Ogata, H. et al. (Eds.) (2015). Proceedings of the 23rd International Conference on Computers in Education. China: Asia-Pacific Society for Computers in Education

A simulation-based learning environment for learning debugging

Yun-Jen HU [a], Po-Yao CHAO [a*]
[a]*Department of Information Communication, Yuan Ze University, Taiwan*
*poyaochao@saturn.yzu.edu.tw

Abstract: With the advancement of technology, more people start to learn how to code. Debugging skills are important to learn programming. We found that traditional software debugging tools are difficult for novices in fixing bugs. However, most of the programming learning support systems emphasize on language syntax structures rather than debugging strategies. In this paper, we design a simulation-based environment which provides novices with debugging scaffoldings. The scaffoldings are aimed to help novices detect and correct logic errors in programs by visualizing execution results. With the visualized debugging scaffoldings, we hope it can help novices improve debugging performance and debugging capabilities.

Keywords: programming, debugging strategies, simulation-based

1. Introduction

With the popularity of computer use and advance of information technology, programming hax become a necessary skill for career success. According to the U.S.A Bureau of Labor calculated, 2010 to 2020 the number of job openings of programmer is expected to growth 30% substantially, the other job only growth 14% (Lockard & Wolf, 2012). Therefore, leaning programming has been a trend. During programming most of time is spent in modifying and debugging existing codes in order to compile codes successfully (Jiang & Su, 2007). When a programming bug appears, a programmer need to locate a bug and fix it for the sake of a program successfully executing. There are often lots of bugs that need to be removed when compiling a program. When novices encounter programming errors, they usually spend lots of time to figure out where the problems are. However, it is difficult for them to confirm a problem whether a syntax error or logic error. Due to most of programming courses are aiming at syntax and algorithms, students need to gain their debugging skills based on practice and accumulated experiences. Moreover, the traditional compilers (e.g. Visual C#, Dev C) provide abstract error messages, which are appropriate for professionals rather than novices. On the other hand, despite there has many simulation-based learning programming systems (e.g. Scratch, Alice), these systems are more focus on training the logics of programming than on debugging skills. Given generating solutions to programming problems is a vital capability to become an effective programmer, the purpose of this paper is to design a simulation-based learning environment which provides novices with debugging scaffoldings. The scaffoldings are aimed to help novices detect and correct logic errors in programs by visualizing execution results. With the visualized debugging scaffoldings, we hope it can help novices improve debugging performance and debugging capabilities.

2. A simulation-based learning environment

Debugging involves the process of locating and correcting bugs (Ko & Myers, 2003). According to Araki, Furukawa, and Cheng (1991), the process of debugging is divided into four main parts: (a) initialize hypotheses based on error messages provided by a typical programing environment; (b) modify a hypothesis; (c) select a hypothesis and verify hypothesis; and (d) compile the program to check if bugs fixed, if a program executes unsuccessfully, select another hypothesis to fix bugs until bug fixed.

Based on the aforementioned debugging process, we proposed a simulation-based learning environment where a robot is given instructions in order to collect objects in the environment as efficient as possible. Because the system aims at facilitating debugging in simulated environments, the robot is initially given incomplete instructions. Learners are asked to modify the given instructions so that the robot can complete given tasks. As shown in in Figure 1(a), on the left side of the figure is an edit panel, learners can add, delete and modify instructions. If there exists at least one bug in the given instructions, a hint of bug shows up to notify learners that bugs exist in the given instructions. On the right side is an area presenting the main task. Learners observe the result of simulation regarding the instructions in the edit panel.

The simulated environment has an 8 by 8 grid placing objects of different attributes. Some of them are essential items needed to be collected. Some of them are obstacles. Objects in the environment include (1) instruction card: Each task has one or more instruction cards. To enable debugging, instructions in the card are incomplete to cause programming errors. Learners need to correct instructions to make the robot reach goals. (2) Items to be collected: Each task the robot must collect specific amounts of items, such as flowers or berries. (3) Stone: Stone obstructs the robot to move forward. It can be exploded by bombs. (4) Bomb: to blow up stones. The robot loses power when the robot encounter to a bomb.

After users explore the problems in farm, they can raise hypothesis and modify instructions accordingly. Available instructions to control the robot were divided into four categories: (1) Move: makes robot move one step forward/backward or turn right/left ； (2) Process: make Robot perform actions on items, such as pick the item, bomb the stone, clean the item ； (3) Calculate: Robot can count the number of item and memorize the result； (4) Detect: Robot can detect status of items in the environment. All of the instructions shown as Figure 1(b).

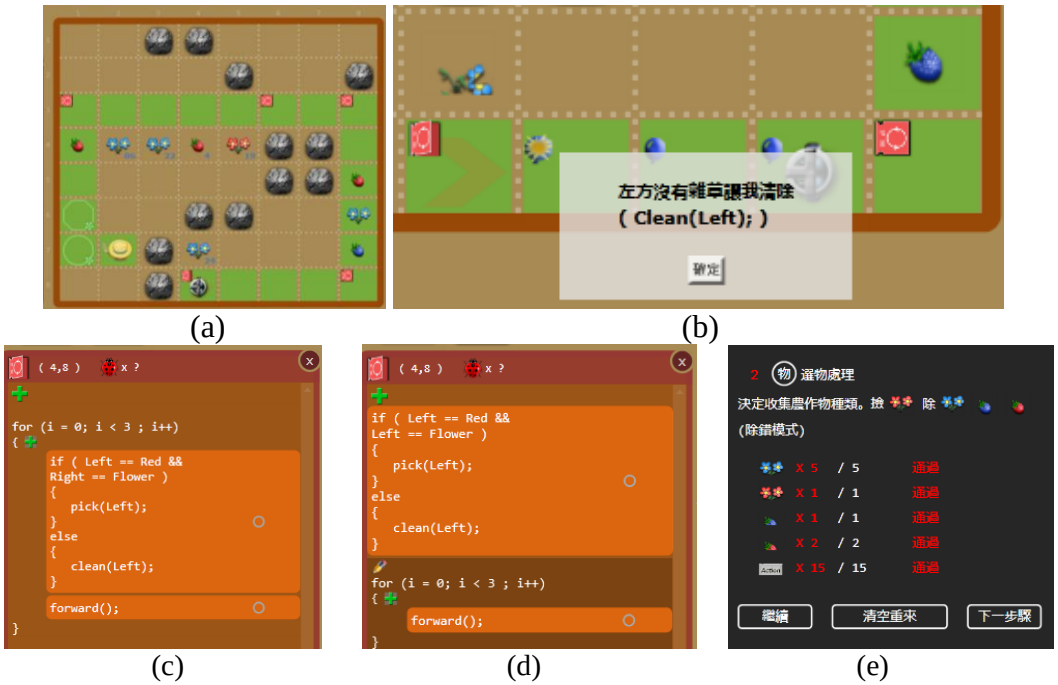
(a) (b)

Figure 1. The interface of robot-farm.

3. Problem-solving in the simulation environment

To describe the process in solving problems in the simulation environment, we take a problem as an example. A learner is to modify instructions in an instruction card so that the robot will successfully collect all the red flowers, clean out other item, and arrive to the farmer finally. The example task is shown as Figure2 (a).

At the beginning, a student may press execute button first and found robot clean the blue flower on its left and forward then several errors appear, it says “there is no item to clean” shown as Figure2 (b) . We wonder know what happened so we click the instruction card to see the instructions shown as Figure2 (c). Bug hint tells there has bug in this card. The student found the if-condition instruction is inside the loop, which enabled four times when robot execute this card but there was only one item in robot’s left side. He knew that the place of if-condition instruction is wrong so he deleted the if-condition instruction inside and create a new one in the front of the loop shown as Figure2 (d). Execute again, and this time he reached the goal shown as Figure2 (e).



(a) (b) (c) (d) (e)

Figure 2. Problem-solving of simulation environment.

4. Applications

To smoothly incorporate these errors and corresponding debugging skills in practical instruction, we also assign problems with different levels from easy to more complex. The learners will learn to debug increasing complex errors in the simulation-based environment. With proposed debugging scaffoldings, leaners are encouraged to discover problematic state, establish hypotheses, and eventually verifying their solutions. Since these debugging skills are core to the programming skills, we hope the learners can finally transfer the learned debugging skills to practical software development process.

Acknowledgements

This research was partially funded by the Ministry of Science and Technology under MOST103-2511-S-155-001-MY2

References

Araki, K., Furukawa, Z., & Cheng, J. (1991). A general framework for debugging. *Software, IEEE, 8*(3), 14-20.

Jiang, L., & Su, Z. (2007). *Context-aware statistical debugging: from bug predictors to faulty control flow paths.* Paper presented at the Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering.

Ko, A. J., & Myers, B. A. (2003). *Development and evaluation of a model of programming errors.* Paper presented at the Human Centric Computing Languages and Environments, 2003. Proceedings. 2003 IEEE Symposium on.

Lockard, C. B., & Wolf, M. (2012). Occupational employment projections to 2020. *Monthly Lab. Rev., 135*, 84.