
# Implementation and evaluation of a simulator and debugger for physical computing environments

Tom Neutens
UGent - Electronics and Information Systems - IDLab - imec
Ghent, Oost-Vlaanderen
Tom.Neutens@UGent.be

Juta Staes
UGent - Electronics and Information Systems - IDLab - imec
Ghent, Oost-Vlaanderen
Juta.Staes@UGent.be

Francis wyffels
UGent - Electronics and Information Systems - IDLab - imec
Ghent, Oost-Vlaanderen
Francis.wyffels@UGent.be

## ABSTRACT

In this poster abstract we present the design and evaluation of a simulation and debugging environment for a graphical programming interface. The environment is designed to be used within a physical computing context enabling users to detect errors faster and more efficiently. Finally, we evaluate its effect on learning progress and show a slight difference in the learning curve when learning programming with or without the debugger.

## CCS CONCEPTS

• **Computer and Information Science Education → Computer science education**;

## KEYWORDS

computer science education, physical computing, debugging, simulation, robotics, STEM education

**ACM Reference format:**
Tom Neutens, Juta Staes, and Francis wyffels. 2017. Implementation and evaluation of a simulator and debugger for physical computing environments. In *Proceedings of ACM Woodstock conference, Nijmegen, Netherlands, November 2017 (WiPSCE '17),* 2 pages.
https://doi.org/10.1145/3137065.3137089

## 1 INTRODUCTION

Graphical programming environments like Scratch and Google Blockly are fast becoming a key instrument in teaching children how to program. For novice programmers, these graphical environments have multiple advantages over textual programming environments. They are more intuitive [7, 9], leave users with a higher feeling of satisfaction [3], facilitate the understanding of more advanced computer science topics and strengthen the learner's motivation and self-efficacy [2]. However, the main disadvantage of these graphical tools is that they do not provide any mechanisms for finding mistakes. To fix their code, learners use techniques like: reading through their script, experimenting with their script, trying to rewrite the script or finding an example that works [4]. As any programmer knows, finding errors using these techniques can be difficult. Therefore, professional programming environments provide multiple debugging tools which expedite the error correction process. However, these tools are designed to be used by professional programmers and might be too complex for young beginner programmers.

Traditionally, debugging is learned as a complementary skill during the process of learning how to program. However, debugging is difficult to master, therefore, it is necessary to teach children how to debug their programs. Early research into the debugging process [5] suggests it has the following steps: (1) Comparing the program behaviour with the desired behaviour. (2) Identifying the difference between the observed and desired behaviour. (3) Locating the source of the error. (4) Correcting the error. Of these four steps, locating the error has been shown to be the most difficult part of the process [8]. Consequently, facilitating bug location might help novice programmers during their early learning process.

In this paper, the design and evaluation of a debugging and simulation environment for Dwenguino (an Arduino compatible microcontroller platform) is discussed. We first provide an overview of the design criteria for the debugger and simulator, afterwards we give an overview of the setting in which the simulator was tested and finally we evaluate if the simulator helped the students during the learning process.

## 2 DEBUGGER DESIGN

Currently, many sophisticated debugging environments are used by professional programmers. However, the complexity of these systems might be bewildering for novice programmers. Therefore, we chose to implement the debugging environment based on the following principles [6, 11]: (1) Favoring low floor over high ceiling, namely, keeping the barrier to entry low over the inclusion of sophisticated possibilities. (2) Selecting the right level of abstraction for the target learners, specifically, novice programmers in the two final years of primary school. (3) Visualize ongoing execution and computation result. Additionally, we want the learners to be more motivated when using our environment. Therefore, we implemented our debugger within a physical computing context [1]. Since on-hardware debugging is often challenging, we chose to provide a simulator alongside the debugger so children can easily check the behaviour of their program.

We kept the barrier to entry for the debugger low by limiting its functionality. Similar to [10], we provide learners with a simple *step* button which lets the users execute their program block by


*WiPSCE '17, November 2017, Nijmegen, Netherlands*

ACM ISBN 978-1-4503-5428-8/17/11.
https://doi.org/10.1145/3137065.3137089

block. However, we decided not to implement the breakpoint block since we did not want to mix programming code with debugger usage. Additionally, the principle of breakpoints might be difficult to understand by novice programmers. Therefore, we chose to implement a pause button and provide the ability to playback their program at slower speeds. This enables them to stop the program close to the area where they assume the bug is located.

By providing the learners with a simulator that implements two predefined scenario's, we hide the complexity of the Dwenguino microntroller board. This is preferable since an infinite amount of robots can be built using this board. Providing the learners with this set of robot abstractions facilitates the learning process. Both scenarios include a two-wheeled riding robot. In the first scenario, the robot has an infinite field of movement. In the second scenario the movement of the robot is restricted by four surrounding walls, the distance to these walls can be determined using a sonar sensor mounted on the robot. Additional to these scenarios, the interface includes a simulation of the microcontroller board's basic functionality. This includes: 9 LEDs, 5 I/O-microswitches, 1 reset button, an LCD and a buzzer.

For both scenarios, a visual representation of the robot is shown on screen enabling learners to see the behaviour of their program. Stepping through their program lets the children incrementally analyze the behaviour of their program facilitating the error location process. Additionally, the visual simulation speeds up the debugging process since it takes less time than uploading the code to the microcontroller and running it in the real world[1].

## 3 EXPERIMENTAL SETUP

To evaluate the debugger we performed an experiment in four classes of the last two years of primary school totalling 74 participants (n = 74). The classes were from two different primary schools, in each school, one class of the second to last year and one class of the last year were included in the experiments. None of the children had any programming experience. Consequently, before the experimental workshop, all children participated in a basic programming workshop. In this workshop, the learners used our graphical programming environment to perform several basic operations on the physical microcontroller board. The participants were separated into an experimental (n = 33) and control group (n = 41). The control group did not have access to the debugger and simulator while the experimental group did, both groups did have access to the physical hardware. During the first workshop, the control group executed their programs directly on the microcontoller while the experimental group was encouraged to use the debugger by telling them to test their program in the simulator before running it on the real hardware.

During the second workshop, both control and experimental group had to solve the same problems. To evaluate the effect of the debugger, children participating in the second workshop took a pre- and post-test. This test evaluated the learners' programming ability. Children are presented with a set of multiple choice questions that show a program and ask how the program behaves.

[1]Source code and executables are available at: https://github.com/dwengovzw/Blockly-for-Dwenguino

## 4 RESULTS AND DISCUSSION

Evaluating the pretest results shows that, on average, the experimental group scored lower than the control group. This could be a result of the extra information about the debugger those learners had to process. However, when comparing the average test scores on the post test the experimental group scores higher. This indicates that learners using the debugger have more difficulties at the beginning of the learning process but catch up with the control group after figuring out how to effectively use the debugger. Although promising, these results are not statistically significant with an $\alpha = 0.09$ when comparing the learning progress of both groups. However, they encourage further investigation into the topic of debugging environments for children.

## REFERENCES


[1] Mikko Apiola, Matti Lattu, and Tomi A Pasanen. 2010. Creativity and intrinsic motivation in computer science education: experimenting with robots. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*. ACM, 199–203.

[2] Michal Armoni, Orni Meerbaum-Salant, and Mordechai Ben-Ari. 2015. From scratch to "real" programming. *ACM Transactions on Computing Education (TOCE)* 14, 4 (2015), 25.

[3] Tracey Booth and Simone Stumpf. 2013. End-user experiences of visual and textual programming environments for Arduino. In *International Symposium on End User Development*. Springer, 25–39.

[4] Karen Brennan and Mitchel Resnick. 2012. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada*. 1–25.

[5] Sharon McCoy Carver and David Klahr. 1986. Assessing children's LOGO debugging skills with a formal model. *Journal of educational computing research* 2, 4 (1986), 487–525.

[6] Sayamindu Dasgupta, Shane M Clements, Abdulrahman Y Idlbi, Chris Willis-Ford, and Mitchel Resnick. 2015. Extending Scratch: New pathways into programming. In *Visual Languages and Human-Centric Computing (VL/HCC), 2015 IEEE Symposium on*. IEEE, 165–169.

[7] Hiromichi Endoh and Jiro Tanaka. 1998. Integrating data/program structure and their visual expressions in the visual programming system. In *Computer Human Interaction, 1998. Proceedings. 3rd Asia Pacific*. IEEE, 453–458.

[8] Sue Fitzgerald, Gary Lewandowski, Renee McCauley, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education* 18, 2 (2008), 93–116.

[9] IA Neag, DF Tyler, and WS Kurtz. 2001. Visual programming versus textual programming in automatic testing and diagnosis. In *AUTOTESTCON Proceedings, 2001. IEEE Systems Readiness Technology Conference*. IEEE, 658–671.

[10] William Robinson. 2016. From Scratch to Patch: Easing the Blocks-Text Transition. In *Proceedings of the 11th Workshop in Primary and Secondary Computing Education*. ACM, 96–99.

[11] Ian Utting, Stephen Cooper, Michael Kölling, John Maloney, and Mitchel Resnick. 2010. Alice, greenfoot, and scratch–a discussion. *ACM Transactions on Computing Education (TOCE)* 10, 4 (2010), 17.