Robot ON!: A Serious Game for Improving Programming Comprehension

Michael A. 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

A number of educational games have been created to help students programming. Many of these games focus on problem solving and the actual act of writing programs, while very few focus on programming comprehension. We introduce a serious game called Robot ON! aimed at players who have never programmed before. Unlike other serious programming games, Robot ON! focuses on comprehension rather than problem-solving challenges; players do not actually write any programs, but are instead given the task of demonstrating their knowledge and understanding of a program's behavior. Robot ON! includes tools that allow players to demonstrate understanding of variable values, data types, program statements, and control flow. We include an evaluation plan to assess Robot ON!'s playability, enjoyment, and benefits to program comprehension.

Keywords

programming, software engineering, computer science, education, serious games, game-based learning, code walkthrough, code review, program comprehension

1. INTRODUCTION

A serious game is a game designed specifically for a purpose other than entertainment [5], such as education or training. Serious games usually provide learners with an opportunity to develop skills in an environment free from obtrusive observation and stress. For many of today's learners who have previous experience with games, a serious game has the potential to provide significant motivational and educational advantages over traditional lectures.

Software engineering and computer science are a natural fit for the application of serious games. For example, the task of programming is very similar to puzzle-solving games, where a player needs to identify the best solution for a given challenge. This is fortunate because programming itself tends to generate a great deal of frustration for learners, which can lead to students withdrawing from computer science programs [2].

Although many serious games about programming already exist, they typically deal with teaching program writing and creation to novices. However, novices tend to have substantial problems when working with programs they did not write themselves [6]. Novices lack *programming comprehension.* For example, novices can reuse sample code from a textbook, but that does not necessarily mean they understand what it does or how it works.

We hope to address the problems of accessibility, frustration, and the need for programming comprehension skills by creating a puzzle-based serious game, Robot ON!. The goal of Robot ON! is to help students achieve programming comprehension learning outcomes while being enjoyable rather than tedious. Robot ON! was designed in accordance with the ACM and IEEE joint computer science curricula [1]. The tasks that players must complete in the game require players to demonstrate understanding of computer programming in order to succeed. Players will fail and have to retry a level if they do not correctly follow each instruction and demonstrate program comprehension; finishing the game is only possible once a player has completed all of the game's tasks and shown that they understand all of the source code they encounter.

In the remaining sections of our paper we will present background on serious educational games and program comprehension (Section 2), our Robot ON! serious game (Section 3), and open research questions as well as our plan for evaluation (Section 4).

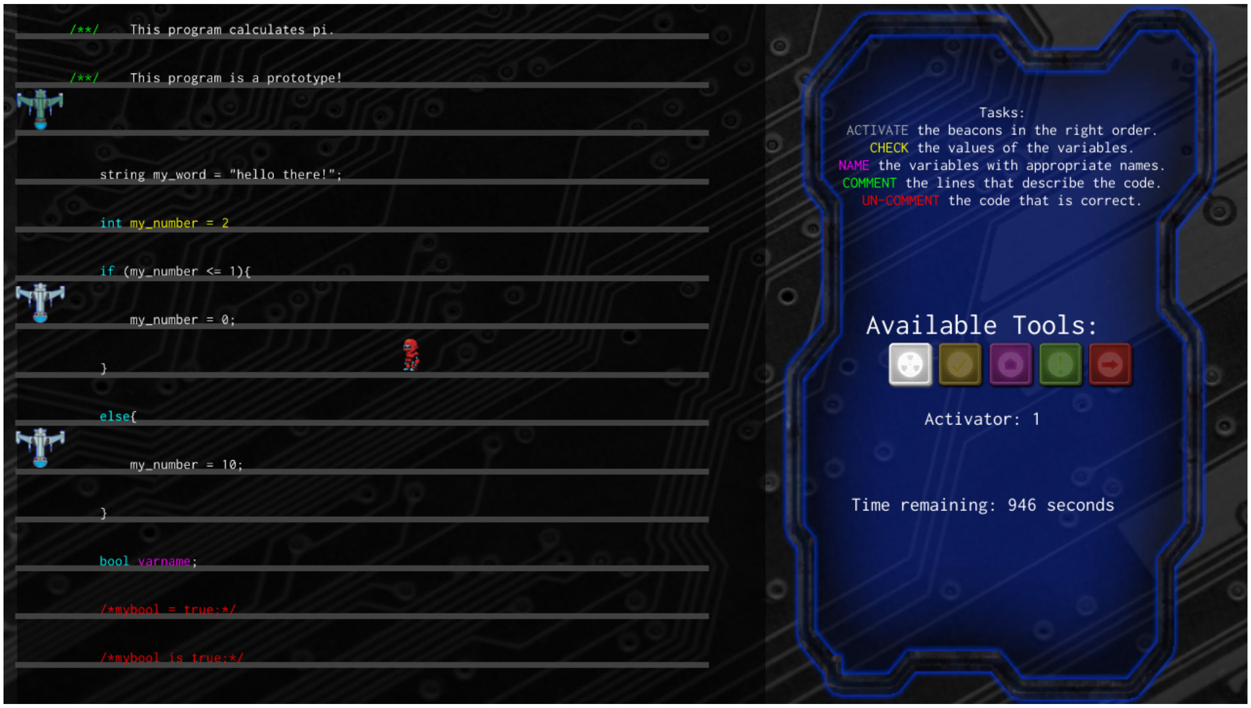
2. BACKGROUND

2.1 Serious Games for Programming

A wealth of games exist for helping novices learn how to program, typically through writing programs inside games. Players must write source code that solves a particular set of problems, and game environments allow them to observe the results of their attempts in a visually appealing way. Games such as EleMental [3] provide students with opportunities to learn through puzzle-solving game play. In EleMental, players learn recursion through writing code for a depth first search of a tree data structure.

However, writing code to meet specific requirements is not the only form of game play; the developers of Code Hunt [12] made their game successful by providing players with the expected results of a correct solution, and requiring them to fill in the blanks of an unknown and incomplete program.

GAS'16, May 16 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4160-8/16/05...$15.00
DOI: http://dx.doi.org/10.1145/2896958.2896962



```
/**/    This program calculates pi.

/**/    This program is a prototype!


     string my_word = "hello there!";

     int my_number = 2

     if (my_number <= 1){

         my_number = 0;

     }

     else{

         my_number = 10;

     }

     bool varname;

     /*mybool = true;*/

     /*mybool is true;*/
```

Tasks:
ACTIVATE the beacons in the right order.
CHECK the values of the variables.
NAME the variables with appropriate names.
COMMENT the lines that describe the code.
UN-COMMENT the code that is correct.

Available Tools:

Activator: 1

Time remaining: 946 seconds

Figure 1: A screenshot of the Robot ON! game

The Robot ON! game interface is divided into two regions: (1) The code region (left) is the area where the user controls the avatar to navigate source code and complete their assigned investigative tasks, (2) The sidebar (right) is used to provide the user with information about the tasks they must complete, their available tools, the active tool, and time remaining in the level.

Although real programming involves the creation of code to meet certain specifications, novices may find Code Hunt's approach to be more effective for learning.

Although the majority of programming games require players to write their own code, this is not necessarily the only successful approach to teaching computer science. A subset of serious games use graphical user interfaces (GUIs) to allow players to write programs using a drag-and-drop technique. One of the first studies that examined the use of GUI programming was Alice2 [8], which allowed users to drag and drop tiles that represent blocks of code. This approach has some trade-offs, as it alleviates some of the frustration associated with learning the syntax of a new programming language, but does not help students to learn how to write programs in a real language.

Very few programming games focus on programming tasks aside from writing. One example of this is our debugging-based game called RoboBUG [9], that served as our inspiration for Robot ON!. In RoboBUG, players learned real debugging techniques by completing a series of different levels that do not require any code to be written. For example, one of the RoboBUG levels introduces the use of breakpoints to students who have never used a real debugging tool. Unfortunately, players must be able to comprehend C++ programs in order to play the game, which limits the potential audience. RoboBUG's game engine, media elements, and control schemes were used as part of the Robot ON! game, but the games differ in their purpose, underlying story, gameplay, available tools, levels, and tasks. We propose using Robot ON! to enhance program comprehension before introducing introductory students to RoboBUG.

2.2 Program Comprehension

Programming comprehension is the ability of a programmer to understand and distinguish what a program is doing and what it is supposed to do [7]. We chose to focus on programming comprehension in order to help students learn to debug, as debugging tasks may take up to 50% of a program's development time [4]. According to Gugerty, programming comprehension skills are significantly more important than a programmer's knowledge of debugging strategies when it comes to debugging a program. Aside from debugging, program comprehension is a critical skill for being able to quickly understand code written by others, particularly when a new programmer has joined an existing development team, or when they are required to work on existing code [10].

3. THE ROBOT ON! GAME

Robot ON![1] (see Figure 1) is a serious game we designed for first-year software engineering or computer science students learning C++. We chose to design Robot ON! as a puzzle-type game, as puzzles have been shown to be effective tools for both helping to learn material as well as demonstrating higher level concepts such as critical thinking and problem-solving skills [11].

[1] available at: https://github.com/sqrlab/RobotON.

Table 1: The tools and associated tasks in Robot ON!

Tool	Task
Activator	The player must activate a sequence of beacons in the correct order. Activating the beacons in the wrong order, including activating a beacon too many times, causes the player to fail the level. This task teaches the concept of **control flow** in combination with loops and if statements.
Checker	The player must assess the value of certain variables and type the expected output into a prompt window. The player fails if they do not type in the correct answer. This task is a check to ensure players are keeping track of **variable values** in the program.
Namer	The player must determine appropriate names for variables used in the program. Calling a string variable 'myInt' or calling a boolean variable 'size' will lead to failure. This task is used to see if players understand the context of the program and to ensure they can identify variable **data types**.
Commenter	The player must identify the lines in the program that describe the behavior of the code. Commenting incorrect lines or trying to comment a line of code that is not a description will result in failure. This task allows players to verify a program's behavior and learn how to distinguish code from text.
Un-Commenter	The player must select comments that contain code and un-comment the correct code. Players who un-comment code that causes the program to work incorrectly will fail. This task demonstrates the use of comments other than for description, and requires players to understand how code can be changed.

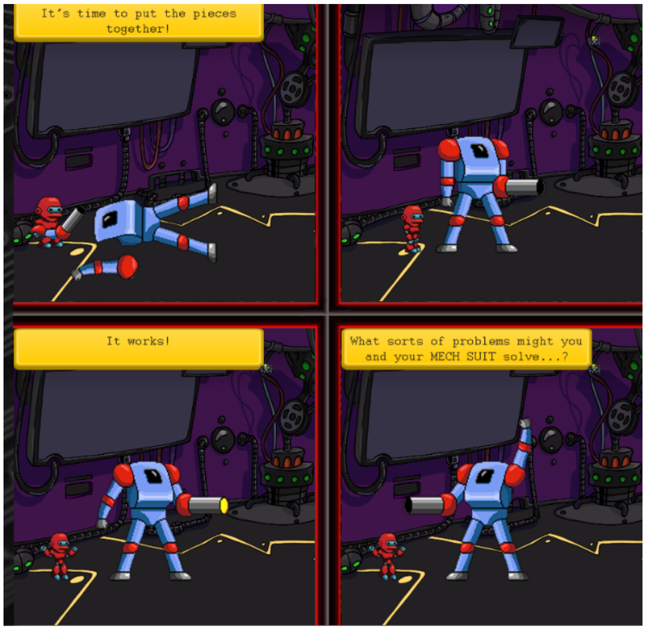

It's time to put the pieces together!
It works!
What sorts of problems might you and your MECH SUIT solve...?

Figure 2: A Robot ON! comic storyboard used to reward players who have beaten the game

The game involves a player taking the role of a scientist attempting to activate an unearthed 'Mech Suit'. In each level the player must accomplish a series of comprehension tasks to eventually activate all of the Mech Suit's systems (see Figure 2). Once the player has accomplished all the tasks in a level, they successfully activate a new robot system and proceed to the next level.

Robot ON! was implemented in C# using the Unity game engine and open source media elements. A player's progress through the game is recorded in a set of log files that allow players to replay any levels that they have already completed. Although the standard version of Robot ON! is based on programming in C++, it 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. C++ was selected as the default language because it is the first programming language taught to students at the University of Ontario Institute of Technology, and this will be the pool of students from whom we will select our participants for the evaluation study. New levels are specified using XML and can be customized with respect to different aspects of a level, including time limit, available tools, required tasks, and source code.

3.1 Game Mechanics

Robot ON! includes five different tools, which can be used in any combination inside game levels (see Table 1). These tools are introduced to players one at a time, using custom designed tutorial levels with simple examples of their use. Over the course of the game, players become familiar with the the tools and corresponding puzzle tasks.

In order to finish a level, the player must complete a set of assigned comprehension tasks. Source code on the screen is color-coded to match the color of the tools and corresponding tasks shown on the right side of the screen. A player can fail a level if he or she expends more tools than necessary, uses tools incorrectly, or does not complete the tasks within the time allotted. As the game progresses, the player is given tools and comprehension tasks used to learn control flow (*activator* tool), learn code behavior (*commenter* and *un-commenter* tools), learn variable purpose (*namer* tool), and learn data flow (*checker* tool).

3.2 Game Extension Points

At present, Robot ON! includes a demonstration of each tool in the context of game play. Using XML, Instructors can add additional levels to Robot ON! to incorporate program comprehension with source code customized to the level and the background of their students. For example, the source code used in a computer science course may be different than the source code used in a course for digital media, game development, or general science students.

When extending or customizing Robot ON!, we suggest instructors design levels using the following template:

- *Level 1* - The structure of a program and how to describe it with the Commenter tool.

- *Level 2* - An introduction to the different types of variables with the Namer tool.
- *Level 3* - Variable assignments, expressions, and calculations with the Checker tool.
- *Level 4* - Deciphering a program and solving problems using the Un-Commenter tool.
- *Level 5* - Learning loops and if statements with the Activator tool for control flow.

4. ROBOT ON! EVALUATION PLAN

To evaluate Robot ON!, we plan to perform a new user study addressing the following open research questions (RQs):

- **RQ1:** *Is the Robot ON! game playable by undergraduate students?*
- **RQ2:** *Do students enjoy playing the Robot ON! game?*
- **RQ3:** *Does Robot ON! give players sufficient skills to understand C++ (i.e., achieve learning outcomes with respect to programming comprehension)?*

Our evaluation will involve student subjects who do not have familiarity with computer programming. Participants will have mixed demographics, including gender and race, but participants will be between 18-20 years of age. We propose a two part user study to evaluate the efficacy of Robot ON! and to answer the research questions. The first half of our user study will be a 5 participant experiment in which we ensure that Robot ON! is playable by someone with no programming experience (**RQ1**). Participants will take part in a 1 hour study session where they will be observed playing the Robot ON! game for at least 30 minutes. Following the game play, the participants will then take part in a 20 minute interview where they will answer both structured and unstructured questions about their experience. We plan to use the interview to determine which levels of the game are too difficult, and to identify any problems with the game's controls and interface. The goal is for Robot ON! to be playable before its efficacy as a serious game is measured.

Next we will evaluate the game's ability to help students achieve learning outcomes (**RQ2**) as well as the user experience (**RQ3**). The second half of our study will involve a larger sample size than the first (15+ participants) and will take approximately 1 hour to complete. Participants in the second half of the study will take pre-tests and post-tests on their programming skills and knowledge as well as their emotional states. Our hypothesis is that players will feel better after playing the game and will have a higher aptitude for programming after playing as well. We hope to find that Robot ON! is an overall positive experience that includes some amount of learning.

5. SUMMARY & CONCLUSIONS

We have presented the Robot ON! game as a possible solution to the problem of teaching programming comprehension to novices. Our planned evaluation of Robot ON! will allow us to assess the game with respect to both learning outcomes and enjoyment. If Robot ON! is shown to be a successful game, we will use it as part of a multi-game package that helps novices with no programming experience to gain the knowledge and skills required to read and understand unfamiliar code, as well as to efficiently identify flaws and bugs in computer code. In the future, we hope to perform a longitudinal study that will include Robot ON! as an tool for students in a introductory programming course. We expect that improving programming comprehension will directly affect how students learn to program in combination with traditional lectures.

6. ACKNOWLEDGEMENTS

This research was partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC). We would like to give special thanks to Dr. Bill Kapralos for his assistance with this project.

7. REFERENCES

[1] *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.* ACM/IEEE-CS Joint Task Force on Computing Curricula, 2013.

[2] R. Bryce. Bug Wars: a competitive exercise to find bugs in code. *J. of Computing Sciences in Colleges*, 27:43–50, 2011.

[3] A. Chaffin et al. Experimental evaluation of teaching recursion in a video game. In *Proc. of the ACM SIGGRAPH Symp. on Video Games (Sandbox '09)*, pages 79–86, 2009.

[4] 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.

[5] S. Deterding et al. From game design elements to gamefulness: defining gamification. In *Proc. of 15th Int. Academic MindTrek Conf.: Envisioning Future Media Environments*, pages 9–15, Sept. 2011.

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

[7] L. Gugerty and G. Olson. Debugging by skilled and novice programmers. *ACM SIGCHI Bulletin*, 17(4):171–174, 1986.

[8] C. Kelleher, D. Cosgrove, and D. Culyba. Alice2: programming without syntax errors. In *Proc. of the 15th Annual Symp. on the User Interface Soft. and Tech.*, pages 3–4, 2002.

[9] M. A. Miljanovic. RoboBUG : A Game-Based Approach to Learning Debugging Techniques. MSc thesis, University of Ontario IT, Canada, 2015.

[10] V. Ramalingam and S. Wiedenbeck. An empirical study of novice program comprehension in the imperative and object-oriented styles. In *Proc. of 7th Work. on Empirical Studies of Programmers*, pages 124–139, 1997.

[11] 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.

[12] 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.