

# **Detonicon - A Networked Game Utilizing AI to Study the Effects of Latency Compensation**

A Major Qualifying Project

Submitted to the Faculty of

**WORCESTER POLYTECHNIC INSTITUTE**

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

in

Computer Science

and

Interactive Media & Game Development

by

---

Hung Hong

---

Antony Qin

Date: 25 April 2019

Report Submitted to:

---

Professor Mark Claypool, Advisor

Worcester Polytechnic Institute

**Abstract**

Online video games are pervasive and as a result, latency deteriorates the gaming experience. While latency compensation techniques exist, the problem is that not many developers can quantify the actual effects of latency compensation. Our goal is to precisely measure the effects of latency compensation in games. We created a networked game for use as a testbed and built a latency simulator and latency compensation into our game. In addition, we implemented AI with the ability to play the game like a human player to enable automated testing. Finally, we evaluated latency compensation by running a bot study to find out how players perform, and a user study to find how players feel. The results show that players using latency compensation performed better and also felt the game was less difficult.

# Table of Contents

|  |    |
|--|----|
| Abstract .....                           | 1  |
| Table of Contents .....                  | 2  |
| List of Figures .....                    | 4  |
| List of Tables .....                     | 6  |
| Chapter 1: Introduction .....            | 7  |
| Chapter 2: Methodology .....             | 9  |
| 2.1: Game Development .....              | 9  |
| 2.1.1: Alpha Build .....                 | 11 |
| 2.1.2: Alphafest .....                   | 14 |
| 2.1.3: Final Build .....                 | 15 |
| 2.2: AI Implementation .....             | 18 |
| 2.2.1: Bot Personality .....             | 18 |
| 2.2.1: Pathfinding .....                 | 20 |
| 2.3: Latency .....                       | 24 |
| 2.3.1: Latency Simulator .....           | 25 |
| 2.3.2: Latency Compensation .....        | 26 |
| Chapter 3: Evaluation .....              | 29 |
| 3.1: Bot Study Procedure .....           | 29 |
| 3.2: User Study Procedure .....          | 32 |
| Chapter 4: Results & Analysis .....      | 35 |
| 4.1: Bot Study Results & Analysis .....  | 35 |
| 4.2: User Study Results & Analysis ..... | 44 |
| Chapter 5: Postmortem .....              | 53 |
| 5.1: What Went Wrong .....               | 53 |
| 5.2: What Went Well .....                | 54 |
| Chapter 6: Conclusion .....              | 55 |
| Chapter 7: Future Work .....             | 56 |
| References .....                         | 57 |

Appendix A .....58  
Appendix B .....60  
Appendix C .....62

## List of Figures

|  |    |
|--|----|
| Figure 2.1.1: Dragonfly Title Screen .....   | 10 |
| Figure 2.1.2: Precision and Deadline .....   | 10 |
| Figure 2.1.3: Super Bomberman 5 .....  | 11 |
| Figure 2.1.1.1: Detonicon Screenshot (Alpha Build) .....                                 | 12 |
| Figure 2.1.3.1: Detonicon Screenshot (Final Build) .....                                 | 17 |
| Figure 2.2.1.1: Bot Decision Tree .....  | 19 |
| Figure 2.2.2.1: Pathfinding with Breakable Wall Avoided .....                            | 21 |
| Figure 2.2.2.2: Pathfinding with Breakable Wall Considered .....                         | 22 |
| Figure 2.2.2.3: Path Smoothing in Horizontal Direction .....                             | 23 |
| Figure 2.2.2.4: Path Smoothing in Vertical Direction .....                               | 23 |
| Figure 2.3.1: Latency Summarized .....   | 24 |
| Figure 2.3.1.1: Latency Simulator Queue .....  | 25 |
| Figure 2.3.2.1: Client Movement Without Latency Compensation .....                       | 27 |
| Figure 2.3.2.2: Client Movement Using Latency Compensation .....                         | 28 |
| Figure 4.1.1: Bot Win Rate vs Latency and Latency Compensation .....                     | 35 |
| Figure 4.1.2: Bot Survivability vs Latency and Latency Compensation .....                | 36 |
| Figure 4.1.3: Spaces Moved vs Latency and Latency Compensation .....                     | 37 |
| Figure 4.1.4: Movement Commands Sent vs Latency and Latency Compensation .....           | 37 |
| Figure 4.1.5: Number of Bumps vs Latency and Latency Compensation .....                  | 38 |
| Figure 4.1.6: Number of Rebounds While Using Latency Compensation .....                  | 40 |
| Figure 4.1.7: Distance Rebounded While Using Latency Compensation .....                  | 40 |
| Figure 4.1.8: Ratio of Number of Rebounds and Number of Spaces Moved .....               | 41 |
| Figure 4.1.9: Number of Times Hit by Explosion vs Latency and Latency Compensation ..... | 42 |
| Figure 4.1.10: Number of Bombs Placed vs Latency and Latency Compensation .....          | 43 |
| Figure 4.1.11: Number of Power-ups Picked Up vs Latency and Latency Compensation .....   | 43 |
| Figure 4.2.1: Perceived Smoothness vs Latency and Latency Compensation .....             | 45 |
| Figure 4.2.2: Perceived Responsiveness vs Latency and Latency Compensation .....         | 45 |
| Figure 4.2.3: Noticeability of Visual Glitches vs Latency and Latency Compensation ..... | 47 |

|  |    |
|--|----|
| Figure 4.2.4: Number of Times Rebounded Player and Bot Comparison .....      | 48 |
| Figure 4.2.5: Number of Spaces Rebounded Player and Bot Comparison .....     | 48 |
| Figure 4.2.6: Number of Spaces Moved Player and Bot Comparison .....         | 49 |
| Figure 4.2.7: Perceived Difficulty vs Latency and Latency Compensation ..... | 50 |
| Figure 4.2.8: Player Survivability vs Latency and Latency Compensation ..... | 51 |
| Figure 4.2.9: Player Survivability Compared with Bot Survivability .....     | 52 |

## List of Tables

|   |    |
|---|----|
| Table 2.1.1.1: Detonicon Game Elements .....      | 13 |
| Table 2.1.2.1: Alphafest Survey Data .....        | 14 |
| Table 2.1.3.1: Feature Priority .....             | 16 |
| Table 3.1.1: Detonicon Client Statistics .....    | 30 |
| Table 3.1.2: Testing Machine Specifications ..... | 31 |
| Table 3.1.3: Bot Study Tests .....                | 32 |
| Table 3.1.4: User Study Tests .....               | 34 |

## Chapter 1: Introduction

Network connectivity enables online games to be prevalent and people enjoy playing them. However, this can mean online games are constrained by a network connection. As a result, latency, or the delay in communication, is a common challenge in online gaming since it can reduce responsiveness of the gameplay and consistency of the game world [1].

In response to the issue of latency, latency compensation techniques have been developed in order to improve online gaming performance and experience [2]. Many different types of latency compensation techniques are practiced in games of all genres, such as First-Person Shooter (FPS) or Massive Multiplayer Online Role-playing Game (MMORPG). Aspiring game developers that are interested in developing online games may consider development of latency compensation for their infrastructures to deliver enjoyable games to their players.

Despite its prevalence, there are few quantified measurements as to how latency compensation benefits gameplay performance and experience over a range of games and player actions. How much does latency compensation help with scoring? With moving the avatar? With shooting? How much does latency compensation help enjoyment? Answers to such questions may help developers weigh the costs and benefits of implementing latency compensation.

The primary goal of our project is to precisely quantify the effect of latency compensation in games by experimentally measuring the effects of latency on gameplay and comparing it to the same gameplay with latency compensation. Secondary goals include producing a game with a range of actions, bots with human-like behavior, and a framework for experiments.

In order to achieve our objectives, we created a networked game called *Detonicon* and implemented latency as well as latency compensation into the game for testing purposes. We also implemented AI in order to execute automated tests. Finally, we ran experiments and did analysis on the data from both a bot study and a user study.

Based on results from 300 bots runs and 19 users our findings validate our knowledge and expectation of latency compensation in games. Specifically, latency compensation improves player survivability by about 10% and improves the perceived responsiveness by about 40%.



The rest of this report is organized as follows: **Chapter 2: Methodology** describes the process of game development, the AI implementation, the latency simulator and the latency compensation implementation; **Chapter 3: Evaluation** presents the evaluation procedure, how we collect data, both from bot study and user study, and what it means for effectiveness of latency compensation; **Chapter 4: Results & Analysis** analyzes our findings on latency compensation and player performance and experience; **Chapter 5: Postmortem** provides a postmortem of the project; **Chapter 6: Conclusion** summarizes our conclusions from our project; and **Chapter 7: Future Work** describes possible future work.

## Chapter 2: Methodology

In order to precisely measure the effects of latency compensation, we:

1. Created a networked game (Chapter 2, Section 2.1)
2. Implemented bot AI to play the game similar to a human player (Chapter 2, Section 2.2)
3. Implemented a latency simulator and latency compensation into the game (Chapter 2, Section 2.3)
4. Evaluated latency and latency compensation with a bot study and a user study (Chapter 3)

Our first step is to create a networked game to act as a testbed and an infrastructure to produce further work utilizing our structure. Our next step is to implement a latency simulator to allow us to simulate specific latencies for clients without third-party applications. Additionally, we implemented a type of latency compensation into our game in order to test its effects on players. We created bot AI to emulate human players, enabling repetitive tests without the need for actual playtesters. Finally, we ran a bot study in order to gather objective data (how the player performs) and a user study to gather subjective data (how the player feels).

### 2.1 Game Development

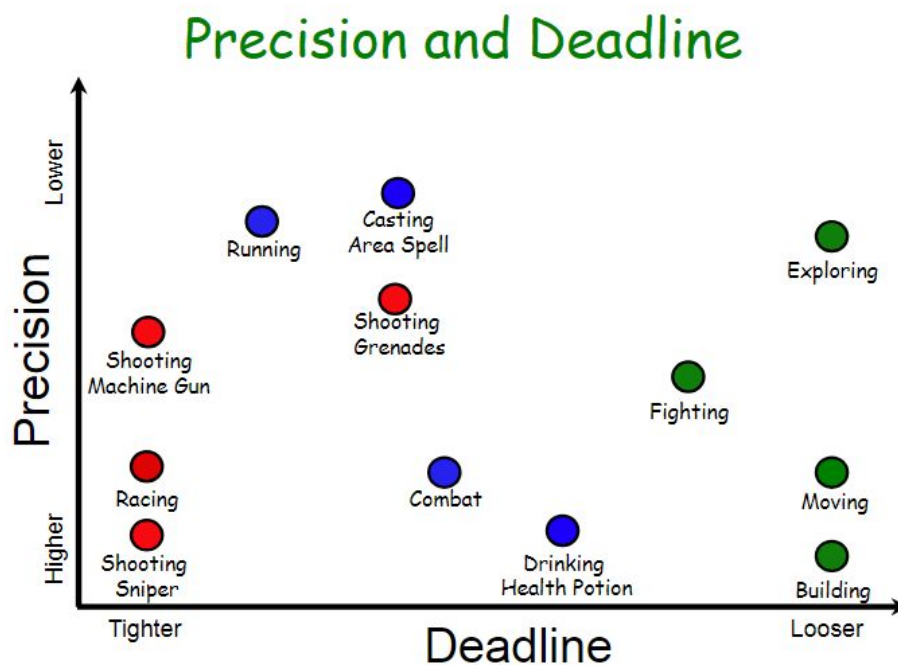
This section discusses how we came up with the game idea and developed our game. We designed a game to use as a research tool for us to study the effects of latency with the following requirements:

1. The game must be created using the Dragonfly engine
2. The game must be a networked game
3. The game must have gameplay that is affected by latency
4. The game should be simple to play and understand



**Figure 2.1.1: Dragonfly Title Screen**

The Dragonfly engine is an ASCII-based C++ game engine to teach game development [3]. **Figure 2.1.1** shows the title screen of the engine and showcases the ASCII art style. Dragonfly was primarily made as a learning tool, but can be used for research.



**Figure 2.1.2: Precision and Deadline [1]**

Precision, or how much accuracy a player action requires, and Deadline, or how long a player action requires, determine how an action is affected by latency [1]. A graph showing how actions are scaled to Precision and Deadline is shown in **Figure 2.1.2**. Actions closer to the origin (bottom right corner) are more affected by latency. Since latency has a variable effect on

different types of games, latency compensation could also have a variable effect on different types of games. A game with flexible Precision and Deadline for actions enables testing the effects of latency compensation for a range of game types.

Since Dragonfly is ASCII-based, we decided on a 2D design as that art style lends itself to ASCII. We decided against Co-op, as that would require the development of different bot AI for allies and enemies. Rather than players with different skills, unique strengths and weaknesses, we decided that players should have the same actions for simpler evaluation. Items scattered across the map allowed players to obtain different abilities. This design philosophy allowed for a single, consistent AI to be used to emulate human player action.



**Figure 2.1.3: Super Bomberman 5 [4]**

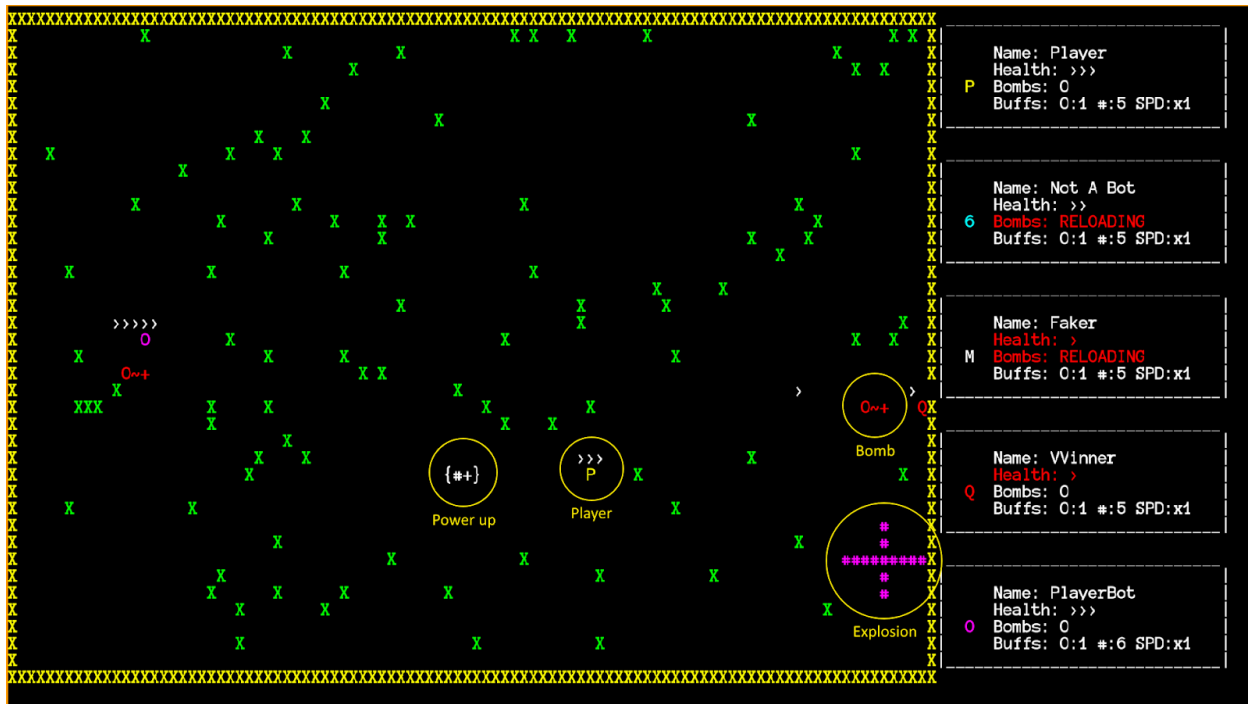
After deliberation, we settled on a design akin to the franchise *Bomberman* (Hudson Soft, 1983). A picture of the game *Super Bomberman 5* (Hudson Soft, 1997) that served as inspiration is shown in **Figure 2.1.3**. *Detonicon* is a battle-royale type of game that revolves around up to 5 players who each have the ability to drop a bomb at their feet that explodes on a timer. Precision is modifiable with bomb explosion size and Deadline is modifiable with bomb timer duration. The goal in *Detonicon* is to be the last one standing.

### 2.1.1 Alpha Build

The alpha build of a game is typically “feature-complete”. This means that the mechanics of the game are all present in the build. The alpha build may not be “asset-complete”, meaning it

does not have all the art and sound intended for the final build of the game in at present. The alpha build of our game included the basic mechanics of:




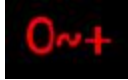
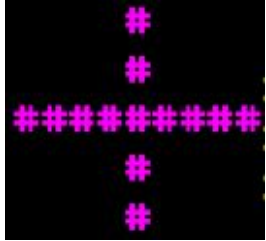

- Movable player
- Placing & kicking bombs
- Enemy bot AI, up to 4
- HUD to display stats of each player
- Breakable & Unbreakable walls
- 1 Map
- Power-ups
- Win / Lose Condition



**Figure 2.1.1.1: Detonicon Screenshot (Alpha Build)**

Figure 2.1.1.1 shows a screenshot of the game's alpha build. Table 2.1.1.1 sums up several key elements in *Detonicon*.

Table 2.1.1.1: Detonicon Game Elements

| Game Element     | Sprite  | Purpose   |
|------------------|---|---|
| Breakable Wall   |    | Serves as an obstacle, can be destroyed by a bomb explosion   |
| Unbreakable Wall |    | Serves as an obstacle, cannot be destroyed by a bomb explosion  |
| Player           |    | <ul style="list-style-type: none"> <li>- Player avatars represented by 'O', 'Q', 'P', 'M', or 'G' are randomly assigned to each player</li> <li>- Can be moved using 'W', 'A', 'S', 'D' or arrow keys</li> <li>- HP bar is expressed with '&gt;' per each HP, and is on top of the character</li> </ul>   |
| Bomb             |    | <ul style="list-style-type: none"> <li>- Can be placed by any player using 'SPACE' key, or be kicked by running over the bomb</li> <li>- The number of bombs for each player depends on the individual bomb count</li> <li>- Explodes after a certain period of time</li> </ul>   |
| Explosion        |  | <ul style="list-style-type: none"> <li>- Created by a bomb after a certain period of time in a '+' shaped pattern</li> <li>- Length of the bomb placed by each player depends on their bomb power</li> <li>- Inflicts 1 damage to the player(s) in range</li> <li>- Destroys breakable wall(s)</li> </ul>   |
| Power-ups        |  | <ul style="list-style-type: none"> <li>- 4 types of Power-ups: <ul style="list-style-type: none"> <li>● Increase bomb power by 1</li> <li>● Increase bomb count by 1</li> <li>● Restore health by 1</li> <li>● Increase move speed</li> </ul> </li> <li>- Can be picked up by any player by running over them</li> <li>- Individual stats are updated to the HUD</li> </ul> |

The game ends once there is one player left alive. In the case where the remaining players die at the same time, the game ends in a tie.

### 2.1.2 Alphafest

Alphafest is a social gathering hosted by the IMGD program at WPI where we presented our alpha build. As stated on the WPI website, “Alphafest is a chance for the entire IMGD community to hang out together, show off project work in progress, conduct playtesting, gather feedback and eat pizza!” [5]. The primary purpose of participating in Alphafest was to elicit feedback from players unfamiliar to *Detonicon* in order to improve the game. We collected feedback through the instrumentation of paper surveys, given to players after they played *Detonicon*. The questions and answers collected from the survey are shown in **Table 2.1.2.1**.

**Table 2.1.2.1: Alphafest Survey Data**

| Question   | Scale  | Surveys |   |   |     |     |   | Average |
|--|--------|---------|---|---|-----|-----|---|---------|
|  |        | 1       | 2 | 3 | 4   | 5   | 6 |         |
| How clear were the sprites in conveying information?               | 1 to 4 | 3       | 3 | 2 | 1   | 2   | 3 | 2.33    |
| How clear was the HUD in conveying information?                    | 1 to 4 | 2       | 3 | 3 | 2   | 3   | 4 | 2.83    |
| How clear were the instructions in conveying information?          | 1 to 4 | 4       | 2 | 2 | N/A | 4   | 4 | 3.2     |
| How human-like were the bots?                                      | 1 to 4 | 2       | 2 | 4 | 1   | 2   | 2 | 2.17    |
| How was the length of a game?                                      | 1 to 4 | 3       | 4 | 4 | 4   | 3   | 3 | 3.5     |
| How was the size of the map?                                       | 1 to 4 | 3       | 3 | 2 | 4   | 3   | 3 | 3       |
| If you played this game over a network, would lag affect the game? | 1 to 4 | 3       | 4 | 4 | 4   | N/A | 2 | 3.4     |

In addition to the paper surveys, we noted additional comments given by players at the time of the playtesting as well as feedback from those who observed the game simply being played. Our main takeaways from the survey and feedback were as follows:

- The map was too big
  - Players could not reliably fight each other
  - The time to finish a game was too long
- The interface was unclear
  - Players could not find their controlled character easily
  - Players did not know what each power-up did
  - Players did not understand the meaning of the HUD
- Players did not read the instructions before playing
- The bot AI needs more work
  - The bots were too predictable and had the tendency to endanger themselves
  - A few commented the bots were “unfair” during the late game, and had too much space to dodge explosions

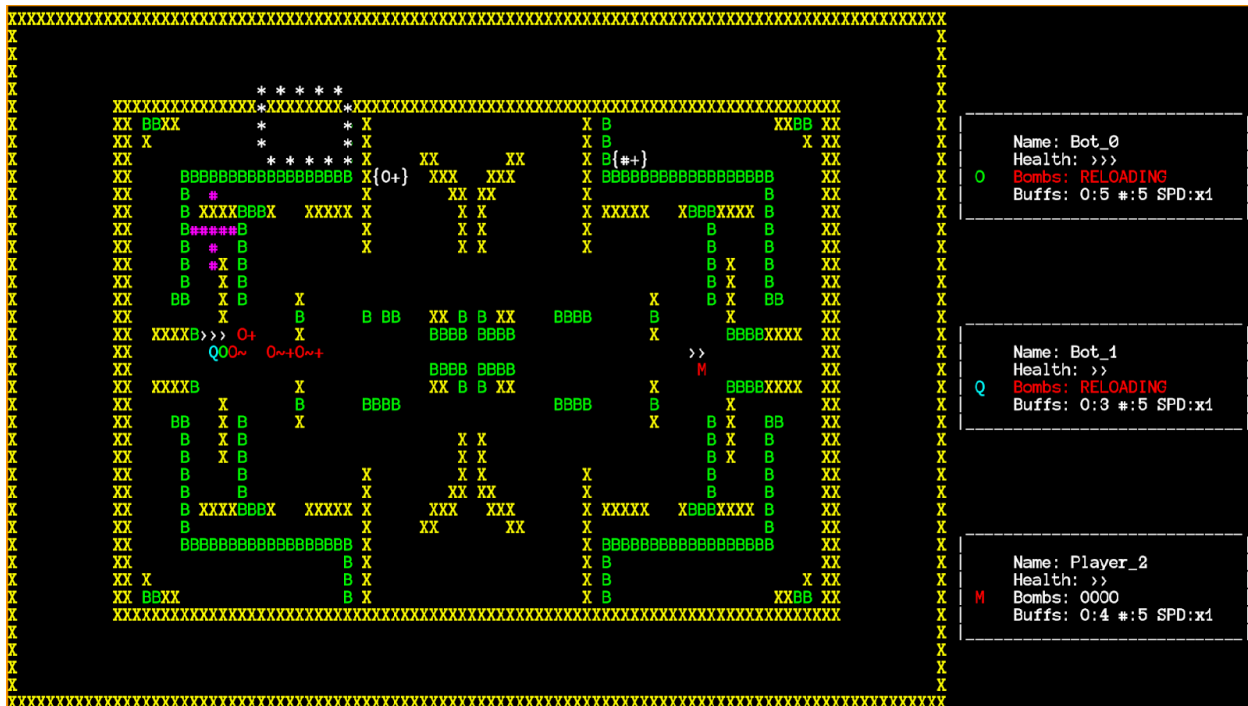
### **2.1.3 Final Build**

We improved Detonicon by taking into account the feedback gained from Alphafest. We tabulated features to address the issues cropped up at Alphafest, ranked by importance and given a difficulty to implement rating. The resulting **Table 2.1.3.1** was used to determine priority and the order in which we implemented features. All the tabulated features were eventually implemented into the game.



**Table 2.1.3.1: Feature Priority**

| <b>Feature To Implement</b>  | <b>Issue Feature Addresses</b>  | <b>Importance of Feature</b> | <b>Difficulty to Implement Feature</b> | <b>Implemented in Final Build</b> |
|--|---|------------------------------|--|-----------------------------------|
| Shrinking map border as the game goes on.  | - The map was too big.<br>- Players had ample space to run instead of fighting others.                        | HIGH                         | HARD                                   | Yes                               |
| Arrows point to and highlight the player-controlled character on game start.     | Players could not find the player-controlled character on game start.   | HIGH                         | EASY                                   | Yes                               |
| Shake HUD of character that takes damage.  | Players could not quickly tell which character was getting damaged without reading the HUD of that character. | MEDIUM                       | EASY                                   | Yes                               |
| Improve bot AI, especially danger detection of enclosing walls or explosions.    | Bot AI during Alphafest was not smart enough.   | HIGH                         | HARD                                   | Yes                               |
| Spawn a quick text snippet on power-up pick up that says what the power-up does. | Players could not tell what each power-up did.  | HIGH                         | MEDIUM                                 | Yes                               |
| Spawn a little spectacle firework on character deaths, and many on game victory. | To add flavor and flair to deaths and victory.  | LOW                          | EASY                                   | Yes                               |
| Add multiple maps, and a way to create a read them into the game.                | To improve the variety of maps and add additional strategy.   | MEDIUM                       | MEDIUM                                 | Yes                               |
| Add a way to read bot personalities into the game before it starts.              | To add an easy way to test different bot AI.  | LOW                          | EASY                                   | Yes                               |
| Change game from the debug build to a release build.                             | Improves resource utilization and optimization during gameplay.   | HIGH                         | EASY                                   | Yes                               |



**Figure 2.1.3.1: Detonicon Screenshot (Final Build)**

**Figure 2.1.3.1** shows a screenshot of *Detonicon*'s final build after taking into account the feedback gained from Alphafest. Most of the improvements were to make the game more understandable. Unbreakable walls are still represented by a yellow 'X' character, but now breakable walls are represented by the green 'B' character. This adds another level of distinction other than color to tell them apart. Characters that spawn have a graphic of arrows pointing to them so players can tell what character they control. Picking up power-ups now spawns a message such as 'BOMB POWER UP' so players know how their character was just strengthened. Our map design improved to be more structured and balanced. Most importantly, over the course of a game, the outer walls shrink in. If the walls reach each other in the middle, the game automatically ends. With this new mechanic, players are forced together, making it harder to dodge bomb explosions. Moreover, the game is guaranteed to end at a reasonable and tweakable time, allowing us to gather more tests during the evaluation.

## 2.2 AI Implementation

Since the goal of our project is to study the effectiveness of latency compensation through our game *Detonicon*, implementing artificial intelligence (AI) in order to emulate player behavior allows study on how our compensation technique performs objectively from automated tests where bots play against each other.

### 2.2.1 Bot Personality

We intend for our bots to resemble human behavior as much as possible. In other words, bots should choose the same actions as do human players in the game. Below are some actions that players normally do in *Detonicon*:

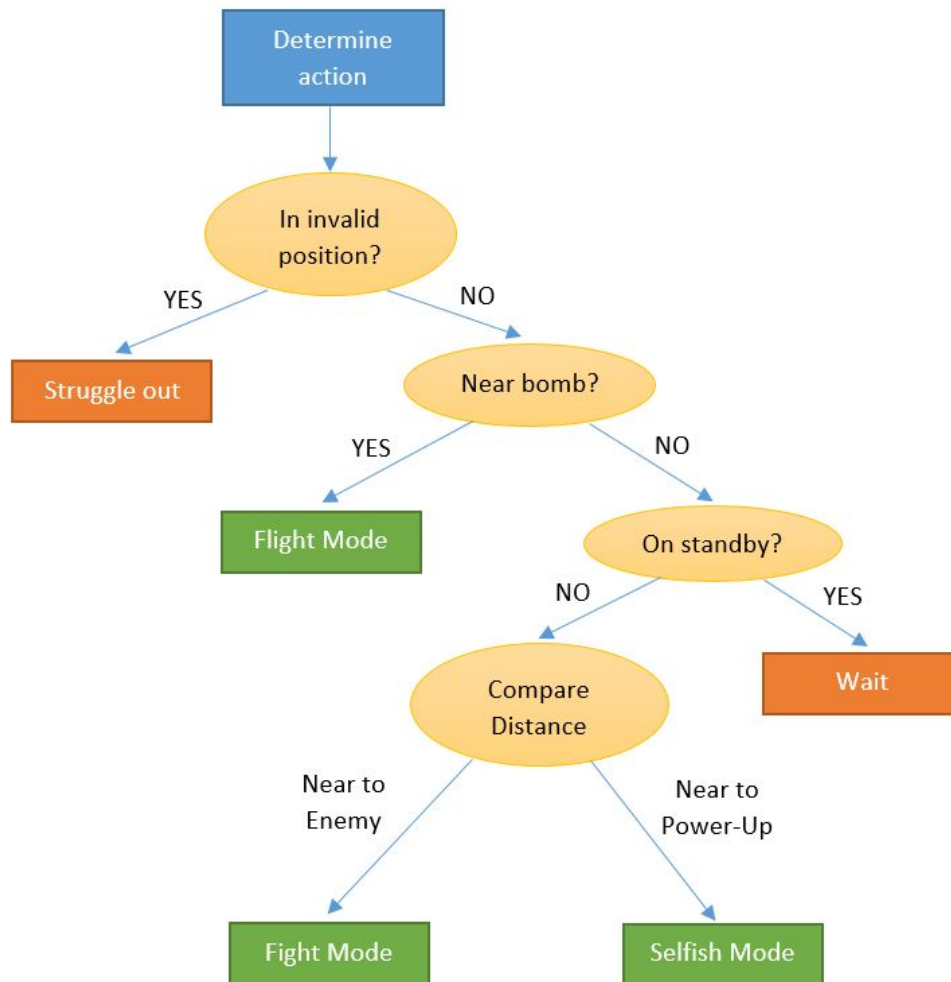
- Move towards other players
- Move towards power-ups
- Move away from bombs
- Move away from the enclosing walls
- Place bombs to attack other players
- Place bombs to break walls
- Kick bombs
- Obtain power-ups

Based on this list of actions, bots should have 3 main modes. Each mode represents the current state of the bots and dictates the appropriate action during each state:

- **Fight Mode:** The bot moves offensively to attack other players by approaching them and placing bomb near them.
- **Flight Mode:** The bot moves defensively to avoid getting hit by bomb explosions and wait safely until the explosions are over.
- **Selfish Mode:** The bot moves to obtain power-ups to gain an advantage in stats.

We built a decision tree to determine which mode the bot chooses, as shown in **Figure 2.2.1.1**, based on its surrounding environment as well as its personalized values assigned externally. These personalized values served as a mean to infuse personalities to the bots so that their behaviors are unique similar to human behaviors. The two personality values are:

- Brave Value. This value determines bot aggressiveness. The higher the Brave Value, the more likely that the bot engages in Fight Mode.
- Smart Value. This value determines bot awareness. The higher the Smart Value, the more accurately the bot skirts around the ranges of bomb explosions during Flight Mode.



**Figure 2.2.1.1: Bot Decision Tree**

The current location of the bot plays an important role in determining the course of action that the bot should undertake. From **Figure 2.2.1.1**, the bot starts with a simple check of whether its current location is invalid. An invalid position is either an out-of-bounds location (which should not happen in the game), or within the shrinking wall boundary. Since players do not

want to be crushed by a shrinking wall, a struggle movement is triggered by the bot to get out of the wall boundary. Going down the decision tree, the next decisions are based on how close is the bot to other key elements of the game, namely bombs, other players, and power-ups. The bot's personality values also determine the probability to enter a branch of the tree. The decision tree also supports a waiting action, in the case the bot has not completed certain decision yet. For example, the bot could wait after placing a bomb instead of deciding to move somewhere else.

### 2.2.2 Bot Pathfinding

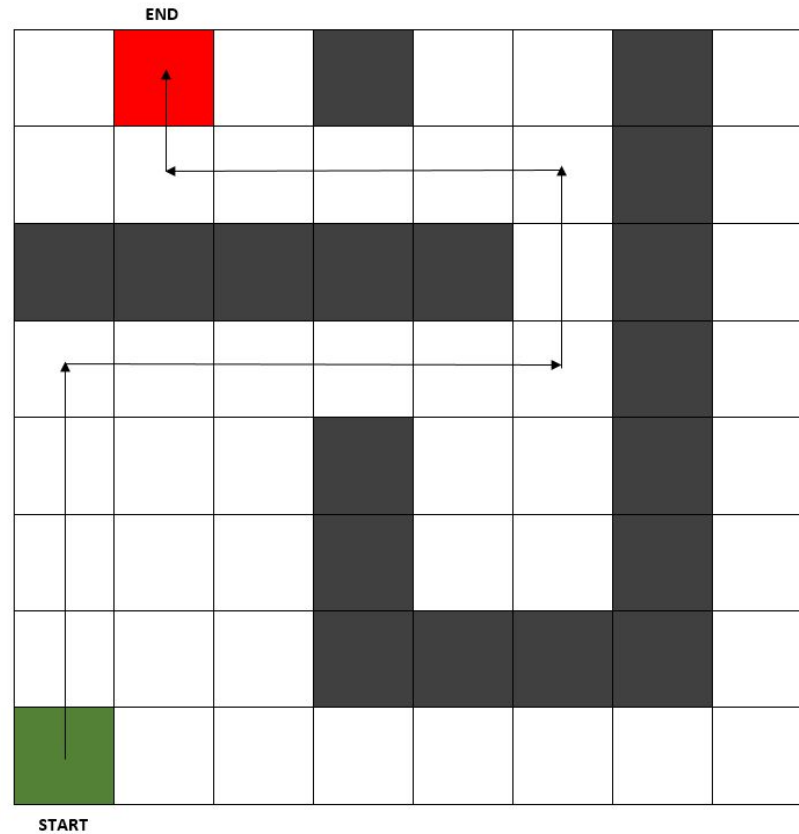
At first, we wanted to utilize the built-in pathfinding feature in Dragonfly. However, we found out that we would need a more customized algorithm due to the technical aspect of how players control movement in Detonicon as well as the existence of breakable walls in the game. In response, we developed our own pathfinding algorithm built on top of the A\* algorithm [6].

After inspecting the surrounding environment and adjusting to the appropriate mode, the bot determines its destination accordingly. The next task is to find the most efficient way to get from its current position to that destination using the A\* algorithm. A\* search aims to find a path from one starting node to another destination node with the smallest cost. Specifically, when building a path, A\* will try to minimize the following cost:

$$f(n) = g(n) + h(n)$$

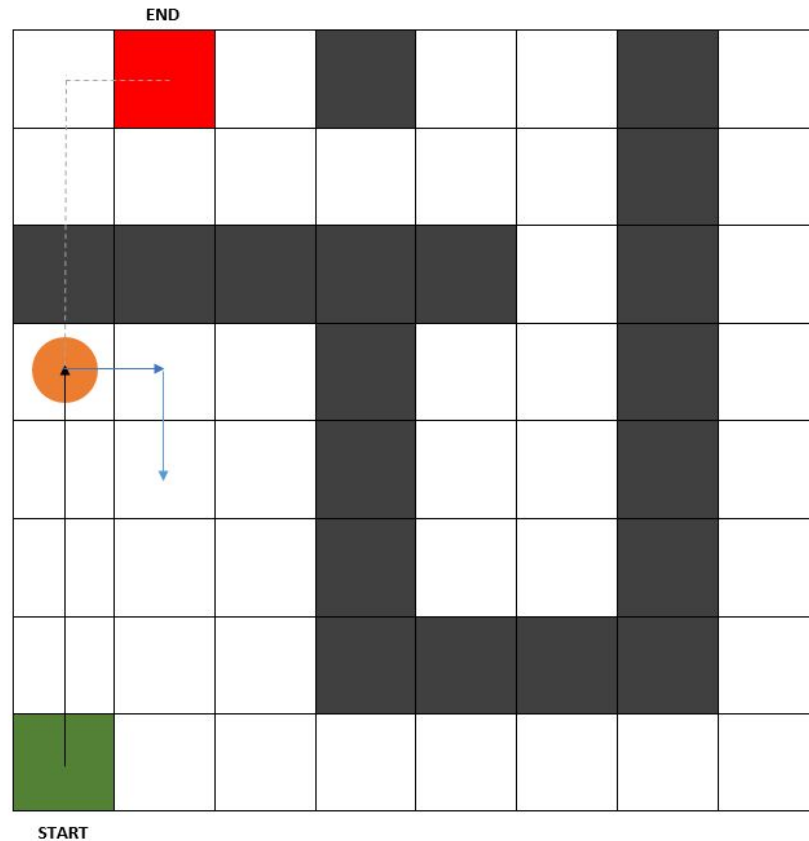
For the cost equation shown above,  $n$  is the next node on the path,  $g(n)$  is the cost of the path from the starting node to node  $n$ , and  $h(n)$  is a heuristic function estimating the smallest cost required from the  $n$ th node to the destination node. The A\* algorithm terminates once it discovers the lowest cost path that connects the starting node to the destination node, or when there is no path possible. The Euclidean Distance is our choice of heuristic for our algorithm because it is quickly computed.

In order to customize the A\* algorithm, we had the algorithm consider the existence of breakable walls, generating two separate searches, if needed, as illustrated in **Figure 2.2.2.1** and **Figure 2.2.2.2**.



**Figure 2.2.2.1: Pathfinding with Breakable Wall Avoided**

As shown in **Figure 2.2.2.1**, the bot first generates a path from Start position (green square) to End position (red square). This first path tries to avoid as many obstacles as possible which include both breakable and unbreakable walls. If the first path is found, the bot is not required to place any bombs in order to reach its destination and does not generate a second path.

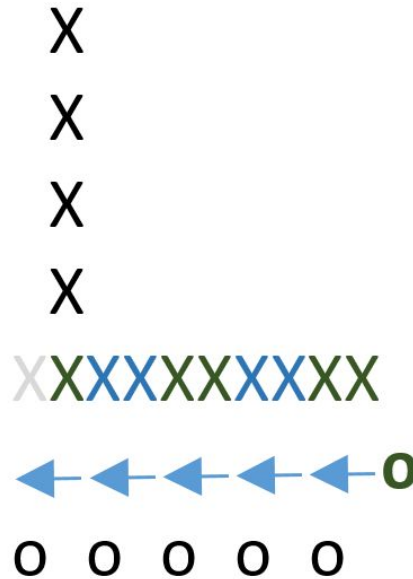


**Figure 2.2.2.2: Pathfinding with Breakable Wall Considered**

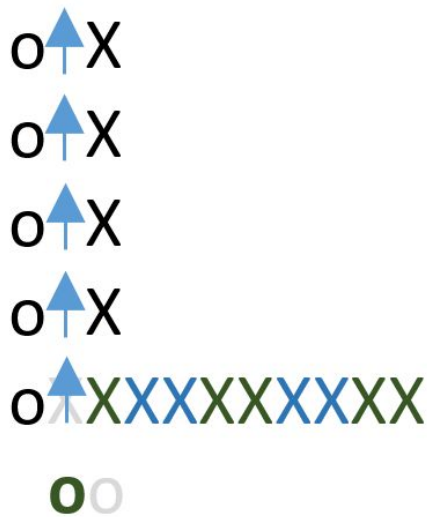
In the scenario where there is no direct path from Start to End, the bot generates a path that may go through breakable walls. Specifically, the algorithm treats breakable walls as if they were unobstructed spaces, allowing free travel. As illustrated in **Figure 2.2.2.2**, a direct path can be generated from Start to End, with the path passing through a breakable wall. The bot travels along its path and is halted by a wall. The bot proceeds to place a bomb at the interrupted location (shown as an orange circle) and switches to Flight mode, which calculates a path and brings the bot to safety (shown as blue arrows). This scenario is common in *Detonicon* where the bot may be surrounded by breakable walls.

After creating a path, there is still an additional step that is required in order to make the bot move along the path seamlessly. In *Detonicon*, each horizontal movement is a 2-unit shift, whereas each vertical movement is a 1-unit shift. Owing to the ASCII nature of the Dragonfly engine, units in the horizontal direction are closer together than the units in the vertical direction (kerning and line spacing, respectively). *Detonicon's* movement design creates a visual

impression that speed is uniform in any direction. However, since our pathfinding algorithm creates a path that is a list of adjacent coordinates, we have to perform an additional step of “path smoothing” so that our bot can understand the path information and move accordingly.



**Figure 2.2.2.3: Path Smoothing in Horizontal Direction**



**Figure 2.2.2.4: Path Smoothing in Vertical Direction**

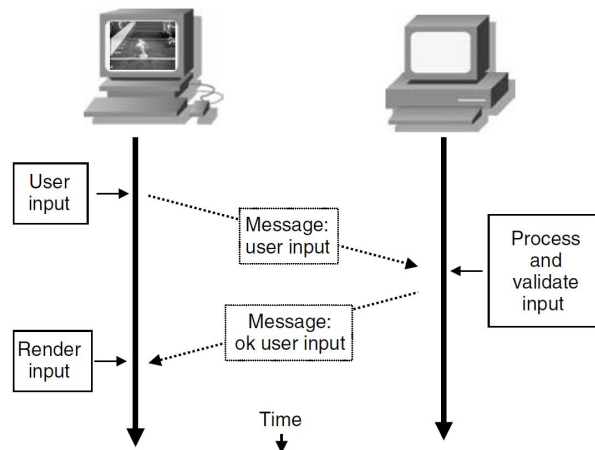
Path Smoothing is a process where the path generated from the pathfinding algorithm is interpreted as a list of “directions” instead of a list of coordinates. **Figure 2.2.2.3** shows an example of path smoothing when moving in a horizontal direction, where X represents the



coordinates within the path and O represents the location of the bot after a move has been made. Basically, the bot compares its current location with the next location on the path coordinate list to determine the direction that the bot should move. If the next coordinate in the path is the same as its new location after the move is made, that coordinate is skipped. This allows the bot to be able to keep up with the generated path and move along the path smoothly. In the case where there is a change in direction, as shown in **Figure 2.2.2.4**, the bot compares its location with the current path coordinate and turns accordingly. This ensures the bot does not move backwards and continues on its path accordingly.

## 2.3 Latency

Latency is the time required for a signal to go from a sender to a receiver and back (the round trip time). As shown in **Figure 2.3.1**, the client on the left does not render input until after receiving a server (shown on the right) ok response, which takes some time (latency). As a result, gamers who experience high latency can see their games stutter, have low frame rate, and perform poorly.



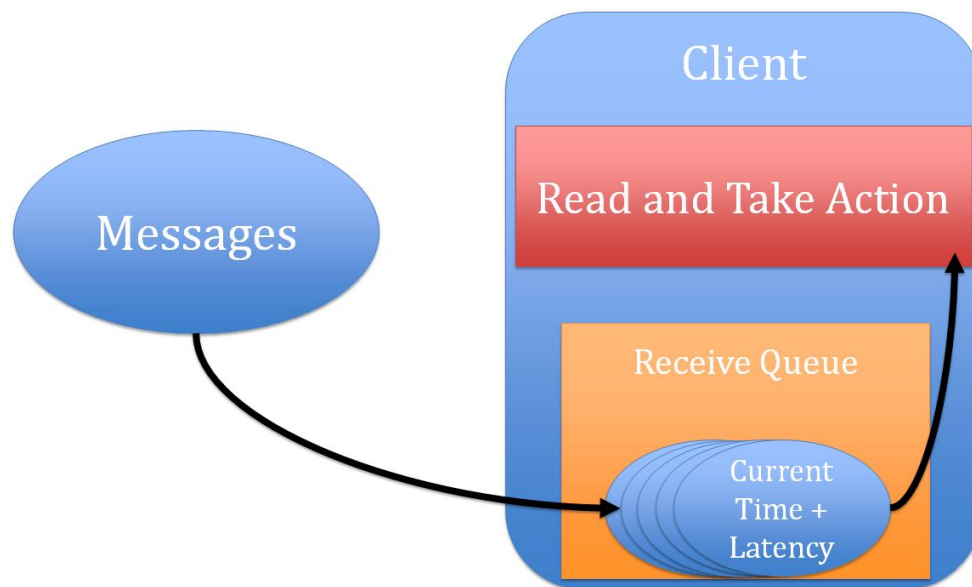
**Figure 2.3.1: Latency Summarized [2]**

Our game Detonicon relies on a Client-Server architecture. First, the server starts up the game and waits. Each player runs a separate client that connects to the started server. Once all the clients have connected and are ready to play, the server begins the game. The server is authoritative, meaning all actions that have any effect on the outcome of the game must be

approved by and happen on the server. A client sends messages to the server to request an action, such as moving the player. The server takes in client request messages and updates the game. The server then sends messages to the clients to update their game world. Finally, the clients render the world based on their copy of the world. A client experiencing high latency takes longer to send and receive messages to and from the server than clients experiencing low latency.

### 2.3.1 Latency Simulator

In order to test the effectiveness of latency compensation in our game, we needed a way to add latency to the clients. We wanted clients to be reliably burdened with a controlled amount of latency for testing and did not want to rely on third-party applications. In addition, we wanted the latency simulator to allow for clients with different latencies on the same machine with the server to allow for easier testing. This way, our project would be a self-sufficient package that could be further developed in the future. We simulated latency with a message queue, shown in **Figure 2.3.1.1**.



**Figure 2.3.1.1: Latency Simulator Queue**

Each client is assigned a latency. Each client processes messages from the server only after their assigned latency time has passed. Messages sent from the server are captured by a *receive queue* on the client, shown in **Figure 2.3.1.1**. Each time a message is added to the *receive*

*queue*, a timestamp is noted that is the sum of the current time and the assigned latency of the client, in milliseconds. Every game loop, the *receive queue* is iterated through, and any messages that have their timestamp less than or equal to the current time are removed from the *receive queue* and processed by the client to update their game state. Latency to the server is simulated with another queue, the *send queue*, where all the messages the client sends are placed in the *send queue* and assigned a timestamp to send equal to the sum of the current time and the client's assigned latency. Every game loop, the *send queue* is iterated through, and any messages that have their timestamp less than or equal to the current time are removed and sent to the server.

### 2.3.2 Latency Compensation

Some latency compensation algorithms are well known [2]. For example:

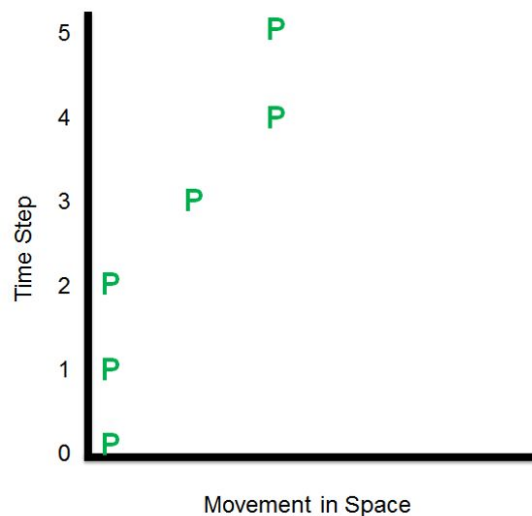
- Player Prediction: The client predicts what player actions the server will allow.
- Opponent Prediction (Dead Reckoning): The client predicts opponent actions the server will provide.
- Time Delay: The server waits to process requests until all clients are ready.
- Time Warp: The server rolls back time, applies a previous client request, and rolls time forward again.

We did not implement opponent prediction as predictions are usually determined using a directional velocity, and our game has no concept of momentum - opponents can just change direction with no penalty. Time delay was also not selected so as not to add a handicap to clients with no latency. Time warp implementation would be out of scope for our project. In the end, we implemented player prediction for the movement action in *Detonicon*.

Without player prediction latency compensation in *Detonicon*, movement in *Detonicon* is as follows:

1. The player (client) inputs a movement request, for example, by pressing 'W' for up
2. The client packages up the movement request and sends it to the server
3. The request waits (depending on the client's assigned latency) in the *send queue* before being sent to the server
4. The server receives the movement request and either approves or rejects it

5. If the request is approved, the server sends a message back to all clients to move the appropriate avatar on their screen
6. The message waits (depending on each client's assigned latency) in each client's *receive queue* before being processed
7. Each client processes the message and moves the character upwards on the screen
8. If the server rejected the request, the client's character simply does not move with the 'W' input command



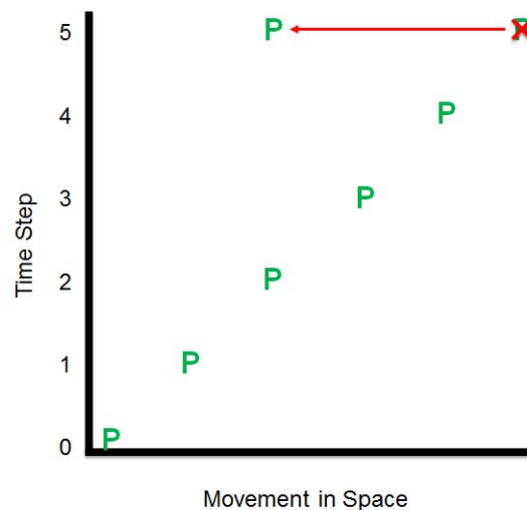
**Figure 2.3.2.1: Client Movement Without Latency Compensation**

**Figure 2.3.2.1** shows a representation of a client moving without latency compensation. The x-axis represents the character's increasing movement in space after inputting a movement command. The y-axis represents increasing time. The 'P' marks the character. The graphic shows that the client's character does not move immediately after inputting a movement command. Instead, the client must wait for the server to approve the movement command and then the character moves through space as shown in the graphic.

With player prediction latency compensation, movement is as follows:

1. The player (client) inputs a movement request, for example, 'W'
2. Assuming that move was valid, the player's avatar moves upwards on the client's screen
3. The client packages up the movement request and sends it to the server
4. The request waits (depending on the client's assigned latency) in the *send queue* before being sent to the server

5. The server receives the movement request and either approves or rejects it
6. If the request is approved, the server sends a message to all other client's to move the character up on their screen
7. The message waits (depending on each client's assigned latency) in each client's *receive queue* before being processed
8. If the request is rejected (for example, trying to move into an occupied space that the client does not know of because it is behind due to latency), the server sends a reject message back to the client, containing the last valid position for that character on the server. The server then ignores all messages from that client until a movement request originates from the last valid position
9. Once the client gets a message of rejection popped from the *receive queue*, the controlled character is rebounded (teleported) back to the last valid position the server packaged in the message



**Figure 2.3.2.2: Client Movement Using Latency Compensation**

**Figure 2.3.2.2** shows a representation of a client moving using latency compensation. The x-axis represents the character's movement in space after inputting a movement command. The y-axis represents increasing time. The 'P' marks the character. The graphic shows that the client's character moves immediately after inputting a movement command. The client does not wait for the server to approve the move. However, in cases where the movement request is

rejected, the player is rebounded back to a prior position (indicated in the figure by the red 'x'). While feeling responsive, this avatar rebounding may be visually jarring in some cases.

## Chapter 3: Evaluation

Evaluation of the effects of latency compensation was done in two stages. The first stage gathered data from bots and the second stage gathered data from human participants.

### 3.1 Bot Study Procedure

The bot study was done to gather objective statistics of *Detonicon* gameplay in order to understand how players perform with latency compensation. It involved a group of bots repeatedly playing the game, where one bot was under a combination of latency and maybe latency compensation, and the other bots all had zero latency. At match end, statistics from the game were recorded in a log file for each client in the game. A table of statistics printed to a client's log file is detailed in **Table 3.1.1**.

**Table 3.1.1: Detonicon Client Statistics**

| Statistic                 | Summary of Statistic  |
|---------------------------|---|
| Match ID                  | The ID of the match that was just played  |
| Lag (MS)                  | How much latency the client was assigned to simulate. Round trip time would be Lag (MS) multiplied by 2 |
| # of Clients              | The number of clients the server serviced during the match  |
| Is Using Lag Compensation | Either true or false, whether or not the client was utilizing latency compensation                      |
| Client Socket ID          | The assigned socket ID that uniquely identifies a client  |
| Class                     | Either Bot or Player, whether or not the client was played by AI or a human                             |
| Brave Value               | Used to determine bot aggressiveness, -1 if the client is a player                                      |
| Smart Value               | Used to determine bot resourcefulness, -1 if the client is a player                                     |

|                             |   |
|-----------------------------|---|
| Map ID                      | The ID of the map the match was played on   |
| Game Length (MS)            | The length of the game in milliseconds  |
| Time Survived (MS)          | The length of time the client survived in milliseconds  |
| # of Times Hit by Explosion | The number of times the client was damaged by a bomb explosion  |
| # of Times Bumped into Wall | The number of times the client inputted a movement command that attempted to move into a wall                         |
| # of Powerups Picked Up     | The number of power-ups the client picked up  |
| # of Bombs Placed           | The number of bombs the client placed   |
| # of Spaces Moved           | The number of spaces the client moved   |
| # of Move Cmds Sent         | The number of movement commands the client sent to the server   |
| # of Times Rebounded        | The number of times the client received a reject message from the server and had to teleport back to a prior position |
| # of Spaces Rebounded       | The total distance in spaces the client teleported  |
| Is Victor                   | Either true or false, whether the client won the game or not  |



The study was performed on one machine in order to simplify logistics for an automated study. The specifications of the machine used we used is shown in **Table 3.1.2**.

**Table 3.1.2: Testing Machine Specifications**

|                  |  |
|------------------|--|
| Operating System | 64 bit Windows Embedded 8.1 Industry Pro |
| Processor        | Intel Core i5-4690K @ 3.5 GHz            |
| Memory           | 16 GB RAM                                |
| Graphics         | NVIDIA GeForce GTX 970                   |

Because the study was performed on one machine, we limited the number of bots per match to 3 to prevent overloading the computer. In order to automate the bot study, we created a script that spawns a server and the bots with one bot under a specific amount of latency (with or without latency compensation). After the game ends, the script continues for a specified number of games. We also created another script that calls the first script but changes the specified latency and whether or not to use latency compensation after the specified number of games is up.

At 1280 millisecond RTT (time to send a message from the client to the server and back) of latency, we felt the game was almost unplayable, so we kept that at the cap. The set of RTT values (all in milliseconds) we tested are 0, 80, 160, 320, 640, and 1280. Our bot study tests are shown in **Table 3.1.3**.

**Table 3.1.3: Bot Study Tests**

| Number of Runs | Lagged Bot's Assigned Latency (ms) | Using Latency Compensation |
|----------------|------------------------------------|----------------------------|
| 30             | 80                                 | Yes                        |
| 30             | 80                                 | No                         |
| 30             | 160                                | Yes                        |
| 30             | 160                                | No                         |
| 30             | 320                                | Yes                        |
| 30             | 320                                | No                         |
| 30             | 640                                | Yes                        |
| 30             | 640                                | No                         |
| 30             | 1280                               | Yes                        |
| 30             | 1280                               | No                         |

### 3.2 User Study Procedure

The user study was done to gather subjective statistics of *Detonicon* gameplay in order to understand how players feel with latency compensation. It involved a human player playing against a group of bots, where the human player was under some combination of latency and maybe latency compensation and the bots had zero latency. Like the bot study, at match end, statistics from the game were recorded to a log file for each client in the game. In addition, after each game, the user was asked to take a short survey to rate on a scale of 1 to 6 some certain aspects of the game. The study was done using two machines. One computer was set up with a script to automatically load the next game of *Detonicon* for the user. The other computer was used for the user to take the surveys. A more detailed procedure for the user study is documented below as a snippet from the Informed Consent Form given to users at the start of the study, found in **Appendix A**.

1. The participant will be welcomed into the experiment area to sit in a chair and be given the Informed Consent Form.

2. Once the participant completes the Informed Consent Form, the participant will be directed to a computer to complete a Google Survey to record the demographics of the participant. No identifying information is asked or recorded.
3. An investigator will then start the game Detonicon for the participant at a computer, leaving it on the title screen of the game.
4. An investigator will ask the participant if they have any questions, and if they do not, they will be directed to begin playing the game by pressing 'P' on the keyboard.
5. The participant will play through a game of Detonicon, which is guaranteed to take less than 4 minutes.
6. Afterward, the participant will be directed to complete a session of a Google Survey that asks about their play of the game.
7. The investigators will repeat steps 3 to 6 for several times in order to test the game with varying amounts of latency and latency compensation.
8. Once the tests have been completed, or 30 minutes have passed, whatever comes first, the participant will be thanked for their time and effort.
9. The participant leaves the experiment area.

The questions on the demographics survey taken before each user study contained a series of questions to gather the user's demographics and gaming experience. A full copy of that survey is found in **Appendix B**. The after game survey contained these four questions:

- How smooth was the game?
  - This question was intended to assess how well the game seemed to run for the player. How was their play? How did the enemies move? How did the game look?
- How responsive was the game?
  - This question was intended to assess how responsive the game seemed to players under the effects of latency and latency compensation.

- How noticeable were the visual glitches?
  - Player prediction latency compensation could sometimes rebound an avatar on the screen if the movement was rejected. This question was intended to assess how often the users noticed visual artifacts.
- How hard was the game?
  - This question was intended to assess how difficult the game seemed to players under the effects of latency and latency compensation.

The user study had the participant play 11 games. The tested latency and latency compensation was the same as in the bot study to better compare the objective and subjective data. The script used in the user study shuffled the order in which users tested the game so that each participant did not get used to their current latency. Our bot study tests are shown in **Table 3.1.4.**

**Table 3.1.4: User Study Tests**

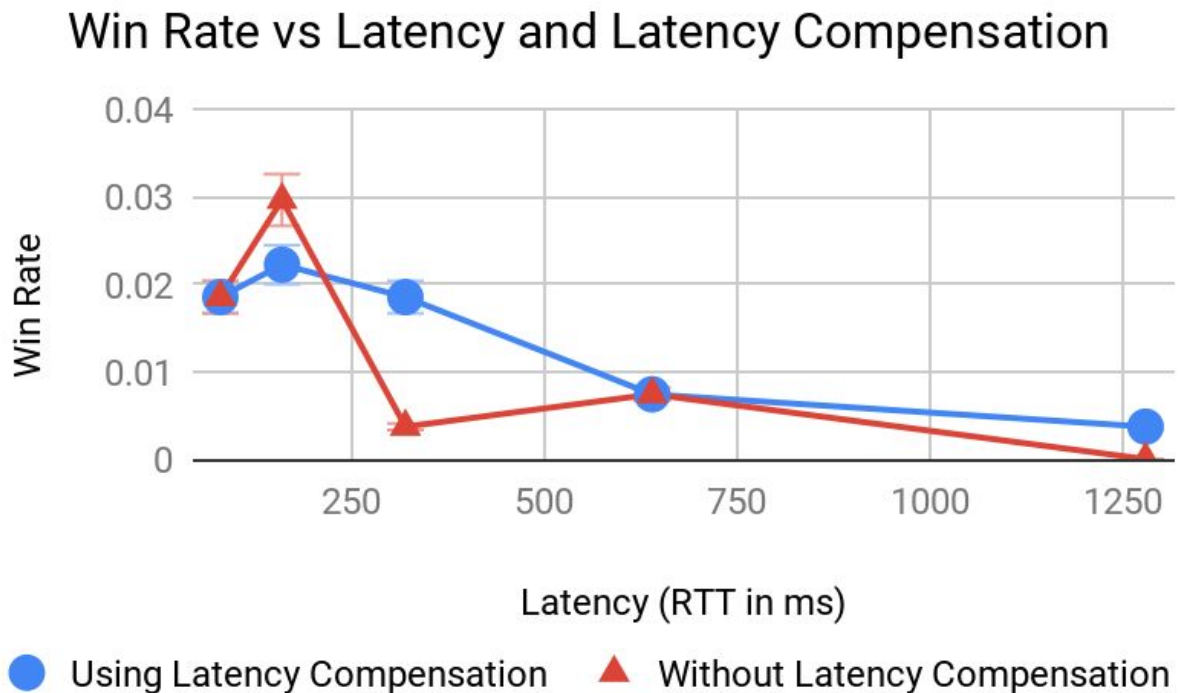
| Number of Runs | Lagged Player's Assigned Latency (ms) | Using Latency Compensation |
|----------------|---------------------------------------|----------------------------|
| 1              | 0                                     | N/A                        |
| 30             | 80                                    | Yes                        |
| 30             | 80                                    | No                         |
| 30             | 160                                   | Yes                        |
| 30             | 160                                   | No                         |
| 30             | 320                                   | Yes                        |
| 30             | 320                                   | No                         |
| 30             | 640                                   | Yes                        |
| 30             | 640                                   | No                         |
| 30             | 1280                                  | Yes                        |
| 30             | 1280                                  | No                         |

## Chapter 4: Results & Analysis

The bot study and the user study returned a lot of data. For the objective data shown in **Table 3.1.1**, scripts were used to parse through log files and then transferred to spreadsheets. For the user study surveys, the information was automatically transferred to spreadsheets using Google Forms. We found from our analysis of the data that latency compensation has an impact on both performance and experience.

### 4.1 Bot Study Results & Analysis

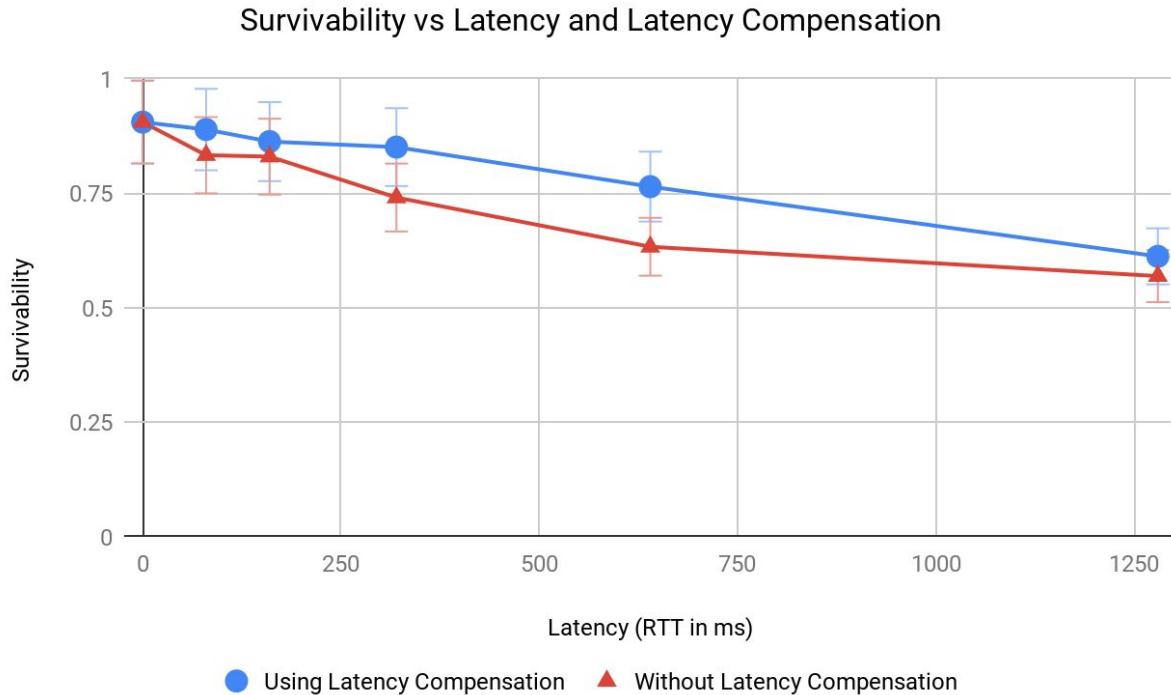
The win rate for bots is significantly skewed towards bots with no latency. 30 out of the 300 (10%) of played games were tied and not considered, but bots with no latency won about 87% (235/270) of the un-tied games during the bot study.



**Figure 4.1.1: Bot Win Rate vs Latency and Latency Compensation**

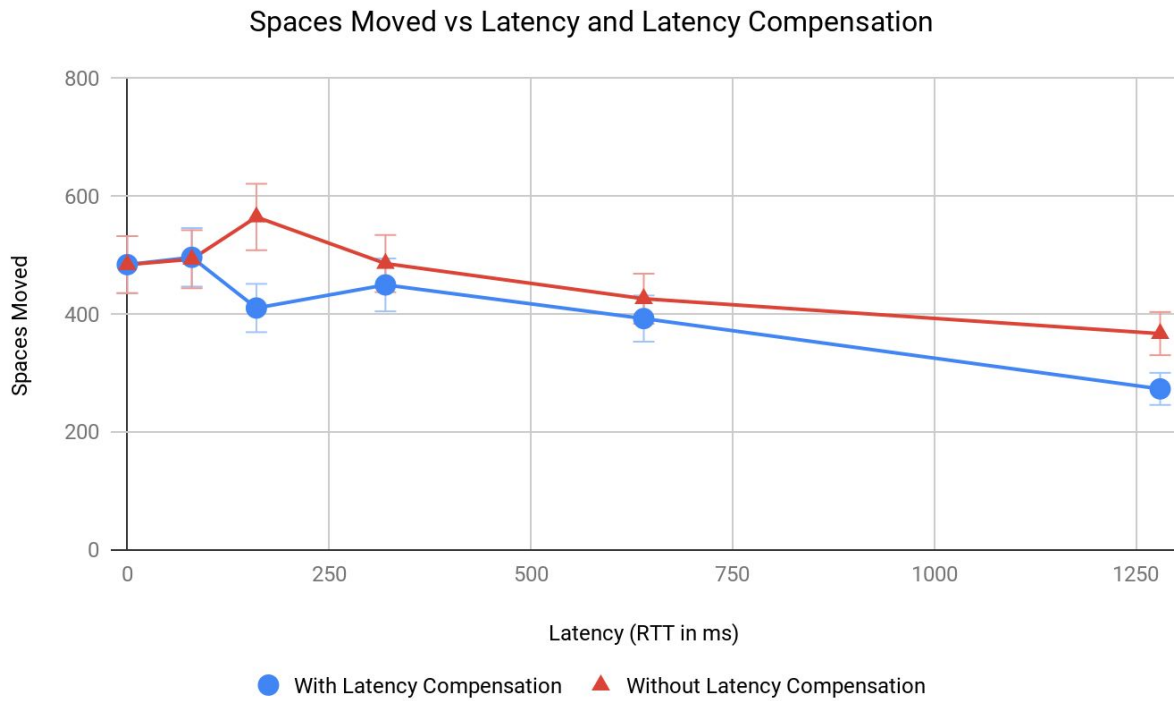
**Figure 4.1.1** shows a graph of the win rates of bots under latency with and without using latency compensation. The x-axis shows the amount of latency while the y-axis shows the win rate. The error bars show standard error. Latency is measured in milliseconds and is the RTT

(time to send a message from the client to the server and back). The figure shows a downward trend of win rate as latency increases. The maximum amount of wins of a bot under latency was 8 from 160 ms of RTT. Note, the sample size is only 35/270 or 12.9% of games. Increasing the number of games to gather more samples could be valuable in future work.

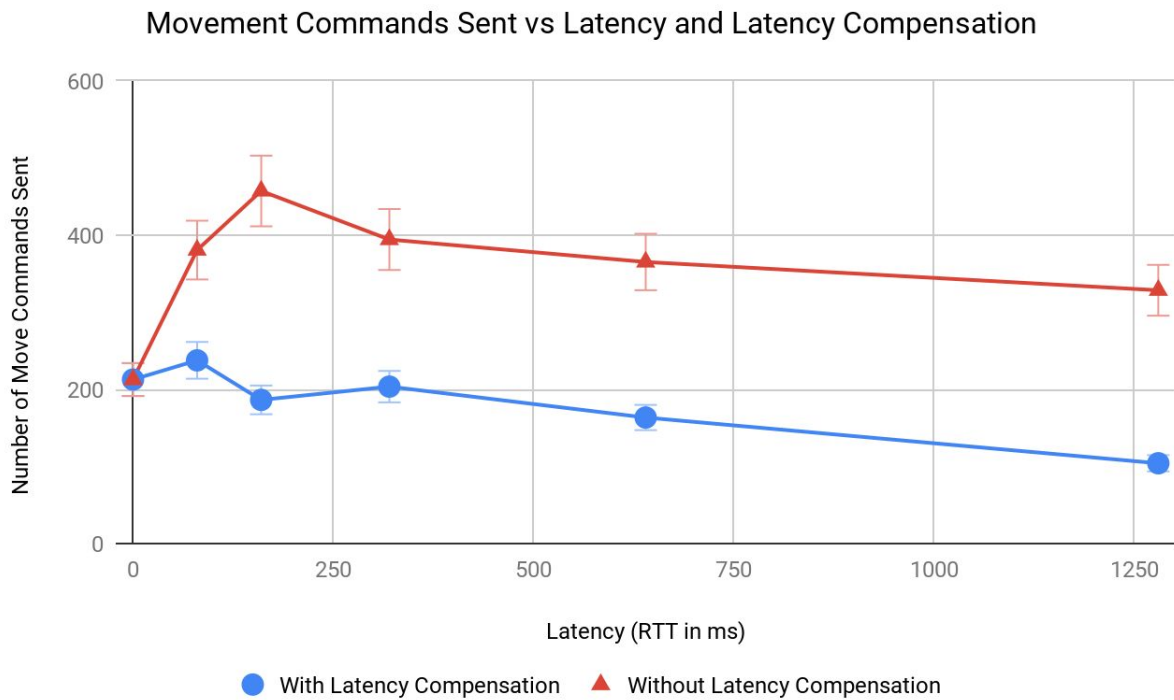


**Figure 4.1.2: Bot Survivability vs Latency and Latency Compensation**

**Figure 4.1.2** shows the average survivability of the bot versus latency and latency compensation. Survivability is a ratio of how long the bot survived over how long the game lasted. A value of 1 is the best. The x-axis shows the latency during the game while the y-axis shows the survivability. The error bars show standard error. The red line containing red triangles shows the survivability without latency compensation while the blue line containing blue circles shows the survivability with latency compensation. As shown in the figure, survivability decreases as latency increases. Furthermore, survivability while using latency compensation is consistently better than without. On average, survivability increased 9.47% while using latency compensation versus without latency compensation.



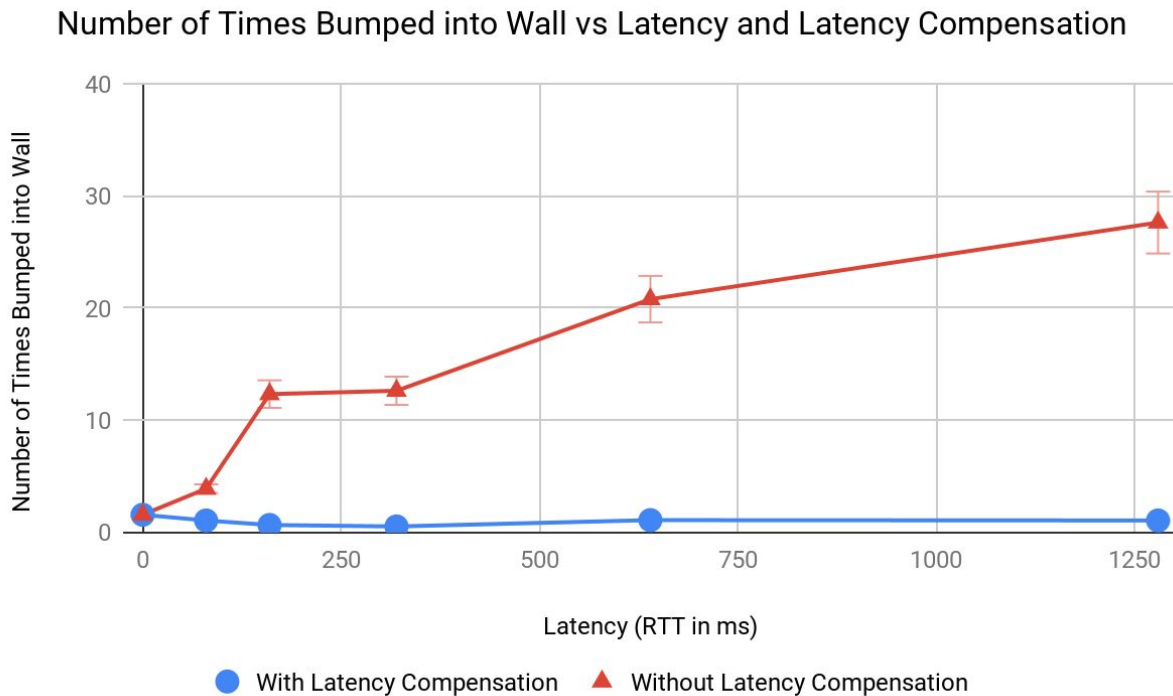
**Figure 4.1.3: Spaces Moved vs Latency and Latency Compensation**



**Figure 4.1.4: Movement Commands Sent vs Latency and Latency Compensation**

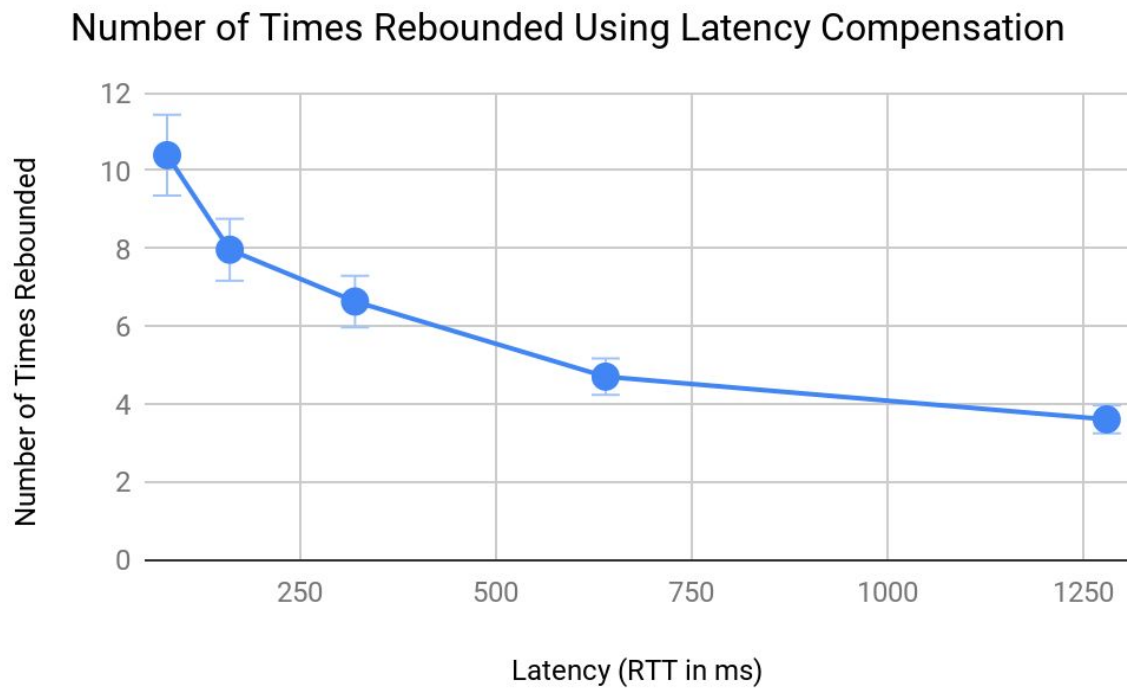
**Figure 4.1.3** shows the average number of spaces the player moved while **Figure 4.1.4** shows the average number of movement commands (requests to move) the player sent to the server over the course of each game. The x-axis of each figure shows the latency during the game. The y-axis of **Figure 4.1.3** shows the number of spaces the player moved while the y-axis of **Figure 4.1.4** shows number of movement commands the player sent. As latency increases, both number of spaces moved and number of movement commands sent decrease. The number of spaces moved and number of movement commands sent while using latency compensation is consistently lower than without. Since there is no delay to a movement input, perhaps the use of latency compensation allows for more accurate travel, which lowers number of spaces moved and number of movement commands sent. And without latency compensation, the player would spam (repeatedly send movement input) before finally moving, overshoot their desired path, and have to send additional movement commands to correct their location. This would increase the number of spaces moved and number of movement commands sent.



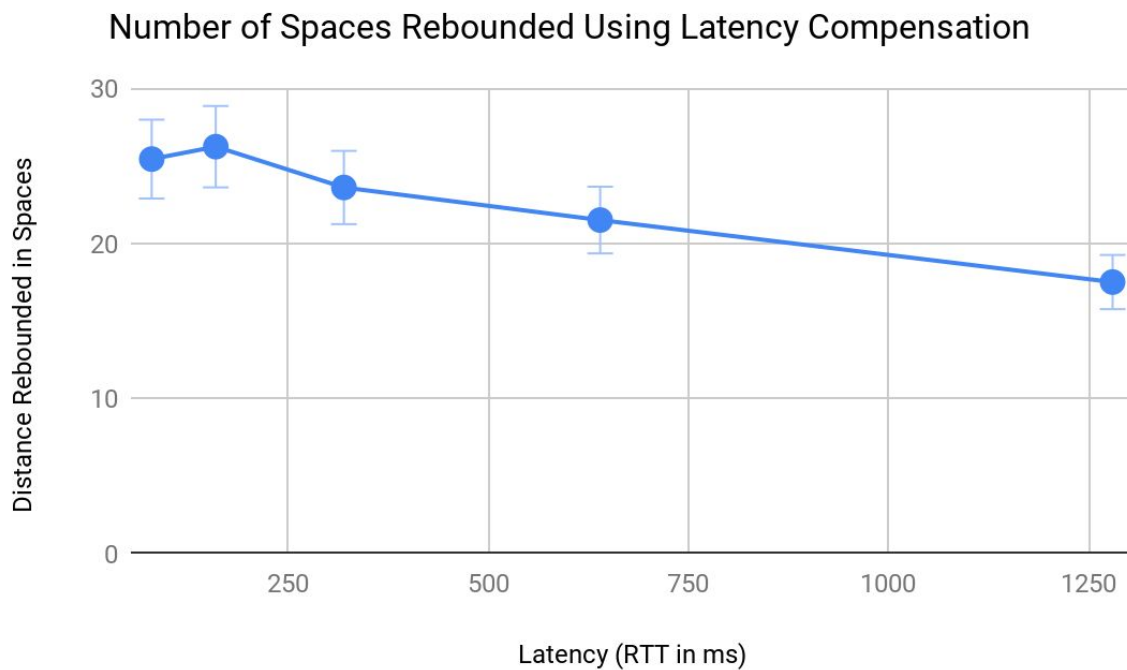


**Figure 4.1.5: Number of Bumps vs Latency and Latency Compensation**

**Figure 4.1.5** shows the number of times a player bumped into a wall versus latency and latency compensation. A player bumps into a wall when issuing a movement command that attempts to travel directly into a wall. The x-axis of the figure shows the latency during the game while the y-axis shows the number times the player bumped into a wall. Without latency compensation, the number of times the player bumps into a wall increases as latency increases. This could be due to the overshooting of a desired movement path due to the delay. With latency compensation, the number of times a player bumps into a wall is extremely stable, with an average of 0.969 times per game across all tested latencies, less than one misstep per game.

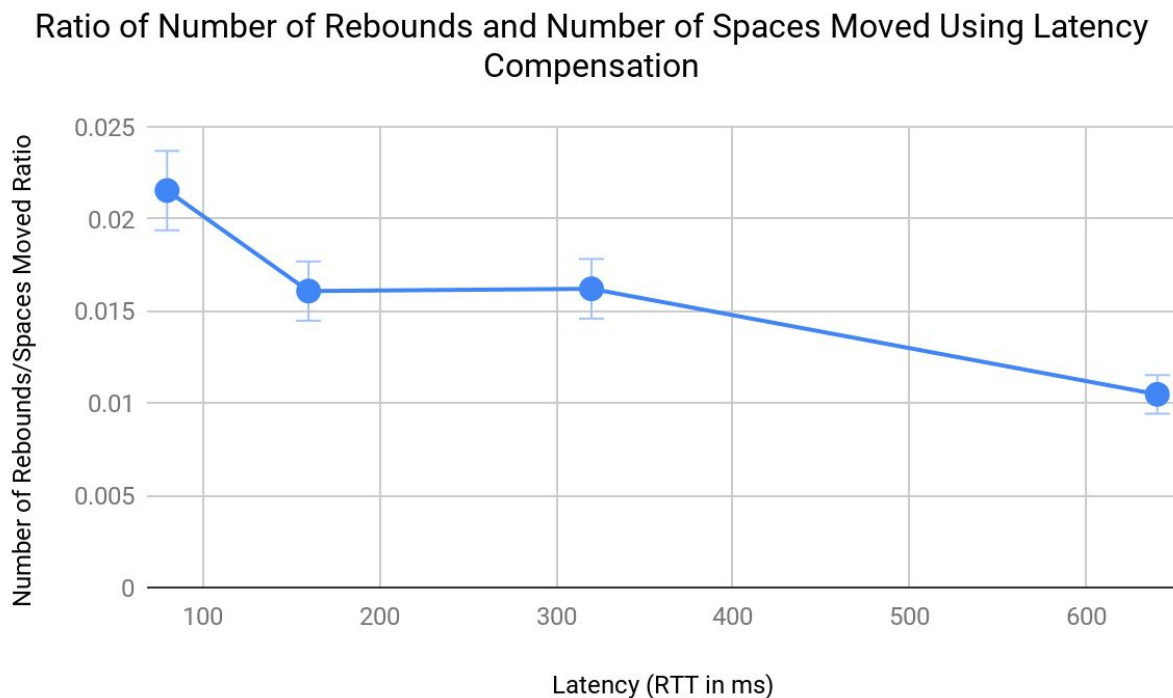


**Figure 4.1.6: Number of Rebounds While Using Latency Compensation**



**Figure 4.1.7: Distance Rebounded While Using Latency Compensation**

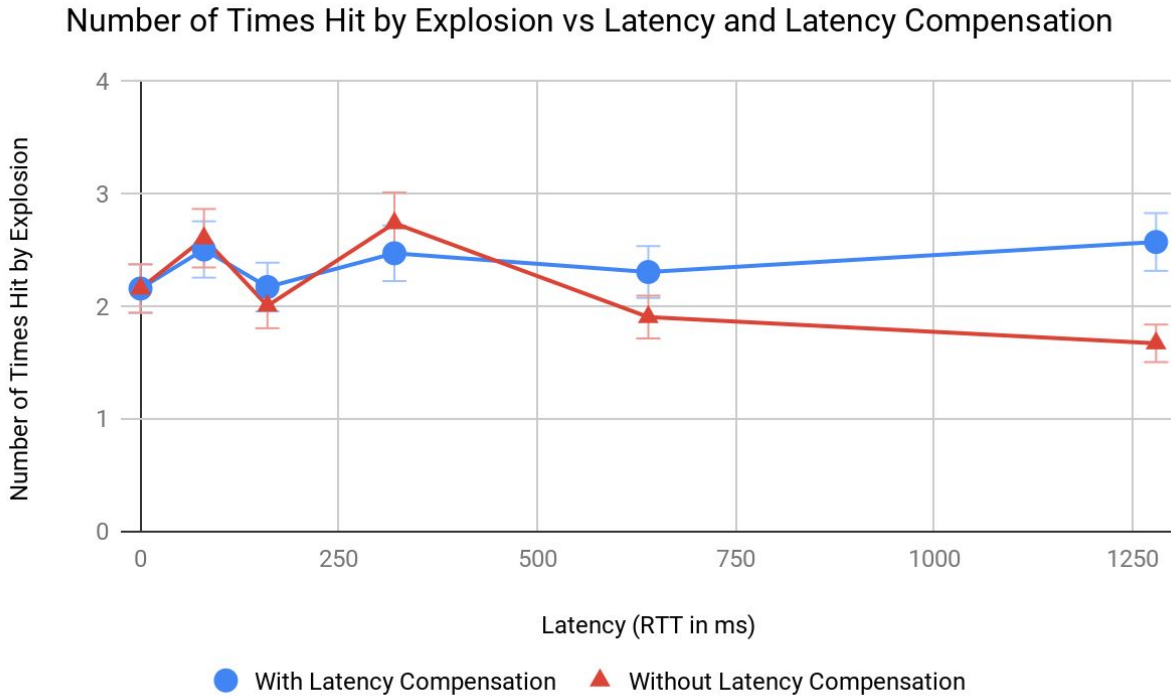
A drawback of player prediction latency compensation is the visual glitches that are experienced by the player. The player can be rebounded back to a previous position if the server rejects their movement request. On the y-axis, **Figure 4.1.6** displays the average number of times rebounded (number of times the server rejects a movement request) and **Figure 4.1.7** displays the average total distance rebounded (distance from where the player was to where the player is teleported to when rebounded) in spaces. The x-axis of both figures shows the latency during the game. Before the bot study, we assumed that the larger the delay, the more frequent and apparent the rejects should be as the client gets further out of sync with the server. However, both number of times rebounded and number of spaces rebounded trend downward as latency increases. As we saw in **Figure 4.1.3** the number of spaces travelled decreased as latency increased. With less movement means less opportunity for rebounds, and perhaps that was why rebounds decreased as latency increased.



**Figure 4.1.8: Ratio of Number of Rebounds and Number of Spaces Moved**

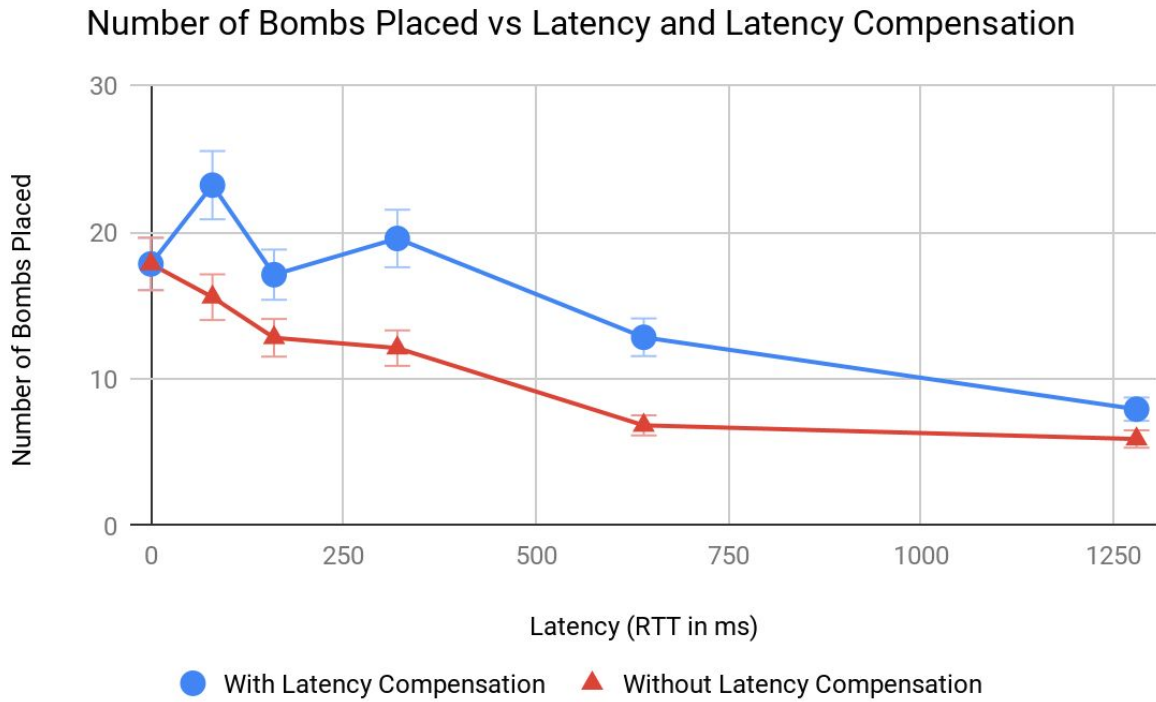
**Figure 4.1.8** displays the ratio of number of times the player was rebounded versus the number of spaces moved. The x-axis of the figure shows the latency during the game while the y-axis shows the ratio of number of rebounds to number of spaces moved. A lower ratio means

that the player was able to move more spaces before being rebounded. The figure shows that the ratio decreased as latency increased. This is consistent with our theory that as latency increases, the client has an easier time moving more spaces before getting a delayed reject message from the server and being rebounded.

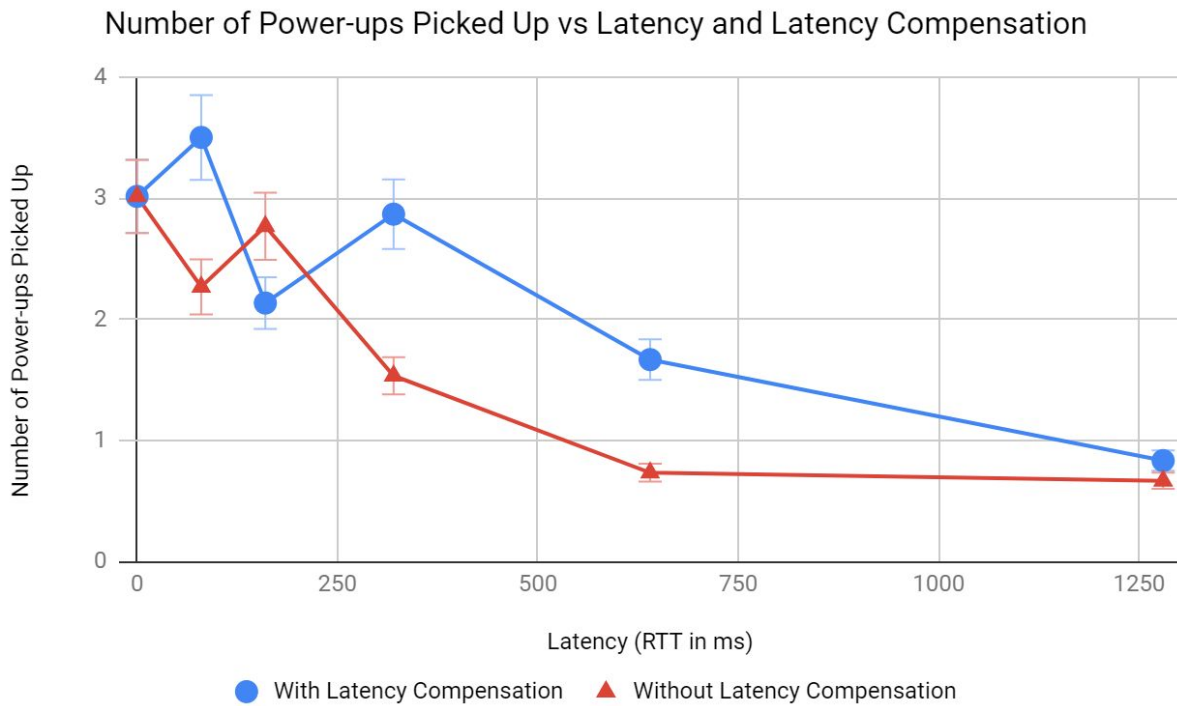


**Figure 4.1.9: Number of Times Hit by Explosion vs Latency and Latency Compensation**

**Figure 4.1.9** displays the number of times the player took damage by being hit with a bomb explosion. The x-axis of the figure shows the latency during the game while the y-axis shows the number of hits. The graph shows that with or without latency compensation, the number of times the player was hit by an explosion is pretty similar. There is also not much of a trend with how often the player gets hit as latency increases. This may be due to the low number of times a player can get hit by an explosion (only 3 times without picking up a health pack) and most games ending with all but one of the bots dying (mostly from being hit by 3 explosions). Therefore, there is no major chance for variance.



**Figure 4.1.10: Number of Bombs Placed vs Latency and Latency Compensation**



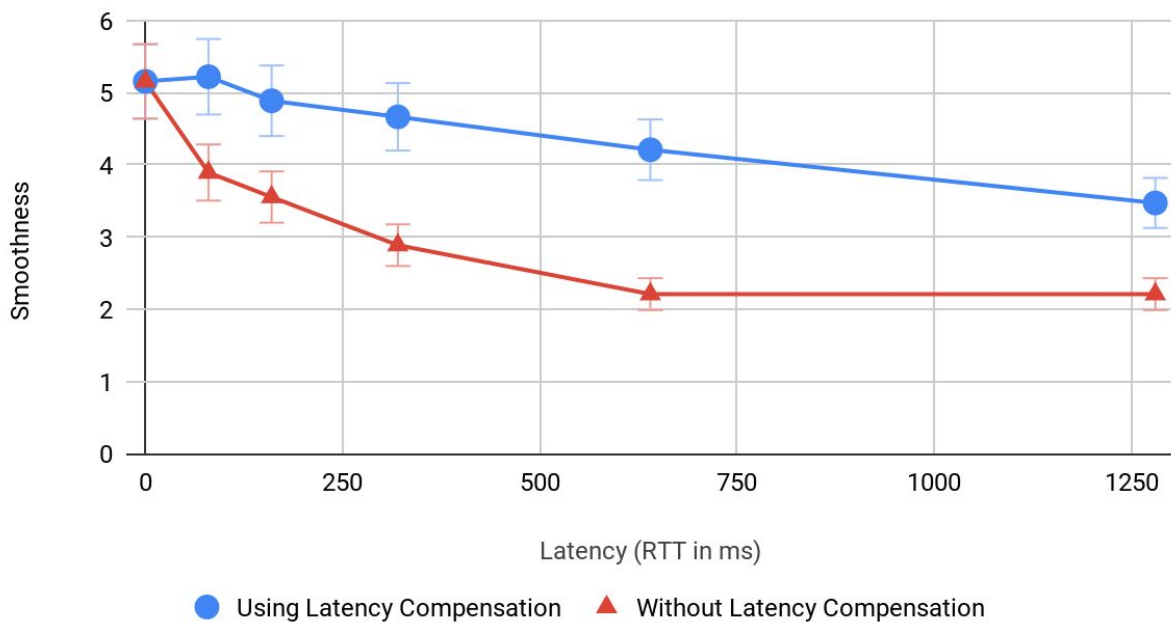
**Figure 4.1.11: Number of Power-ups Picked Up vs Latency and Latency Compensation**

**Figure 4.1.10** shows the average number of bombs placed while **Figure 4.1.11** shows the average number of power-ups picked up over the course of the game. The x-axis of both figures shows the latency during the game. The y-axis of **Figure 4.1.10** displays the number of bombs placed while **Figure 4.1.11** displays the number of power-ups picked up. Both the number of bombs placed and the number power-ups picked up show a downward trend as latency increases. This is most likely due to the player moving fewer spaces as latency increases, as shown in **Figure 4.1.3**. Moving fewer spaces means decreased likelihood of being near another player and placing a bomb. Moving fewer spaces means decreased likelihood of being near a power-up and picking it up. **Figure 4.1.3** also shows more spaces moved with latency compensation than without, and that is reflected in **Figure 4.1.10** and **Figure 4.1.11** because the player places more bombs and picks up more power-ups with latency compensation than without. The only outlier is from the 160 millisecond latency tests. Future work could explore performance at 160 milliseconds of latency.

## 4.2 User Study Results & Analysis

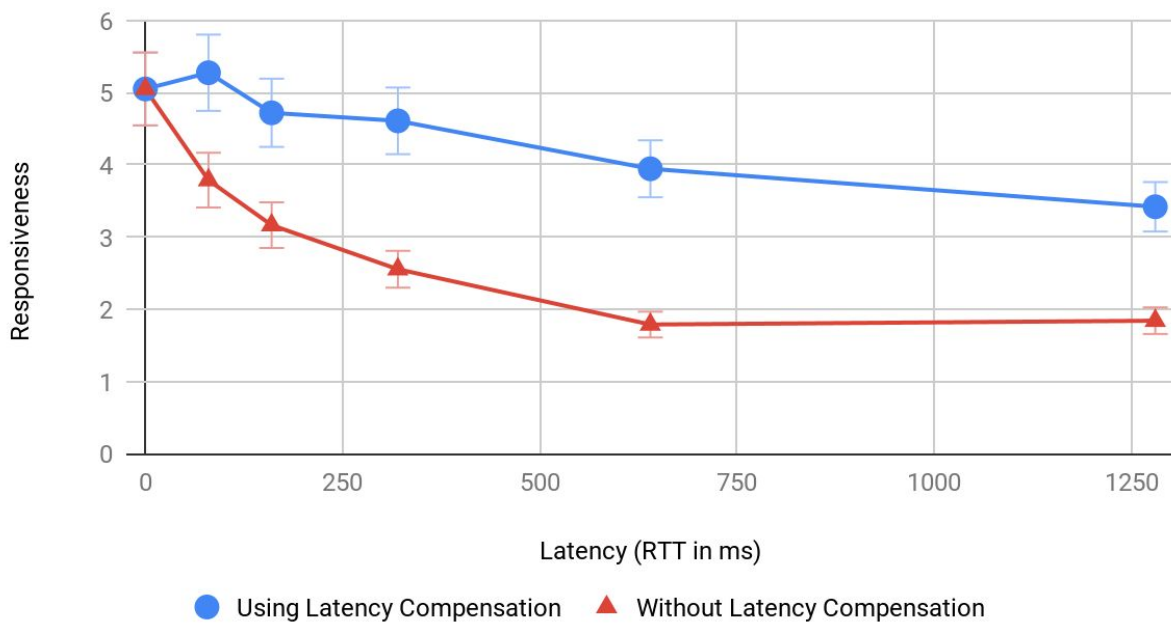
Each participant of the user study was first tasked to first complete a demographics survey before playing *Detonicon*. The participant pool was made up of a total of 19 participants. The participants were all WPI students and ranged in age from 18 to 24 with an average age of 19.8. Of the participants, 17 (89.5%) were male and 2 (10.5%) were female. The group was mostly proficient in online multiplayer games. Most were comfortable playing games with the keyboard only. The participants were familiar with a variety of game genres, but only 2 of the 19 (10.5%) had any experience with *Detonicon*'s genre, a maze-based game. A full breakdown of the demographic survey results can be found in **Appendix C**.

## Smoothness vs Latency and Latency Compensation



**Figure 4.2.1: Perceived Smoothness vs Latency and Latency Compensation**

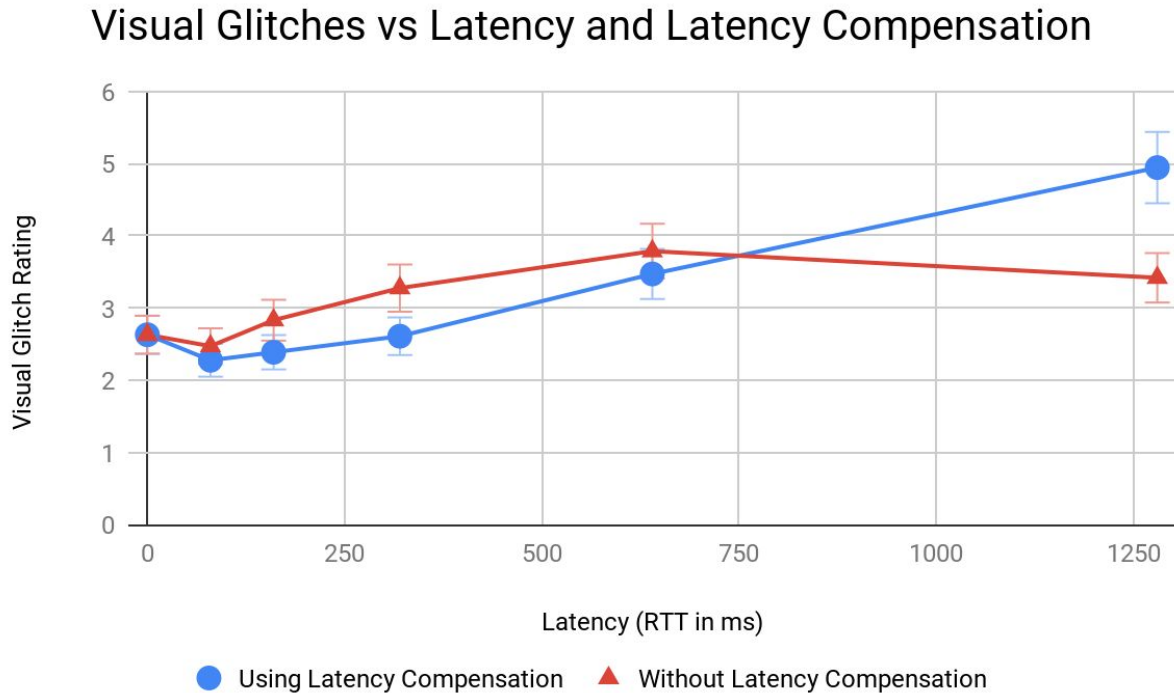
## Responsiveness vs Latency and Latency Compensation



**Figure 4.2.2: Perceived Responsiveness vs Latency and Latency Compensation**

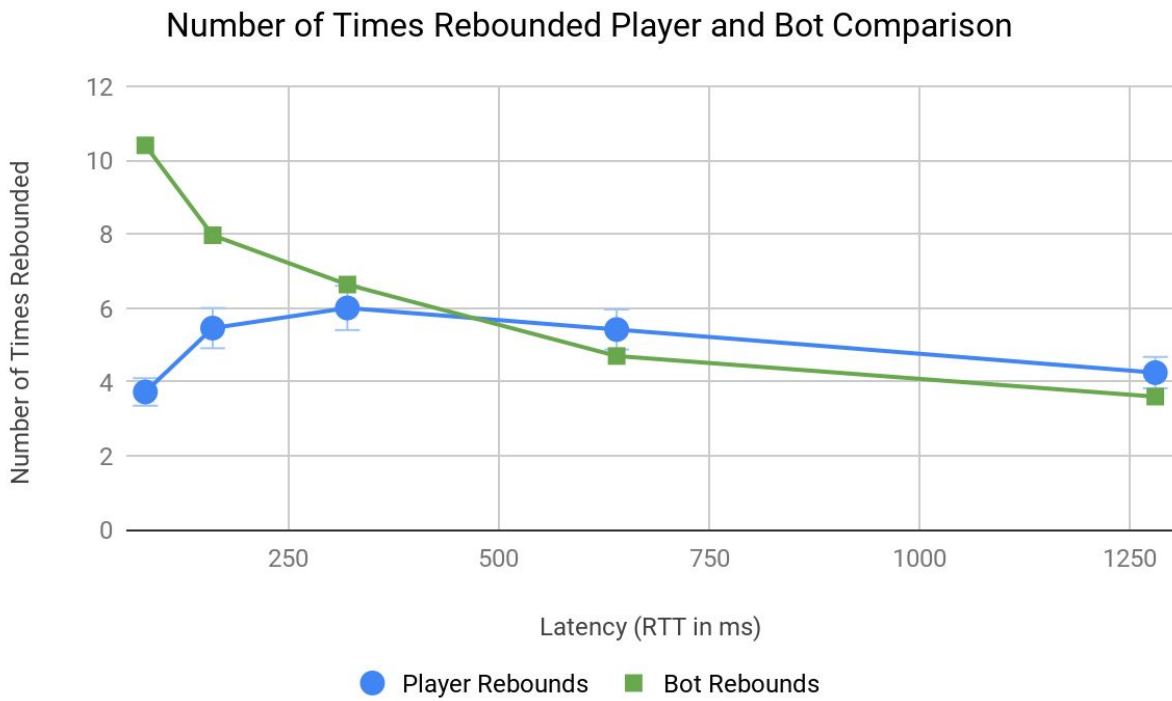
Users were asked to rate smoothness and responsiveness on a scale of 1 (low) to 6 (high) after each game. Smoothness was a description of how well the game felt to the user. Responsiveness was description of how quickly the game responded to key presses. The x-axis of **Figure 4.2.1** and **Figure 4.2.2** shows the latency during the game. The y-axis of **Figure 4.2.1** shows the user rated smoothness while the y-axis of **Figure 4.2.2** shows the user rated responsiveness. Both perceived smoothness and responsiveness decrease as latency increases. As seen in both figures, the smoothness and responsiveness while using latency compensation is consistently better than without. On average, while using latency compensation, smoothness increased 34.9% and responsiveness increased 41.3%.



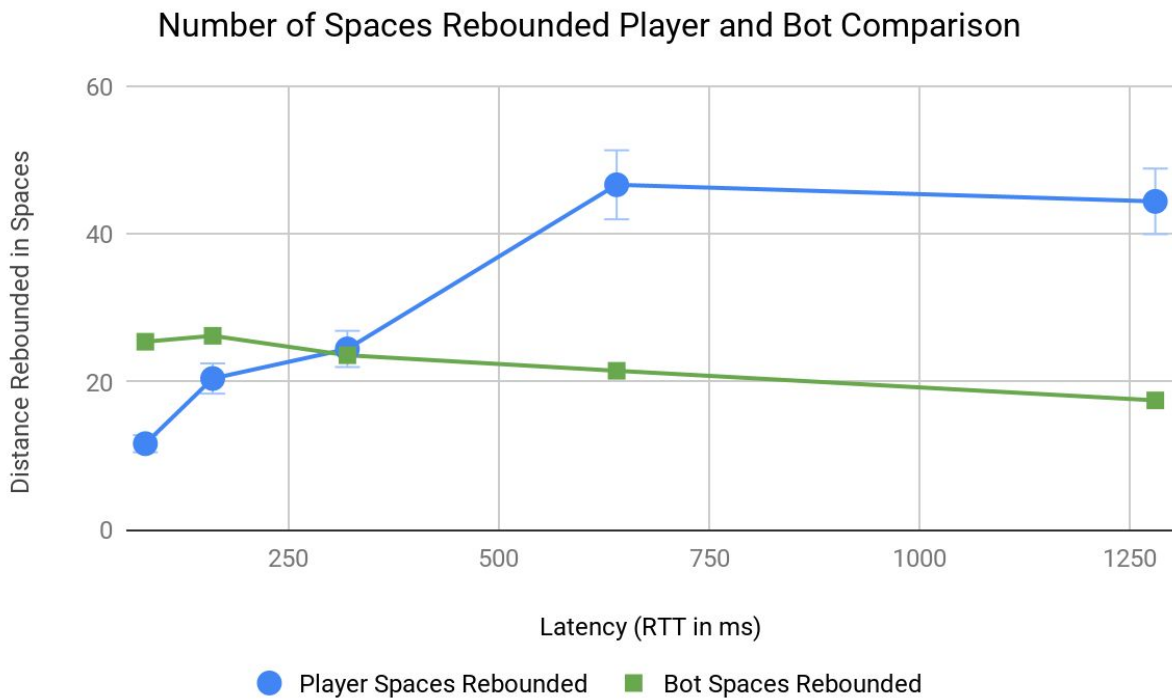


**Figure 4.2.3: Noticeability of Visual Glitches vs Latency and Latency Compensation**

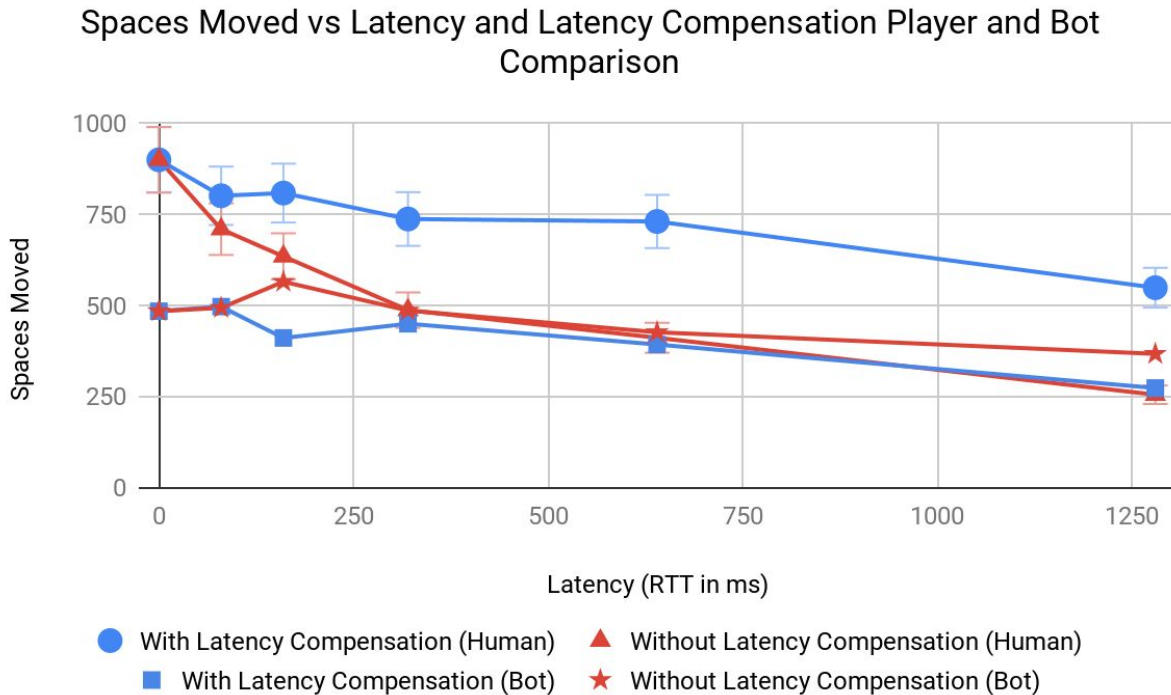
Users were asked to rate the noticeability of visual glitches on a scale of 1 (low) to 6 (high) after each game. The type of latency compensation we used (player prediction) in *Detonicon* could result in rebounding if the server rejects a player inputted move. The x-axis of **Figure 4.2.3** shows the latency during the game while the y-axis shows the user rated noticeability of visual glitches. Noticeability of visual glitches trends upward as latency increases. We hypothesized that visual glitch noticeability should increase with latency compensation because of the added rebounding from server rejects. However, users rated visual glitch noticeability higher while not using latency compensation except for the test of 1280 millisecond latency. Perhaps this is due to poor explanation of the survey question as well as the art style. Users may have assumed that an unresponsive, lagging game had more visual glitches. **Figure 4.2.1** and **Figure 4.2.2** shows that smoothness and responsiveness are consistently better with latency compensation, and perhaps that allowed users to look past any visual glitches.



**Figure 4.2.4: Number of Times Rebounded Player and Bot Comparison**

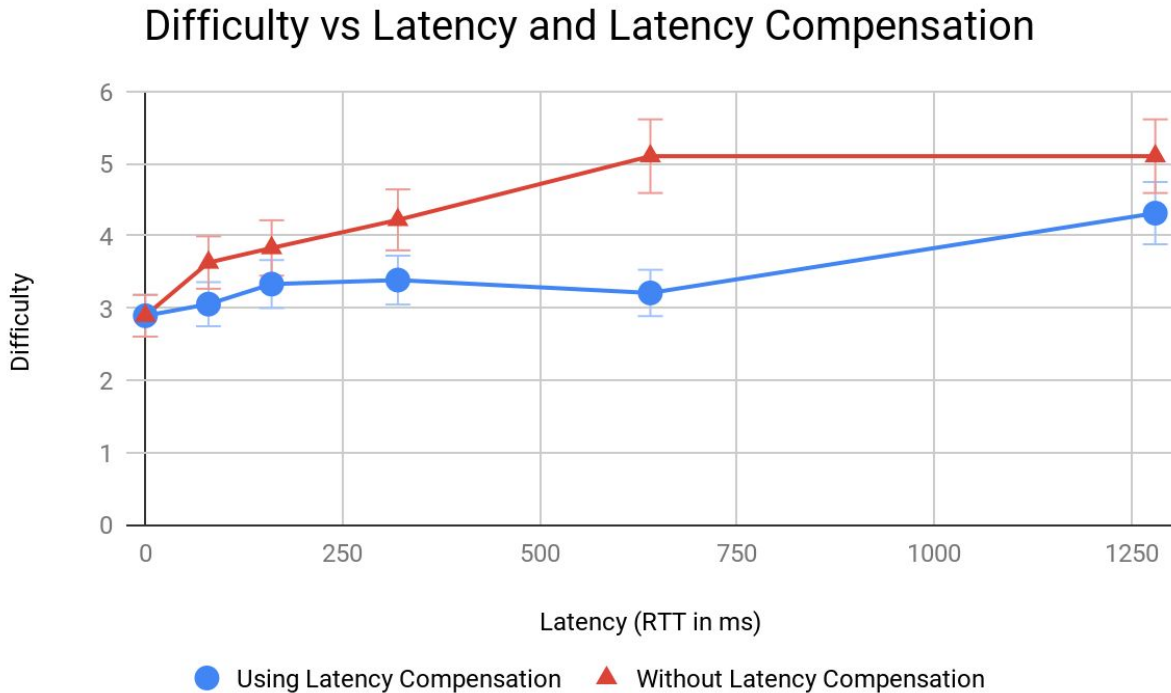


**Figure 4.2.5: Number of Spaces Rebounded Player and Bot Comparison**



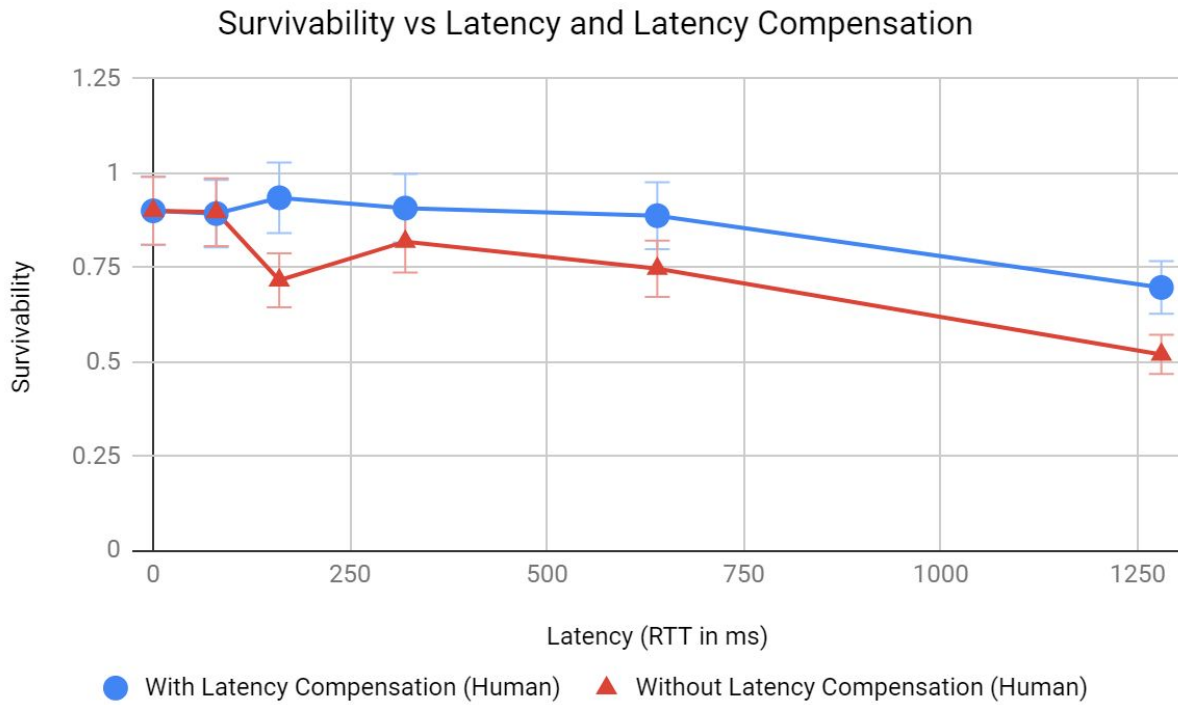
**Figure 4.2.6: Number of Spaces Moved Player and Bot Comparison**

In addition to the subjective rating, objective statistics of player performance was also recorded for each game of the study. We wanted to analyze the actual number of visual glitches for each test, so we analyzed the number of rebounds and distance rebounded shown in **Figure 4.2.4** and **Figure 4.2.5**, respectively. The figures also overlay the stats from the bot study as comparison. The number of times rebounded trends downward for users and bots to a similar level. Although the number of times rebounded for the bots starts higher than a player's at a lower latency. Perhaps the bots skirt too closely to the shrinking wall, causing more rebounds at lower latency. The number of spaces rebounded for users trends upward, to an average max much higher than that of bots. **Figure 4.2.6**, which compares bot spaces moved with that of user's, shows that players move much more than bots do under latency compensation. This might mean that bots do not explore as much on the map so their number of spaces rebounded becomes fewer than that of users as the latency grows large. **Figure 4.2.6** also shows that users move more with latency compensation than without, which is opposite that of bots. This shows that the bot's behavior may need to be tweaked further in order to more closely resemble that of human players.



**Figure 4.2.7: Perceived Difficulty vs Latency and Latency Compensation**

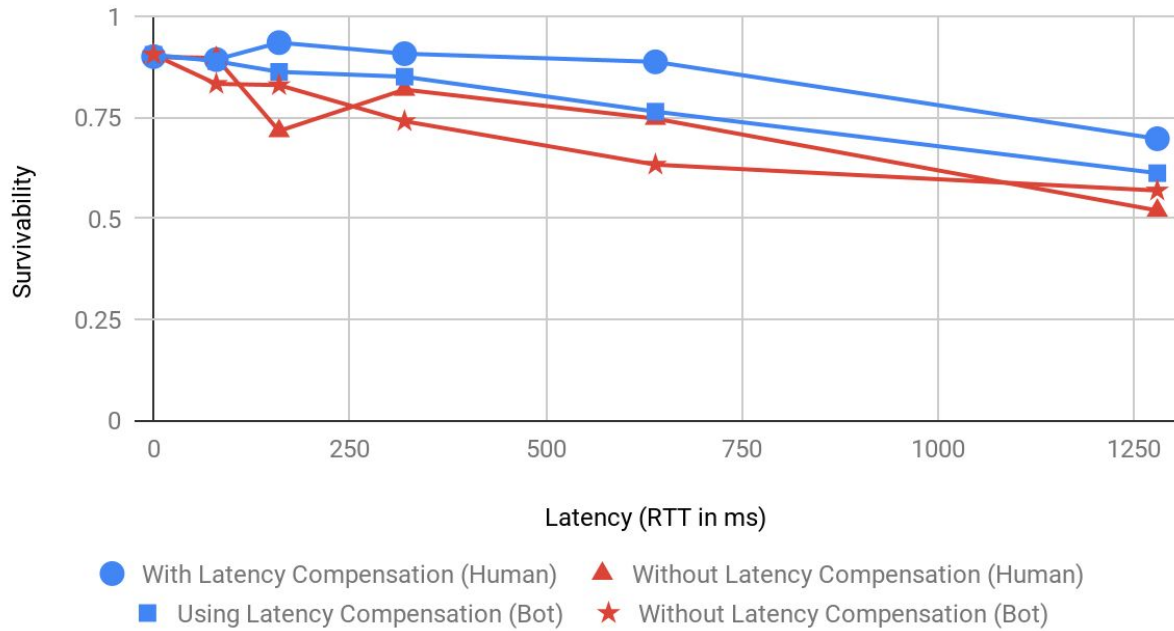
Users were asked to rate the difficulty of the game on a scale of 1 to 6 after each game. A rating of 6 is a more difficult game. The x-axis of **Figure 4.2.7** shows the latency during the game while the y-axis of **Figure 4.2.7** shows the user rated difficulty. The error bars show standard error. The red line containing red triangles show the user's rating while playing without latency compensation while the blue line containing blue circles show the user's rating while playing using latency compensation. Perceived difficulty increases as latency increases. Values of perceived difficulty while using latency compensation are consistently better than without. On average, players thought the game was 27.2% less difficult with latency compensation versus without latency compensation.



**Figure 4.2.8: Player Survivability vs Latency and Latency Compensation**

The x-axis of **Figure 4.2.8** shows the latency during the game while the y-axis shows the survivability. Survivability decreases as latency increases. Survivability with latency compensation is consistently better than without. This correlated well with **Figure 4.2.7** that showed user rated difficulty. With latency compensation, users thought the game was easier and also had greater survivability.

### Survivability vs Latency and Latency Compensation Player and Bot Comparison



**Figure 4.2.9: Player Survivability Compared with Bot Survivability**

**Figure 4.2.9** shows player survivability from the user study overlaid on top of bot survivability from the bot study. The results are very similar, with a trending decrease in survivability as latency increases and a consistent better performance with latency compensation.

## Chapter 5: Postmortem

We worked for a school year on this project, broken up into 4 terms. Over the first two terms we built *Detonicon* and added some bot AI. In the third term we implemented networking and improved bot AI. In the final term we held evaluations.

### 5.1 What Went Wrong

Our game was not properly optimized. Each wall is a separate object and makes the number of objects per game over 2 thousand. During testing we used a map with a smaller number of walls. Perhaps we could have created bigger walls to build together in order to keep the number of objects down. This also caused issues when we were implementing networking as the socket would get overflowed by synced wall messages. Moreover, bots were given a large amount of time per game loop to compute their entire path instead of a small amount of time and calculating a piece of the path per game loop. This would cause the game to stutter when the bots were in a trapped room and had to consider every single reachable space. *Detonicon* has the capability to play with 5 clients and 1 server. However, because we evaluated on one machine, we limited the number of clients to 3 instead of 5 during studies. When we played with 5 clients one machine, the computer could not keep up and the game regularly stuttered. Therefore we could not test the game at maximum capacity.

The bots did not play quite to the level of a human player. As shown in the **Chapter 4.2**, the bots results were slightly different than that of humans. Additionally, bots still had trouble in closed maps (trapped in a small space) so we ended up holding the tests on a more open map.

Originally, we planned to have our participants test the game by playing against each other in a PvP experience. However, due to logistic difficulties, we were unable to conduct the test, and decided to focus our user study on player versus bots only.

During evaluation, our game had a small chance of crashing. When the game crashed, no objective statistics were recorded in a log file. For the bot study, the script was built robust enough to continue with the next test. We later redid any crashed tests in order to obtain an equal amount of data for each run. For the user study, we also continued with the next test. However, in an effort to save time, we did not ask the participant to redo the test. In the end, 5 of the 209

games (each of the 19 participants played 11 games) played in the user study crashed. Therefore we lack 2.39% (5/209) of the potential data. The cause of the crash was a heap corruption error, and unfortunately we were not able to fix the bug.

## 5.2 What Went Well

We were able to build *Detonicon*, a complex game with over 5000 lines of code, over 40 classes, and over 30 sprites.

During the networking implementation we ran into problems with the socket overflowing. We solved the socket overflow problem by taking many objects (such as walls) off of automatic sync. The server would send custom messages to have the clients sync large amounts of data. For example, instead of sending each wall object over the network, the server just sends one message to spawn a specific map to the client.

We initially used a third party program called *clumsy* in order to simulate latency [7]. However, the application did not allow for multiple different latencies on the same machine. Additionally, since it was a third-party application, it was difficult to pull out the assigned latency and record it in the same log file with all the other statistics shown in **Table 3.1.1**. Not only did the implementation of the latency simulator work well, it also solved all of the problems from working with *clumsy*.

The user study went well. There was always enough room in the computer lab that we used to hold the study, so every participant was able to play *Detonicon* on one machine and take the survey on another. It only took around 20 minutes for each participant instead of the projected 30. Participants were positive and polite. Even if the game crashed, we were able to quickly move on to the next test and keep the user study going smoothly.



## Chapter 6: Conclusion

As long as online games exist, so will latency since it is impossible to completely remove the delay between the server and client for every player. Latency compensation algorithms do exist to ameliorate latency, but unfortunately their exact effects over a range of games and player actions are not quantified. The goal of our project is to precisely measure the effects of latency compensation with a scientific approach.

We created an online game called Detonicon that provides a platform to carry out our research. We built the ability to simulate latency directly into the game, giving us freedom from depending on third party applications to simulate latency and allowed evaluation on one machine. We implemented player prediction as a latency compensation algorithm. We implemented bots with AI behavior to play like human players to allow for automated tests. Finally, we ran two studies to evaluate the effect of latency compensation. Our bot study gathered objective data over the course of 300 games. Our user study gathered subjective data over the course of 209 games.

From our results, we found that player prediction latency compensation improves player's performance. On average, players with latency compensation survive an average of 9.47% longer (**Figure 4.1.2**). Subjectively, players have a better experience while using latency compensation. On average, players with latency compensation feel that the game is 27.2% less difficult (**Figure 4.2.7**). Players also feel the game is 34.9% smoother (**Figure 4.2.1**) and 41.3% more responsive (**Figure 4.2.2**) with latency compensation. Player prediction latency compensation can have visual artifacts. However, players actually noticed visual glitches less often when there was latency compensation (**Figure 4.2.3**).

## Chapter 7: Future Work

Future work could run more experiments. For example, **Figure 4.1.11** at 160 milliseconds of lag shows that with latency compensation, the number of power-ups picked up is fewer than without latency compensation. This effect is not shared by the other tested latencies, so more runs could potentially reveal more information. We tested only one map during evaluation, but future work could test with different maps, perhaps a map with more constraints. Our user study had 19 participants before it ended, but could be extended to gather more subjective data. In **Chapter 4.2**, the data revealed that bot results were slightly different than human results. Improving the bot AI to closer emulate the play of humans could also be valuable. Fixing the bug that crashed the game (see **Chapter 5.2**) could reduce the amount of lost data.

The addition of a different type of latency compensation such as time warp could offer a comparison to the effectiveness of player prediction latency compensation.

Although our results showed that player prediction latency compensation did improve player performance and experience, its effects might not carry over to a different type of game. *Detonicon* is a 2D maze-based game, a niche genre that does not look like popular online genres such as FPS (First Person Shooter) or MOBA (Mobile Online Battle Arena). Future work could involve building another game as an infrastructure and then testing the effects of latency compensation on it.

## References

- [1] Mark Claypool and Kajal Claypool. Latency Can Kill: Precision and Deadline in Online Games, In Proceedings of the First ACM Multimedia Systems Conference (MMSys), (Invited paper), Scottsdale, Arizona, USA, February 2010.
- [2] Armitage, Grenville, Mark Claypool, and Philip Branch. Networking and Online Games: Understanding and Engineering Multiplayer Internet Games. Chichester, England Hoboken, NJ: John Wiley & Sons, 2006. Print.
- [3] Mark Claypool. Dragonfly - Program a Game Engine from Scratch, Interactive Media and Game Development, Worcester Polytechnic Institute, 2014. Online at:  
<http://dragonfly.wpi.edu/book/>
- [4] Super Bomberman 5. Hudson Soft, 1997.
- [5] Events & Conferences. (n.d.). Retrieved April 24, 2019, from  
<https://www.wpi.edu/academics/departments/interactive-media-game-development/events-conferences>
- [6] Belwariar, R. (2018, September 07). A\* Search Algorithm. Retrieved April 24, 2019, from  
<https://www.geeksforgeeks.org/a-search-algorithm/>
- [7] Tao, C. (n.d.). Clumsy 0.2. Retrieved April 24, 2019, from <https://jagt.github.io/clumsy/>

# Appendix A

## Informed Consent Agreement for Participation in a Research Study Form

**Investigators:** Hung Hong & Antony Qin

**Contact Information:**

Hung: [hphong@wpi.edu](mailto:hphong@wpi.edu)

Antony: [aegin@wpi.edu](mailto:aegin@wpi.edu)

**Title of Research Study:** A Networked Game Utilizing AI to Study the Effects of Latency Compensation

**Sponsor:** Professor Mark Claypool

**Introduction:** You are being asked to participate in a research study. Before you agree, however, you must be fully informed about the purpose of the study, the procedures to be followed, and any benefits, risks or discomfort that you may experience as a result of your participation. This form presents information about the study so that you may make a fully informed decision regarding your participation.

**Purpose of the study:** This study attempts to learn about how specific network conditions and latency compensation affect a user's quality of experience and performance in a networked maze-based game.

**What you will be asked to do (Procedure):** In this experiment, you will play the game Detonicon several times and answer questions about your experience of the gameplay.

1. The participant will be welcomed into the experiment area to sit in a chair and be given the Informed Consent Form.
2. Once the participant completes the Informed Consent Form, the participant will be directed to a computer to complete a Google Survey to record the demographics of the participant. No identifying information is asked or recorded.
3. An investigator will then start the game Detonicon for the participant at a computer, leaving it on the title screen of the game.
4. An investigator will ask the participant if they have any questions, and if they do not, they will be directed to begin playing the game by pressing 'P' on the keyboard.
5. The participant will play through a game of Detonicon, which is guaranteed to take less than 4 minutes.
6. Afterward, the participant will be directed to complete a session of a Google Survey that asks about their play of the game.
7. The investigators will repeat steps 3 to 6 for several times in order to test the game with varying amounts of latency and latency compensation.
8. Once the tests have been completed, or 30 minutes have passed, whatever comes first, the participant will be thanked for their time and effort.
9. The participant leaves the experiment area.

**Risk and Benefits:** There are no anticipated risks beyond those encountered in everyday life if you participate in this study. There are no benefits to you in participating in this study.

**Your participation in this research is voluntary:** Taking part in this study is completely voluntary. If you choose to be in the study, you can withdraw at any time without consequences of any kind. Participants can choose to skip any questions, but if the participant does not provide enough information, their answers will be discarded. Participating in this study does not mean that you are giving up any of your legal rights.

**Your answers will be confidential:** Any report of this research that is made to the public will not include your name or any other individual information by which you could be identified. The records of this study will be kept private. Recordings will be destroyed after transcription, and records will be kept in an electronic database. The data collected in this study will be used to further understand the effects of latency compensation.

**Compensation or treatment in the event of injury:** The study will not generate more than minimal risk to the participant. Any injury that results from this study will be reported to campus, and in the event of an emergency, WPI campus police. You do not give up any of your legal rights by signing this statement.

**If you have questions or want a copy or summary of the study results:** Contact us at our email: [hphong@wpi.edu](mailto:hphong@wpi.edu) or [aeqin@wpi.edu](mailto:aeqin@wpi.edu). You will be given a copy of this form to keep for your records. If you have any questions about whether you have been treated in an illegal or unethical way, contact the IRB Chair (Professor Kent Rissmiller, Tel. 508-831-5019, Email: [kjr@wpi.edu](mailto:kjr@wpi.edu)) or the Human Protection Administrator (Gabriel Johnson, Tel. 508-831-4989, Email: [gjohnson@wpi.edu](mailto:gjohnson@wpi.edu)).

**Statement of Consent:** By signing this consent form and proceeding with the survey, you confirm that you have read the above information, and have received answers to any questions you may have. You affirm that you are 18 years of age or older, and you consent to take part in this research study.

---

Study Participant Name (Please Print)

---

Study Participant Signature

---

Date

---

Signature of Investigator who Explained this Study

---

Date



**6. Rate your ability in multiplayer online games. \****Mark only one oval.*

|        |                       |                       |                       |                       |                       |                       |        |
|--------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|--------|
|        | 1                     | 2                     | 3                     | 4                     | 5                     | 6                     |        |
| Novice | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Expert |

**7. Rate your experience with playing PC games using the keyboard only. \****Mark only one oval.*

|        |                       |                       |                       |                       |                       |                       |        |
|--------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|--------|
|        | 1                     | 2                     | 3                     | 4                     | 5                     | 6                     |        |
| Novice | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Expert |

**8. How many hours per day do you play video games? \****Mark only one oval.*

- Less than 1 hour
- 1 to 5 hours
- 6 to 10 hours
- More than 10 hours

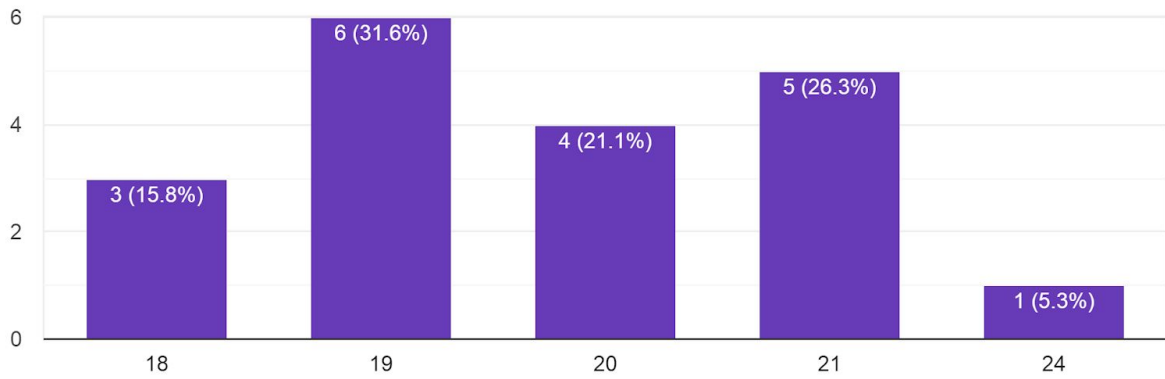
**9. Select all the game genres you play often. \****Check all that apply.*

- Real Time Strategy
- Massively Multiplayer Online
- Role-playing Game
- First Person Shooter
- Multiplayer Online Battle Arena
- Maze Game
- Other: \_\_\_\_\_

## Appendix C

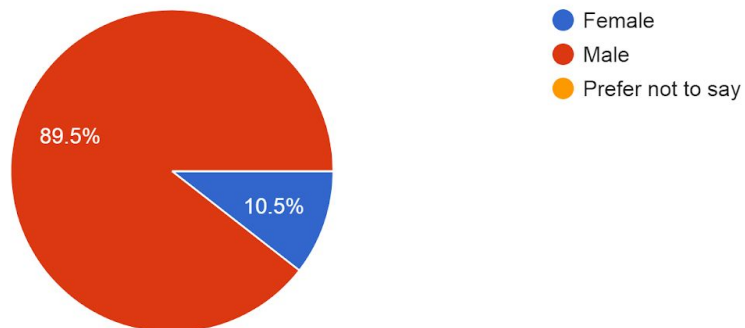
### How old are you?

19 responses



### What is your gender?

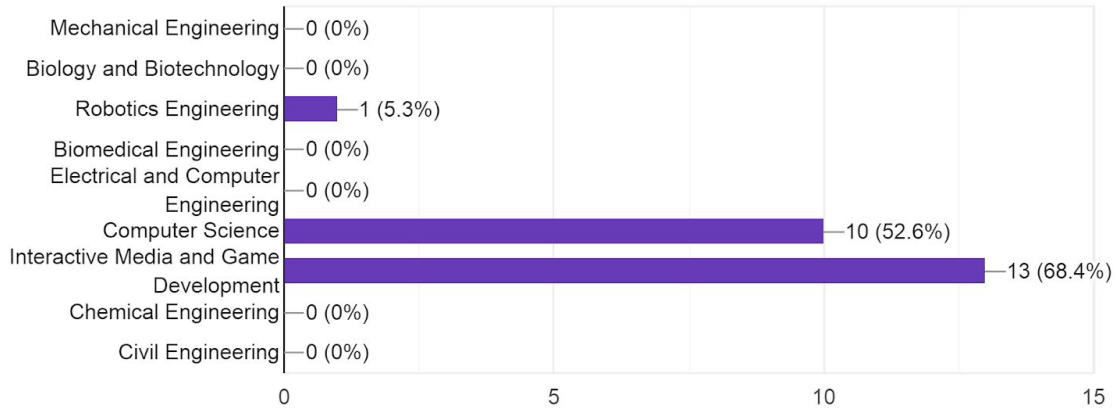
19 responses





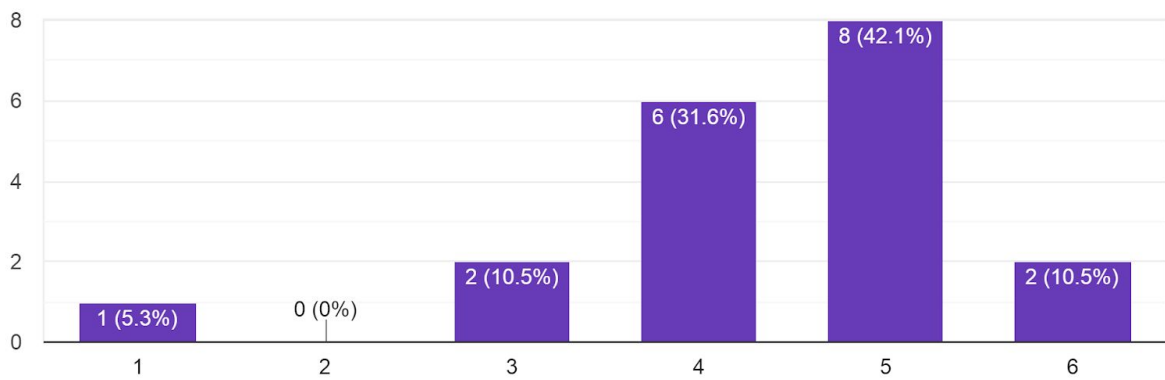
## What is your Major?

19 responses



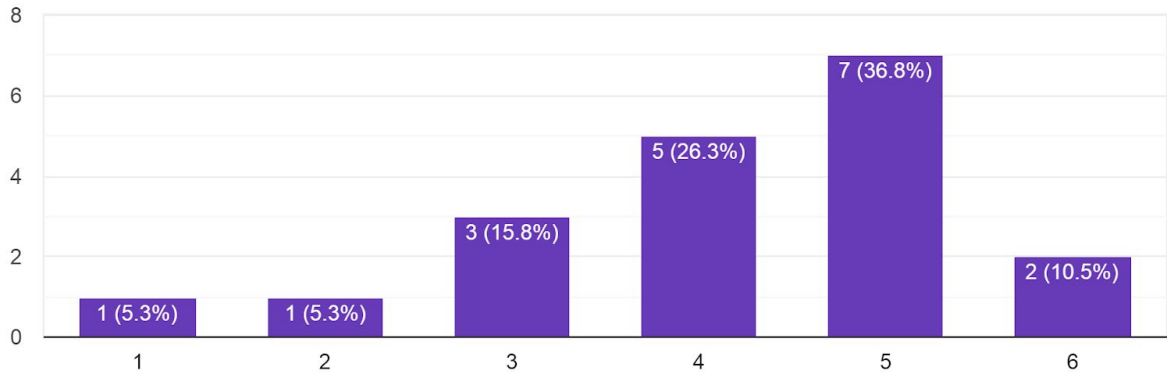
## Rate your ability as a computer gamer.

19 responses



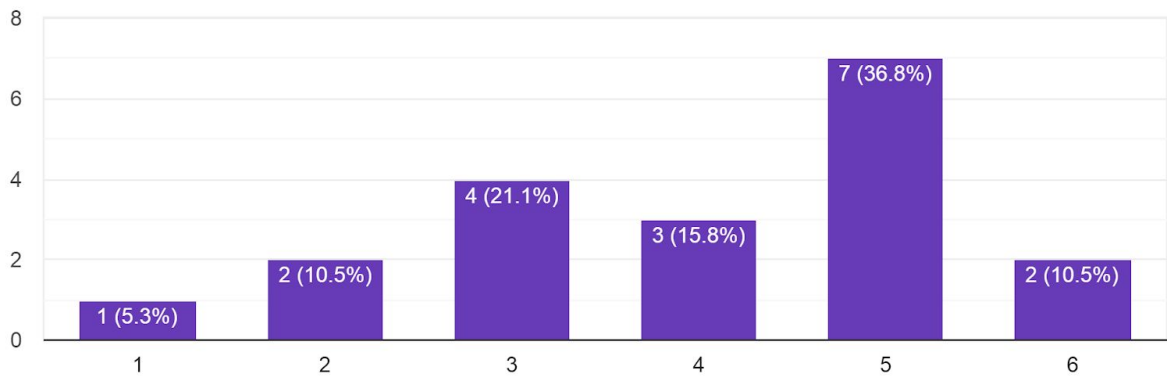
### Rate your ability in multiplayer online games.

19 responses



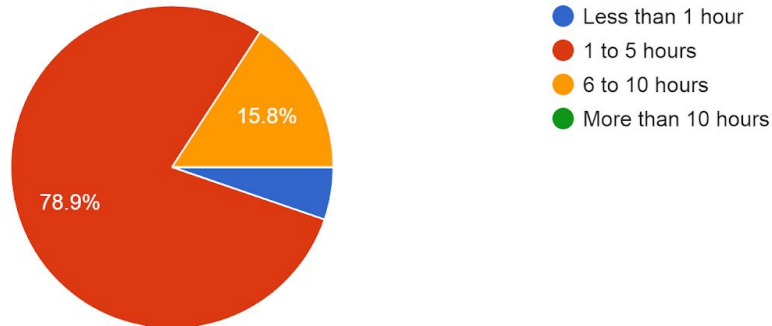
### Rate your experience with playing PC games using the keyboard only.

19 responses



## How many hours per day do you play video games?

19 responses



## Select all the game genres you play often.

19 responses

