FootPrint

Motivation

Standard debuggers do not offer the option of stepping back and interactive visualization. Oftentimes programmers set the breakpoint too far or step through the debugger too fast and miss the step they wanted to examine and have to restart. For example, imagine a CS student who is trying to debug their program. They want to look at what is in their array at line 15, but they accidentally advance to line 16 when the array is discarded. With FootPrint they could simply examine the state of the array by looking at the program state from line 15, which FootPrint would store for reference. Moreover, developers could have a better understanding of their code and the data structure.

Current Approach

Currently, time travelling debuggers and other plugins (e.g data visualization plugins) exist that show the history of variables, but they have a multitude of issues. For one, many existing plugins are memory intensive, storing information such as stack frames, exceptions, method history, and logs. Chronon Time Travelling Debugger (n.d.) is one such plugin with these limitations. Furthermore, Chronon also saves execution paths and time stamps for the program, increasing the memory needs. Chronon is not unique in these features; many existing plugins store this information including JIVE (n.d.) and UndoDB (2019). JIVE is a data visualization plugin for Java for Eclipse that also allows the user to see a history of their variables. UndoDB is a time travelling debugger for C/C++ that also tracks memory and register states. All of these solutions are limited by their memory usage, storing superfluous information that FootPrint does not.

In addition, these plugins are bloated with features, such as storing unnecessary information, that make them difficult for developers to use and learn. For example, Chronon user Mik01aj wanted “to see and understand … what happens in ALL classes” but found it to be “so ridiculously slow”(2014). Providing this sort of functionality, while supposedly good for the user, in the end causes frustration. Further, in personal use of these plugins, many team members have found them to be unintuitive and unreasonably complicated for what we are trying to achieve with them. Finally, one critical issue with current solutions is that many are expensive and are targeted towards corporations rather than individual users. Chronon, for example, has a free open source version, but their Personal plan is a subscription costing $60 a year. UndoDB, on the other hand, has no option for individuals to use or purchase. JIVE and other free plugins exist, however, many still suffer from the issues listed above.

Our Approach

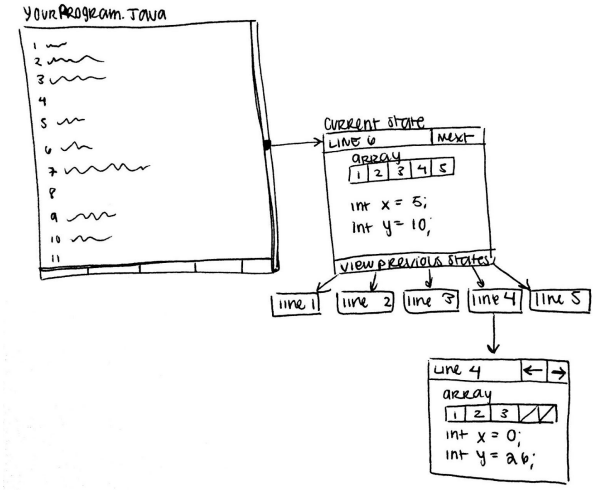
FootPrint would simply store the states of the variables and allow the user to view these states during the program execution. FootPrint uses the idea of “recording” the program as opposed to actually undoing the changes in order to catch irreproducible bugs and address concerns of irreversible execution such as network calls or file changes (“Reverse Debugging”). When the developer wants to “step back”, they would pause the program’s execution and display the previously stored information. This lets the developer see what just happened. Then, the developer can resume execution. With other solutions like Chronon, a developer would have to wait for the program to finish executing in order to replay the events. This would be time consuming if they only want to debug the beginning of the code. FootPrint modifies this idea and incorporates the recording into the live debugging session. Additionally, to address the problem of a memory, FootPrint only records variable states. FootPrint would be an Intellij plugin for Java developers.

While FootPrint does not allow the user to actually undo execution, it still allows developers to view the history of their variables which is often times what they are interested in. It also has deceased memory use as opposed to storing the whole program state. This approach also allows the user to do live debugging by stepping backwards or forwards, instead of simply recording the execution. Overall, FootPrint will serve as a user-friendly, lightweight, and simple way for Java developers to view the history of their variables.

Impact

We think Footprint could be useful for all Java programmers using Intellij. Everyone needs to debug, and it is always more convenient to have the option to view the history of your variables. However, for beginner programmers who may be overwhelmed by the many features of current solutions or their lack of focus, our solution would be especially useful for them since it is would be user-friendly and specific to one issue. Overall, this means less time and frustration spent on debugging. We can measure the performance of FootPrint by measuring and comparing memory use and user feedback. Our goal is to create a lightweight debugger that will be more efficient and use less memory that the current solutions, while still being a powerful solution.

Risks and Rewards



Although our idea seems simple enough, it may be difficult to implement. For example, we may run into roadblocks when trying to find efficient ways to store variable states. There is also the question of what exactly we are going to store, and also designing the user interface. In addition, there is the question of how to display this data. We would like to integrate data visualization into FootPrint and have it be more interactive like the built-in debugger, however this would be a challenge to implement. On the other hand, we can have a text-based display, which would be easier to implement but less user-friendly. For now, we will focus on the backend implementation (the actual storage of variable history) and decide on how to display this info later on to minimize risk.

Despite this, we think that our solution is feasible. Many other plugins and programs offer the feature of seeing variable history, so it is not an unreasonable goal. Our implementation is also quite simple; storing the history of variables in a cache and then returning this data to the user. Also, although we have not built an Intellij plugin before, there is a lot of support online on doing so. To minimize risks, we will implement basic features first and then add more as time permits.

Cost and Time

Since FootPrint will build on some of the functionalities from IntelliJ's debugger (like extracting variable states), we expect to be able to finish this project within 6 weeks. The first four weeks will be

spent on building and testing the backend of extracting and caching information while the remaining two weeks will be spent on building and testing a user interface.

Checks for Success

The midterm to measure FootPrint project is to finish the building process of FootPrint's backend. At this point, we will have fully implemented and tested the storage of debugging information. The final exam to check for success will be after the UI design of FootPrint and the launch of the project. We will do experiments with the users for feedbacks to further improve FootPrint.

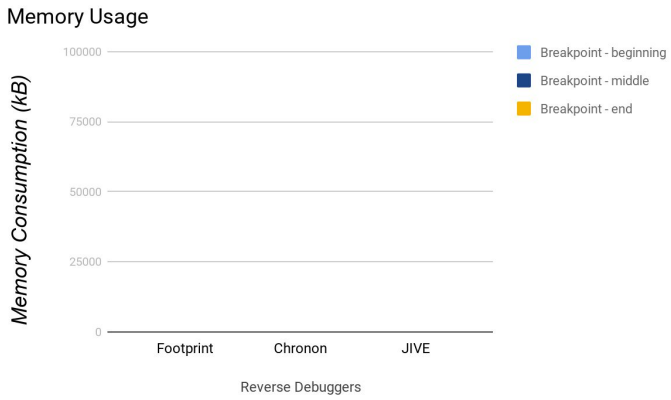
Scientific and practical interest

As Engblom (2012) pointed out, reverse debugging has been discussed since the very beginning of computer programming, however it was long ignored because of the challenges in its implementation. Although the growing advances in memory size, computing power, and debugging technology have been making reverse debugging semi-practical, the current reverse debuggers are mostly built for experienced developers. So the need for a user-friendly tool which could help beginners to understand, learn and implement their code is often ignored. Building a tool that could display and store the information from reverse debuggers for beginners is of practical interest, and the solution is ignored by current tools.

Experimental Methodology

Performance Experiment:

Run FootPrint, Chronon, and JIVE on an open source program in debugging mode. Set breakpoints in various places in the program (i.e. beginning, middle, end) and record the memory consumption of each reverse debugger plugins. The plugin that takes up the least memory in all three cases is the one with the best performance.



User-Friendliness Experiment:

We will gather a small sample of developers who have never been exposed to Chronon and JIVE and have them test Footprint, Choronon, and JIVE on the same program and see which one they think offers the most intuitive and understandable UI through a user survey. Through the results of the survey, we can see if FootPrint achieves its goal of being more user-friendly than current solutions on the market.

Week-by-week schedule

Week 5: Architecture and Implementation plan
We will outline the architecture plan for Footprint. We will decide on a UI and give a mockup. Furthermore, since we are extracting info from the built-in debugger, we will research the architecture of that. We will also decide on what data structures to use for our backend. We will also designate team assignments e.g. evaluation and implementation teams.

Week 6: User manual + begin implementation
User manual - ReadMe.MD (Sections: About, How to Download, How to Use)
Implementation - Finish and test module that will extract variable states from the debugger. Finish and test module that will store the variable states.

Week 7: Build and Test
We will complete and do testing of the UI. Further, we will create a user survey to determine usability and stability of FootPrint. At this point, FootPrint's basic features should be usable.

Week 8: Initial Results
We will compile results of user feedback survey and previous test results. Based on these, we will add to the test suite and fix any apparent bugs. We will run the outlined experiments to gauge the memory performance of FootPrint compared to other products.

Week 9: Draft Final Report
At this point FootPrint should be complete and fully tested. We may add more example programs to run FootPrint on, but otherwise coding should be complete.

Week 10: Finalize Project Report
Finish final commits on Git and finalize report.

References

About JIVE. (n.d.). Retrieved, January 27, 2019, from https://cse.buffalo.edu/jive/

Chronon Time Travelling Debugger. (n.d.). Retrieved January 27, 2019, from http://chrononsystems.com/products/chronon-time-travelling-debugger

Google Patents(2013), Google, Retrieved January 27, 2019, From https://patents.google.com/patent/US8997059B2/en

Engblom, J. (2012, September). A review of reverse debugging. In System, Software, SoC and Silicon Debug Conference (S4D), 2012 (pp. 1-6). IEEE.

Mik01aj. (2014, November 24). How can I track everything (all code) with Chronon? Retrieved from https://stackoverflow.com/questions/27103092/how-can-i-track-everything-all-code-with-chronon

UndoDB. (2019). Retrieved January 27, 2019, from https://undo.io/products/undodb/

Time Spent: 10 hours