S2 robot and a GUI - A possible approach to teach text-based programming to middle or high school students

Olivia Nche-Eyabi and Sekou L Remy
Clemson University
oncheey@clemson.edu, sremy@clemson.edu

S2 robot and a GUI - A possible approach to teach text-based programming to middle or high school students

1 Introduction

Some middle and high school students are already familiar with a certain level of programming using graphical user interfaces (GUI). However, in college or in the industry, programmers have to learn text-based programming. In this research, we propose and investigate a possible approach to introduce text-based programming to middle or high school students in order to give them a head start for college. GUIs alone are good introductory tools but will not suffice to teach text-based programming. Robotics programs are also quickly becoming commonplace in high schools due in part to the fact that learning with robots has the ability to provide rapid feedback and hands-on learning. As such, educational robots offer an exciting way [4, 11, 12] to address many challenging concepts which typically discourage or hinder students' interests in Science, Technology, Engineering, and Mathematics (STEM) [2, 7, 11]. We can harness the excitement, the interdisciplinary and pro-active nature of robots to teach text-based programming to middle or high school students. In this study, we combined a robot and a GUI to investigate a possible approach to teach text-based programming to high school students. We found that an approach, which used robotics programming as a channel, GUI as an introductory tool and reverse engineering/scaffolding to facilitate a transition to text-based programming, yielded the best results in the post-test for high school students.

2 Reverse Engineering/Scaffolding

Reverse engineering facilitates the modeling of a system by taking apart the current model and examining it to understand its distinctive components [18]. This can be done in small modules to simplify the process. Whereas, scaffolding requires one to break up the learning into chunks and provide a learning structure, for each chunk [17]. In our research, we used reverse engineering to examine the program created on the GUI in order to understand its distinct modules so that we could reproduce them on the Java IDE. We did so by breaking apart the code in small modules and analyzing each module carefully in the light of both Java and the GUI. We then employed scaffolding to rebuild the code on a Java IDE, working with the small chunks at a time. For each program chunk, we wrote a simple program in Java that implemented the same concepts learnt from the GUI project. However, in the latter case, it was structured to accomplish a different task. For example, the GUI pizza project involved loops and control statements amongst others. During the reverse engineering phase, we re-examined these concepts independently and then explained how they could be implemented in Java.

The pizza project required the S2 to draw a circle as one of its modules. This was accomplished by programming the S2 to draw an arc and then placing that piece of code

in a loop whose counter was set to three. Thus, this segment of the program dealt specifically with a loop. We isolated this portion in our reverse engineering/scaffolding transition phase and analyzed the unique features presented here. We then discussed these features in the Java context and proceeded to write a program that would loop through a list of numbers and sort out the prime numbers in the list. So the concept of loops was transferred to Java from the GUI and implemented in a different scenario. This process was reversed when we transitioned from the Java IDE to the GUI. In essence, reverse engineering allowed us to examine the GUI program with the purpose of rebuilding a similar model in Java, while the process of scaffolding facilitated the creation of this new model a little chunk at a time.

3 Programming Challenges

Programming is a very important and inevitable component of computing and robotics because it creates the code that instructs the robot to execute a series of actions [10]. However, traditional text-based programming has several challenges [13], which can render the learning process an agonizing one for the novice programmer. The process of programming requires the programmer to have declarative and procedural knowledge of the syntax of a language. The programmer should also possess the ability to memorize, understand, solve problems, deal with abstraction and think logically [5, 13, 16]. Furthermore, a programmer should not only possess problem-solving skills and knowledge of the programming tools of a language but should also have effective strategies for designing and implementing the program [1, 5, 16]. With the traditional approach in programming education, the students are first taught the basics of a programming language and then guided towards effective strategies for the whole programming process. The main source of difficulty here is not necessarily the syntax or understanding of concepts, but rather basic program planning. There is a difference between programming knowledge and programming strategies. Understanding a programming concept is one thing and being able to use it appropriately in a program is another [1]. Typically, a student can possess the former but will still struggle to execute the later. This implies that students may know the syntax and semantics of individual statements, but might have trouble combining these features into useable programs. Even when they know how to solve the current problem, they have trouble translating it into an equivalent computer program [1].

Deek et al. developed a problem-solving approach, which handles this problem and helps students to learn both concept knowledge and strategies for using them at the same time. In their approach, language features were introduced to students only in the context of specific problems. This was done in little bits at a time so as not to overwhelm the students. They found that this method was able to improve the performance of the students in their course work and boosted their overall programming confidence [1, 9].

We apply this concept in formulating an approach that will address some of the difficulties associated with programming for the young learner such that the struggles do not become a deterrent to some of them who would otherwise be interested in computing. Our method facilitates the learning of programming concepts and their application within the context of a particular problem simultaneously. A successful approach could potentially encourage more students to pursue computer science in college. Furthermore, when they get to college, programming will no longer be a new idea but a repetition to

some extent, of concepts that they have already interacted with [16]. This will further increase the percentage of students who will be retained for graduation [13]. There is also a shifting trend from the visual programming approach to a text-based approach. Microsoft's preference for a code first approach in Entity Framework, an object-relational mapping tool, is evidence to this changing pattern [19].

4 The S2 Scribbler Robot

We used the Scribbler 2 (S2) robot shown in Figure 1 for this research. The S2 is a robot, which is capable of drawing different shapes, letters, numbers and pictures, as it drives. It has a pen port to enable this activity. The pen port can hold a Sharpie or any similar marker [14, 15]. The S2 can be programmed by changing the Propeller source code in the S2 BASIC-like Spin language to suit a desired purpose or by using the Graphical User Interface (S2 GUI) tile-based programming tools.

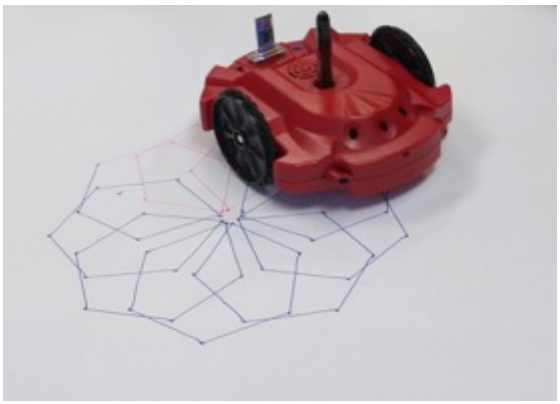

Figure 1. S2 robot drawing a shape

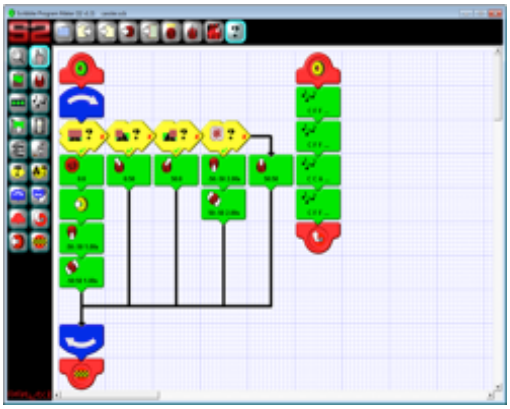

Figure 2. Scribbler 2 robot GUI

The S2 GUI in Figure 2 is a graphical drag-and-drop interface, which is a good platform to introduce students to programming [14, 21].

5 Previous Research

Some research has been done in this area and our intention is to add to the knowledge base, which has been created by previous works. Weintrop et al investigated how high School students perceived blocks-based programming. They conducted a 10-week experiment that allowed the students to experience and use different blocks-based environments. They used cognitive interviews and surveys to gauge the perceptions of the students. Their findings show that students generally perceived blocks-based programming to be easier than the text-based alternative. The natural language labels on the blocks, the shapes and colors of the blocks, the drag-and-drop composition mechanism, and the ease of browsing the blocks library were some of the reasons cited for this perceived ease of use. Students also identified drawbacks to the blocks-based programming approach including issues of authenticity, lack of expressive power, and challenges in authoring larger, more sophisticated programs. They also found that the differences high school students see between blocks-based and text-based programming span the visual interface, the types of programs that can be authored, as well a different programming practices that each representation supports [21].

The difference between their work and ours is that they sought to investigate if students thought blocks-based programming was easier than text-based programming and what they perceive as the differences between blocks-based and text-based programming. Although the authors were able to show that students perceive blocks-based programming to be easier than text-based programming and that it is a good introductory tool, no quantitative assessment was done to support this.

We take the study a step further to show that while blocks-based programming is a good introductory and easy to use tool, it does not exclusively prepare the students for text-based programming in college and industry. We perform a quantitative study, which shows that mastery of GUI based programming does not necessarily translate to an understanding of text-based programming.

Wang, X. et al [20] describe another approach, which involves using the Scratch GUI as an auxiliary to a textbook. The students are initially allowed to try out a project on the Scratch GUI and then the concepts involved in this project are later explained to them. In this approach, the authors propose that a teacher should start off by using a pre-existing example of a project on the Scratch GUI to teach the students how to design a project. After this process, the students should then be encouraged to design their own projects from their daily life experiences. Then the teacher should transfer to a textbook example with the traditional text-based code. He should then combine the textbook material with the scratch project to elucidate on the key programming concepts involved.

The authors carried out no experiment to substantiate their theory. The work was intended in part to provide a certain reference for similar research in teaching programming to high school students. This paper typically builds a teaching mode with Scratch for teaching programming to high school students.

In our case, we performed an experiment whose results show that Introducing programming with a GUI and then transitioning to text-based programming can improve students' performance.

6 Method

6.1 Target Sample

The target sample was middle and high school students. The experiment was conducting with students from the Anderson Christian School, which is a private school. The students who participated in this study were students from the 6^{th}, 8^{th} and 10^{th} grades. There were about 47 participants who had no prior experience in programming. They were almost evenly distributed across gender lines and their ages ranged from 10 – 14 years.

6.2 Pre- and Post-tests

At the beginning of the experiment, the students were given a pre-test, comprising of questions in Java. This test included questions to test their knowledge of concepts like variables, control statements, loops, and operators, to name a few. At the end of the learning period, the same quiz was administered as the post-test. The posttest was intended to capture the knowledge gained through the teaching process.

6.3 Experimental Procedure

The school authorities allowed us to come in during the Computer Science class time which was typically used to teach the students Microsoft Office. Their rational for this

was that our research would serve as an enrichment activity for the students. The CS class held at different time slots on different days and so we met each grade independently. For five weeks, we worked with these students, testing different approaches to teach text-based programming. We divided the students into four groups. For each grade, we divided the class into two groups (A and B), on the first day of the experiment. We randomly selected participants by letting them draw either letters A or B from a basket. For the 6^{th} and 8^{th} graders, those in group A constituted the GUI only group (group1). Those in group B became the participants in the text-based only group (group 2). In the high school class, group A participants were placed in the GUI before text-based group (group 3) while group B students were retained in the text-based before GUI treatment group (group 4). The high school students alone made up about half of the participants. So setting up the experiment this way allowed each of the experimental groups to have about the same number of students.

The students in group 1 (GUI only) were taught how to program using the S2 GUI only. At the beginning of the process, we taught the students how to use the S2 GUI and then proceeded to teach them how to program the robot to draw different shapes. In this phase, the students were taught how to draw simple shapes like squares, octagons, hexagons etc. Moving on from here, we showed them how to combine three or more of the same shapes to generate a different artifact like the one seen in Figure 1. In other cases, we paired different shapes to form a new one. An example of this was the house, which was created by programming the S2 to draw a combination of a triangle, a rectangle and a square in the shape of a house. We worked on several small projects over the five-week period and when we felt that the students were ready to do their own projects independently, we encouraged them to do so. We guided the students through a brainstorming phase to decide which project they would like to work on and formulated the steps that they would need to execute their projects. Then they went on to program the S2 to draw the shapes which they had conceived. After this exercise was completed, we engaged them in a final project. This project included several concepts of programming like algorithms, loops and control statements. In this final project, The S2 was programmed to draw a pizza, cut it up into 8 slices and then follow a path to deliver it to a home [14].

The second group of students (traditional text-based only) was taught text-based programming in Java, using the traditional method only. In their case we also worked on projects, which encompassed the same concepts that the GUI only group projects covered except that these were done in Java. With this group, we started off by introducing the Java language to them and providing an explanation of a Java IDE. In this case we used the NetBeans platform. Our next step was to download a Java IDK/IDE bundle. Once we had everything set up, we gradually introduced key concepts to them like variables, loops, control statements etc. We proceeded to teach each concept in further depth and worked on practical examples, which incorporated these concepts. So throughout the duration of the experiment, we taught this group how to program, using only the traditional text-based method.

For the third group (GUI before text-based), we used the S2 GUI as an introductory tool to teach the students programming just like we did with the first group. So we started off by teaching them how to program the S2 to draw simple shapes and then progressed to

more complicated ones. After this, we worked on the pizza final project. Then we transitioned them to text-based coding, working backwards from the GUI project to text-based coding in Java. At this point, we employed the process of reverse engineering/scaffolding to facilitate the transition from GUI to text-based coding in Java. We took the GUI code apart and re-examined it carefully in the context of Java so as to rebuild something similar to it. Then we employed scaffolding to rebuild a similar code in Java. The point was not necessarily to reproduce the exact GUI program but to reproduce something similar, which explained the same concepts that were presented formally in the GUI. Prior to this though, we had given them an overview of Java. We then delved into the Core Java concepts, which were introduced in the S2 GUI programming context. We showed them how to write a code in Java, which incorporated the same ideas as those in the GUI projects. Some of these concepts included variables, control statements and loops. While we did this, we continued to make references to how these concepts applied to the GUI example that they had already seen before.

The fourth group (text-based before GUI) started off by learning text-based coding in Java directly, using the traditional method like the second group did. In this case, we also gave them an overview of the Java language, and then introduced some key concepts. Then we proceeded to teach them how to write simple programs. For the first half of the five weeks of the experiment, this group was administered the same treatment as group 2. During the second half, we transitioned them to the S2 GUI using our reverse engineering/scaffolding method.

Group	**Pre-Test Ave. Score %**	**Post-Test Ave. Score %**
GUI only	40.4	46.9
Text-based only	45.4	51.4
GUI first before Text-based	66.4	84.9
Text-based first before GUI	66.4	69.0

Table 1: Results from pre- and post-test

In the GUI phase, they received the same treatment as group 1. The difference between groups 3 and 4 was that the students in group 4 moved from the Java IDE to the S2 GUI, a reverse of what happened in group 3.

7 Results

As the figures in Table 1 show, we found that group 1 students, who learnt programming, using the GUI only performed very poorly in the post-test. Those who were taught how to program in Java, using only the traditional approach (group 2), performed a little bit better than the “GUI only” group. We also found that the students in group three who were taught programming with the GUI as an introductory tool before transitioning to the traditional text-based method, had a better performance in the post-test than group 4 students, who were taught text-based programming using the traditional method first

before transitioning to the GUI.
M=84.9, 95% CI [79.0, 90.7]

8 Discussion

Our quantitative study supports the idea that blocks based programming has limitations. The suggestion that it is inauthentic compared to conventional text-based programming is an indicator that it is insufficient in itself to impute text-based knowledge to students [23]. In our GUI only group, participants spent the entire five-week period working on several projects on the S2 GUI exclusively. At the end of this period, we see that their performance in a text-based test was quite low. This affirms the notion that an understanding GUI programming does not necessarily translate to an understanding of text-based programming. So while GUI based programming serves well as an introductory tool, it does not in itself prepare the students for text-based programming.

Our approach was to introduce the students to programming using the S2 GUI and then transition them to text-based programing in the context of specific problems. The problem in the GUI case was to program the S2 to draw different shapes. In Java, we did not program a robot to draw shapes (in this research), although this could be a valid option. On the contrary we applied the same concepts learnt from the GUI projects to Java programs, to accomplish different tasks and solve different problems. Some examples included writing a program in Java that would accept information for a class directory and print out any data that was requested. Another one was a simulation of the ATM machine.

It is worth noting that there is a grade difference between the participants in this study, which could account for the wide disparity in the pre-test results. Although middle and high school students participated in the study, the approach we suggest in this research was tested only with high school students in this first part of our research. The first two groups were 6^{th} and 8^{th} graders and the 3rd and 4^{th} groups were high school students. That is why the results of the last two groups are considerable higher in both tests. Maybe this could be an indication that age plays a role in how easily students are able to grasp programming concepts. All of these students had no prior programming experience.

9 Limitations and Future Works

The GUI before text-based method, which we propose in this research was tested amongst high school students in contrast to the text-based before GUI approach. We did not test this approach with middle school students. The logistical set up in the school did not permit us to have a blended class of both middle and high school students. It will be interesting to carry out this experiment in a different scenario where each group could have a mixed multitude of students from middle and high school. In the future, we could also test this method amongst middle school students exclusively to see how they perform. A further extension of this study would be to perform this entire experiment with only high school students. In that case we would test all four approaches on high school students and see how they perform in each treatment level. Right now we know that in two treatment levels (GUI before text-based and Text-based before GUI) they score high in the former.

10 Conclusion

We found that students who were taught programming using the S2 GUI as an introductory tool before transitioning to Java, using reverse engineering/scaffolding, performed better in their post-test than the students in the reverse approach. This leads us to conclude that the approach, which uses robotics programming as a channel, GUI as an introductory tool, and reverse engineering/scaffolding to facilitate a transition to text-based programming can improve students' performance in text-based programming. The programming experience both with the GUI and text-based required the students to apply problem solving techniques, algorithmic and computational skills. They had to find suitable solutions to the problems and establish logical algorithms for the projects. This programming experience therefore also served to hone these important skills in the students [14], helping to prepare them for college or industry. The process of testing and debugging the code in both scenarios also helped to foster problem solving and assessment skills. We hope that our research will encourage computer science training in general and text-based programming in particular in middle and high schools. This could possibly translate to a higher rate of enrollment and retention of computer science students in college.

11 Acknowledgments

We thank the Upper School Principal of Anderson Christian School, Mrs. Chris Maddox for allowing us to work with the students of this institution. We would also like to thank Mr. Harrison Bolton and the entire staff for their immense support throughout this period. Finally we thank the 6th, 8th and 10th grade students who were eager to learn how to program the S2 robot and made our research possible.

References

[1] Ala-Mutka, K., "Problems in Learning and Teaching Programming", Codewitz Needs Analysis: Institute of Software Systems, https://www.cs.tut.fi/~edge/literature_study.pdf

[2] Alvarez Caro I. (2011). VEX Robotics: STEM Program and Robotics Competition Expansion into Europe Published by Springer Berlin Heidelberg. Page 2

[3] Ayanian, N. et al. (2010). Sparking a lifelong interest in engineering through a summer academy in robotics. International Design Engineering Technical Conferences and Computers and Information in Engineering Conference ASME 2010

[4] Bell, t. et al. (2012). Computer science unplugged, robotics, and outreach activities. ACM technical symposium on Computer Science Education 2012

[5] Boyle, T. et al. (2003). Using Blended Learning to Improve Student Success Rates in Learning to Program. Journal of Educational Media. 2003

[6] Chen, M et al. (2010). Instructional Simulations for Teaching High School Computer Science Concepts: A Technology Acceptance Perspective Digital Game and Intelligent Toy Enhanced Learning (DIGITEL), 2010 IEEE

[7] Connaughton, R. and Modlin, M. 2009. A Modular and Extendable Robotics Platform for Education 2009 ASEE/IEEE Frontiers in Education Conference

[8] Culler, D et al (2014). Computing Visions for 2025. Computing Research Association; http://cra.org/uploads/documents/events/snowbird/2014slides/2025_Snowbird_CCC_session.pdf

[9] Davies, S. (1993). Models and theories of programming strategy. International Journal of Man-Machine Studies, 39, pp. 237-267

[10] Dutta S. and Mathur R. 1996. Computer Programming – A Building Block of Stem. ISEC 2011

[11] Eubanks, A. et al. (2011). A comparison of compact robotics platforms for model teaching. Journal of Computing Sciences in Colleges ACM 2011

[12] Henkel Z. et al. (2009) Exploring Computer Science through Autonomous Robotics. FIE 09 IEEE
[13] Jenkins, T. "On the difficulty of learning to program." In LTSN_ICS Conference. 27-29 August 2002. Proceedings of the 3rd Annual LTSN Information and Computer Science Conference.
[14] Nche-Eyabi, O. et al. (2016) Drawing with Robots: An Experience Report (Fundamental). ASEE Annual Conference & Exposition
[15] Parallax Inc. Scribbler 2 Robot. 2016 https://www.parallax.com/product/28136
[16] Piteira, M and Costa, C. (2013). Learning Computer Programming: Study of difficulties in learning programming. Proceedings of the 2013 International Conference on Information Systems and Design of Communication (ISDOC) 2013 ACM
[17] Quintana, C. et al. (2002). A Case Study to Distill Structural Scaffolding Guidelines for Scaffolded Software Environments. SIGCHI Conference on Human Factors in Computing Systems
[18] Stroulia, E. and Systa, T. (2002). Dynamic analysis for reverse engineering and program understanding. ACM SIGAPP Applied Computing Review
[19] Tim A. Entity Framework goes 'code first' as Microsoft pulls visual design tool. (2014) http://www.theregister.co.uk/2014/10/23/entity_framework_goes_codefirst_only_as_microsoft_shutters_yet_another_visual_modelling_tool/
[20] Wang X and Zhou Z. (2011) The research of situational teaching mode of programming in high school with Scratch. Information Technology and Artificial Intelligence Conference (ITAIC), 2011 IEEE Joint International
[21] Wientrop, D and Wilenski, U. (2015). To Block or not to Block, That is the Question: Students' Perceptions of Blocks-based Programming ACM IDC 2015.