Design Lessons From *Binary Fission*: A Crowd Sourced Game for Precondition Discovery

Kate Compton, Heather Logas, Joseph C. Osborn, Chandranil Chakrabortti, Kelsey Coffman, Daniel Fava, Dylan Lederle-Ensign, Zhongpeng Lin, Jo Mazeika, Afshin Mobramaein, Johnathan Pagnutti, Husacar Sanchez, Jim Whitehead, Brenda Laurel
University of California, Santa Cruz
1156 High Street
Santa Cruz, California
(831) 459-0111
kcompton@soe.ucsc.edu, hlogas@soe.ucsc.edu, jcosborn@soe.ucsc.edu, cchakrab@ucsc.edu, kercoffm@ucsc.edu, dfava@soe.ucsc.edu, dlederle@soe.ucsc.edu, linzhp@soe.ucsc.edu, jmazeika@ucsc.edu, mobramaein@soe.ucsc.edu, jpagnutt@ucsc.edu, hsanchez@soe.ucsc.edu, ejw@soe.ucsc.edu, blaurel@soe.ucsc.edu

John Murray
SRI International
333 Ravenswood Avenue
Menlo Park, Californa
(650) 859-5186
jxm@sri.com

ABSTRACT

This paper introduces the formal software verification game, *Binary Fission*. After outlining the problem space of formal software verification games, we give a brief overview of *Binary Fission*. We then go into detail about several important design goals we had in mind, such as affording rapid decision making, based on other software verification games and our own past experience. We detail how *Binary Fission* achieves these design goals, and then talk about several design lessons we learned. We discuss lessons learned, both in the parts of the game that performed well, and in the parts that did not quite work as intended.

KEYWORDS

games with a purpose, game design, formal verification

INTRODUCTION

Formal software verification is the task of proving that a software system has particular properties. Some examples of provable properties are that the software will never enter an

Proceedings of 1st International Joint Conference of DiGRA and FDG

unsafe state, or that the software is immune to a malicious attack. The general approach in software verification is to create a mathematical model of the software, then reason over the model with the aid of automated tools. If the model is accurate, than the statements provided by the reasoning tools are true of the software. Currently, creating these mathematical models is time consuming and requires a high degree of training in mathematics and computer science. The demand for formal software verification, therefore, easily out strips the ability of a small collection of specialists to verify code.

Several efforts to crowd source the formal verification problem have begun. This paper describes the design of such a crowd-sourcing effort, the puzzle game *Binary Fission*. We focus on key game design goals that we believe are important for crowd sourced formal verification games, such as state representation and affording rapid decision making. We also discuses the reception of *Binary Fission* and some design lessons we learned in creating the game, from real-time player chat to reward systems.

RELATED WORK

The design for *Binary Fission* comes from several different sources that were all intertwined into the final game. First and foremost, the game was designed as a citizen science game. It was one of the Phase 2 games from the CSFV (Crowd Sourced Formal Verification) program (Dean et al. 2015). Therefore, the game's primary end goal was to produce good formal verification results. With this in mind, we looked at the designs and results from the previous CSFV games, including *Flow Jam* (Dietl et al. 2012), *Ghost Map* (Watro et al. 2014)(Moffitt et al. n.d.) and *Xylem* (Logas et al. 2014). These games provided obvious examples of existing designs to serve as templates for *Binary Fission*.

Additionally, we drew inspiration from other citizen science games and other games with a purpose (or GwaPs). One classic example of these games is *FoldIt*, developed by the University of Washington (Khatib et al. 2011). This game crowd sourced the problem of protein folding, and its design presented a novel set of challenges (Cooper et al. 2010). *FoldIt*, and other games like it, have been highly praised for their successes in solving computationally intractable problems. However, as Tuite discusses, there are several very easy traps that a designer can fall into while designing a GwaP (Tuite 2014).

Some of these design traps are: unresponsive or slow feedback to the player, game mechanics that do not assist or even impede the science goals of the game, treating players as human processing units, and being opaque about the science goal of the game.

There has been a recent push towards using crowdsourcing as a tool for helping the field of software engineering as a whole. In a survey by Mao et al., the authors describe a wide variety of methods, platforms and goals that approaches in this domain as a whole (Mao et al. 2015). While not all of the techniques described are games, the use of GwaPs for software engineering work definitely falls under this umbrella. Tools designed to make verification easier (Schiller and Ernst 2012)(Akiki et al. 2013) have lowered some barriers to entry for real world use of software verification, and provide examples of possible human-friendly ways to present formal verification problems to users. Additionally, there have been other systems (Li et al. 2012) outside of the CSFV program that have attempted to use games as a tool for assisting verification.

Our biggest influence comes from *Xylem* itself, as not only is *Binary Fission* focused on discovering similar results, but many of the developers worked on both projects. While *Xylem*'s designers chose to put the mathematics behind the game on display (Logas et al. 2015), *Binary Fission* presents the players with a more abstracted view of its puzzles, drawing inspiration from games like Puzzle Pirates and marble maze toys for its design.

BINARY FISSION

In *Binary Fission*, players are presented with a set of blue and orange quarks which are originally mixed together inside the nucleus of an atom. The goal of the player is to isolate the quarks into as pure sets as possible. The players are given a set of filters which are capable of splitting the atom's nucleus (and the quarks contained in it) into two sections. Different filters create different splits, and it is the player's job is to decide which filters to apply, and in what order.

By recursively applying filters, players build a binary decision tree. The game starts with a root node representing the original nucleus (where all quarks are mixed together). Applying the first filter causes the root node to be split into two child nodes, each containing a certain chunk of the original quarks. Players continue to apply filters until nodes contain only blue or only orange quarks, or until a depth limit is reached.

Players earn points based on how effectively they can isolate the quarks from the mixture in the original nucleus. Each level has a point goal which players can achieve in order to go to the next level. Alternatively, players can click on a skip button to automatically go to the next level. Players start as recruits and get promoted as they accumulate more points over their lifetime in the game. The game has also a chat window which the players can use to interact with each other, as shown in figure 1.

The quarks and filters are more than a set of gameplay elements with interesting properties. Quarks represent the values of variables at a given point in the program. Blue quarks come from sets of variable values that come from normally terminating executions. Orange quarks come from sets of variable values that lead to a crash or assertion violation.

Filters, then, are logical predicates which are either satisfied or falsified by the quarks. For instance, a quark from a program that only featured two integer variables could be represented by a pair $< x, y >$ such as <3, 4> or $< 10, 0 >$. A filter designed to work with these quarks could represent statements such as $x < y$, $x17$ or $y + x = 0$.

By playing *Binary Fission*, players are building a decision tree that separates good and bad program states. Once a player produces a classification tree, it is easy to read out logical expressions that characterize good states and bad states, and those expressions constitute likely program invariants. However, in *Binary Fission*, the classification trees are typically partial; some leaf nodes only contain good states, some only contain bad states, while others contain a mixture. Therefore, the conjunction of predicates that link the root to a pure blue node describes a set of states that satisfy program assertions, and express a likely invariant. A single player solution can contain several such paths. By extension, the disjunction of paths to pure blue (good) nodes across all player solutions forms the consensus, likely invariant. This results in an expression of the form:

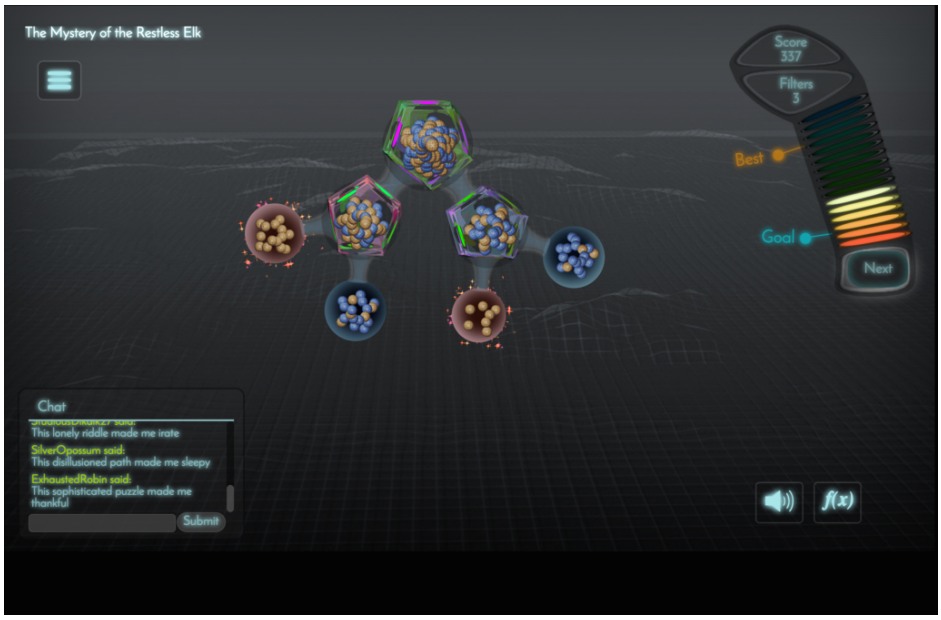

Figure 1: The general game play screen. Players create decision trees that sort the quarks into a blue pile and an orange pile

$$ConjuctionOfPureGoodPredicates_1 \vee ... \vee ConjuctionOfPureGoodPredicates_n$$

Because these expressions are induced from data, they are only likely, or candidate, invariants. Determining whether an expression is an actual program invariant requires logical proof. Candidate invariants produced by *Binary Fission* are tested by passing them through the CBMC model checker: an automated tool that calculates the logical effect of each program statement on the candidate invariant, and determines if the end result implies that the desired postconditions are true.

SOFTWARE VERIFICATION TASK

A key problem in software verification is to find program abstractions that are sufficiently precise enough to enable the proof of a desired property. However, these abstractions need to be sufficiently general enough to allow an automated tool to reason about the program. Automated techniques find such abstractions by identifying suitable program invariants—i.e., mathematical statements about the program that are true regardless of input. Unfortunately, the space of possible abstractions is essentially infinite and it is not currently possible to find useful predicates via automated methods. The human process for finding invariants relies on highly skilled people, schooled in formal methods, to reason from the purpose of programs towards possible predicates. However, this approach has an issue of scale: millions of programs could benefit from formal verification, while there are only a few thousand such experts worldwide. Automated methods rely on search and expectations to constrain the predicate invention process. White box techniques leverage knowledge about program content to propose candidate invariants, while black box methods search a space of templates (often boolean functions of linear inequalities) using comparatively little knowledge of program structure.

Recent work on classification techniques employ data to constrain predicate invention. Here, the objective is to induce a boolean expression over a base set of predicates that admits "good" program states (inputs that satisfy desired properties encoded as assertions) while excluding all "bad" states (input that violates such assertions on execution). These techniques output likely invariants that can be tested by static or dynamic analysis methods to determine if they are invariant conditions of the underlying program. The key issue in this approach is generalization; useful invariants are broad statements, while classification methods tend to overfit the data. Moreover, the data on good and bad program states required to achieve robust generalization is in short supply, as program sampling is itself a hard task.

Binary Fission addresses the subtask of precondition mining; it assumes a set of annotations that encode the desired program properties, and seeks predicates that imply the annotations hold under program execution. Players function as classification engines by collectively composing likely invariants without any awareness that they are performing program verification.

DESIGN GOALS

State Representation

One of the challenges with formal verification is the sheer size of software. Programs have many variables, often encompassing several data types. In addition, we want to find predicates for both individual and combinations of variables. Therefore, we want to be able to work with and reason over very large collections of variable-to-value mappings. For *Binary Fission* in particular, we also want to be able to represent valid and invalid variable-to-value mappings.

Not only are there a lot of variables that we want to reason about in as many combinations as possible, but each variable has a large range on its potential data type. We need some representation that allows for reasoning about strings and for the same reasoning to apply to dictionaries, floating point numbers, or structured data. Presenting this sort of information in terms of mathematical equations or operations is very verbose—players might be looking at pages of equations for a single problem.

Binary Fission attacks this problem with the quark and filter abstractions. Quarks are a very compact representation of data; we 'abstract away' all of the information a player does not need to sort the quark with a filter. Quarks are just a bit of data (that the player does not see) labeled as 'good' or 'bad'. This lets us represent all kinds of data types without their being any real mechanical or visual changes in the problem, as the data itself is unimportant—it is the labeling that players see. We can also represent many examples of 'good' and 'bad' for a single variable.

Filters are another point of abstraction. We only let players see how the collections of quarks split after applying a filter, rather than the actual property that the filter encodes. This, again, lets us represent a lot of different potential filters (as long as the hidden data in the quark matches the hidden property in the filter) for any problem in a visually compact way.

Rapid Decision Making in Large Decision Spaces

In addition to the challenges of representing a program state in a homogeneous way, we also want the representation to afford rapid decision making. Function predicates are equations

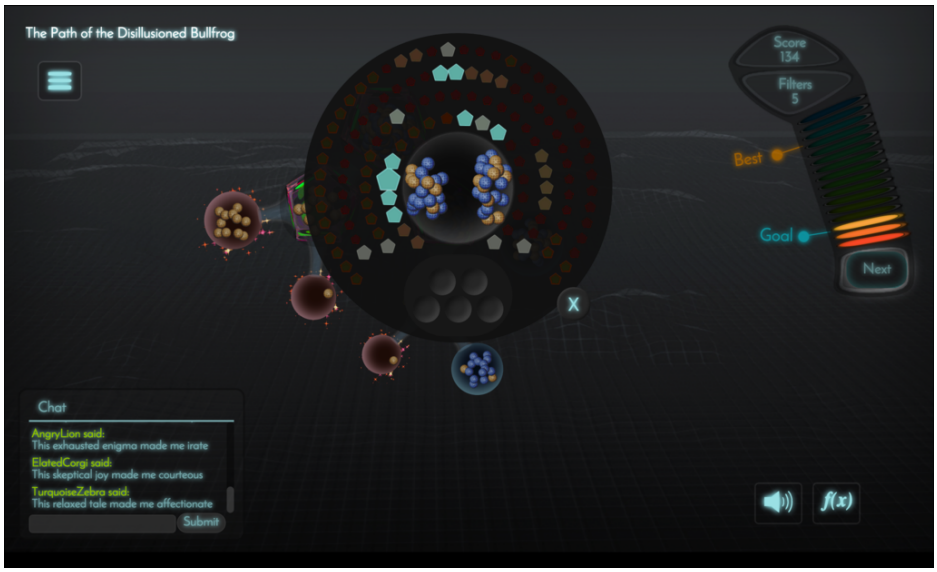

Figure 2: The Iris view of *Binary Fission*. In the center, players see how the current filter sorts the quarks. Spread around this visualization are the various filters, highlighted based on how well they sort the current set of quarks based on the current highlighting heuristic

over variables and program states, but interpreting and understanding these equations is a difficult task, even for specialists. To compound this problem, we want to be able to reason about many potential predicates. So, if understanding one predicate is a complex cognitive task, understanding a group of predicates may be, if not insurmountable, off-putting to a large portion of potential players. Partially, this is due to the difficulty of the task, but another contributing factor is math anxiety in our potential player base (Lee 2009). For a formal software verification game, we need some sort of abstraction that allows for reasoning about predicates that is easier to interpret than a raw mathematical equation.

Our solution to this was the Iris View, shown in figure 2. In this view, players can mouse over a potential filter and see how it will separate the quarks. Players are able to scan multiple filters with one mouse gesture. This rapid feedback loop helps players make a choice that separates the quarks the way they want quickly. In addition, we provide three highlighting strategies for filters: we can highlight the filters that separate the quarks such that there is a large group of only one color (purity), we can highlight the filters that separate the quarks such that they have roughly the same amount in each new cluster (balance), and we can highlight the filters that separate the quarks to maximize player score (utility).

Rapid Construction of Predicates

For some game mechanics, it is important to give players feedback as quickly as possible (e.g. movement in a endless runner game). Although this fast feedback paradigm is not always imperative of puzzle games (a genre in which both *Binary Fission* and *Xylem* are members), rapid user feedback on puzzle progress can often help players feel engaged and more comfortable in trying out novel new ideas or concepts (Compton and Mateas 2015).

For formal software verification, establishing preconditions over predicates is a task we want users to be able to complete very quickly, in order for them to feel safe in trying out novel new ideas and concepts to construct novel preconditions. There is also a component of sheer usefulness—a player that can come up with twice as many decision trees for some particular timestep is roughly twice as valuable to the primary end goal as a player who came up with less predicates, assuming the trees are of similar usefulness.

Binary Fission affords rapid problem progress in two ways. First, the mechanics for solving problems themselves are simple. The game focuses on point and click mechanics, rather than slower drag and drop mechanics. Second, the game makes it very easy to drop into a puzzle and move on to the next one. The tutorial is streamlined, and the player is never further than one screen away from a puzzle at any time.

Both design choices in the Iris View and screen navigation link back to the formal verification task. Players in *Binary Fission* are exploring a large space, and we want to encourage players to look in corners that automated methods ignore. Having players being able to rapidly experiment with new solutions helps fulfill this goal.

Citizen Science Audience

Some of *Binary Fission*'s design goals were inspired, in part, by past work performed on *Xylem*. One of the big evolutions was a shift in intended audience. The intended audience for *Xylem* was a general gaming "niche-casual" (Logas et al. 2014) audience. For *Binary Fission*, we targeted a new citizen science audience.

This new citizen science target audience came with a different set of game design considerations. Citizen scientists (roughly defined as users of web services such as Zooniverse [1]) are primarily interested in the science goals behind a GwaP, rather than a strong narrative framing or extensive world building. This casts citizen scientists as similar players to general puzzle game players (who often care more about puzzle mechanics than explanations as to why they are solving puzzles or other narrative framing), and as such, we designed *Binary Fission* with these sorts of players in mind.

However, we did not want to intimidate users by exposing all of the underlying mathematics that *Binary Fission* uses to create its puzzles—a lesson that was learned from *Xylem*. As stated earlier, the expressive quark abstraction completely masks all of the underlying data (what the quarks and filters actually represent under the hood), presenting the player with nothing more than a simple sorting puzzle.

In addition, we wanted to try and leverage the crowd by building a community around *Binary Fission*. Communities around games can be useful for developing a consistent user base, bootstrapping new players to work on relevant problems and providing assistance to each other on difficult problems. Several proposed systems for community involvement were discussed and designed, such as sharing problems and compound predicates across players for real time collaboration. We also considered motivating players through team competitions. However, due to time constraints on the project, we ended up implementing a simple, realtime chat feature.

1. http://www.zooniverse.com

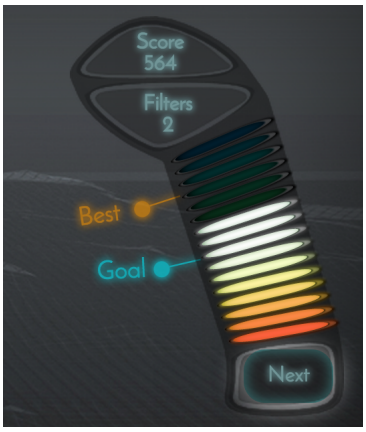

Figure 3: The score widget of *Binary Fission*. The bar on the right fills up as a player's score increases. The teal Goal marker shows on the bar a baseline number of points a player must achieve to 'score' the problem. The orange Best marker on the bar shows what the current overall high score for that problem is. The Next button is not shown until the player's current score is greater than or equal to the goal score

Reward Systems

It is common for puzzle games to reward players in points, and then encourage more play by beating a high score for a particular puzzle, or accumulating the most points total in the game currently (e.g, leaderboards). Points also serve as a handy progress marker for the player, so they can see how close they are to finding a solution. However, a consistent design difficulty with formal verification games is that progress on a problem is hard to chart. Tied to the 'charting forward progress problem' is a difficulty in gauging how hard a problem is. Common sense metrics, like 'problems that require more steps to solve are harder' no longer fit, as it is difficult to figure out what a forward step looks like. This makes assigning how many points a problem is worth is also hard. The point assignment difficulty makes presenting the player with a difficulty curve that makes sense (i.e. players should start with easier problems and progress to harder ones) a challenge.

Our solution to this three-pronged (hard to define forward progress, hard to define a metric for problem difficulty, and hard to define a difficulty curve) problem was heavily inspired by the game *Phylo* (Kawrykow et al. 2012). The way scores are presented to the player is shown in figure 3. Players do not need to completely sort each problem to make a meaningful contribution. As such, a minimum useful score is set—10% of the theoretical total possible points for the problem. When players exceed this threshold on the puzzle, they may move on to the next puzzle. We also present the best score received so far on a particular puzzle, so that players can try to be the current best solver, and friendly competition can help inspire better and better solutions.

$$N \times \sum_{i \in \text{leaf nodes}} \left(purity_i^A \times size_i^B \right) \quad (1)$$

Scoring in *Binary Fission* is based around two primary metrics: purity and size (1). Here, *purity* is the maximum of the percentage of good states and the percentage of bad states in a node, *size* is the total number of quarks contained within a node. This calculation provides the game's core tension between fewer broad-strokes filters (which typically maximize the size of the nodes at the expense of the purity) and more precise filters that only remove a small number of quarks (which create very pure, but very small, nodes).

The exponents *A* and *B* are arbitrary constants (set to 1.9 and 1.4, respectively), and *N* is a value that increases with the overall count of pure nodes, but decreases with the maximum depth of the tree. To get the theoretical maximum score for a problem, we assume the existence of a filter that will completely sort the quarks, and calculate the score from that theoretical filter.

Binary Fission imposes a 5-level depth limit on player generated classification trees, which bounds the complexity of the resulting classifiers. The depth limit and scoring mechanics influence players to produce as many pure nodes as early as possible. This guides players towards producing useful and general classification trees. Each classification tree produced through *Binary Fission* is typically partial: some leaf nodes only contain good states, some only contain bad states, while others contain a mixture. In addition, the solutions are idiosyncratic, as the players generally employ different subsets of filters during game play.

Finding Strong Invariants

The space of potential program invariants is vast. This is one of the reasons we want to crowd source invariant finding—we hope that people will look at interesting places in the invariant space that automated methods miss. However, people in general will not, by default, shift towards interesting or useful parts of the solution space. *Binary Fission* tries to lead people towards useful parts of the space by trying to consider invariants that sit on the boundary between true program behavior and impossible program behavior. Program invariants can be thought of as tight over-approximations of actual program behavior. A particular invariant may permit more properties than are actually true during program execution, so we want to find invariants that approximate actual program behavior as much as possible.

Trying to facilitate consideration in this space, we generate the bad quarks by mutating the code slightly and then generating data that fits the mutated code. Because the bad quarks are highly similar to the good quarks, players compose invariants that are tightly related to actual code behavior.

Aesthetic Goals

The visual design for *Binary Fission* was chosen to invoke a "sciencey" feel, specifically by invoking a design motif that is reminiscent of modern cyberpunk visuals, and merging them with a feeling of organic life. All of *Binary Fission*'s visual elements work with this

concept, from the light-on-dark color scheme to the use of curves and smooth growth when opening the Iris view. We chose this aesthetic to help demonstrate that the game has a simple visual metaphor for computer science, which relates to the actual scientific task. In addition, such an aesthetic design does not require any complex narrative framing or world building, as players can already visualize doing a computer science task with a system that invokes popular computer science symbology.

This aesthetic is also minimal and clean. Although some puzzle games have thrived with an emphasis on difficult to interpret user interfaces and complicated user interaction patterns, our desire to tap into a citizen science audience and focus on the formal verification task pushed us towards a simple, clean visual aesthetic.

In addition to the aesthetic itself, we focused our efforts on finding mechanics for play that were as simplistic as possible while still allowing the depth of experience that the game needed. To this end, the Iris View featured quarks scattering into their two piles as the player dragged their cursor over the various filters available to them in each problem. Selecting a filter was done with a simple mouse click, and opening the iris view only required a single click as well.

RECEPTION

At time of writing, *Binary Fission* was played by 972 players, who came up with 2963 solutions. It is interesting to note that 655 solutions (22%) came from 160 players (16.5%) playing on July 21st, which coincided with the publication of a BBC article on *Binary Fission* and the CHEKOFV project as a whole[2]. We use *Binary Fission* to analyze the implementation of a Traffic Collision Avoidance System (TCAS). Such systems alert pilots to the threats of a mid-air collision posed by nearby aircraft, and suggest avoidance maneuvers. It is critical for these systems to satisfy safety properties which can be represented as postconditions embedded into the code. We tackle a subtask of the verification process, which is, to find suitable function preconditions. Function preconditions are statements about program state that, if they hold on entry to the function, they then ensure that the postconditions are not violated.

The version of TCAS used in our evaluation has become a common subject to verification methods and test case generation systems since being incorporated into the Software-artifact Infrastructure Repository (Rothermel et al. 2006).

We created game levels based on seven out of the nine functions in TCAS. Logical predicates were generated though game play and, in the background, we analyzed these predicates on two main counts: whether they can be used as function preconditions, and if so, whether these preconditions are useful to the formal verification process.

Our analysis shows that players discovered several function preconditions that precluded the safety postconditions in TCAS from being violated. Figure 4 shows three preconditions found with *Binary Fission*. There were 3,108 clauses created by players from game play across all levels, 191 of them qualified as program preconditions.

2. See http://www.bbc.com/news/business-33519194 for the article

```
(not(Other_Tracked_Alt > Own_Tracked_Alt))
    and (Up_Separation < Positive_RA_Alt_Thresh[Alt_Layer_Value])

(Other_Tracked_Alt > Positive_RA_Alt_Thresh[Other_Capability])
    and (Down_Separation >= Up_Separation)
    and (not(Up_Separation <= Positive_RA_Alt_Thresh[Alt_Layer_Value]))
    and (Other_Tracked_Alt > Own_Tracked_Alt)

(not(Other_Capability == 2))
    and (not((Down_Separation == 800) or (Down_Separation == 600)
                                      or (Down_Separation == 500)))
    and (Down_Separation != Positive_RA_Alt_Thresh[Alt_Layer_Value])
    and (not(Other_Tracked_Alt > Own_Tracked_Alt))
    and (Up_Separation < Positive_RA_Alt_Thresh[Alt_Layer_Value])
```

Figure 4: Example crowd sourced preconditions found through *Binary Fission* game play.

As opposed to long, complicated logical predicates, the preconditions that *Binary Fission* players discovered were human readable; we believe that players had a big role in curbing the complexity of these predicates.

Because players are encouraged to build shallow trees and are forced to work within a maximum depth limit, they are also encouraged to create large pure splits, therefore needing less predicates to filter more of the good program states.

We assess the generality of the crowd sourced preconditions by measuring their coverage against test data that was not used in game level constructions (this is data not seen by players). The more of the "good" program state data explained by the found program preconditions, the more general are the logical statements and the more utility they offer. The best-case scenario is for the precondition to accept all good states.

For the seven functions in TCAS, we computed the percentage of "good" states in the test set that were accepted by the logical disjunction of the preconditions found by players. As shown in table 1, the results are promising. *Binary Fission* players were able to find function preconditions with noteworthy levels of generality, yet there is room for improvement.

LESSONS LEARNED

Chat

Unfortunately, players did not use the real time chat system very much in *Binary Fission*, and as a consequence, the game did not build much of a community around it. Looking at how chat was used, we realized that players would often send a message out in chat, but not see a reply. After this happened once, players were highly unlikely to use the chat feature again.

This is mostly a consequence of a sparse player base for *Binary Fission*. We still believe that community building through in-game communication can occur in sparse player environments. However, the system should be closer to a messaging service (where an immediate

Function	% accepted good states
ALIM	53.7%
alt_sep_test	21.2%
Inhibit_Biased_Climb	20.0%
Non_Crossing_Biased_Climb	20.3%
Non_Crossing_Biased_Descend	36.6%
Own_Above_Threat	0
Own_Below_Threat	0

Table 1: Testing the player-found preconditions' generality by comparing the number of good states accepted versus the total number of good states in the test set.

response isn't expected) than a chat channel. Another useful feature would have been to set up alerts, so that a developer could see when someone sent a message to chat, and respond accordingly. We also believe that our originally planned community building features (such as problem sharing) would have helped community building and social engagement.

Abstraction of formal verification

One of the strengths of *Binary Fission* is how well it abstracts away the messy details of formal software verification and presents it as a simple sorting problem. This allows for *Binary Fission* to represent a wide range of data types and predicates, and get input data from a wide variety of programs. This also keeps *Binary Fission* accessible to a wide audience.

However, this abstraction also turned players off to playing the game. A subset of players wanted to know more about the mathematical underpinnings behind the quarks, filters and sorting, and felt frustrated by the lack of information they could get about any particular problem. Finding the correct balance between expressibility and flexibility is a core problem in crowd sourced formal verification games. In addition, it became more difficult to leverage the innate human skill of spacial intuition at such a high level of abstraction.

Reward systems

While *Binary Fission* had a leveling system for the player (over time, a player earns points and achieves new ranks within the game) it fails to provide several other important utilities for player retention. First, while the overall top score for each level is shown the player, the player has no indication of what their personal top score is, much less whether or not they have attempted to solve a given puzzle before at all. Without this in place, playing puzzles can feel like an exercise in shouting into the void—a player can solve a puzzle as much as they want, but unless they are tracking their scores separately on their own, there is no meaningful indication of progress until they manage to create a solution that scores higher than the current top score.

This use of personal best could also inform the scoring function, as well as the decision trees that particular player generates. If the player is seeing a problem again, we may only let them submit solutions that score better than their original contributions. If the player's

current best is visualized out via *Binary Fission*'s score widget (the current version is shown in figure 3), we can motivate players to "close the gap" between their best scores and the current high score, as well as concretely showing them how much better their current score is compared to our baseline. This mix of competition and positive reinforcement may improve player retention.

FUTURE WORK

The main goal of our work on *Binary Fission* was to introduce crowd-sourcing as a promising approach to invariant discovery. From this perspective, the entire game is an exploration of a simple conjecture, i.e., that many non-expert individuals have the desire and ability to provide insight into verification tasks when they are presented in a suitable form. This conjecture appears to holds for *Binary Fission*. If it generalizes, related games will provide leverage on additional verification tasks, and crowd sourcing will offer an avenue for expanding the reach of verification technology.

The work on *Binary Fission* can be continued down three major paths going forward. The first, and most obvious, would be a direct application of the lessons learned to a new iteration of *Binary Fission*. This could take the form of adding in things such as a non-realtime chat feature or alerts and personal score tracking. Another unmentioned, but important part to improve would be the classification tree. We could allow it to use the pure bad nodes in addition to pure good nodes to improve the formal invariant check. One could also imagine a visualization scheme such that a future formal software verification game could take more advantage of human spacial intuition and show more of the mechanical underpinnings of the problem without resorting to long equations, program snippets or other forms of software representation that are hard to reason over.

The second major thrust would be to be an industrial strength version of this game, designed to work over actual code input by the users. This system would use a suite of automated techniques to provide predicates, good and bad program states. Additionally, it would allow for the crowd sourced candidate invariants to be tested as invariants for a client's source code. This would require a significant amount of effort to just extend *Binary Fission*'s current system to the limits of modern automated techniques. However, even then, there would still be a large set of problems that the system would be unable to address.

The third strategy would be to create a suite of games that allow for the players to work on several different avenues for addressing the problem of invariant search. For instance, in one game players could define various primitive statements that could be imported into *Binary Fission* and composed into more complex statements. From there, other games could be implemented to test the strength of the outputs of *Binary Fission*, and this data could be used to inform the problems from the first game, thus creating a closed loop that gradually increases the strength of the various invariants over time.

The design for *Binary Fission* is built around a solid core, and any of these improvements could see great future results.

CONCLUSION

The design of *Binary Fission* presented several challenges. The reasoning space that needed to be represented was vast, but players still needed to be able to move rapidly about it in

order to solve puzzles. We needed to design a reward system that managed to allow for incremental progress on a problem in a highly complex domain. Even with these constraints, we wanted player solutions to be useful for the task of formal software verification and aspired to keep mechanics simple and accessible, while fitting a cyberpunk aesthetic.

Several of our features ended up not working as well as we would have liked—we found the use of a real-time chat service did not foster the sense of community we had hoped, due to the sparsity of our player base. Although our software abstraction was powerful, several players wished for deeper view as to what was going on behind the hood. Although our scoring mechanic did manage to reward incremental progress, it failed to retain players due to a lack of context provided.

However, overall, *Binary Fission* did lead to promising results on the core task of formal software verification, with the crowd managing to find novel, legible and useful program invariants.

ACKNOWLEDGEMENTS

This material is based upon work supported by the United States Air Force Research Laboratory (AFRL) and the Defense Advanced Research Projects Agency under Contract No. FA8750-12-C-0225. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of AFRL or DARPA.

BIBLIOGRAPHY

Akiki, Pierre, Arosha Bandara, and Yijun Yu. 2013. "Crowdsourcing user interface adaptations for minimizing the bloat in enterprise applications." In *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems,* 121–126. ACM.

Compton, Kate, and Michael Mateas. 2015. "Casual Creators." In *Proceedings of the Sixth International Conference on Computational Creativity,* edited by Hannu Toivonen, Simon Colton, Michael Cook, and Dan Ventura, 228–235. Provo, Utah: Brigham Young University, June.

Cooper, Seth, Adrien Treuille, Janos Barbero, Andrew Leaver-Fay, Kathleen Tuite, Firas Khatib, Alex Cho Snyder, Michael Beenen, David Salesin, David Baker, et al. 2010. "The challenge of designing scientific discovery games." In *Proceedings of the Fifth international Conference on the Foundations of Digital Games,* 40–47. ACM.

Dean, Drew, Sean Gaurino, Leonard Eusebi, Andrew Keplinger, Tim Pavlik, Ronald Watro, Aaron Cammarata, et al. 2015. "Lessons Learned in Game Development for Crowdsourced Software Formal Verification." In *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15).* Washington, D.C.: USENIX Association, August. `https://www.usenix.org/conference/3gse15/summit-program/presentation/dean`.

Dietl, Werner, Stephanie Dietzel, Michael D Ernst, Nathaniel Mote, Brian Walker, Seth Cooper, Timothy Pavlik, and Zoran Popović. 2012. "Verification games: Making verification fun." In *Proceedings of the 14th Workshop on Formal Techniques for Java-like Programs,* 42–49. ACM.

Kawrykow, Alexander, Gary Roumanis, Alfred Kam, Daniel Kwak, Clarence Leung, Chu Wu, Eleyine Zarour, Luis Sarmenta, Mathieu Blanchette, Jérôme Waldispühl, et al. 2012. "Phylo: A citizen science approach for improving multiple sequence alignment." *PloS one* 7 (3): e31362.

Khatib, Firas, Seth Cooper, Michael D Tyka, Kefan Xu, Ilya Makedon, Zoran Popović, David Baker, and Foldit Players. 2011. "Algorithm discovery by protein folding game players." *Proceedings of the National Academy of Sciences* 108 (47): 18949–18953.

Lee, Jihyun. 2009. "Universals and specifics of math self-concept, math self-efficacy, and math anxiety across 41 PISA 2003 participating countries." *Learning and Individual Differences* 19 (3): 355–365.

Li, Wenchao, Sanjit A Seshia, and Somesh Jha. 2012. "CrowdMine: towards crowdsourced human-assisted verification." In *Proceedings of the 49th Annual Design Automation Conference,* 1254–1255. ACM.

Logas, Heather, Richard Vallejos, Joseph Osborn, Kate Compton, and Jim Whitehead. 2015. "Visualizing Loops and Data Structures in Xylem: The Code of Plants." In *Games and Software Engineering (GAS), 2015 IEEE/ACM 4th International Workshop on,* 50–56. IEEE.

Logas, Heather, Jim Whitehead, Michael Mateas, Richard Vallejos, Lauren Scott, Dan Shapiro, John Murray, Kate Compton, Joseph Osborn, Orlando Salvatore, et al. 2014. "Software Verification Games: Designing Xylem, The Code of Plants." In *Foundations of Digital Games (FDG 2014).*

Mao, Ke, Licia Capra, Mark Harman, and Yue Jia. 2015. "A Survey of the Use of Crowdsourcing in Software Engineering." *RN* 15:01.

Moffitt, Kerry, John Ostwald, Ron Watro, and Eric Church. n.d. "Making Hard Fun in Crowdsourced Model Checking."

Rothermel, Gregg, Sebastian Elbaum, Alex Kinneer, and Hyunsook Do. 2006. *Software-artifact infrastructure repository.*

Schiller, Todd W, and Michael D Ernst. 2012. "Reducing the barriers to writing verified specifications." *ACM SIGPLAN Notices* 47 (10): 95–112.

Tuite, Kathleen. 2014. "GWAPs: Games with a Problem." In *Proceedings of the 9th International Conference on the Foundations of Digital Games.*

Watro, Ronald, Kerry Moffitt, Talib Hussain, Daniel Wyschogrod, John Ostwald, Derrick Kong, Clint Bowers, Eric Church, Joshua Guttman, and Qinsi Wang. 2014. "Ghost Map: Proving Software Correctness using Games." *SECURWARE 2014:* 223.