn: Building CS1 learning games for retention. *ACM 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

Becker, B. A., & Quille, K. (2019, February). 50 years of CS1 at SIGCSE: A review of the evolution of introductory programming education research. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)* (pp. 338–344). Minneapolis, MN, USA: ACM. https://doi.org/10.1145/3287324.3287432

Ben-Ari, M. (2001). Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching*, *20*(1), 45-73. https://www.learntechlib.org/primary/p/8505/

Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *ACM SIGCSE Bulletin*, *39*(2), 32-36. https://doi.org/10.1145/1272848.1272879

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

Chiodini, L., Santos, I. M., Gallidabino, A., Tafliovich, A., Santos, A. L., & Hauswirth, M. (2021, June). Curated inventory of programming language misconceptions. *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education, Volume 1 (ITiCSE '21)* (pp. 380–386). Virtual Event, Germany: ACM. https://doi.org/10.1145/3430665.3456343

Code.org. (n.d.). https://code.org/educate/gamelab

CodeMonkey.org. (n.d.). www.codemonkey.com

Cote, A. C. (2020). *Gaming sexism: Gender and identity in the era of casual video games*. New York University Press. https://doi.org/10.18574/nyu/9781479838523.001.0001

Crow, T., Luxton-Reilly, A., & Wuensche, B. (2018, January). Intelligent tutoring systems for programming education: A systematic review. *Proceedings of the 20th Australasian Computing Education Conference (ACE '18)* (pp. 53–62). Brisbane, Queensland, Australia: ACM. https://doi.org/10.1145/3160489.3160492

Csikszentmihályi, M. (2018). *Flow: The psychology of optimal experience*. CreateSpace Independent Publishing Platform.

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

Denny, P., Luxton-Reilly, A., Tempero, E., & Hendrickx, J. (2011, March). CodeWrite: Supporting student-driven practice of Java. *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)* (pp. 471–476). Dallas, TX, USA: ACM. https://doi.org/10.1145/1953163.1953299

du Boulay, C., O'Shea, T., & Monk, J. (1981). The black box inside the glass box: Presenting computing concepts to novices. *International Journal of Man-Machine Studies*, *14*(3), 237–249. https://doi.org/10.1016/S0020-7373(81)80056-9

Edmondson, C. (2009). Proglets for first-year programming in Java. *ACM SIGCSE Bulletin*, *41*(2), 108-112. https://doi.org/10.1145/1595453.1595486

Farooq, M. S., Khan, S. A., Ahmad, F., Islam, S., & Abid, A. (2014). An evaluation framework and comparative analysis of the widely used first programming languages. *PLoS ONE*, *9*(2), e88941. https://doi.org/10.1371/journal.pone.0088941

Fouh, E., Akbar, M., & Shaffer, C. A. (2012). The role of visualization in computer science education. *Computers in the Schools*, *29*(1-2), 95-117. https://doi.org/10.1080/07380569.2012.651422

Franklin, D., Salac, J., Thomas, C., Sekou, Z., & Krause, S. (2020, March). Eliciting student scratch script understandings via scratch charades. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)* (pp. 780–786). Portland, OR, USA: ACM. https://doi.org/10.1145/3328778.3366911

Game2Learn. (2012). *Game2Learn lab archives*. North Carolina State University, Department of Computer Science. https://eliza.csc.ncsu.edu/archives.html

Harel, I., & Papert, S. (1991). *Constructionism: Research reports and essays, 1985–1990*. Ablex Publishing Corporation.

Hundhausen, C. D., Douglas, S. A., & Stasko, J. T. (2002). A meta-study of algorithm visualization. *Journal of Visual Languages & Computing*, *13*(3), 259-290. https://doi.org/10.1006/jvlc.2002.0237

Jeliot 3. (n.d.). *Publications*. http://cs.joensuu.fi/jeliot/pub.php

Johnson, C., McGill, M., Bouchard, D., Bradshaw, M. K., Bucheli, V. A., Merkle, L. D., Scott, M. J., Sweedyk, Z., Velázquez-Iturbide, J. A., Xiao, Z., & Zhang, M. (2016, July). Game development for computer science education. *Proceedings of the 2016 ITiCSE Working Group Reports (ITiCSE '16)* (pp. 23–44). Arequipa, Peru: ACM. https://doi.org/10.1145/3024906.3024908

Kafai, Y. B., & Burke, Q. (2015). Constructionist gaming: Understanding the benefits of making games for learning. *Educational Psychologist*, *50*(4), 313-334. https://doi.org/10.1080/00461520.2015.1124022

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., Kowalchuk, R. K., Lowman, L. L., Petoskey, M. D., Keselman, J. C., & 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

Kiili, K. (2006). Evaluations of an experimental gaming model. *Human Technology: An Interdisciplinary Journal on Humans in ICT Environments*, *2*(2), 187-201. https://jyx.jyu.fi/bitstream/handle/123456789/20195/1/HT_2006_v02_n02_p_187-201.pdf

Kölling, M., Quig, B., Patterson, A., & Rosenberg, J. (2003). The BlueJ system and its pedagogy. *Computer Science Education*, *13*(4), 249-268. https://doi.org/10.1076/csed.13.4.249.17496

Koulouri, T., Lauria, S., & Macredie, R. D. (2015). Teaching introductory programming: A quantitative evaluation of different approaches. *ACM Transactions on Computing Education*, *14*(4), Article No.: 26. https://doi.org/10.1145/2662412

Liu, A. S., Schunn, C. D., Flot, J., & Shoop, R. (2013). The role of physicality in rich programming environments. *Computer Science Education*, *23*(4), 315-331. https://doi.org/10.1080/08993408.2013.847165

Loksa, D., Margulieux, L., Becker, B. A., Craig, M., Denny, P., Pettit, R., & Prather, J. (2022). Metacognition and self-regulation in programming education: Theories and exemplars of use. *ACM Transactions on Computing Education*, *22*(4), Article No.: 39. https://doi.org/10.1145/3487050

Luxton-Reilly, A., Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J., & Szabo, C. (2018, July). A review of introductory programming research 2003-2017. *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018)* (pp. 342–343). Larnaca, Cyprus: ACM. https://doi.org/10.1145/3197091.3205841

Major, L., Kyriacou, T., & & Brereton, O. P. (2012). Systematic literature review: Teaching novices programming using robots. *IET Software*, *6*(6), 502 – 513. https://doi.org/10.1049/iet-sen.2011.0125

Malliarakis, C., Satratzemi, M., & Xinogalos, S. (2014). Educational games for teaching computer programming. In C. Karagiannidis, P. Politis, & I. Karasavvidis (Eds.), *Research on e-learning and ICT in education: Technological, pedagogical and instructional perspectives* (pp 87–98). Springer. https://doi.org/10.1007/978-1-4614-6501-0_7

McWhorter, W. I., & O'Connor, B. C. (2009, March). Do LEGO® Mindstorms® motivate students in CS1? *Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE '09)* (pp. 438–442). Chattanooga, TN, USA: ACM. https://doi.org/10.1145/1508865.1509019

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with Scratch. *Computer Science Education*, *23*(3), 239-264. https://doi.org/10.1080/08993408.2013.832022

Miljanovic, M. A., & Bradbury, J. S. (2018, November). A review of serious games for programming. *Proceedings of the 4th Joint International Conference on Serious Games (JCSG 2018)*. Darmstadt, Germany: Springer, Cham. https://doi.org/10.1007/978-3-030-02762-9_21

Mohanarajah, S. (2018). Increasing intrinsic motivation of programming students: Towards fix-and-play educational games. *Issues in Informing Science & Information Technology*, *15*, 69-77. https://doi.org/10.28945/4027

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., & Velázquez-Iturbide, J. A. (2003). Exploring the role of visualization and engagement in computer science education. *ACM SIGCSE Bulletin*, *35*(2), 131-152. https://doi.org/10.1145/782941.782998

National Science Foundation & Institute of Education Sciences. (2013, August). *Common guidelines for education research and development*. https://www.nsf.gov/pubs/2013/nsf13126/nsf13126.pdf

Nelson, G. L., & Ko, A. J. (2018, August). On use of theory in computing education research. *Proceedings of the 2018 ACM Conference on International Computing Education Research (ICER '18)* (pp. 31–39). Espoo, Finland: ACM. https://doi.org/10.1145/3230977.3230992

Papert, 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*. John Wiley and Sons. https://dl.acm.org/doi/10.5555/539521

Pears, A., & Rogalli, M. (2011, November). mJeliot: A tool for enhanced interactivity in programming instruction. *Proceedings of the 11th Koli Calling International Conference on Computing Education Research (Koli Calling '11)* (pp. 16–22). Koli, Finland: ACM. https://doi.org/10.1145/2094131.2094135

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

Price, B. A., Baecker, R. M., & Small, I. S. (1993). A principled taxonomy of software visualization. *Journal of Visual Languages & Computing*, *4*(3), 211-266. https://doi.org/10.1006/jvlc.1993.1015

Qian, Y., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education*, *18*(1), Article No.: 1. https://doi.org/10.1145/3077618

Robins, A. (2015). The ongoing challenges of computer science education research [Editorial]. *Computer Science Education*, *25*(2), 115-119. https://doi.org/10.1080/08993408.2015.1034350

Robins, A. V. (2019). Novice programmers and introductory programming. In S. A. Fincher, & A. V. Robins (Eds.), *The Cambridge handbook of computing education research* (pp. 327 - 376). Cambridge University Press. https://doi.org/10.1017/9781108654555.013

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

Robocode. (n.d.). https://robocode.sourceforge.io/

Ryan, R. M., & Deci, E. L. (2000). Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary Educational Psychology*, *25*(1), 54-67. https://doi.org/10.1006/ceps.1999.1020

Saito, D., Sasaki, A., Washizaki, H., Fukazawa, Y., & Moto, Y. (2017, November). Program learning for beginners: Survey and taxonomy of programming learning tools. *Proceedings of the 9th International Conference on Engineering Education (ICEED)*. Kanazawa, Japan: IEEE. https://doi.org/10.1109/ICEED.2017.8251181

Shabalina, O., Malliarakis, C., Tomos, F., & Mozelius, P. (2017, October). Game-based learning for learning to program: From learning through play to learning through game development. *Proceedings of the 11th European Conference on Games Based Learning* (pp. 571-576). Graz, Austria: Academic Conferences and Publishing International Limited. https://www.diva-portal.org/smash/get/diva2:1147690/FULLTEXT01.pdf

Shabalina, O., Vorobkalov, P., Kataev, A., & Tarasenko, A. (2008). Educational games for learning programming. *Educational games for learning programming, Book 6. International book series: Information science and computing* (pp. 79-83). Institute of Information Theories and Applications FOI ITHEA. http://hdl.handle.net/10525/1136

Sharmin, S. (2022). Creativity in CS1: A literature review. *ACM Transactions on Computing Education*, *22*(2), Article No.: 16. https://doi.org/10.1145/3459995

Shorn, S. P. (2018, June). Teaching computer programing using gamification. *Proceedings of the 14th International CDIO Conference* (pp. 1-10). Kanazawa, Japan: The CDIO™ Initiative. http://www.cdio.org/knowledge-library/documents/teaching-computer-programming-using-gamification

Smith, J. M. (2022, July). How do students learn to program? Investigating theory and practice with learning analytics. *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '22), Volume 2* (pp. 640-641). Dublin, Ireland: ACM. https://doi.org/10.1145/3502717.3532110

Soloway, E., & Spohrer, J. C. (1989). *Studying the novice programmer*. L. Erlbaum Associates. https://dl.acm.org/doi/book/10.5555/576212

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. https://cs.wellesley.edu/~tinkerblocks/CCSCNE14-AI2-events-first-paper.pdf

United States Bureau of Labor Statistics. (2022, September 9). Software developers, quality assurance analysts, and testers. *Occupational Outlook Handbook* [Online]. https://www.bls.gov/ooh/Computer-and-Information-Technology/Software-developers.htm

Vahldick, A., Mendes, A. J., & Marcelino, M. J. (2014, October). A review of games designed to improve introductory computer programming competencies. *Proceedings of the IEEE Frontiers in Education Conference (FIE)* (pp. 1-7). Madrid, Spain: IEEE. https://doi.org/10.1109/FIE.2014.7044114

Villareale, J., Biemer, C. F., El-Nasr, M. S., & Zhu, J. (2020, September). Reflection in game-based learning: A survey of programming games. *Proceedings of the 15th International Conference on the Foundations of Digital Games (FDG '20)* (Article No.: 81). Bugibba, Malta: ACM. https://doi.org/10.1145/3402942.3403011

Weinberg, G. M. (1971). *The psychology of computer programming*. Van Nostrand Reinhold.

Wikipedia. (n.d.). *Educational programming languages*. https://en.wikipedia.org/wiki/Category:Educational_programming_languages

Yadin, A. (2011). Reducing the dropout rate in an introductory programming course. *ACM Inroads*, *2*(4), 71-76. https://doi.org/10.1145/2038876.2038894

Zweben, S., & Bizot, B. (2022, May). 2021 Taulbie survey: CS enrollment grows at all degree levels, with increased gender diversity. *Computing Research News*, *34*(5), 2-82. Computer Research Association. https://cra.org/wp-content/uploads/2022/05/2021-Taulbee-Survey.pdf

APPENDIX A: PRE/POST TEST

Pretest: Conditional Branching.

Choose the best answer.

Q1. Which of the following is a valid java code to check whether the value in the integer variable x is an even number and if so, to display the text 'even'?

a	if (x % 2 = 0) System.out.print("even");	b	if (x % 2 = 0); System.out.print("even");
c	if (x % 2 == 0) System.out.print("even");	d	if x % 2 == 0 System.out.print("even");

Q2. Consider the code below. What will be printed?

```
x =50;
if (x < 500)
  System.out.print (" less than 500");
else if (x < 100)
   System.out.print (" less than 100");
```

A	less than 500	b	less than 500 less than 100
C	less than 100	d	Nothing will be printed

Q3. Which of the following java code correctly calculates the discount on a purchase? The discount will be 5% if the purchased amount is greater than $100, and it will be 10% if the purchase exceeds $500.

A	if (purchaseAmount <= 100)
	discount = 0;
	else if (purchaseAmount <= 500)
	discount = 0.05;
	else
	discount = 0.1;
		b	discount = 0;
			if (purchaseAmount > 100)
			discount = 0.05;
			else if (purchaseAmount > 500)
			discount = 0.1;

C	if (purchaseAmount <= 100)
	discount = 0;
	else (purchaseAmount > 100)
	discount = 0.05;
	else (purchaseAmount > 500)
	discount = 0.1;
		d	discount = 0;
			if (purchaseAmount > 500)
			discount = 0.01;
			else
			discount = 0.5;

Q4. Consider the following code (with improper indentation). What will be printed?

```
int score = 70, code = 0;
if (score > 80)
  if (code == 0)
       System.out.print( " pass");
else
  System.out.print( " fail")
```

A	pass	b	fail
C	pass fail	d	Nothing will be printed

Q5. Consider the following code. What will be printed?

```
int score = 70, code = 0;
if (score > 80) {
    if (code == 0)
        System.out.print( " pass");
} else
        System.out.print( " fail");
```

A	Pass	b	fail
C	pass fail	d	Nothing will be printed

Q6. Consider the following code. What will be printed?

```
int x= 0;
x = (x != 0) ? 0 : 1 ;
System.out.print( x );
```

A	1	b	0
C	10	d	Nothing will be printed

Q7. Consider the following code.

```
int daysInMonth=31;
switch (month) {
   case 1: case 3: case 5: case 7: case 8: case 10: case 12:
                                          daysInMonth = 31;
                                          break;
  case 4: case 6: case 9: case 11: daysInMonth = 30; break;
  case 2:  daysInMonth = 28; break;
   default: daysInMonth = 0;
 }
 System.out.print(daysInMonth);
```

What will be printed, if month = 4, month = 2, month = 1, and month = 13 respectively?

A	30 28 31 0	b	28 28 31 0
C	31 31 31 0	d	0 0 0 0

Q8. Consider the following code.

```
int daysInMonth=31;
switch (month) {
   case 1: case 3: case 5: case 7: case 8: case 10: case 12: daysInMonth = 31;
  case 4: case 6: case 9: case 11: daysInMonth = 30;
  case 2:  daysInMonth = 28;
   default: daysInMonth = 0;
 }
 System.out.print(daysInMonth);
```

What will be printed if month = 4, month = 2, and month = 1 respectively?

A	30 28 31 0	b	28 28 31 0
C	31 31 31 0	d	0 0 0 0

APPENDIX B: QUESTIONNAIRE

BASICS

1. Gender (select one): Male ○ Female ○ Other ○

2. Indicate your education level in programming:

Less than a year ○	1 year ○	2 years ○	3 years ○	more than 3 years ○

3. Indicate your level of experience in playing games (any genre):

Less than a year ○	1 year ○	2 years ○	3 years ○	more than 3 years ○

4. Indicate your level of experience in playing casual games (simple games like Angry Birds):

Less than a year ○	1 year ○	2 years ○	3 years ○	more than 3 years ○

GAME-RELATED

5. How many times did you play this given game?

more than 3 times ○	3 times ○	2 times ○	once ○	never ○

6. The game used in this study is just a prototype created for research purpose: it is not an industry standard game. Keeping this in mind, indicate your experience in using this game.

very boring ○	boring ○	neutral ○	like it ○	like it very much ○

7. Suppose for your next formative assessment/revision you have the option of choosing between a game-based revision and a traditional paper-based revision. Which option will you choose?

never game-based ○	sometimes game-based ○	either is okay ○	mainly game-based ○	always game-based ○

AUTHORS

Dr. Selvarajah (Mohan) Mohanarajah received his Ph.D. in Computer Science from the Massey University, New Zealand. He is currently serving as a Professor of Computer Science at University of North Carolina at Pembroke, NC, USA. His research interests include AI in Computer Science Education, Educational Games, Cybersecurity and Machine Learning.

Dr. Thambithurai Sritharan received his Ph.D. in Mathematics from the University of Sussex, U.K. Senior Lecturer at the University of Colombo, School of Computing, University of Colombo, Colombo, currently teaching Mathematics and Theoretical Computer Science courses.