RoboBUG: A Serious Game for Learning Debugging

Michael Miljanovic
Software Quality Research Lab
University of Ontario Institute of Technology
Oshawa, ON, Canada
michael.miljanovic@uoit.ca

Jeremy S. Bradbury
Software Quality Research Lab
University of Ontario Institute of Technology
Oshawa, ON, Canada
jeremy.bradbury@uoit.ca

ABSTRACT

Debugging is an essential but challenging task that can present a great deal of confusion and frustration to novice programmers. In general, Computer Science education does not sufficiently address the challenges that students face when identifying bugs in their programs. To help students learn effective debugging techniques and to provide students a more enjoyable and motivating experience, we have designed the RoboBUG game. RoboBUG is a serious game that can be customized with respect to different programming languages and game levels.

CCS Concepts

•**Social and professional topics** → **Computer science education;** •**Software Engineering** → *Testing and Debugging;* •**Applied computing** → Computer games;

Keywords

debugging, programming, software engineering, computer science, education, serious games, game-based learning.

1. INTRODUCTION

The majority of tool research related to programming and software development focuses on advancing the state-of-the-art in software developer practices. We advocate that the development of programming and software development learning tools is equally important as software developers need to first learn best practices before they can effectively perform them.

It is essential that programmers who seek to write reliable, high quality source code be able to efficiently identify and repair **bugs** in program code [21]. The process of fixing these bugs (i.e. **debugging**) has been shown to consume up to 50% of the time of a large software project [5]. Furthermore, the ability to debug code is not easily acquired, and experts have a significant advantage over novices [3]. Game-based learning has already been implemented and proven effective for computer programming education [9, 13] which suggests that it may also prove useful in debugging education. Serious games should not only help players achieve learning outcomes but should also be a fun and positive experience. Previous studies have shown that motivated and engaged learners will perform more effectively [8, 15]. The benefits of serious games suggest they may be an effective way to counter the frustration typically associated with debugging.

Novices programmers enrolled in first year computer programming courses, are often unprepared for the task of debugging. This lack of preparedness is compounded by the fact that there is no established set of best practices for teaching debugging [7]. There is a lack of online resources dedicated to helping novices learn to debug [3]. Even students with a good understanding of how to write programs still struggle with debugging [1]. These students may have the ability to fix bugs in their programs, but only after accomplishing the more difficult task of first finding the bugs.

In general, the lack of accessibility to debugging techniques leads students to have a primarily negative experience, even when given the opportunity to learn debugging [17]. This fact is particularly problematic when students conclude that debugging skills are based on aptitude and are unable to be learned [4]. We hope to address the problem of accessibility and frustration with debugging by creating a puzzle-based serious game, RoboBUG, that will help students achieve debugging learning outcomes in an enjoyable rather than tedious way.

The creation of RoboBUG required us to address a number of challenges, including:

1. **Game design:** How can debugging activities be represented as game tasks/actions? How can these tasks be connected to produce enjoyable and cohesive gameplay?

2. **Game learning data:** What debugging topics and learning materials should be used in the game to achieve the desired learning outcomes while minimizing frustration?

3. **Game evaluation:** How do we design our study to effectively assess debugging learning as well as level of enjoyment?

In addition to these challenges, we needed to consider if the combination of game design and learning data would allow the player to retain debugging concept information after the game's completion. In the remaining sections of our paper we will present background on debugging and game-

The 48th ACM Technical Symposium on Computer Science Education 2017 March, Washington, USA

DOI: 10.1145/1235

Figure 1: A screenshot of the RoboBUG game

The RoboBUG game interface is divided into two regions: (1) The code region (left) is the area that the user controls the avatar to navigate the source code, use different debugging tools and eventually find bugs, (2) The sidebar (right) is used to provide the user with information about available debugging tools, the currently active tool and the time remaining in the level. In addition to the above regions, the RoboBUG game also utilizes dialogs (bottom) to provide contextual information to the user such as feedback from debugging activities.

based learning (Section 2), our RoboBUG serious game (Section 3), the results of a preliminary evaluation (Section 4.1) as well as our two-part full evaluation (Section 4.2). All of our evaluations were conducted with undergraduate students at the University of Ontario Institute of Technology (UOIT).

2. BACKGROUND

2.1 Debugging

The design of any learning tool must first consider the material that needs to be included. In our case, we selected a combination of static and dynamic debugging techniques for inclusion in the RoboBUG game. The techniques we chose to include were:

1. **Code Tracing**: The act of reading through code to make sure it is behaving properly

2. **Print Statements**: Displaying internal program status information as output to determine where faults occur

3. **Divide-and-Conquer**: Systematically separating source code into sections in an effort to isolate a bug

4. **Breakpoints**: Pausing a program during run-time to check the internal value of variables at specific times

2.2 Game-based Learning

Serious games for Computer Science is a popular research area [14, 20] especially with respect to learning how to write computer programs. Games such as Code Hunt [18] help learners develop their skills through puzzle tasks that require players to write programs in order to solve a specific problem. Serious programming games usually focus on the act of writing programs, or to creating programs using a drag-and-drop interface, as seen with Program Your Robot [10]. Robot ON! [12] is an uncommon example of a puzzle-based game that does not require any programs to be written. Instead, Robot ON! focuses on program understanding and comprehension by requiring players to read source code written by someone else.

3. THE ROBOBUG GAME

RoboBUG[1] (see Figure 1) is a serious game intended to be played by first-year computer science students learning to debug. We chose to design RoboBUG as a puzzle-type game, as puzzles have been shown to be effective for both helping to learn material as well as demonstrating higher level concepts such as critical thinking and problem-solving skills [16].

RoboBUG was implemented in C# using the Unity game engine[2] and open source media elements. Although the standard version of RoboBUG is based on debugging in C++, it

[1]available at https://github.com/sqrlab/robobug.
[2]https://unity3d.com/

Table 1: The levels and tools in the default C++ version of RoboBUG

Level	Tools	Description
Level 1	Bugcatcher	The Mech Suit is unable to stand because it cannot correctly calculate the physical forces acting upon it. The player must practice **code tracing** by identifying bugs while reading through source code that calculates the average value of a set of physical forces.
Level 2	Bugcatcher, Activator	The Mech Suit is failing to correctly identify the most dangerous bugs that appear in its viewing area. The player needs to use **print statements** to identify program bugs in an algorithm that sorts the externally viewable bugs from most to least dangerous.
Level 3	Bugcatcher, Activator, Warper, Commenter	The Mech Suit vision system has been infected by bugs and no longer functions at all. To fix it, the player must search for a bug in the robot's visual color database. Since the database is large, the player will need to employ a **divide-and-conquer** strategy and comment out different blocks of source code.
Level 4	Bugcatcher, Activator, Breakpointer, Warper	The Mech Suit is not able to correctly calculate which bugs are closest in proximity. This is the most challenging level, requiring the player to use several debugging tools to locate bugs across multiple functions. This includes the use of **breakpoints** to display variable values and program state at run-time while locating the bug in a distance calculation function.

Figure 2: A RoboBUG comic storyboard used to advance the game plot and setup a new game level

The RoboBUG game utilizes a plot to engage the user in the debugging learning tasks. Each level begins with a four pane comic that provides plot details. For example, in the above comic, bugs have infected the mech suit's vision system and the user is tasked with detecting the bugs in the relevant mech suit source code.

also includes a framework that allows instructors to create their own levels using different programming languages and insert these new levels into the game with minimal effort. New levels are specified using XML and can be customized with respect to different aspects of a level, including time limit, available tools, output text, and source code.

The game involves a player taking the role of a scientist attempting to purge bugs from their infected 'Mech Suit' and save the world from an alien bug invasion. In each level the player must fix a particular part of their Mech Suit (e.g. vision system) by figuring out where the bug is hidden (see Figure 2 for a plot example). the game is finished once the player has completed all levels, found all of the bugs, and has a working Mech Suit.

The default version of RoboBUG includes four levels that teach different debugging techniques in C++ (see Table 1). Each of these levels includes: a tutorial that introduces new debugging tools, 2-3 subproblems that contain small debugging tasks and partial source code and a final challenge that combines the tools introduced in the tutorials and the knowledge gained from the subproblems. The final challenge will typically involve detecting a bug in the full program.

A player's progress through the game is recorded in a set of log files that allow gameplay to resume should they choose to exit and play again later. Prototype testing has shown that the game with the default levels takes approximately 30 minutes to complete. RoboBUG has been used with the default levels during an introductory programming course at UOIT.

3.1 Game Levels

In developing the default version of RoboBUG, we thought about the order and content that should be included in the game levels. We chose to first introduce code tracing as it can be used in combination with other techniques and it is not too time-consuming due to the short length of the example programs. Next, we selected the use of print statements in order to help novices learn to identify program behavior at run-time. This was followed by the strategy of divide-and-conquer, where novices can reduce the search space for bugs by commenting out code that is guaranteed to contain no bugs. Finally, we adapted some features of a debugger so that novices can learn the concepts of breakpoints and the value of observing a program's execution state during run-time.

3.2 Game Mechanics

In order to complete a level, the player must navigate their avatar in the code region (see Figure 1) using the arrow keys. Once a bug has been found, the player uses the *bugcatcher* tool to "capture" the bug at a specific line of code. If the location is incorrect, the player fails the level and must start it again. The player can also fail if he or

she expends all of the available tools, or does not complete the level within the time allotted. As the game progresses, the player is given additional kinds of tools that can activate print statements (*activator* tool), comment out source code (*commenter* tool), set and trigger breakpoints (*breakpointer* tool) and jump to different regions of the program (*warper* tool). Completion of each level requires the player to use the tools available before using the 'bugcatcher' tool to complete the level.

4. ROBOBUG EVALUATION

Unfortunately, serious games, including those in the Computer Science education literature, tend to be published without a proper evaluation [11] making it difficult to assess their impact on learning. In order to determine the efficacy of a game for learning, the most effective type of evaluation is a user study [6].

4.1 Preliminary Evaluation

A preliminary evaluation of an early RoboBUG prototype assessed the value of the game in comparison with traditional assignment-based learning. Participants were split into two random groups: a control group of 12 participants who completed a short assignment, and an experimental group of 11 participants who played the RoboBUG game. All participants were undergraduate students at UOIT with knowledge of the C++ programming language. Both the assignment and the game were based on the same debugging techniques (see Section 2.1) and similar source code. Although the evaluation found no significant difference with regards to achieving learning outcomes, participants tended to find RoboBUG to be more 'fun'. However, the first prototype was also found to be too complicated and many participants were not able to complete the game without hints. The current version of RoboBUG was updated to address these issues, by subdividing levels, reducing complexity, and providing opportunities for players to fail and replay the levels.

4.2 Full Evaluation

To evaluate the current version of RoboBUG, we conducted a pair of user studies addressing the following research questions (RQs):

- **RQ1:** *Is the RoboBUG game playable by undergraduate students?*
- **RQ2:** *Does RoboBUG actual improve a student's understanding of debugging techniques (i.e., achieve learning outcomes)?*
- **RQ3:** *Do students enjoy playing the RoboBUG game?*

Our evaluation involved the participation of first year computer science students at UOIT who had familiarity with C++. Participants were between the ages of 18 and 25, with mixed demographics, gender, and race.

4.2.1 Evaluating RQ1

The first half of our user study was a 5 participant experiment to ensure that RoboBUG was an accessible and playable game (**RQ1**). Participants individually took part in a 1 hour session during which they were observed playing the RoboBUG game for at least 30 minutes. Following the game play, the participants completed a 20 minute interview where they answered both structured and unstructured questions about their experience, including:

- What did you learn about debugging that you didn't know before?
- What aspect/part of the game was most enjoyable?
- What aspect/part of the game was the most frustrating?
- What aspect/part of the game was most innovative?
- What aspect/part of the game would you like to see improved?

During the interview, participants provided feedback about parts of the game where they became stuck or frustrated. The goal was for RoboBUG to be playable before its efficacy as a serious game is measured.

Overall, the game was viewed positively by the participants, who particularly enjoyed the game elements that differentiated RoboBUG from real debugging. During the interviews, the participants gave the following opinions:

- *"[I enjoyed] trying to test my skills with how good I am with debugging."*
- *"It's a great tool, that's what I can say."*
- *"The way that the divide and conquer was set up was pretty cool."*
- *"The inclusion of breakpoints was kind of innovative."*
- *"[The warper tool] was interesting because I thought all of the code would be in one class."*
- *"I think that the warper/commenting, being able to zip between different segments of code was really good."*

Our study identified some important playability issues with the game, including control problems and concerns with some of the game's levels. In particular, participants in all our evaluations had significant challenges trying to debug a level containing an off-by-one index bug. This bug was cited by participants to be especially frustrating, due to players having trouble identifying print statements that would help them find the bug. There was also some confusion with the way that the game handled commenting, as players did not realize that a persisting error meant the bug was **not** commented out. A frequently requested change to the game was the idea of a 'hint' system that would provide better feedback to players who fail to complete levels.

4.2.2 Evaluating RQ2 and RQ3

In the second half of our study we evaluated the game's ability to help students achieve learning outcomes (**RQ2**) as well as the user experience (**RQ3**). This part of our study involved a larger sample size (14 participants) and took approximately 1 hour to complete. Participants in the second half of the study first completed the Positive and Negative Affect Scale [19], which assesses the user's feelings (see Figure 4). The PANAS scale in our study contained 12 positive words and 13 negative words. They then completed a pretest to evaluate their debugging skills prior to playing the

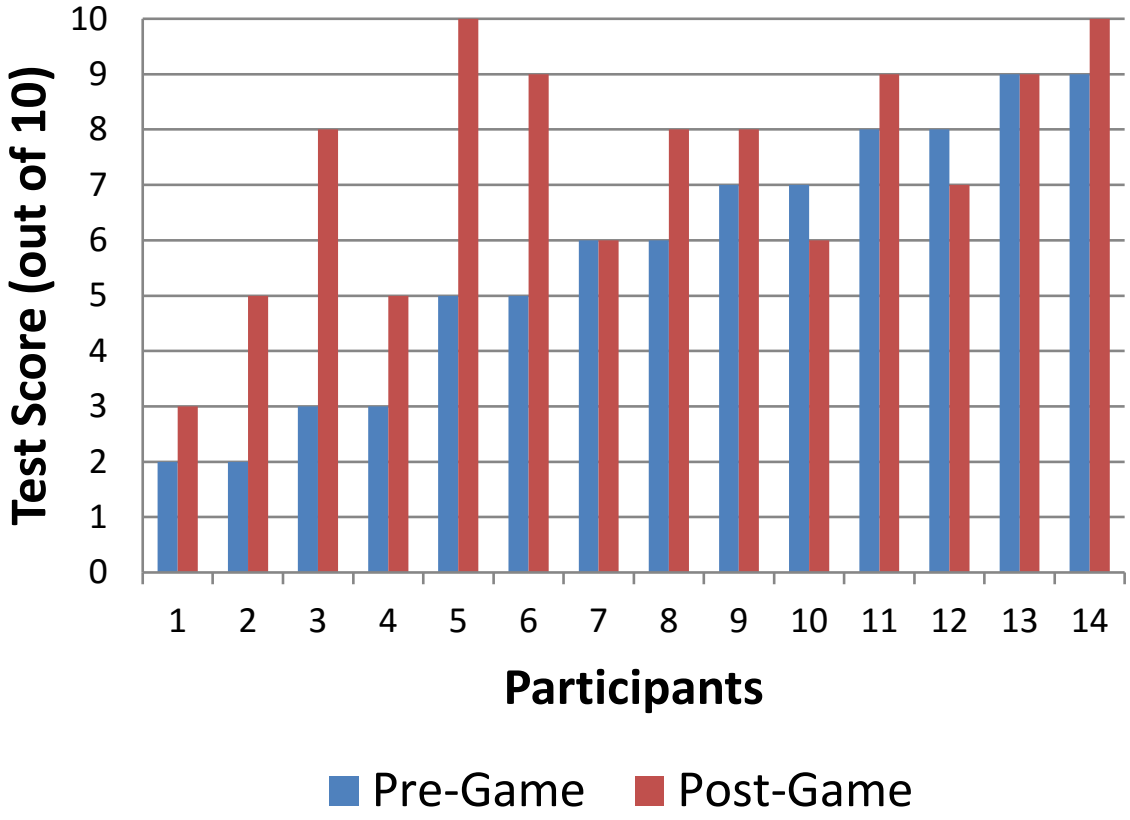

Figure 3: Skill test scores from before and after the RoboBUG game was played.

1 Very Slightly / Not at all	2 A Little	3 Moderately	4 Quite a bit	5 Extremely

1. Interested	_____	5. Strong	_____
2. Distressed	_____	6. Guilty	_____
3. Excited	_____	7. Scared	_____
4. Upset	_____	8. Enthusiastic	_____

Figure 4: Positive-Negative Affect Scale (PANAS) [19]

The Positive-Negative Affect Scale is a self-evaluation assessment of affect based on words that associate with positive or negative emotions. Total affect is calculated by adding all of the positive or negative item ratings, with higher scores representing higher affect levels.

Paired-samples t-test				
	Mean	Std. Dev.	t(13)	p value
Test Scores			3.0970	0.0085
Pre-Game	5.71	2.46		
Post-Game	7.36	2.10		
Positive Affect			2.1272	0.0531
Pre-Game	45.93	5.40		
Post-Game	42.93	8.92		
Negative Affect			0.7555	0.4634
Pre-Game	18.86	5.67		
Post-Game	20.21	10.64		

Figure 5: Paired-samples t-test

There was a significant increase in debugging test scores after the game was played. No significant changes in positive or negative affect scores were observed.

RoboBUG game. This evaluation test included ten multiple choice questions about the four debugging techniques used in the game (see Section 2.1). After the game, participants completed the evaluation test and PANAS questionnaire again to assess if there were any changes.

We analyzed the pre and post test data and PANAS data using a paired samples t-test (see Figure 5). Our results indicate that RoboBUG helps students to achieve the debugging learning outcomes (see Figure 3). Players became familiar with the nature of the debugging techniques, and were able to practice debugging and solve problems in a satisfying manner. Unfortunately, there was a decrease in positive affect and an increase in negative affect (non-statistically significant), indicating that the game still led to some user frustrations. It is possible that the game remains less frustrating than real debugging tasks, but our observations of participants suggest that the lack of a hint system and the difficulty

Positive Keyword	Average Change	Negative Keyword	Average Change
Interested	-0.64	Anxious	-0.21
Enthusiastic	-0.64	Nervous	-0.14
Alert	-0.50	Guilty	-0.07
Excited	-0.36	Stressed	-0.07
Determined	-0.36	Depressed	-0.07
Attentive	-0.36	Scared	0.00
Proud	-0.14	Distressed	0.07
Inspired	-0.14	Hostile	0.07
Happy	-0.07	Jittery	0.29
Confident	0.00	Afraid	0.29
Active	0.07	Irritable	0.36
Strong	0.14	Upset	0.43
		Ashamed	0.43

Figure 6: Positive-Negative Affect Scores

This graph shows the average change of affect for all participants based on each question on the PANAS. Positive scores indicate that participants associated ***more*** *with that emotion after playing RoboBUG, and negative scores indicate that players associate* ***less*** *with that emotion after playing RoboBUG.*

of the tasks were major challenges for participants. Ultimately, our game still requires participants to debug code and completely removing frustration related to debugging is still an open problem.

5. SUMMARY & CONCLUSIONS

We have presented the RoboBUG game as a possible solution to the challenge of learning debugging in first year Computer Science. Our evaluation of RoboBUG showed that the game helps students to achieve learning outcomes, but has a non-statistically significant impact on affect.

In addition to RoboBUG, we have also developed a prequel game called Robot ON! that will allow RoboBUG to be played by users who have no previous programming experience [12]. Future work on RoboBUG could include adding a hint system, a points system for competitive play, adding a cooperative multi-player mode, and improving the replayability by using program mutation [2] to generate random bugs each time a level is played. We are also planning a longitudinal study of RoboBUG in a first year programming course at UOIT.

6. ACKNOWLEDGEMENTS

This research was partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

7. REFERENCES

[1] M. Ahmadzadeh, D. Elliman, and C. Higgins. An analysis of patterns of debugging among novice computer science students. In *Proc. of 10th SIGCSE Conf. on Innovation and Technology in Comp. Sci. Education(ITICSE '05)*, pages 84–88, 2005.

[2] J. H. Andrews, L. C. Briand, and Y. Labiche. Is mutation an appropriate tool for testing experiments? In *Proc. of ICSE 2005*, pages 402–411, May 2005.

[3] E. Carter and G. Blank. Debugging Tutor: Preliminary evaluation. *J. of Computing Sciences in Colleges*, pages 58–64, 2014.

[4] M.-W. Chen, C.-C. Wu, and Y.-T. Lin. Novices' debugging behaviors in VB programming. In *Proc. of Learning and Teaching in Comp. and Eng. (LaTiCE 2013)*, pages 25–30, Mar. 2013.

[5] D. Chuntao. Empirical study on college students' debugging abilities in computer programming. In *Proc. of 1st Int. Conf. on Info. Sci. and Eng. (ICISE 2009)*, pages 3319–3322, Dec. 2009.

[6] H. Desurvire, M. Caplan, and J. A. Toth. Using heuristics to evaluate the playability of games. In *Proc. of CHI '04 - Extended Abstracts*, pages 1509–1512, 2004.

[7] S. Fitzgerald et al. Debugging from the student perspective. *IEEE Trans. on Education*, 53(3):390–396, 2010.

[8] R. Garris, R. Ahlers, and J. E. Driskell. Games, motivation, and learning: A research and practice model. *Simulation & Gaming*, 33(4):441–467, 2002.

[9] R. Ibrahim et al. Students perceptions of using educational games to learn introductory programming. *Comp. and Info. Sci.*, 4(1):205–216, 2010.

[10] C. Kazimoglu, M. Kiernan, L. Bacon, and L. Mackinnon. A Serious Game for Developing Computational Thinking and Learning Introductory Computer Programming. *Procedia - Social and Behavioral Sciences*, 47:1991–1999, 2012.

[11] F. Ke. A qualitative meta-analysis of computer games as learning tools. *Handbook of Research on Effective Electronic Gaming in Education*, 2009.

[12] M. A. Miljanovic and J. S. Bradbury. Robot On!: a serious game for improving programming comprehension. In *Proc. of the 5th International Workshop on Games and Software Engineering*, pages 33–36. ACM, 2016.

[13] M. Muratet, P. Torguet, J.-P. Jessel, and F. Viallet. Towards a Serious Game to Help Students Learn Computer Programming. *Int. J. of Comp. Games Tech.*, 2009:1–12, 2009.

[14] J. O'Kelly and J. P. Gibson. RoboCode & problem-based learning : A non-prescriptive approach to teaching programming. In *Proc. of 11th SIGCSE Conf. on Innovation and Technology in Comp. Sci. Education(ITICSE '06)*, pages 217–221, 2006.

[15] V. J. Shute. Stealth assessment in computer-based games to support learning. In *Computer Games and Instruction*, volume 55, pages 503–524, 2011.

[16] A. Siang. Theories of learning: a computer game perspective. In *Proc. of 5th Int. Symp. on Multimedia Soft. Eng. (ISMSE 2003)*, pages 239–245, Dec. 2003.

[17] B. Simon et al. Debugging assistance for novices. In *Working Group Reports on Innovation and Tech. in Comp. Sci. Education (ITiCSE-WGR '07)*, pages 137–151, 2007.

[18] N. Tillmann and J. Bishop. Code Hunt: Searching for secret code for fun. In *Proc. of 7th Int. Work. on Search-Based Soft. Testing (SBST 2014)*, pages 23–26, 2014.

[19] D. Watson, L. a. Clark, and A. Tellegen. Development

and validation of brief measures of positive and negative affect: The PANAS scales. *J. of Personality and Social Psychology*, 54(6):1063–1070, 1988.

[20] W.-T. Wong and Y.-M. Chou. An interactive Bomberman game-based teaching/learning tool for introductory C programming. In *Proc. of 2nd Int. Conf. on Edutainment*, pages 433–444, Jun. 2007.

[21] A. Zeller. *Why programs fail: a guide to systematic debugging*. Elsevier, 2009.