UNIVERSITY OF CALIFORNIA, SAN DIEGO

Synthesizing Loop Invariants Through a Multiplayer Game

A Thesis submitted in partial satisfaction of the requirements
for the degree Master of Science

in

Computer Science

by

Rohit Jha

Committee in charge:

Professor Sorin Lerner, Chair
Professor William G. Griswold
Professor Ranjit Jhala

2017

The Thesis of Rohit Jha is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

Chair

University of California, San Diego

2017

DEDICATION

To my family.

EPIGRAPH

Life is more fun

if you play games.

—Roald Dahl

Talent wins games,

but teamwork and intelligence wins championships.

—Michael Jordan

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my advisor, Professor Sorin Lerner for giving me the opportunity to work with him, and for his support and advice throughout the course of my degree at the University of California, San Diego. This thesis would not have been possible without the help and suggestions I received from my committee members, Professors Bill Griswold and Ranjit Jhala.

I am extremely grateful to Dimo for his help with the single-player game, and discussions about the multiplayer version. I am also thankful to everyone who took out time to test and participate in playing the games and providing their feedback.

ABSTRACT OF THE THESIS

Synthesizing Loop Invariants Through a Multiplayer Game

by

Rohit Jha

Master of Science in Computer Science

University of California, San Diego, 2017

Professor Sorin Lerner, Chair

Human computation and crowdsourcing have been successfully proposed and applied to solve problems that are difficult to solve due to either the limitations of current computing technology or the existence of only a few experts. A common way to solve problems through human computation and crowdsourcing systems is to convert them into interactive games that target a wide audience of non-experts. Leveraging humans' innate ability to recognize patterns, a few systems have solved the problem of synthesizing loop invariants through single-player puzzle games.

This thesis proposes to apply human computation to synthesize loop invariants by having non-experts play a multiplayer game. We explain the challenges involved in

designing such games and explore the advantages multiplayer games offer over single-player games in terms of players' efficiency of solving problems and the amount of fun they have.

Chapter 1

Introduction

1.1 Human Computation

Luis von Ahn [54] defined human computation as the paradigm of utilizing humans to solve problems that computers cannot solve yet. By treating human brains as processors in a distributed system, where each performs a small unit of computation, complex problems can be solved. Researchers have subsequently adapted human computation to target problems that computers cannot efficiently solve, and may require both humans and machines.

One of the first human computation systems, reCAPTCHA's objective is to actually translate books [55][61], with a side-effect of being used as a security measure. Words that cannot be scanned by an optical character recognition (OCR) software are sent to the reCAPTCHA servers, where each such word is distorted and paired with the original word's image. These words are then served to users to be recognized until a consensus is reached on the correct answer.

Since an increasing part of the global population has access to the Internet, von Ahn proposed using online games in order to encourage participation in human

computation systems, and demonstrated how games could be used to correctly identify objects in images [60], annotate images with description [58], and collect common-sense knowledge from humans as training data for reasoning algorithms [59]. von Ahn emphasized that people play games to be entertained and not because they are interested in solving a small unit of a computational problem [57]. However, the success of studies such as FoldIt [9] and SETI@Home [1] suggests that players are also drawn to games where they can play the roles of citizen-scientists by contributing to complex and interesting problems.

Nevertheless, the greatest challenge to design games that aim to solve difficult problems is keeping the games enjoyable for a wide audience. In order to design such games, the following need to be achieved [29]:

- Identification of tasks which can and cannot be performed efficiently by machines
- Decomposition of complex tasks into relatively small units of computation
- Combining these small units into an entertaining game
- Providing people incentive to have a long-term interaction with the game
- Obtaining relevant and correct results from a vast collection of players' responses

1.2 Games With A Purpose

Games With A Purpose (GWAP) are human computation systems in the form of games wherein people perform tasks computers are unable to perform but as a side-effect of playing [57]. Therefore, for GWAP, the challenge of attracting and retaining players is even more critical since the focus is on entertainment. To achieve this, the game designers must abstract away the underlying domain-specific concepts and jargon, while still prodding the players towards finding correct solutions.

According to the 2017 report on Essential Facts About the Computer and Video Game Industry by the Entertainment Software Association (ESA) [51], nearly two-thirds of US households are home to at least one person who plays three or more hours of games per week. Of the most frequent gamers, 53% engage in online multiplayer games, averaging 11 hours of multiplayer gameplay every week. These statistics indicate that players enjoy competing and collaborating with other players, which can be leveraged by GWAP that offer a multiplayer mode.

1.3 Crowdsourcing

Jeff Howe coined the term crowdsourcing to define tasks that are usually performed by an expert, but which can be outsourced to a large group of (possibly non-expert) people as an open call [26]. The most prominent crowdsourcing platform today is the Amazon Mechanical Turk [2], where a demographically diverse set of participants can be recruited rapidly and inexpensively.

The difference between human computation and crowdsourcing is that in a crowdsourced system the contributors are typically compensated with monetary rewards, whereas players in human computation systems, such as GWAP, play for entertainment. However, crowdsourcing and human computation are not disjoint systems – tasks that can be performed by either a computer or an expert human lie at the intersection, and can be converted into entertaining games.

1.4 Thesis Statement

In formal program verification, loop invariants are typically expressed using predicate logic and are used to prove the correctness of loops and the algorithms that

employ loops. Traditionally, finding loop invariants is the responsibility of either the verification engineer, or the programmer writing the loop. However, it is a challenging task, mentally and economically, to manually find and annotate all the loop invariants of new or existing programs. As a result, this has led to the development of several automated tools and techniques for discovering loop invariants [16][17][21][27][28][46][47]. The other approach to find loop invariants is through human computation and crowdsourcing systems using games, such as CrowdMine [31] and Xylem [35], which convert the problem of finding loop invariants into single-player games.

My thesis is that a hybrid collaborative-competitive multiplayer online game is better than a single-player game for synthesizing loop invariants since it can:

1. generate loop invariants for problems which an individual may find difficult to solve alone

2. generate loop invariants with better throughput, and

3. be more engaging and entertaining

The notion behind the collaborative aspect is to help players find a higher number of loop invariants to quickly achieve the shared goal of solving a problem by building on top of each others' found invariants. The competitive aspect of the game is to motivate players to perform better than their peers by finding stronger and more number of loop invariants, especially since they can compare their progress against others during their games.

1.5 Thesis Contributions

Following are the contributions of this thesis:

1. Demonstrating the advantages of applying human computation in the form of multiplayer games to solve the problem of synthesizing loop invariants

2. Corroborating the benefits of hybrid collaborative-competitive multiplayer games over single-player games to solve difficult problems

3. Exploring the challenges faced while designing such multiplayer games, and some techniques to overcome these challenges

1.6 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 explains the challenges, architecture and game design of the multiplayer game. Chapter 3 covers the details of the experiment and results obtained. Chapter 4 briefly explains a few games that share the same objective of generating loop invariants, and compares them with our multiplayer game. Chapter 5 concludes the thesis with a few potential future works.

Chapter 2

A Multiplayer Game for Loop Invariant Synthesis

2.1 Challenges

In this section, we explore the challenges encountered while designing a multiplayer game for generating loop invariants as a GWAP. We provide examples of how we addressed those challenges with the details explained later in section 2.3.

2.1.1 Non-Experts as Players

As is the case with many human computation systems, the target users are assumed to have no prior experience knowledge of the domain [57]. In our multiplayer game, we abstract away from the players details about the source code, loops and loop invariants, while still conveying to them that the objective is to synthesize loop invariants. For example, one way in which we address this is to avoid the term loop invariant and instead use the term *expression*. Also, we do not expose the source code to any player, which makes our game suitable to find loop invariants in proprietary software.

2.1.2 Visual Interface

A key challenge in designing any game is to make the interface visually appealing to the players. In our game, we provide a minimal interface that is easy for players to navigate and to comprehend features at a glance. To ensure that players understand the game's interface, we walk the players through a short tutorial where the interface components are revealed and their roles explained, one at a time.

2.1.3 Feedback

To maintain players' engagement in a game, a good way is to provide feedback by awarding players points for finding correct solutions. In our multiplayer game we not only add points to players' score for finding new invariants, but we also award bonus points for stronger invariants. Players get feedback on their invariants' validity almost instantly and this ensures that the game's pace is not slowed.

2.1.4 Player Efficiency

A simple measure of players' efficiency is throughput, which is the number of correct answers they provide per unit time. We employ the use of a scoreboard in every game and a global leaderboard to provide a sense of competition to the players. Besides allowing players to keep track of their performance and their standing relative to that of other players, we expect this to motivate them to outperform their peers by finding more invariants faster. To persuade players to generate a variety of invariants, we provide players with power-ups (which change in their strength as the game progresses) if they do not repeat similar invariants.

2.1.5 Scalability

Our multiplayer game needs to be able to handle a large number of concurrent players without any adverse impact on latency. Besides verifying each player's inputs, which is computationally intensive, the implementation at the client and server-sides should support frequent or real-time synchronization of loop invariants found by all the players in every game. To handle this challenge, our architecture handles a large number of players by potentially scaling to multiple load-balanced server instances. The details are explained in section 2.2.

2.1.6 Quality of Loop Invariants

In our multiplayer game, we refrain from having a time limitation or ranking players based on their speed during the game as we want players to provide well-considered responses, with the expectation that this leads to better loop invariants. Additionally, we reward players with bonus points for finding invariants that are stronger than one or more previously found invariants, which are visible to all players in the game. A loop invariant p can be said to be stronger than another loop invariant q if $p \nLeftrightarrow q$ and $p \Rightarrow q$, i.e. if p is not equivalent to q then whenever p holds true, q also holds true. For example, if p is the invariant $y = 3$ and q is the invariant $y > 0$, then we can see that if y is equal to 3, then y is greater than 0, i.e. $y = 3 \Rightarrow y > 0$ and therefore p is a stronger loop invariant than q. If we do not check for equivalence, then players could enter a valid invariant $y > 0$ repeatedly and get bonus points since $y > 0 \Rightarrow y > 0$. However, since $y > 0 \Leftrightarrow y > 0$, players will be alerted that their provided invariant is equivalent to an invariant already found.

2.1.7 Player Enjoyment

Perhaps the most important challenge for designing a GWAP is the amount of enjoyment it provides players with. In [52], the authors explain a model for evaluating players' enjoyment which includes the following elements:

1. *Concentration*: Games should have a high workload, within appropriate levels of players' perceptual, cognitive and memory limits. They should quickly grab the players' attention and maintain focus for as long as possible.

2. *Challenge*: Games should provide challenges to match players' skill levels.

3. *Player skills*: Games must provide tutorials and in-game help to assist players learn easily and have fun at the same time.

4. *Clear goals*: Games should have a clear objective that is immediately apparent to players.

During the tutorials, we explain to the players that their objective is to find expressions (loop invariants) to solve a puzzle and increase their score to move to the top of the scoreboard. Players can understand the operators available to them and also revisit the tutorials at a later stage through the in-game help. As explained later in section 2.3, we have designed the interface such that players focus on the traces and are alerted, only when necessary, to events occurring in the game such as an increase in their score, a new player joining the game and a new expression being found.

2.2 Architecture

The game has a client-server architecture, as shown in Figure 2.1. Players connect to the server through their browser and all their requests to the server are handled by nginx,

which acts a high-performance web server and reverse proxy [44]. Since low latency and high scalability are paramount in a multiplayer game, routing requests through nginx reduces communication overhead and improves handling of a high number of concurrent requests without burdening hardware resources [40].

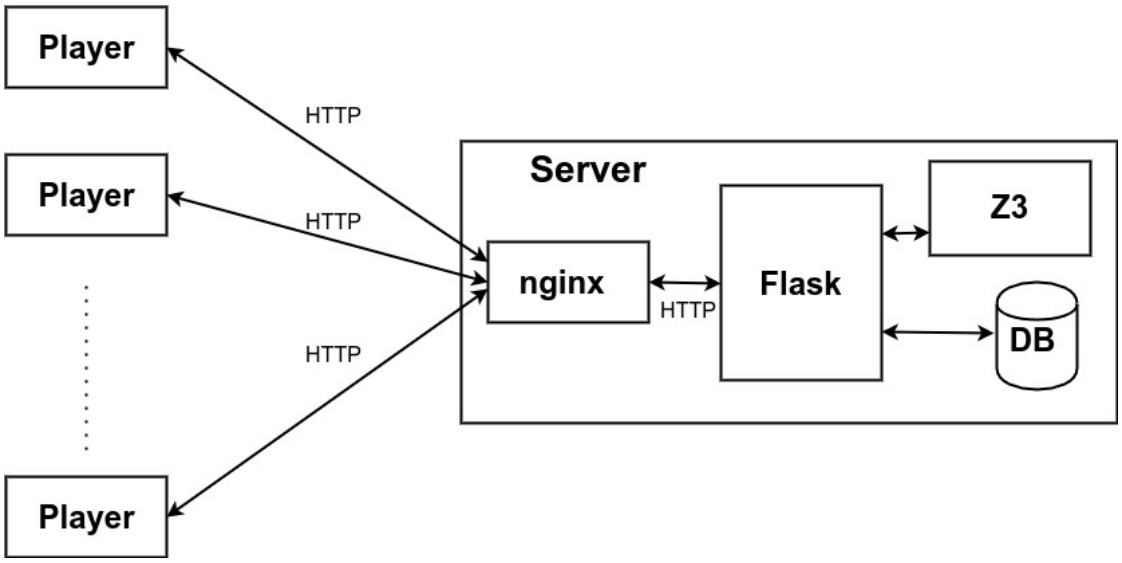

Figure 2.1: Architecture diagram of the multiplayer game

The nginx server forwards HTTP requests for dynamic content to the Flask micro web framework [23], which provides a RESTful API implementation for the multiplayer game. Although our current implementation involves a single Flask instance, we can run multiple Flask instances on one or more commodity servers and have nginx act as a load-balancer to increase scalability and performance.

Each Flask instance maintains a key-value store to synchronize the states of games and communicate the same to all players at fixed intervals of time. This key-value is currently implemented as a Python dictionary, but it can be an in-memory database such as Redis to boost read and write performances [3]. Each game has a unique, randomly-generated game identifier *gameId*, which acts as the key in this key-value store. The value corresponding to every *gameId* has the format shown in Figure 2.2

The request from every player includes the *gameId* and the server responds with

```
{
    "players": [
        {
            "id": "<id>",
            "score": <score>
        },
        {
            "id": "<id>",
            "score": <score>
        },
        ...
    ]
    "expressions": {
        "<levelId>": [
            {"<invariant>": "<HTML representation>"},
            {"<invariant>": "<HTML representation>"},
            ...
        ],
        "<levelId>": [
            {"<invariant>": "<HTML representation>"},
            {"<invariant>": "<HTML representation>"},
            ...
        ],
        ...
    }
}
```

Figure 2.2: Key-value store format to store a game's state

the value corresponding to this *gameId* in the key-value store. The synchronization logic is implemented on the client-side in JavaScript to reduce the load on the server, and the pseudocode is shown in Figure 2.3.

This synchronization logic is implemented as a JavaScript function on the client-side which executes at frequent intervals (currently every 1 second). If a player's local state, which is a copy of the current *gameId*'s value, is not equal to the server's state for the same *gameId*, then the new players and the new expressions found by all the other players are merged into the player's local state. Executing this function at frequent

$localState \leftarrow$ getLocalState()

$serverState \leftarrow$ getServerState($gameId$)

if $localState \neq serverState$ **then**

$localPlayers \leftarrow$ mergePlayers($localState$, $serverState$)

$localExpressions \leftarrow$ mergeExpressions($localState$, $serverState$)

$localState \leftarrow$ ($localPlayers$, $localExpressions$)

updateServerState($gameId$, $localState$)

Figure 2.3: Synchronizing a player's state with the server

intervals may cause a 1-2 second latency for a player's recently found expression to be visible to all other players, but future versions can provide a real-time experience by using the WebSocket protocol to communicate changes in states instead of sending the entire state snapshot over HTTP [7][42].

Currently every level of the game also exists as a Boogie program on the server [6][14][30]. We extract environment information from these programs (such as variable names and values), combine it with the expressions provided by players then use the Z3 SMT solver [11] to verify if these expressions are loop invariants. If yes, then the players are notified and the expressions are added as loop invariants.

2.3 Game Design

After signing in to the game, players are directed to a landing page which lists their username, total points scored so far and their current rank on the leaderboard (Figure 2.4). From the "How To Play" section, players can launch the multiplayer tutorial and then head to the "Play" section to either start a new game or join an existing game, if any (Figures 2.5 and 2.6). Players can view the global leaderboard by clicking on the "Leaderboard" tab, as shown in Figure 2.7.

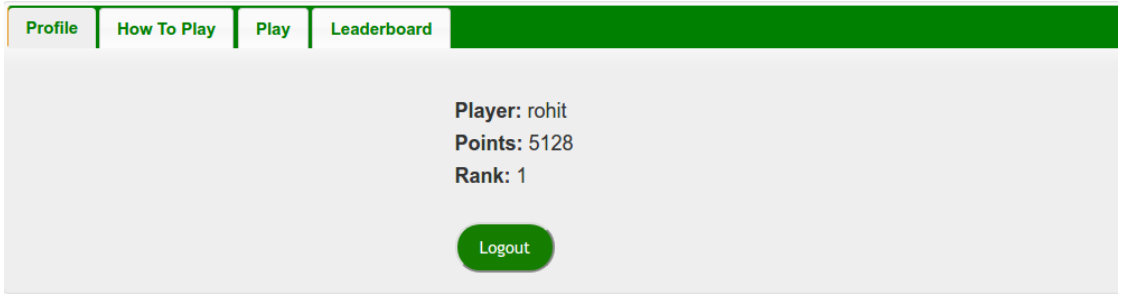

Figure 2.4: Players' landing page after signing in

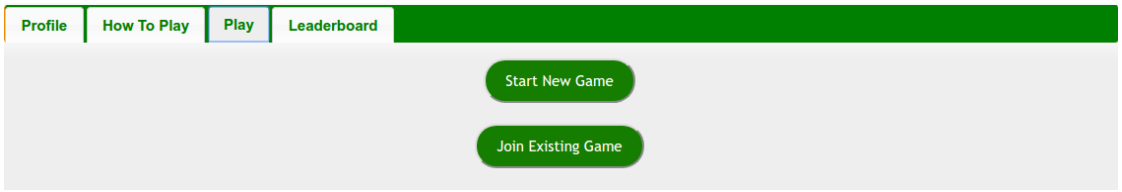

Figure 2.5: Players can either start a new multiplayer game or join an existing game

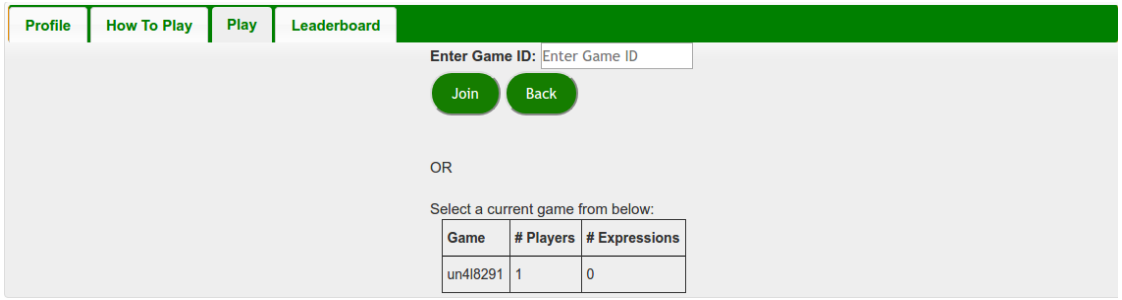

Figure 2.6: Players can join an existing game by selecting it from a list or by entering a known Game ID

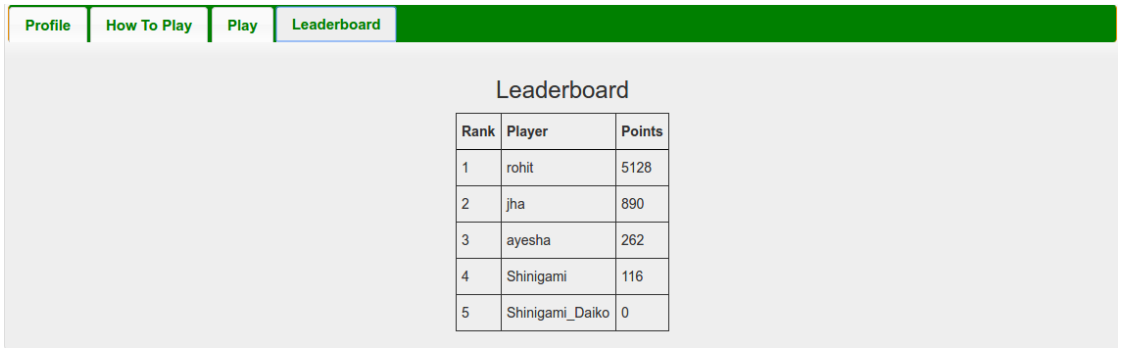

Figure 2.7: The global leaderboard

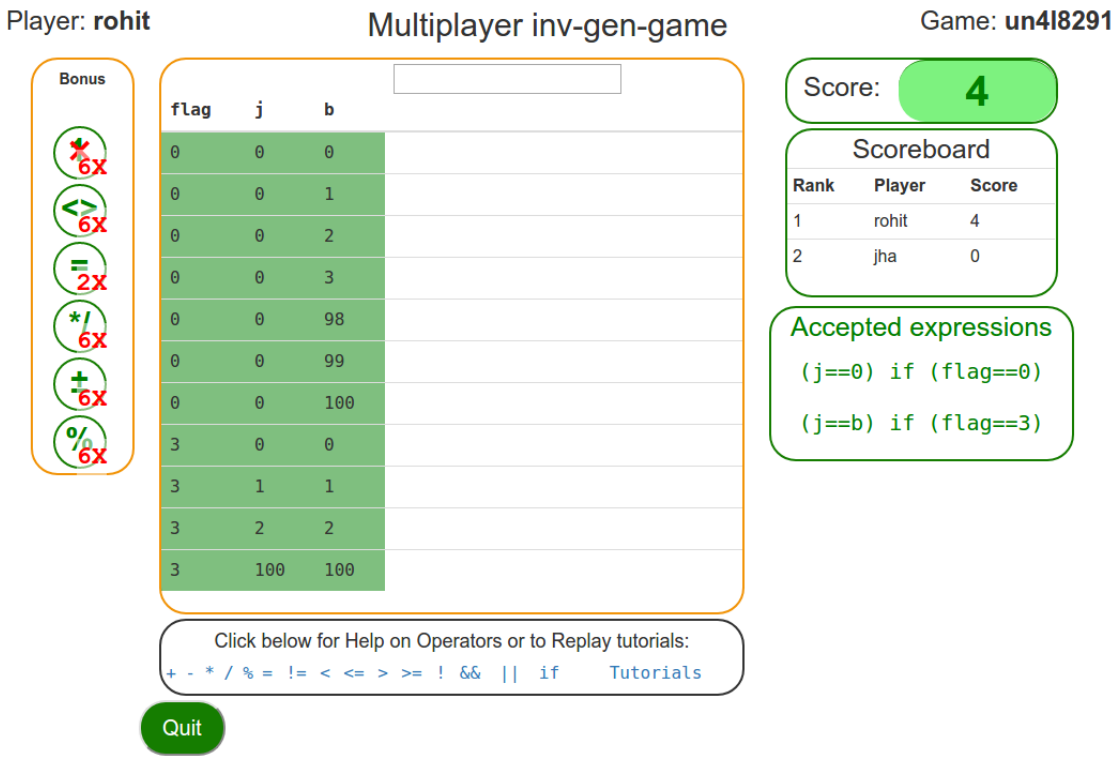

Figure 2.8: The multiplayer game's interface

2.3.1 Game Interface and Play

Once players start or join a game, the game's interface visible to them is the one shown in Figure 2.8. The various components of the game's interface are:

1. *Trace data and input*: Trace data from loops are shown in a tabular form with variable names as column headings and each row showing the value help by all variables in an iteration. As shown in Figure 2.9, a textbox allows players to type their expressions (loop invariants) and view the results of their expressions for every iteration instantly. In case the entered expression is not a loop invariant, the rows corresponding to the iterations where the expression does not hold are highlighted in red. In addition, a message under the textbox explains to the player why the expression is not acceptable. If the expression entered by the player is a loop invariant, the message instead reads "Press enter..." in green.

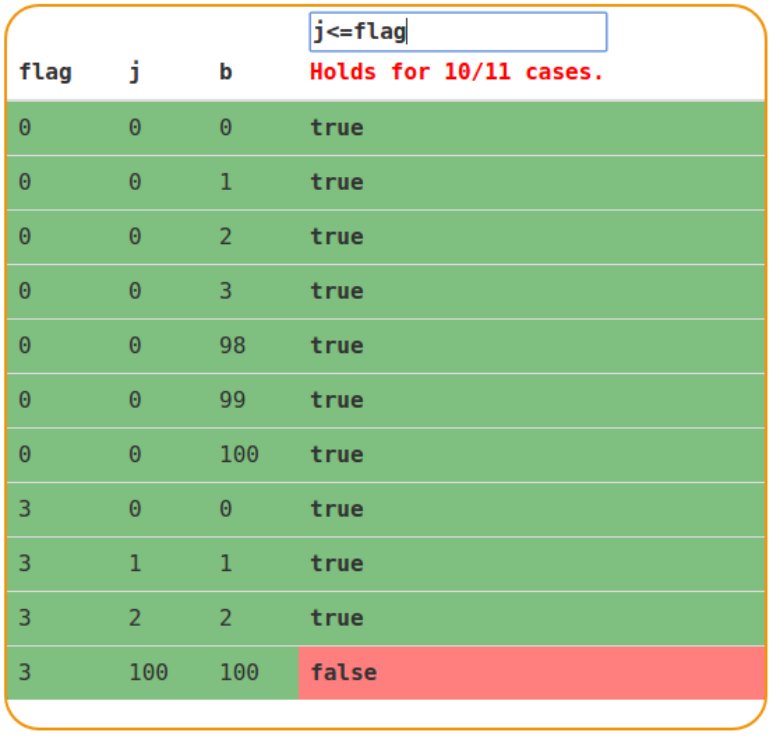

Figure 2.9: A player trying an expression

2. *Accepted expressions*: When players press enter after finding a loop invariant, their expressions is added to the list of accepted expressions and made visible to all players. When an expression found by any player appears as an accepted expression, players are notified by a visual pop-up accompanied by an audio alert.

3. *Bonus*: To encourage players to find more expressions, players are provided with bonus points for using different operators, variables and constants through various power-ups. The strength of these power-ups increases the longer they are not applicable for accepted expressions. For example, not using an inequality operator will increase its strength from 2 to 4 when the player attempts to find the next expression. To help players understand the effect, players are alerted when the power-up is applied or when the strength changes. Another bonus available to players is that they are awarded 10 points when they find an expression stronger

than one or more existing accepted expressions found by any of the players. The basis of determining whether an expression is stronger than another is explained in Section 2.1.6.

In Figure 2.8, the Bonus section on the left lists the available power-ups and their current strengths: 6 times as many points for not using a constant, 6 times as many points for using an inequality operator, 2 times as many points for using equality, 6 times as many points for using either the multiplication or division operators, 6 times as many points for using the addition or subtraction operators, and 6 times as many points for using the modulus operator.

4. *Player score and scoreboard*: Towards the top-right of the screen are the player's current score and a scoreboard ranking players in the current game according to their scores.

5. *Help section*: Towards the bottom of the screen is the help section which players can access at any point while playing the game. Clicking on an operator shows the player the description of the operator and examples to help understand how to use it (Figure 2.10). Clicking on "Tutorial" shows an option to either replay the entire tutorial, replay the part explaining how to use "if" in expressions, or watch a video of the tutorial explaining the use of "if".

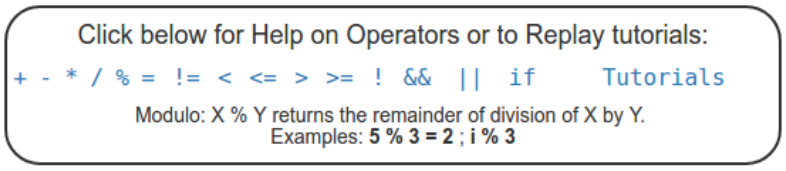

Figure 2.10: Help menu showing description of the modulus operator

Each game is composed of multiple levels, not organized in any order since it is difficult to predict the amount of effort that will be required to solve a level. In order to

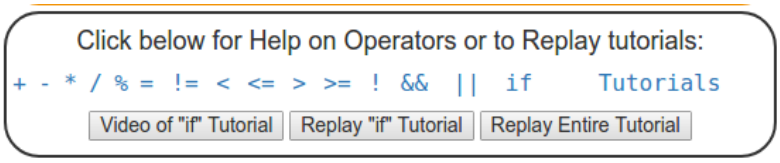

Figure 2.11: Players can launch the tutorial from the game

keep players engaged, we allows players to skip to the next level when at least six loop invariants have been found at a level. Such levels are marked as "completed" but not solved. A level is marked "solved" when the Z3 SMT solver returns that the given set of loop invariants are sufficient to solve the equivalent Boogie program. Once a player has completed a level, they proceed to the next level while the other players at the same level see a button "Skip to Next Level" appear at the bottom of their screen. They can then choose to continue finding stronger expressions or proceed to the next level.

Chapter 3

Evaluation

3.1 Methodology

We randomly divided 14 participants into two groups of seven players each and had them all play the single- and multiplayer versions of the game for 30 minutes each. To counterbalance order effects, we had the players in the first group play the single-player game followed by the multiplayer game in groups of two or three players. We then had the players in the second group play the multiplayer game followed by the single-player game. In each of these games, players were geographically distributed and were unaware of other players' identities, preventing any inadvertent collaboration or cheating. At the end of each game we had the players answer a short survey about their playing experience, and a final survey after playing both games. We describe the results of these surveys and our findings in the next section.

3.2 Results

For all the single- and multiplayer games, we measured the following:

1. the number of levels completed
2. player efficiency in terms of the average time taken to complete levels, the average time taken to find loop invariants, etc.
3. fun and engagement as reported by players

3.2.1 Levels Completed

We measured how many players could complete each level during all the games and show the results in Figure 3.1.

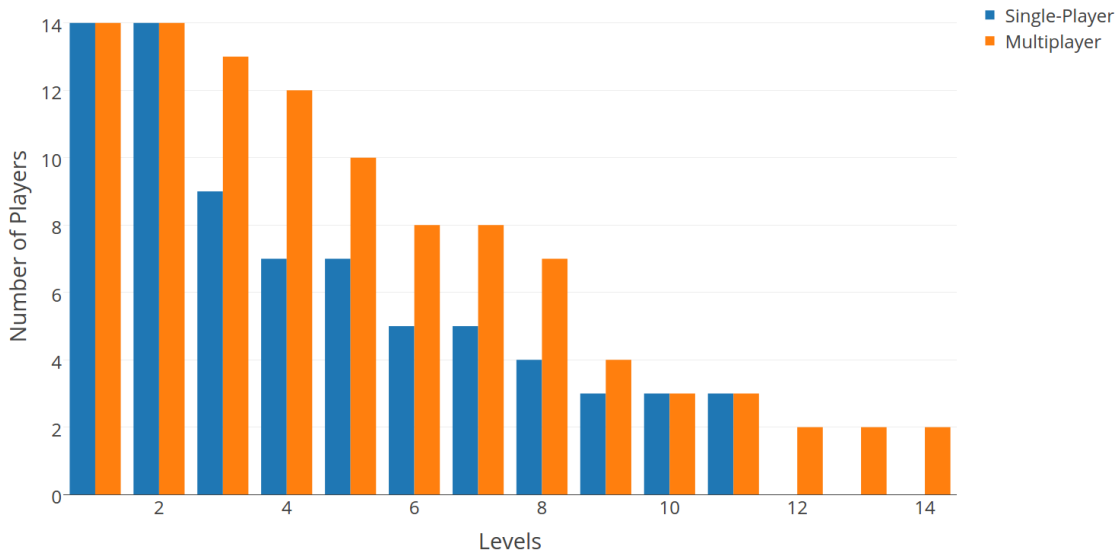

Figure 3.1: Number of players completing each level

Figure 3.1 shows that during both the single- and multiplayer games, all players could complete levels 1 and 2, but it became easier to complete levels 3-9 in the multiplayer version. Levels 10 and 11 were solved by three players in both versions, but levels 12-14 could only be solved in the multiplayer version. These results indicate that more problems could be solved through multiplayer games than single-player games in the same amount of time.

Table 3.1: Players' efficiency for single-player games and multiplayer games involving two or three players

	Single-Player	Two-Player	Three-Player
#Levels solved	2	3	7
Avg. #levels completed	5.286	6.125	8.833
Avg. #invariants found per player	28.571	40.625	45.5
Avg. #invariants found per level by each player	5.674	6.81	5.188
Avg. time per level (s)	364.03	289.686	216.112
Avg. time to find an invariant per player (s)	72	41.865	41.77

3.2.2 Player Efficiency

To determine how players' efficiency varies between single- and multiplayer games, we measured a few indicators that are listed in Table 3.1.

In case of the two and three-player versions of the multiplayer game, players were able to complete 3 and 7 levels out of a total of 14 levels, as compared to the single-player game, where only 2 levels were solved. We can see that for the same duration of play, 15.87% more levels were completed with two players and 67.1% more levels with three players as compared to the single player games. Also, the average number of loop invariants found by players rises significantly in multiplayer games – 42.19% more invariants with two players and 59.25% more invariants with three players. Even though the average number of loop invariants found per player increases in multiplayer games, the average number of invariants found per level by each player reduces when the number of players increases from two to three. The reason for this could be because the contribution of each player to solve a level become smaller as the number of players increases.

In case of a two-player game, the average time to solve a level drops to 79.58% of the time to solve a level in single-player game. This time drops to 59.37% in a multiplayer

game with three players, suggesting that having more players is beneficial to quickly solve problems. Furthermore, in case of multiplayer games with two and three players, the average time to find an invariant by a player drops to 58.15% and 58% of the time taken during single player games, respectively.

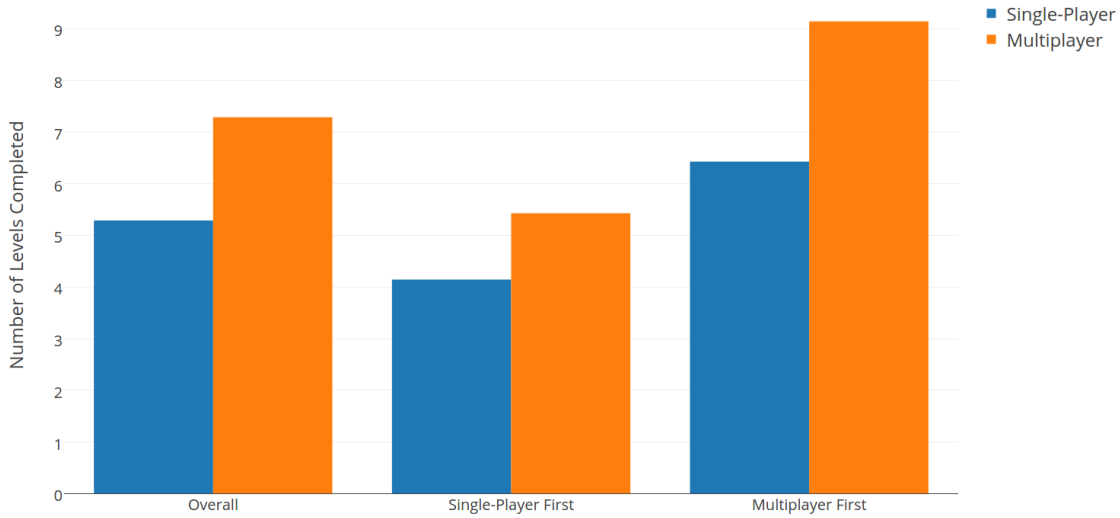

Figure 3.2: Average number of levels completed when playing the single- or multiplayer version first

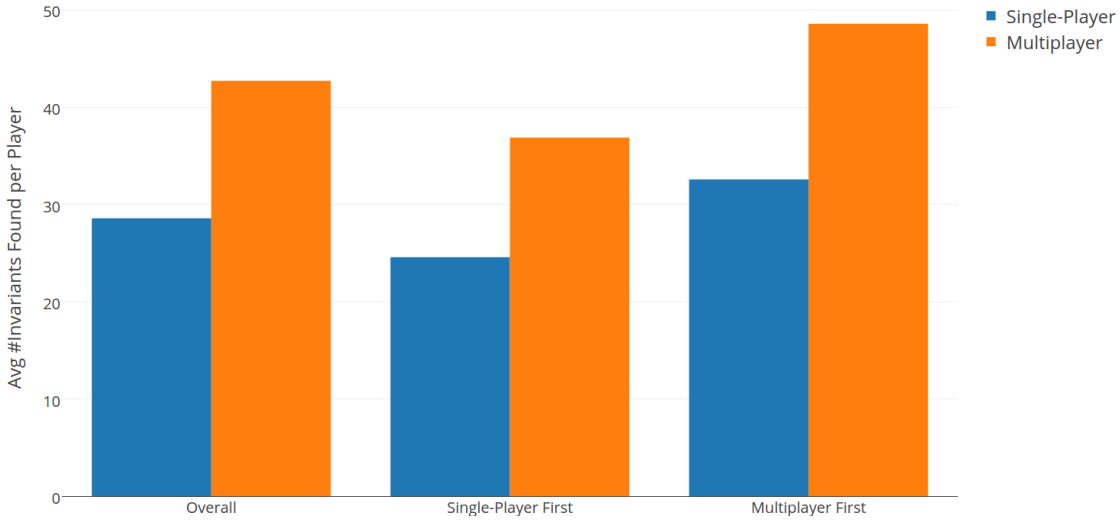

Figure 3.3: Average number of invariants found when playing the single- or multiplayer version first

As we can see from Figure 3.2, the participants who played the single-player

version before playing the multiplayer version completed an average of approximately 4 levels and participants who played the multiplayer version first completed more than 9 levels on average. To understand the effect on number of invariants found by players, we compare the results in Figure 3.3. Participants playing the single-player game first found an average of 24.571 invariants while participants playing the multiplayer game first found an average of 48.571 invariants, which is nearly twice as many. An interesting observation is that, in both these cases, participants who played the multiplayer version first performed better during the single-player version than players who played the single-player version first.

Not only did players playing the multiplayer game first find a larger number of invariants and solve more levels than players playing the single-player game first, but they also did so more efficiently. We can see from Figures 3.4 and 3.5 that participants spent nearly half the time per level (204 seconds) when playing the multiplayer game first, as compared to participants who played the single-player game first. While the collaborative aspect of the game can explain the reduction in time, it does not explain the observation that players switching from the single-player to multiplayer game took longer to solve a level on average. This effect is less pronounced when looking at the average time to find an invariant. Participants playing the multiplayer game took approximately the same time in each of the versions irrespective of the order or playing, as discussed above.

3.2.3 Fun and Engagement

Based on the heuristics for evaluating games designed for enjoyment, we measured how much fun players had while playing, the amount of challenge they faced and the focus the game required of them [36][37]. We asked every player after their single- and multiplayer games how much fun they had on a five-point Linkert scale, with 1 standing for “No fun at all” and 5 for “A lot of fun”. We also asked players how

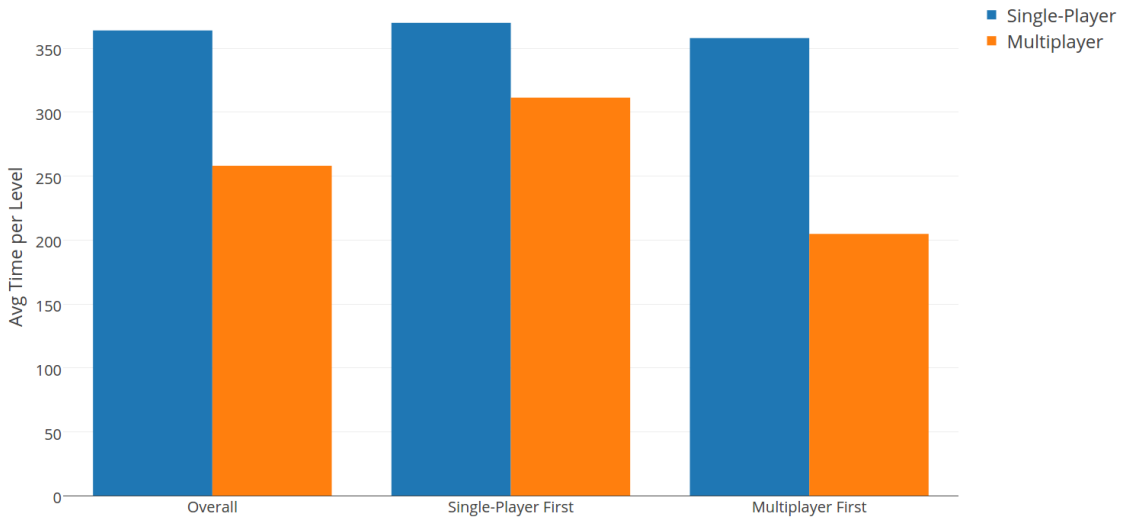

Figure 3.4: Average time spent per level by each player

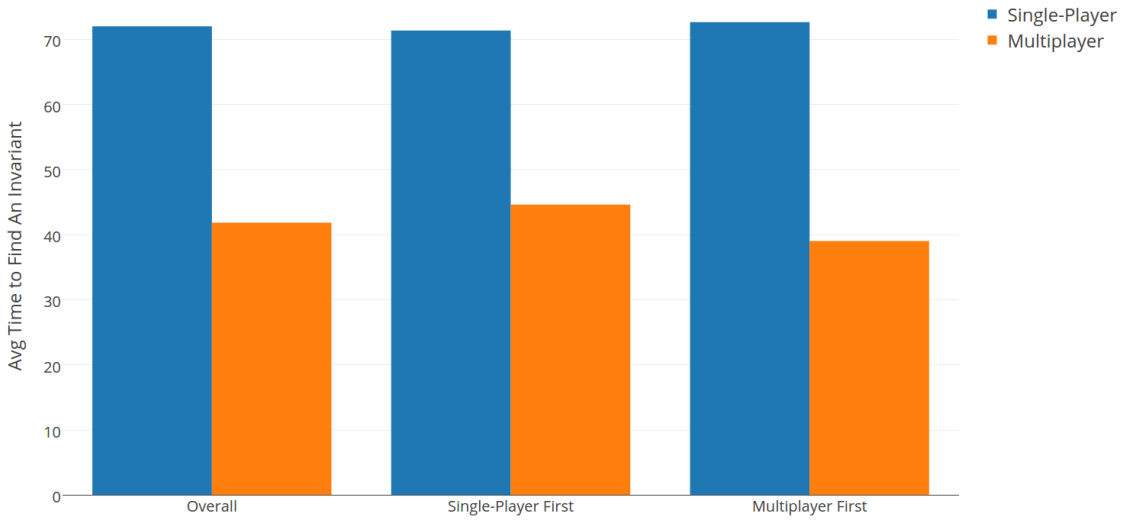

Figure 3.5: Average time spent to find an invariant by each player

challenging they found their games and how much they had to focus while playing on similar five-point Linkert scales. Figure 3.6, 3.7 and 3.8 show the players' responses.

As we can see from Figures 3.6 and 3.7, players had more fun playing the multiplayer version and faced a higher challenge playing it, which can be attributed to the competitive nature of the game. Even though the level of focus/concentration required during the single- and multiplayer games was not the same for every player, the number

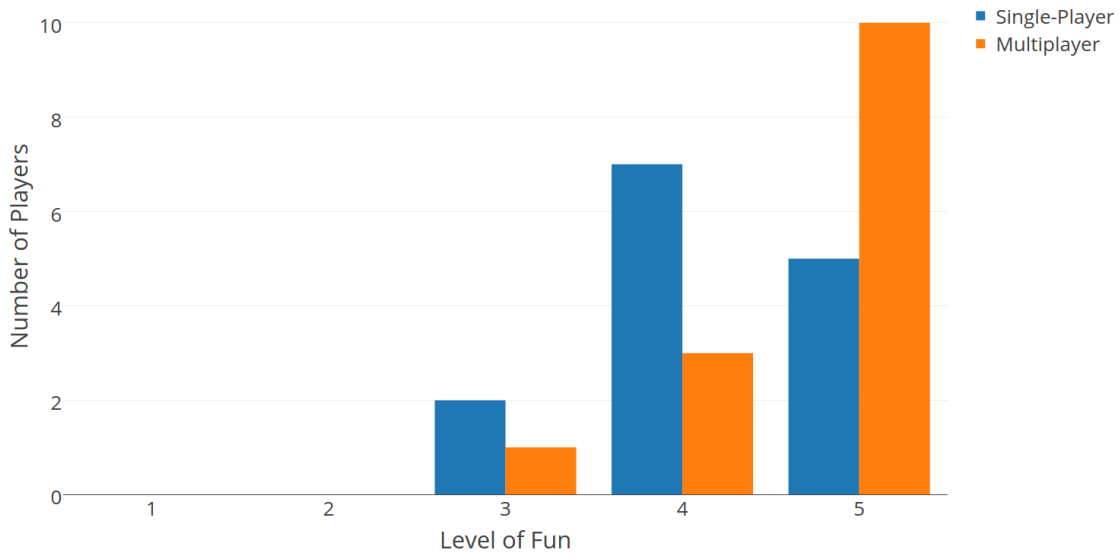

Figure 3.6: Players' reported levels of fun during the single- and multiplayer games

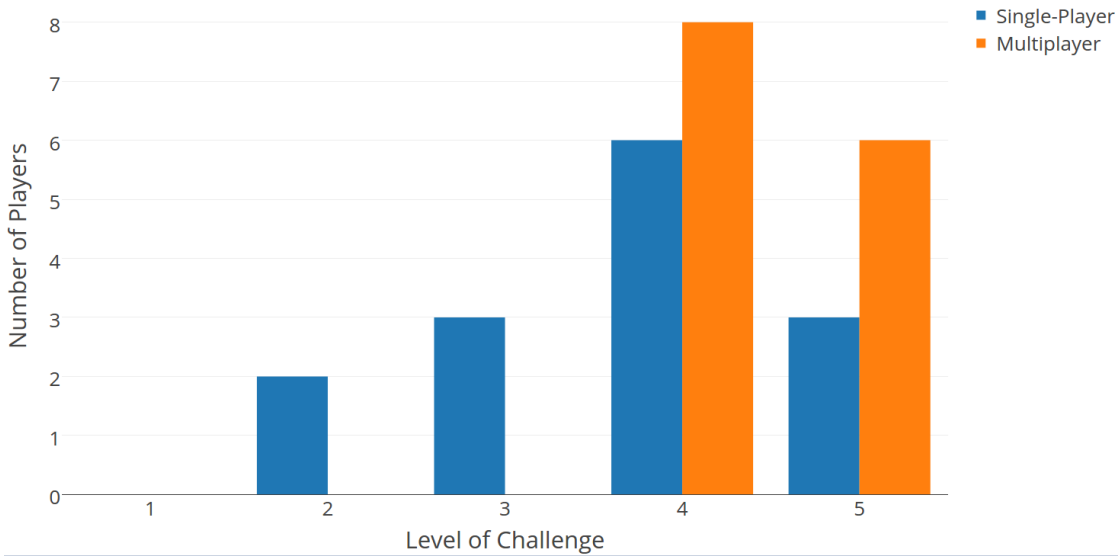

Figure 3.7: Players' reported levels of challenge felt during the single- and multiplayer games

of players reporting each level turned out to be equal. Some players may not have had to focus as much during the multiplayer game to solve a level since collaboration with additional players made the task easier. However, some players who reported a higher

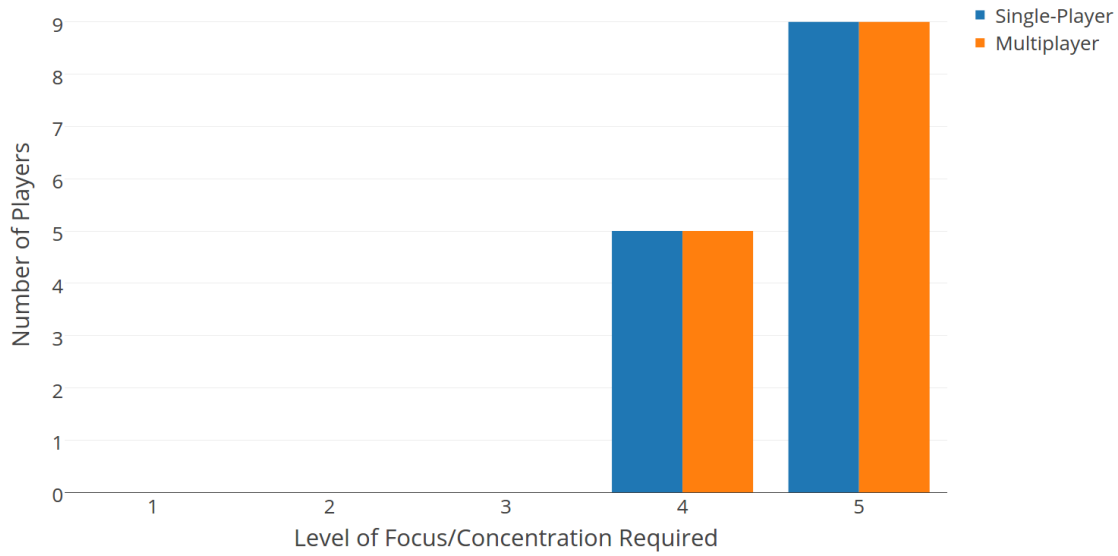

Figure 3.8: Players' reported levels of concentration during the single- and multiplayer games

level of concentration required in their multiplayer games could have felt this due to the competition with other players.

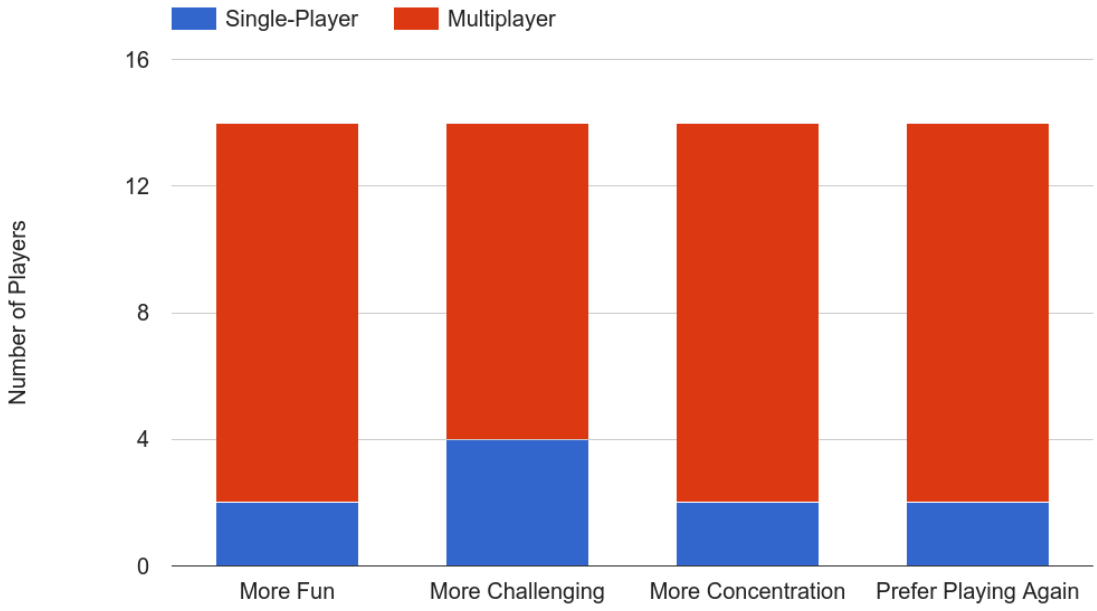

Figure 3.9: Players' reported levels of fun, challenge and concentration during the single- and multiplayer games and the version they prefer to play again

When players had finished playing both the single- and multiplayer games, we

asked them which version they enjoyed playing more, which version they found more challenging, and in which version they had to focus more. Figure 3.9 shows that 85.7% or 12 out of 14 players enjoyed playing the multiplayer version more than the single-player version, 71.4% felt that the multiplayer version was more challenging and 85.7% players had to focus more during the multiplayer game. We also asked players which version of the game they would prefer playing again in this case too 85.7% reported that they would prefer the multiplayer game.

3.2.4 Player Demographics

To understand the skills possessed by players, we asked them about their expertise in mathematics and programming. Figure 3.10 shows that out of the 14 players, 12 had completed a mathematics course as part of their Bachelor's degree, 1 as part of their Master's degree and 1 as part of their PhD. Since our game involves propositional logic, we believe that a graduate-level mathematics course would not put players at any advantage and these players performed around the average in terms of number of loop invariants found and levels completed.

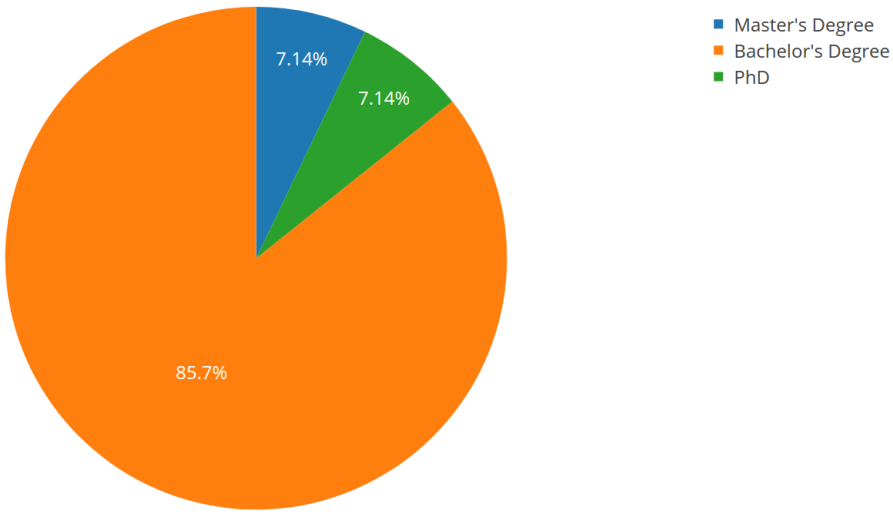

Figure 3.10: Highest education level at which players took a mathematics course

We also asked players about their programming expertise levels and players reported themselves to be have intermediate, advanced or professional expertise, as can be seen in Figure 3.11. However, for these skill levels, we did not notice any indicators that show an advantage to any player, suggesting that an intermediate level of expertise is sufficient to play the game.

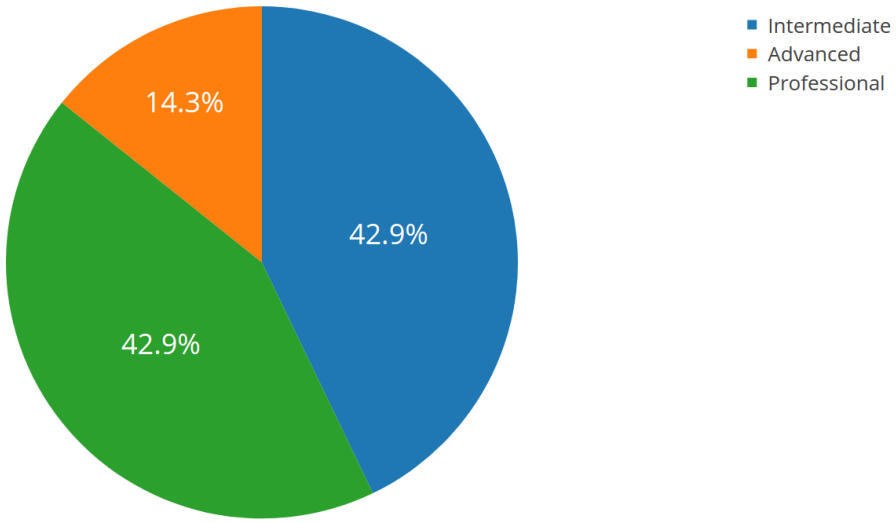

Figure 3.11: Players' reported levels of programming expertise

Chapter 4

Related Work

4.1 CrowdMine

CrowdMine taps into humans' ability to recognize patterns in images and proposes to crowdsource the task of writing specifications for a circuit through Amazon Mechanical Turk [31]. CrowdMine first transforms parts of a trace into images and tasks the participants to find patterns in those images. Each trace is transformed into a 2D image by having the rows represent signals and the columns represent cycles. A player is awarded points based on the number of cells in an identified pattern, but is prevented from submitting patterns that have already been found.

The game succeeds in protecting a circuit's internal details, which can be proprietary. However, it suffers from a few drawbacks:

1. A 15-second time limit per level does not allow players to find complex or long patterns.

2. By encoding different signal values as shades of colors, CrowdMine makes it difficult for players to differentiate between a large number of signal values.

By avoiding any time limitations for levels, our game allows players to find stronger patterns after careful deliberation. This also gives them time to refine their opponents' invariants and move up the scoreboard. Furthermore, players in our game rely on using variable names, their numerical values and operators to easily represent a higher number of possible invariants.

4.2 Xylem: The Code of Plants

Xylem [35] is an iPad puzzle game for players to find loop invariants by comparing plants' growth stages side-by-side, thereby avoiding exposing players to the mathematical aspect. Each variable is represented as a flower and the number of flowers at each loop iteration/stage represents the corresponding variable's value at that loop iteration/stage. While solving these puzzles, players move to different levels and collaborate with other players to solve harder problems [12].

While the abstraction and collaboration attracts a large pool of players, Xylem does suffer from a few drawbacks:

1. Despite targeting a "casual niche" audience, Xylem could only attract players with mathematics or computer science backgrounds. Unfortunately, this meant that the decision to use plants as a visual metaphor did not pay off.

2. Players can view at the most only two plant growth stages side-by-side, significantly limiting the ability to find more (and complex) patterns quickly and requiring players multiple attempts to correctly find invariants.

3. The interface design makes it cognitively difficult for players to work with a large number of variables and a large range of values.

Our game does not rely on visual metaphors but we do provide our audience with tutorials, descriptions and examples to understand and use a variety of operators. Since we represent trace data in a tabular form, with seven or more variables in some levels from at least four loop iterations, players can find all the information they need at a glance to generate loop invariants quickly.

Chapter 5

Conclusion and Future Work

In this thesis, we have shown how a hybrid collaborative-competitive multiplayer game can be used to find loop invariants more efficiently than single-player games, and be even more fun and engaging to play. In our trials, we found that players in the multiplayer games were able to solve more problems than when they played a single-player game, and came up relatively quickly with loop invariants – approximately taking 60% of the time to find 60% more invariants and complete over 67% more levels. We also found that players completed more levels and found more invariants when they played the multiplayer game before the single-player game. These results suggest that, in addition to the collaborative and competitive aspects of the multiplayer game, a higher level of fun and challenge reported during the multiplayer games may be a contributing factor to the increase in players' efficiency.

Future studies can examine the efficacy of having more players per game and determining whether there is a plateau or decline in players' efficiency after a point. By adding support for non-integer values such as Booleans and including quantifiers for working with arrays, future versions of the game will be able to solve a wider variety of problems. While the current implementation lies in the human computation domain,

we expect that it is possible to reach out to a wider audience through crowdsourcing platforms such as Amazon Mechanical Turk. Since studies have shown that the fastest growth in players has been on the mobile platform, adapting the game to work on these platforms may also help attract and retain players to solve problems.

Bibliography

[1] David P Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@ home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[2] Michael Buhrmester, Tracy Kwang, and Samuel D Gosling. Amazon's mechanical turk a new source of inexpensive, yet high-quality, data? *Perspectives on psychological science*, 6(1):3–5, 2011.

[3] Josiah L Carlson. *Redis in Action*. Manning Publications Co., 2013.

[4] Jon Chamberlain, Udo Kruschwitz, and Massimo Poesio. Methods for engaging and evaluating users of human computation systems. In *Handbook of Human Computation*, pages 679–694. Springer, 2013.

[5] R Chandrasekar, E Chi, M Chickering, PG Ipeirotis, W Mason, F Provost, and J Tam. von ahn, front matter. *Proc. SIGKDD HCOMP 2010*, 2010.

[6] Bor-Yuh Evan Chang, Bart Jacobs, Rustan Leino, Mike Barnett, Robert DeLine, and Rob DeLine. Boogie: A modular reusable verifier for object-oriented programs. 2005.

[7] Bijin Chen and Zhiqi Xu. A framework for browser-based multiplayer online games using webgl and websocket. In *Multimedia Technology (ICMT), 2011 International Conference on*, pages 471–474. IEEE, 2011.

[8] Ning Chen. Gate: game-based testing environment. In *Proceedings of the 33rd international Conference on Software Engineering*, pages 1078–1081. ACM, 2011.

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

[10] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM, 1977.

[11] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[12] Drew Dean, Sean Gaurino, Leonard Eusebi, Andrew Keplinger, Tim Pavlik, Ronald Watro, Aaron Cammarata, John Murray, Kelly McLaughlin, John Cheng, et al. Lessons learned in game development for crowdsourced software formal verification. *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*, 2015.

[13] Katherine Deibel. On the automatic detection of loop invariants. *Online]. http://courses. cs. washington. edu/courses/cse503/02wi/papers/deibel. pdf*, 2002.

[14] Robert DeLine and K Rustan M Leino. Boogiepl: A typed procedural language for checking object-oriented programs. 2005.

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

[16] Michael D Ernst, Jake Cockrell, William G Griswold, and David Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering*, 27(2):99–123, 2001.

[17] Michael D Ernst, Jeff H Perkins, Philip J Guo, Stephen McCamant, Carlos Pacheco, Matthew S Tschantz, and Chen Xiao. The daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1):35–45, 2007.

[18] Melissa A Federoff. *Heuristics and usability guidelines for the creation and evaluation of fun in video games*. PhD thesis, Citeseer, 2002.

[19] Robert W Floyd. Assigning meanings to programs. *Mathematical aspects of computer science*, 19(19-32):1, 1967.

[20] Carlo A Furia, Bertrand Meyer, and Sergey Velder. Loop invariants: Analysis, classification, and examples. *ACM Computing Surveys (CSUR)*, 46(3):34, 2014.

[21] Pranav Garg, Daniel Neider, Parthasarathy Madhusudan, and Dan Roth. Learning invariants using decision trees and implication counterexamples. In *ACM SIGPLAN Notices*, volume 51, pages 499–512. ACM, 2016.

[22] David Gries. A note on a standard strategy for developing loop invariants and loops. *Science of Computer Programming*, 2(3):207–214, 1982.

[23] Miguel Grinberg. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2014.

[24] Ashutosh Gupta and Andrey Rybalchenko. Invgen: An efficient invariant generator. In *International Conference on Computer Aided Verification*, pages 634–640. Springer, 2009.

[25] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, October 1969.

[26] Jeff Howe. *Crowdsourcing: How the power of the crowd is driving the future of business*. Random House, 2008.

[27] Deepak Kapur. Automatically generating loop invariants using quantifier elimination. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2006.

[28] Laura Kovács and Andrei Voronkov. Finding loop invariants for programs over arrays using a theorem prover. In *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2009 11th International Symposium on*, pages 10–10. IEEE, 2009.

[29] Edith Law and Luis von Ahn. Human computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 5(3):1–121, 2011.

[30] K Rustan M Leino. This is boogie 2. *Manuscript KRML*, 178(131), 2008.

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

[32] Greg Little, Lydia B Chilton, Max Goldman, and Robert C Miller. Exploring iterative and parallel human computation processes. In *Proceedings of the ACM SIGKDD workshop on human computation*, pages 68–76. ACM, 2010.

[33] Greg Little, Lydia B Chilton, Max Goldman, and Robert C Miller. Turkit: human computation algorithms on mechanical turk. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, pages 57–66. ACM, 2010.

[34] Yanhong A Liu, Scott D Stoller, and Tim Teitelbaum. Strengthening invariants for efficient computation. *Science of Computer Programming*, 41(2):139–172, 2001.

[35] Heather Logas, Jim Whitehead, Michael Mateas, Richard Vallejos, Lauren Scott, Daniel G Shapiro, John Murray, Kate Compton, Joseph C Osborn, Orlando Salvatore, Zhongpeng Lin, Huascar Sanchez, Michael Shavlovsky, Daniel Cetina, Shayne Clementi, and Chris Lewis. Software verification games: Designing xylem, the code of plants. In *FDG*, 2014.

[36] Thomas W Malone. What makes things fun to learn? heuristics for designing instructional computer games. In *Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems*, pages 162–169. ACM, 1980.

[37] Thomas W Malone. Heuristics for designing enjoyable user interfaces: Lessons from computer games. In *Proceedings of the 1982 conference on Human factors in computing systems*, pages 63–68. ACM, 1982.

[38] Kerry Moffitt, John Ostwald, Ron Watro, and Eric Church. Making hard fun in crowdsourced model checking–balancing crowd engagement and efficiency to maximize output in proof by games. In *CrowdSourcing in Software Engineering (CSI-SE), 2015 IEEE/ACM 2nd International Workshop on*, pages 30–31. IEEE, 2015.

[39] Scott B Morris and Richard P DeShon. Combining effect size estimates in meta-analysis with repeated measures and independent-groups designs. *Psychological methods*, 7(1):105, 2002.

[40] Clément Nedelcu. *Nginx HTTP Server: Adopt Nginx for Your Web Applications to Make the Most of Your Infrastructure and Serve Pages Faster Than Ever*. Packt Publishing Ltd, 2010.

[41] Corina S Păsăreanu and Willem Visser. Verification of java programs using symbolic execution and invariant generation. In *International SPIN Workshop on Model Checking of Software*, pages 164–181. Springer, 2004.

[42] Victoria Pimentel and Bradford G Nickerson. Communicating and displaying real-time data with websocket. *IEEE Internet Computing*, 16(4):45–53, 2012.

[43] Alexander J Quinn and Benjamin B Bederson. Human computation: a survey and taxonomy of a growing field. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1403–1412. ACM, 2011.

[44] Will Reese. Nginx: the high-performance web server and reverse proxy. *Linux Journal*, 2008(173):2, 2008.

[45] Leonard Richardson and Sam Ruby. *RESTful web services*. " O'Reilly Media, Inc.", 2008.

[46] Enric Rodríguez-Carbonell and Deepak Kapur. An abstract interpretation approach for automatic generation of polynomial invariants. In *International Static Analysis Symposium*, pages 280–295. Springer, 2004.

[47] Sriram Sankaranarayanan, Henny B Sipma, and Zohar Manna. Non-linear loop invariant generation using gröbner bases. *ACM SIGPLAN Notices*, 39(1):318–329, 2004.

[48] Walt Scacchi and Kendra M Cooper. Research challenges at the intersection of computer games and software engineering. In *Proc. 2015 Conf. Foundations of Digital Games*, 2015.

[49] Walt Scacchi and Kendra ML Cooper. Emerging research challenges in computer games and software engineering. In *Computer Games and Software Engineering*, pages 261–284. Chapman and Hall/CRC, 2015.

[50] Rahul Sharma, Isil Dillig, Thomas Dillig, and Alex Aiken. Simplifying loop invariant generation using splitter predicates. In *International Conference on Computer Aided Verification*, pages 703–719. Springer, 2011.

[51] Entertainment Software Association. Essential facts about the computer and video game industry, 2017.

[52] Penelope Sweetser and Peta Wyeth. Gameflow: a model for evaluating player enjoyment in games. *Computers in Entertainment (CIE)*, 3(3):3–3, 2005.

[53] Marc Vass, John M Carroll, and Clifford A Shaffer. Supporting creativity in problem solving environments. In *Proceedings of the 4th conference on Creativity & cognition*, pages 31–37. ACM, 2002.

[54] Luis Von Ahn. Human computation. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 1–2. IEEE, 2008.

[55] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. Captcha: Using hard ai problems for security. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 294–311. Springer, 2003.

[56] Luis Von Ahn and Laura Dabbish. Labeling images with a computer game. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 319–326. ACM, 2004.

[57] Luis Von Ahn and Laura Dabbish. Designing games with a purpose. *Communications of the ACM*, 51(8):58–67, 2008.

[58] Luis Von Ahn, Shiry Ginosar, Mihir Kedia, and Manuel Blum. Improving image search with phetch. In *Acoustics, speech and signal processing, 2007. icassp 2007. ieee international conference on*, volume 4, pages IV–1209. IEEE, 2007.

[59] Luis Von Ahn, Mihir Kedia, and Manuel Blum. Verbosity: a game for collecting common-sense facts. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 75–78. ACM, 2006.

[60] Luis Von Ahn, Ruoran Liu, and Manuel Blum. Peekaboom: a game for locating objects in images. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 55–64. ACM, 2006.

[61] Luis Von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008.

[62] Ronald Watro, Kerry Moffitt, Talib Hussain, Daniel Wyschogrod, John Ostwald, Derrick Kong, Clint Bowers, Eric Church, Joshua Guttman, and Qinsi Wang. Ghost map: Proving software correctness using games. *SECURWARE 2014*, 223, 2014.