Zan Balcom: zbalcom
Autumn Blackburn: kblack37
Jake Chiang: jchiang2
Alex Davis: ahd2112

Verigames: Crowdsourcing Code Verification Through Video Games

Motivation

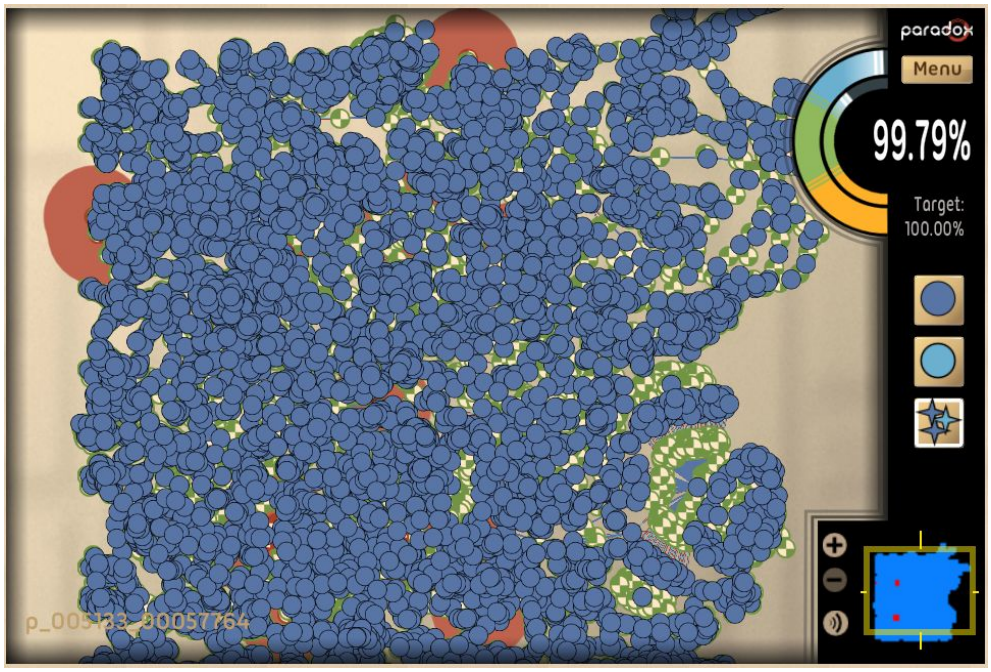
Computer programs have defects, which, as history and experience shows, can be very costly when they survive to be in the final product. For example, in 1996, the Ariane-5 rocket, due to an error casting 64-bit floating point values to 16-bit signed integers, exploded 40 seconds after take off [3]. Program 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 verifying a complete and meaningful program with these tools would take years even with a team of dedicated engineers. Due to these constraints, usually only critical sections of code are verified for correctness.

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[1]. 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 all the balls small enough to flow through the pipes. This in turn signifies that there could be a type error in the code that the pipes represent; a trained software engineer can then look at that part of the code and resolve any errors they find manually. 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.

With a crowdsourcing approach to code verification, the task of identifying potential problem areas in the code for trained engineers to review becomes much more manageable. The current issue with code verification is that it is a laborious project that has to be undertaken by trained engineers and thus verifying 100% of a large program correct would take far too many people-hours of work to consider as an option. With verification games, large swathes of the program can be verified correct automatically by players and so the developers of the program can then utilize their engineers more efficiently. 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, Paradox handles the complexity of large programs by leaving it up to the users to decide how much of the program they want to verify. (Fig. 2, left) shows a game of Paradox being played. The red areas identify constraints that are not currently met by the program. It is the player's job to change the links between the small circles to meet the constraint. This is where the way that Paradox deals with its complexity has a problem in our view, it does not really handle the complexity in any meaningful way. In the game you can use the map in the bottom right hand corner to zoom in on



an area to work on. This is in contrast with how PipeJam works, where the user is only given a small area to work on at a time; with those areas being interconnected to form the verification of the whole larger program. We believe this way of obfuscating the complexity of the program is important to keeping players motivated to play the game and in drawing in new players for the crowdsourcing of verification.

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 this end, we've developed a few different ideas to implement in sequence, starting with updating the current codebase to a new platform to improvements in the game to improve player retention.

The first goal is to update 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 [8] [9]. In addition to this, many of the external libraries the original codebase depended on have fallen out of use or into disrepair, so we will have to find or create replacements for those.

The second goal is to, after the codebase is fixed and we have the prototypes of the game up and running, 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. From this, we can pick the version of the game that has the greatest potential for player engagement and thus will most likely reach the largest audience.

The last goal is to, once we have know which version of the game we're going to move forward with, implement new features to the game to increase player engagement. Secondary objectives have been shown, when implemented a certain way, to increase player engagement [5]. To this end, we plan on implementing a system of secondary objectives in the form of collectible tokens that are obtained in the normal path of solving a level. This will be integrated into the scoring system, with picking up collectibles being another way for players to increase their score. With this scoring system in place, we can also implement a competitive aspect into the game, such as a score- or speed-tracking system so players can compete with their friends and possibly recruit new friends into playing[6]. But with these ideas, we also want to retain the simple-to-grasp nature of PipeJam with the abstractions it has and not make the game too unapproachable for newer players.

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

Continue resolving compiler errors & replacing external dependencies.

Week 5

Continue resolving compiler errors & replacing external dependencies.

Week 6

Have PipeJam/FlowJam games both running and proceed with comparison playtesting.

Week 7 (Midterm) Begin Evaluation Phase

Iterating and playtesting of game to go forward with.

Week 8

Determine which gameplay elements to expand on.

Week 9

Add selected gameplay features to improve player engagement and enjoyment.

Week 10

Playtest new game features, iterate on their feedback.

Week 11 (Final)

Have game version completed.

Challenges and Risks

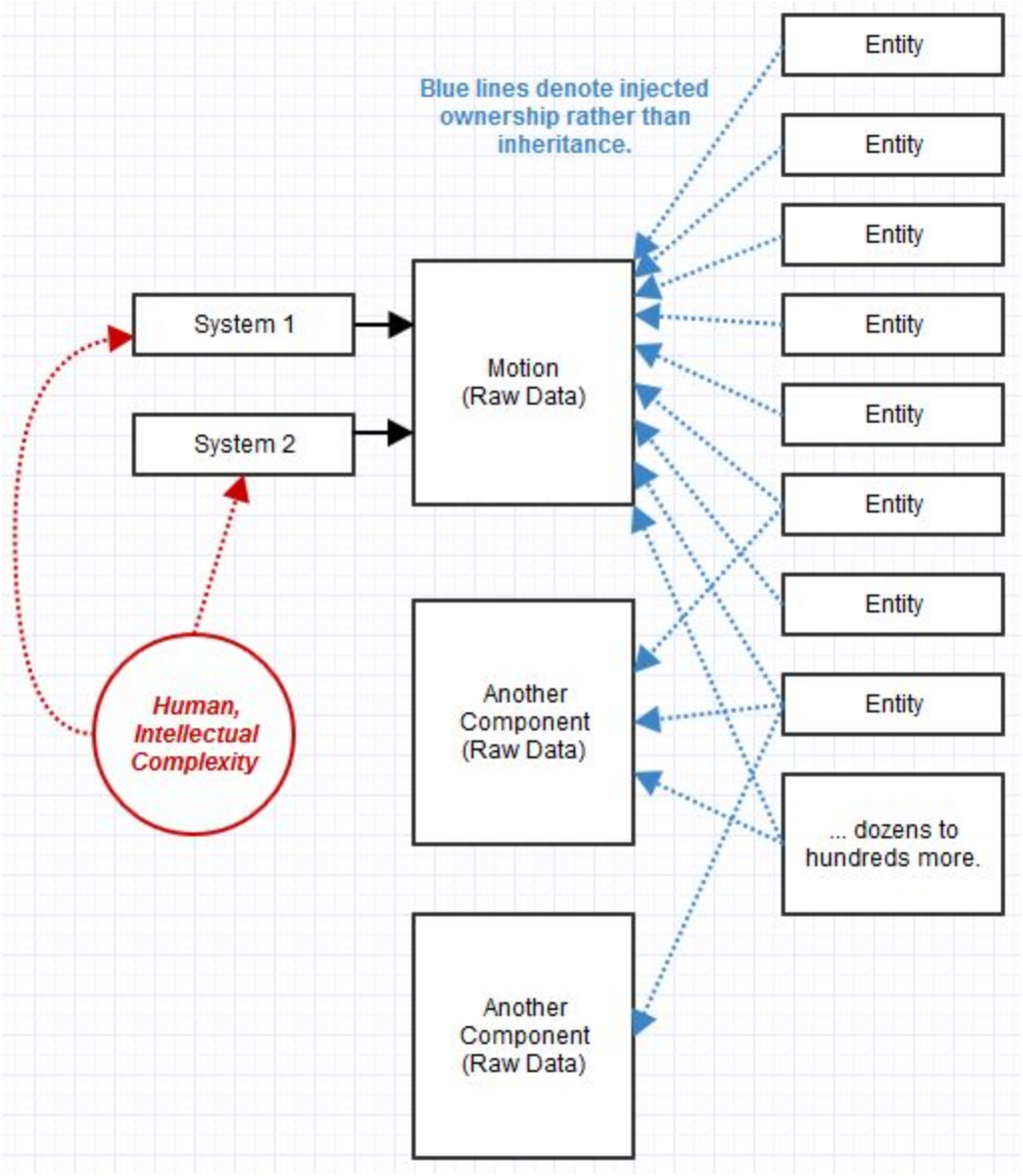
Our most immediate concern involves updating the existing game to a playable state that is open for future development. With Flash support being discontinued and its already limited uses being dropped from modern browsers, adapting the codebase to a more accessible system is both a top priority and one 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.

Another challenge is that the codebase we were given for the game is clearly in a prototype state - even after we finish porting and are able to get the game up and running, there will be significant issues with its extensibility and maintainability. There are significant architectural issues in the codebase that will have to be refactored; specifically, we plan to move the codebase to utilize an entity-component-system architectural pattern.

After the game is available and open to be played, 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. This will require us to evaluate which particular dynamics players respond best to. Once this is known, we can expand

on those game aspects. This could be, for example if players are interested in a competitive dynamic, a comparison among the players' friends of who is completing the most levels or using the least buzzsaws. There could also be a design space for a tertiary game loop with goals such as medals or some form of progression through the "world" of PipeJam. Nonetheless, overcoming this final obstacle will see the utilization of crowdsourcing become a reality.

Architecture



[7]

We wish to move the codebase away from its current, less flexible and maintainable architecture to an entity-component-system architecture. From its current state, the code will require a lot of refactoring. Since it was designed as a prototype and not a codebase to be maintained long-term, it has no definitive architecture that would allow for interesting feature growth.

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
6. **Hunicke, Robin, Marc LeBlanc, and Robert Zubek. "MDA: A formal approach to game design and game research."** *Proceedings of the AAAI Workshop on Challenges in Game AI*. Vol. 4. No. 1. AAAI Press San Jose, CA, **2004**.
7. **Owens, Thomas. "Is it reasonable to build applications (not games) using a component-entity-system architecture?"** *StackExchange* 4 Feb. 2018. Web. 4/12/2018 https://softwareengineering.stackexchange.com/questions/186696/is-it-reasonable-to-build-applications-not-games-using-a-component-entity-syst
8. **Haxe - The Cross-platform Toolkit.** (n.d.). Retrieved from https://haxe.org/
9. **HaxeFoundation**. HaxeFoundation/as3hx. Retrieved from https://github.com/HaxeFoundation/as3hx