Visual Debugging Technology with Pencil Code: Position Paper

Amanda Boss
Harvard College
aboss@college.harvard.edu

Cali Stenson
Wellesley College
cstenson@wellesley.edu

Jeremy Ruten
University of Saskatchewan
jeremy.ruten@gmail.com

ABSTRACT

Pencil Code, a web-based program, utilizes blocks to aid students in creating code. This paper presents work creating a debugging tool to further improve Pencil Code and create an environment where students can better learn to understand code by having a way to visually trace its operations.

1. INTRODUCTION

First-time programmers do not naturally understand how to debug programs, so we propose a tool which helps beginners trace and debug the execution of their code without requiring manual debugging with breaks and print statements. Often times, when students are debugging their programs for the first time, they have the most difficulty with tracing through the program. When Fitzgerald et al. conducted a study, they found that a majority (57%) of the students had the most difficulty with finding the problem with the code [1]. In the same study, structured interviews with the students showed that the most commonly mentioned strategy for debugging code was tracing followed by testing and isolating the problem [1]. Even when first-time programmers do fix one of the bugs in their programs, they often have difficulty applying those same debugging principles for future bugs [2]. This stems from an inability to uniformly isolate bugs and understand code flow [2]. Our tool automates the process of isolating problems in the code and tracing code to offer a dynamic learning experience for the user.

2. OVERVIEW

Our debugging tool is a slider element that depends on a tracing transpiler and works in tandem with code annotations. Figure 3 shows the interaction between the three components.

2.1 Pencil Code

Our debugging tool is integrated into Pencil Code [3], a block-based online programming tool. Pencil Code allows users to convert the blocks into text and enter an open sandbox where they can create any program that is possible with CoffeeScript, JavaScript, and HTML. The environment is ideal for learning and is geared towards first-time programmers of a wide range of ages.

2.2 Tracing Transpiler

To help students see how their code is connected to what is happening when it runs we trace each line as it is animated on the canvas and allow students to highlight a given line to show what the animation looked like when that line ran. Tracing is a subtle, but useful, visual that allows students to see how their program works.To get a complete trace of a student's program as it runs, we pass the student's program through a code instrumenter before running it. Each statement in the program is instrumented with two function calls, one before the statement and one after, that send the location of that statement, as well as values of variables and function calls that are being tracked, to a listener that collects these events into an array containing the full trace of the program.

For example, if the following program is instrumented,

```
var x = 2;
var y = x + x;
```

then the code instrumenter will output a program that looks something like this:

```
ide.trace({type: 'before', line: 1,
vars: [{name: 'x', value: x}]});
var x = 2;
ide.trace({type: 'after', line: 1,
vars: [{name: 'x', value: x}]});
ide.trace({type: 'before', line: 2,
vars: [{name: 'y', value: y}, {name: 'x', value: x}]});
var y = x + x;
ide.trace({type: 'after', line: 2,
vars: [{name: 'y', value: y}, {name: 'x', value: x}]});
```

The code instrumenter works by parsing the input program into an abstract syntax tree (AST), and then adding the instrumented code before and after each node in the AST that is recognized as a statement. The AST is also used to find all variables and function calls within a statement, so that they can be tracked. Variables are tracked by passing their name and value directly into each event. Return values of function calls are tracked by assigning each function call to a temporary variable, and passing the value of the temporary variable into the event. Once the program has been fully

instrumented in AST form, it is compiled and run in the Pencil Code environment. As trace events are raised by the instrumented program, we use the information in each event to visualize what the program is currently doing, and allow the student to go "back in time" to any event to see what state their program was in at that time.

2.3 Visualization of Events

One of the primary features for the visual debugger is an interactive slider element that allows users to have an in-depth understanding of code flow. When the code in the editor panel is compiled and there exists more than two lines of code, the slider element appears. The line number requirement is to ensure sufficient complexity that warrants a debugging slider element. Each "tick" on the slider element represents an event that was traced by the compiler as a result of the inputted code. As users drag and click through the slider they are able to see the corresponding line of code highlighted as well as see the canvas reflect what the turtle animations looked like at that line of code. As users are scrolling through the slider, they are also able to see the arrows and variable tracking that occurred at that point in the code. The goal of this feature is to not only understand the sequence of events that occurred in the users' code, but to allow the users to grasp the fundamentals of important programming concepts. These concepts include, but are not limited to, looping and recursion. As users update their code and re-run their program, the slider element is also updated to reflect their changes. The slider works by keeping track of the state of events as code is run and storing those events so that they can be replayed after the code has run. The events from the transpiler are given to the slider and the slider indexes the events as a new value is selected on the slider and shows a visual of which line was run and what was happening on the screen at that line.

2.4 Code Annotations

Another feature we developed are arrows that help students understand code flow. The code annotations that show arrows were implemented to help students understand how lines of code are being processed. Arrows are deployed by the debugger when it traces an event on a line that does not immediately follow the line before it, as seen in Figure 1.The arrows were created with the idea that students would better understand loops and function calls if an arrow was drawn to direct their attention to the jump in lines since one would normally expect lines to run in order when new to programming. Arrows point out the repetitive nature of loops as well as the way code inside of functions is run on a function call. The arrows are designed to flash on the screen as code runs and to show when the slider highlights a line that an arrow is drawn from. Arrows are assigned to pairs of events from the transpiler that are nonlinear in code lines. When the slider shows the arrows those events that have had an arrow assigned to it will call the arrow function and re-deploy the arrows the user saw during runtime. Variable tracking allows us to show the values of variables as they are updated and passed into functions. As seen in figure 2, we show the variables as a pop-up on the right-hand side of a line of code so that users can see what the value of their variable was at a given point in runtime. Every variable used in an expression is displayed for the user alongside that expression. The return values of function calls used in the expression are also displayed. These return values allow the user to

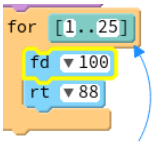

Figure 1: Arrows in Block Mode

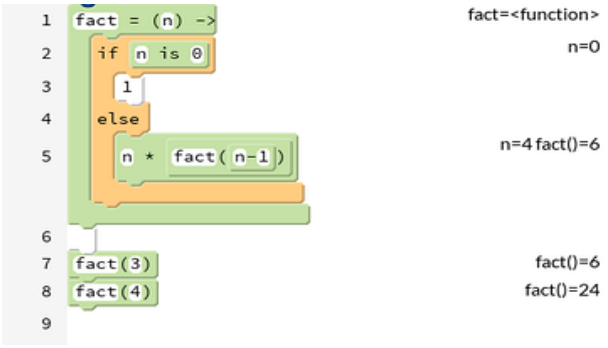

Figure 2: Variable tracking with a factorial function in Pencil Code

be aware of every value going into each expression, helping them track down where bad values might be coming from when they have a bug in their code. In the case of assignment statements, the old value of the variable is displayed before the statement executes, and the displayed value updates with the new value when the statement finishes (the debugger may have stepped into multiple functions in the meantime).

3. RELATED WORK

There are a variety of online programs to help users with the debugging process. We discuss those that influenced our work.

3.1 Python Tutor

One of the most prevalent programs is Python Tutor, a visualization web-embedded tool that allows users to step through the code at execution and view elements currently in the stack and heap [4]. This concept of tracing through the code at execution has proven to be successful in the debugging process as it mimics what a teacher would explain to students on a whiteboard. However, no version of Python Tutor for other languages such as JavaScript,which gives first-time programmers more flexibility to create web-based programs, currently exists. Furthermore, the audience of Python Tutor is geared towards students in an introductory computer science class [4]. Pencil Code is designed to be a self-learning tool for all ages; thus, we have created a more accessible debugger tool than that offered by Python Tutor.

3.2 Gidget

Another debugging tool, primarily geared towards a younger audience, is a web-based game called Gidget [5]. The game consists of a variety of levels centered around helping a character called Gidget. In order to pass each level, users must fix the bug in the code. Users are equipped with a variety of tools such as tracing through the program and going step by step. By gamifying the debugging experience as well as making it more interactive, Gidget is able to appeal to younger first-time programmers [6]. However, Gidget does

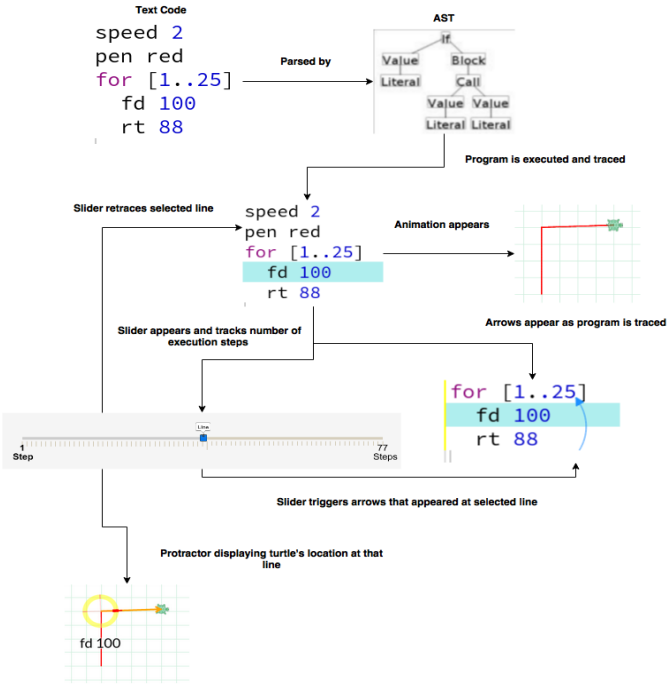

Figure 3: Debugging Flow in Pencil Code

not support an open sandbox environment for users like Pencil Code.

4. USER STUDY

To get a preliminary idea of how students would benefit from our visual debugger we did a study of 7 students. We asked each student to debug a program using the existing Pencil Code site without our features and then to debug another program using a test site with our features. We then asked the students questions about how they debugged code and how they felt about the arrows and slider features we had implemented. Most of the users (6/7) said that they thought the slider helped them understand the code, and some of the users (2/7) found the arrows helpful (the other users did not debug programs with loops or non-linear code flow). At the end of our study we asked the students what techniques they utilized to help them debug (Figure 4). Only one student cited using our tracing tools as a way to help them to debug code. This result could suggest that students do not find our tools helpful; however, we believe the students' practices were affected by using a site without the tools first and not feeling as though they need them the second time around. All of the students found it was important to read code carefully when you are debugging it, and our debugger is designed to emphasize your reading with a visual that further explains how your code works. With further investigation we aim to find that students use our tools to complement how they read code and train themselves to debug things automatically.

5. CONCLUSION

Pencil Code is an open sandbox and our debugger encourages students to fearlessly create programs and see how they work. Our debugger will allow students to improve their programs without the learning curve of understanding breakpoints and logging to help figure out programming mistakes.

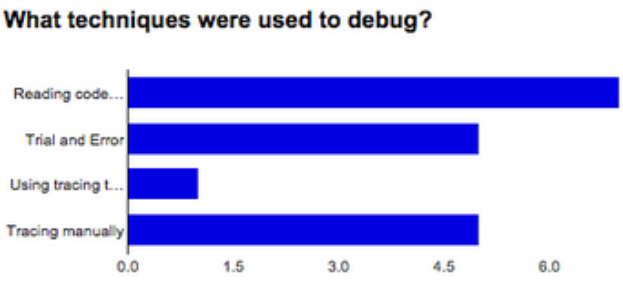

Figure 4: Results from Survey

With the support of our debugger, students can watch as their variables change, see which line of their code is running, and re-watch how their code works. These features create an environment that teaches you how to understand how your code behaves. Knowing how their code behaves will help students figure out what they should change if it does not work in the way they expect. With the debugger, users of Pencil Code can learn to write more complicated programs and become experienced programmers who program easily without the support of a blocks structure.

6. FUTURE WORK

To fully understand the effect our tools have on students we would like to get more user feedback on our debugger tools. This feedback will come from future studies and user responses to our features when they are in production on pencilcode.net. In addition to understanding how people use and appreciate current features, there are myriad of features that we hope to add to our debugging tool. One of the observations that we made during our user study was that participants were more likely to use buttons rather than a slider UI. Similar to Python Tutor, we plan on adding buttons that will perform the same task as the slider and will allow users to go forward and back one step in the program.

7. REFERENCES

[1] Fitzgerald, S., McCauley, R., Hanks, B., Murphy, L., Simon, B., and Zander, C. Debugging From the Student Perspective. *IEEE Transactions On Education, 53* (3). 390-396.

[2] Ahmadzadeh, M., Elliman, D., and Higgins, C. An Analysis of Patterns of Debugging Among Novice Computer Science Students. in *Innovation and technology in computer science education: Proceedings of the 10th annual SIGCSE conference,* (Libson, Portugal, 2005), 84-88.

[3] Bau, D., Bau, D.A., Dawson, M., and Pickens, S.C. Pencil code: block code for a text world. in *Interaction Design and Children: Proceedings of the 14th International Conference,* (Massachusetts, USA 2015), 445-448.

[4] Guo, Philip. Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education, March 2013. Retrieved July 22, 2015,from Python Tutor: http://pythontutor.com.

[5] Lee, M.J. Gidget: An online debugging game for learning and engagement in computing education. in

IEEE Symposium on Visual Languages and Human-Centric Computing, (Melbourne, Australia 2014), 193-194.

[6] Lee, M.J. and Ko, A.J. Personifying programming tool feedback improves novice programmers' learning in *Proceedings of the seventh international workshop on computing education research,* (Rhode Island, USA 2011), 109-116.