Zan Balcom: zbalcom
Autumn Blackburn: kblack37
Jake Chiang: jchiang2
Alex Davis: ahd212
[Papers that reference verification games paper]

Motivation

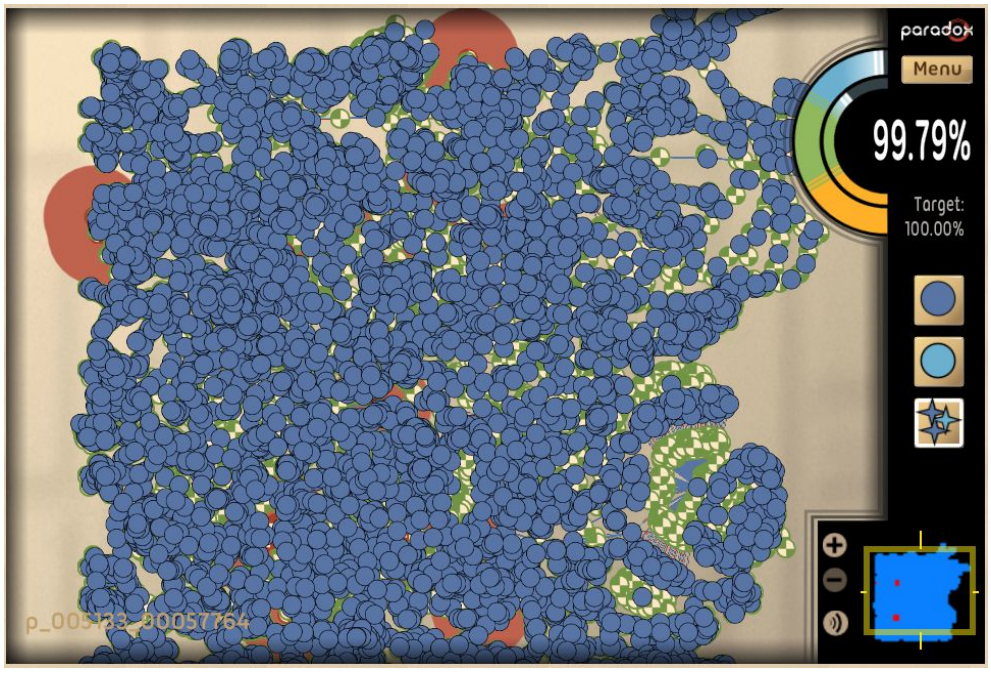
Computer programs have defects, which, as history and experience shows, can be very costly when they survive to be in the final product. For example, in 1996, the Ariane-5 rocket, due to an error casting 64-bit floating point values to 16-bit signed integers, exploded 40 seconds after take off [3]. Program verification is the process by which a program is proved to be free of some of these costly bugs; however, it is expensive to have trained programmers spend the time to evaluate every line of code looking for these defects. This is the way nearly all verification is handled today. There are existing tools that can perform certain amounts of verification checking, but none that can verify 100% of a program.

The main idea of Verification Games is to transform the intricate task of code verification into a game that can be completed with no special training, where a solution to a level provides valuable verification insight. This game can then be deployed to a large audience, crowdsourcing the laborious task of code verification to a population that doesn't need to be trained for it. Pipe Jam (fig. 1, left), is one such game developed by a team at the University of Washington, that uses a series of differently-sized pipes and balls to represent the variables and values, respectively, of a program. Game levels are made by reading a Java program and a specification in the form of a type system and then converting these into corresponding balls and pipes. The goal of the game is to manipulate the width of the pipes to produce a configuration that allows the balls to flow through the whole system. Such a solution equivalently is a proof that the program holds to the specification. One example of this type system in practice would be to look for null pointers in the program, with one width of pipe representing nullable assignments and another being non-null types. By solving the game, the player would be proving that no null value could be assigned to something that shouldn't be null. If the player cannot find a solution, then they can use a "buzzsaw" to make the balls small enough to flow through the pipes. This in turn signifies that a solution may not be possible, and

identifies possible problem areas that need to be evaluated by a trained programmer. In other words, the buzzsaw helps identify areas where a configuration of the type system could not be found that solves the puzzle, which indicates that the specification is not yet met.

Crowdsourcing the main tedious aspects of code verification, i.e. the reading and parsing of hundreds or thousands of lines of code, and identifying potential problem areas so that skilled engineers can examine them is the main motivation behind developing this sort of system. If this could be scaled up to be able to verify large programs, valuable developer time could be spent on other tasks. Currently, however, these verification games are only capable of representing small programs because the number of variables, values, and dependencies quickly becomes unwieldy in meaningful programs. This limits the power of the tool considerably since the number of pipes in larger programs would quickly become unmanageable for a player to deal with.

Other games approach the same sort of crowdsourced verifications with different methods. One is Paradox, which uses the same principles as PipeJam, although it has problems with complexity. (Fig. 2, left) shows a game of Paradox being played. The red areas identify constraints that have not been met by the program. It is the player's job to change the links between the small circles to meet the constraint. The most concerning aspect of this solution is that the barrier of entry would be too high due to its complexity to engage as wide of an audience as would be needed to verify large programs. The design of the game would need a deal of rework to be able to reach larger audiences needed for practical crowdsourcing.



Approach

Our approach to verification games is to focus on distributing the game to as large of an audience as possible. This will place PipeJam into the realm of large scale crowdsourcing: previous applications of crowdsourcing in computing have utilized many different amounts of users, from relatively small number of individuals like TopCoder to 75,000 users in the protein folding solver FoldIt [2].

To achieve these goals, we plan on updating the platform that PipeJam was built on. The original game was written in ActionScript3, which has fallen largely out of favor in the game development world as Adobe has announced that it will stop supporting its Flash Player browser plug-in[4]. We plan on updating the codebase to Haxe using the as3hx library so it can be easily configured to be distributed via iOS, Android, or HTML5, reaching a wider audience.

Another key point of our plan is to make the game engaging to players so they want to keep playing. This could mean abandoning the PipeJam game covered above altogether and changing to one of its other implementations, such as FlowJam. The two games are equally powerful verifiers, but one may be more fun to play or easier to understand to the average player. Another way of engaging the player would be to implement a competitive aspect into the game, such as a score and a score-tracking or speed-tracking system so players can compete with their friends, increasing their engagement. Yet another idea to aid player engagement is to add secondary objectives for them to collect, perhaps integrating with the aforementioned score system. Secondary objectives that are easy to obtain have been shown to increase player engagement levels [5]. We could implement some side system that integrates into (but does not intefere with) the verification functionality of the game, such as collectibles in the pipes. We want to retain the simple-to-grasp nature of PipeJam with the abstractions it has; we want to avoid reproducing the complexity of a game like Paradox.

To this end, after the codebase is fixed and we have the prototypes of the game up and running, we will begin playtesting both implementations of the verification game, PipeJam and FlowJam. We will have a decent amount of people (at least 20) - including those both within and without of our social circle (recruiting from, for example, the HUB on the UW campus) - playtest both games and observe their reactions to playing the game. Based on this data, we can pick an implementation to focus on developing and begin brainstorming and prototyping new additions to the game. This process will happen iteratively until the game is in a state ready to release, if only on smaller programs.

Challenges and Risks

Our most immediate concern involves updating the existing game to a playable state that is open for future development. With Flash support being discontinued and its already limited uses being dropped from modern browsers, adapting the codebase to a more accessible system is both a top priority and one that will require an indeterminate amount of work that will only become clear once the porting is underway. This process will also require becoming familiar with both the ActionScript and Haxe languages, which many of us are new and as of yet unskilled with. The payoff, however, will be a port with stable support and cross platform accessibility. This in turn both makes the audiences necessary for tenable crowdsourcing more reachable and provides a more welcoming development environment for future contributors.

We believe the most serious long-term challenge will be scaling up Pipe Jam by finding a way to simplify large programs into more manageable chunks. Currently, large programs pose a problem when being translated into game levels, as they produce overly complicated puzzles that are still impenetrable to casual players. Solving this problem by modularizing large programs into smaller, player-friendly pieces will be the foremost challenge, as this must be

done in such a way that the solutions found by users for the small parts still satisfy the same specification when reassembled back into the full program. Without careful design, this approach runs the risk of the verification provided by Pipe Jam not actually being sound. Mitigating this will require in-depth knowledge of formal verification to prove our solution is still correct. Despite the challenges, successfully implementing this system will allow the project to reasonably be used on larger programs where crowdsourced verification would be desired the most.

Finally, as we implement these changes and open the game to more players, we will need to remain cognizant of the problems involved in ensuring that the game is entertaining enough to draw a crowd of users. Our collective experience in regard to game design is limited, and will require consistent testing, feedback, and iteration to ensure our end product is enjoyable. Nonetheless, overcoming this final obstacle will see the utilization of crowdsourcing become a reality.

Timeline

N.B. Goal specifics in the second half are dependent on a further analysis of the codebase and cannot currently be determined.

Week 3

Resolve compiler errors for the Haxe port of PipeJam and FlowJam

Week 4

Replace external dependencies in PipeJam and FlowJam

Week 5

Have PipeJam and FlowJam running on sample JSON files

Week 6

Finish playtesting of PipeJam vs. FlowJam

Week 7 (Midterm) Begin Evaluation Phase

Iterating and playtesting of game to go forward with

Week 8

Continued iterating and playtesting

Week 9

Continued iterating and playtesting

Week 10

Continued iterating and playtesting

Week 11 (Final)

Have game version completed

References

1. **Werner Dietl Stephanie, Dietzel, Michael D. Ernst, Nathaniel Mote Brian Walker, Seth Cooper, Timothy Pavlik, and Zoran Popović. Verification Games: Making Verification Fun.**
https://homes.cs.washington.edu/~zoran/verigames-ftfjp2012.pdf, **2012**.

2. **Thomas D. LaToza and André van der Hoek. Crowdsourcing in Software Engineering: Models, Opportunities, and Challenges.** https://pdfs.semanticscholar.org/787c/e8e0973fd21051260385bb27d74f6f53c7f9.pdf, **2016**
3. **Gleick, James. "A Bug and a Crash."** *A Bug and a Crash by James Gleick*, New York Times, **1996,** www.around.com/ariane.html.
4. **Mackie, Kurt. "Adobe and Browser Makers Announce the End of Flash."** *Redmond Magazine* 25 Jul. 2017. Web. 4/5/2018 https://redmondmag.com/articles/2017/07/25/adobe-ending-flash-support.aspx
5. **Erik Andersen, Yun-En Liu, Richard Snider, Roy Szeto, Seth Cooper, and Zoran Popović. On the Harmfulness of Secondary Game Objectives.** http://grail.cs.washington.edu/projects/game-abtesting/fdg2011/fdg2011.pdf, 2011