Exploring Adaptive Time Delay as a Latency Compensation Technique in First Person Shooter Games

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

Completed by:
Benjamin Gelinas
Andrew Hariyanto
Trevor Ng
Sophia Silkaitis

Submitted to:
Mark Claypool

Special Thanks to Shengmei Liu and Samin Shahriar Tokey

Table of Contents

Table of Contents 2
Abstract 3
Section 1: Introduction 4
Section 2. Background 6
    Section 2.1 Client-Server Architecture 6
    Section 2.2: Latency 7
    Section 2.3: Impact of Latency on Fairness, Responsiveness, and Performance 8
    Section 2.4: Time Delay as a Latency Compensation Technique 10
Section 3: Methodology 13
    Section 3.1: Color Clash 13
    Section 3.2: Adaptive Time Delay Implementation 16
        3.2.1: Implementing Regular Time Delay 16
        3.2.2: Adaptive Time Delay Conditions 19
        3.2.3: Adaptive Latency Smoothing Algorithm 20
    Section 3.3: User Study 21
        3.3.1: Preparing the Study 21
        3.3.2: Round System and Delay Configurations 22
        3.3.3: Running the Study 23
        3.3.4: Data Collection 24
Section 4: Analysis 29
    Section 4.1: Data Cleaning 29
    Section 4.2: Demographics 31
    Section 4.3: Performance Results 32
    Section 4.4: Perceived Latency Results 35
    Section 4.5: Perceived Fairness 38
    Section 4.6: Summary of Results 40
Section 5: Limitations 42
    Section 5.1: Game Implementation 42
    Section 5.2: User Study 44
Section 6: Conclusion 45
Section 7: Future Work 46
    Section 7.1: Short Term 46
    Section 7.2: Medium Term 47
    Section 7.3: Long Term 48
References 49
Appendix A: Demographics Survey 51
Appendix B: Perceived Fairness vs. Accuracy 53
Appendix C: Perceived Latency vs. Accuracy 54
Appendix D: Perceived Fairness vs. Perceived Latency 55

Abstract

The gaming industry is one of the fastest growing in the world, with online games giving people more opportunities to play together. However, due to factors such as distance from servers, players also experience high latency in online games which can result in unfairness. A latency compensation technique called Time Delay reduces unfairness by synchronizing client-server communication by artificially delaying low latency players, which increases fairness at the cost of responsiveness for low latency players. Our project implements and tests Adaptive Time Delay, which aims to improve responsiveness over Time Delay yet maintain fairness by equalizing latencies only when players interact with each other. We created a two player first person shooter game to evaluate Adaptive Time Delay and conducted a user study, soliciting 38 students to play, providing performance and Quality of Experience data. Based on our analysis of the data, Adaptive Time Delay lowers players' perception of latency while maintaining performance and fairness.

Section 1: Introduction

The gaming industry is one of the fastest growing in the world, worth over $200 billion in 2022 and projected to increase to $312 billion by 2027 [1]. In the United States alone over 215 million people played video games in 2022, with 66% reporting that they played at least once a week. Of those 215 million players, 42% played shooter games, with many of them being online. Due to the nature of online games, they are susceptible to the negative effects of latency.

Latency is the delay between performing an action and the action being acknowledged and responded to by its recipients [2]. Latency in games is divided into two categories: local latency and network latency. Local latency is the delay inherent in the user's end system, whereas network latency is a delay that occurs during the process of communication between the server and clients. For the purposes of this research, we focus on network latency because it makes up most of the total latency experienced by players [3]. High latency results in a longer time for players to receive feedback on their inputs, thus leading to decreased responsiveness. This becomes a bigger issue when at least two players are involved, since the differing levels of responsiveness due to latency impact player performance and create an unfair situation.

For competitive multiplayer games, fairness is imperative. A game is fair when players have equal foundations on which they can act on to win the game and is especially important for competitive FPS games [4]. The metric for measuring fairness can be quantitative or qualitative depending on use case and target audience. In an online multiplayer setting, players have equal opportunities if the game state is consistent and responsive in each player's client, and the biggest factor of consistency and responsiveness is network latency.

Software algorithms called latency compensation techniques can limit the repercussions players face in higher latencies [5]. One such algorithm is time delay, which introduces delays to messages. This ensures that all messages are received by the server and clients at the same time. In general, the synchronization of input and output increase the overall fairness of the clients at the cost of responsiveness.

With time delay, the player with the lower latency essentially experiences additional artificial latency for the whole duration of the game. However, players are not always interacting with each other; as such constant delay limits the responsiveness for the lower latency player without much, if any, benefit for the higher latency player. The goal of this research is to implement and test a possible modification to time delay to limit its negative effects on responsiveness by turning off delay when players are not interacting with each other. This technique is called Adaptive Time Delay.

The paper is organized as follows: Chapter 2 describes background, Chapter 3 describes the methodology, Chapter 4 analyzes our results, Chapter 5 discusses limitations, Chapter 6 is our conclusion, and Chapter 7 discusses future work.

Section 2. Background

This section will discuss relevant concepts and terminologies including client-server architectures, latency and its impacts on fairness, performance, and responsiveness, as well as time delay as a latency compensation technique.

Section 2.1 Client-Server Architecture

There are three main architectures for networked games: peer-to-peer, server pool, and client-server [6]. In peer-to-peer, one player's computer connects directly to another player's computer without an intermediate game computer processing the game state. This architecture does not require additional computers to support the game, but it does however make it harder to prevent players from cheating and players with less powerful computers can degrade the experience for all. The next architecture, server pool, is where several interconnected servers communicate as peers while each player's client connects to a local server. Server pool can reduce demands on any server as the number of clients increase, but it does also offer additional complexity in sharing info about the game state and can add additional latency for clients.

The last architecture, client-server architecture, has one computer which acts as the server and relays communication between all other player computers [6]. The server holds a well known public IP address and port to make it simpler for players to connect to the game. In this architecture, the client communicates with the server by making a request to which the server fulfills through a response [7]. For example, in an FPS game, a client might send a shoot request to the server. The server validates and executes the request, and then sends the updated game state back to the client. This creates a network feedback loop for client-server communication. For the purposes of this research, the team focus is on the client-server architecture. The client-server architecture is the best fit as it is the most commonly used in games, and as such research on latency can be applied more broadly as opposed to peer-to-peer, which necessitates that both players have similar machines to have fair play. It also allows for the team flexibility to adjust the client, server, or both, compared to server pool, which puts most demands on the server.

Section 2.2: Latency

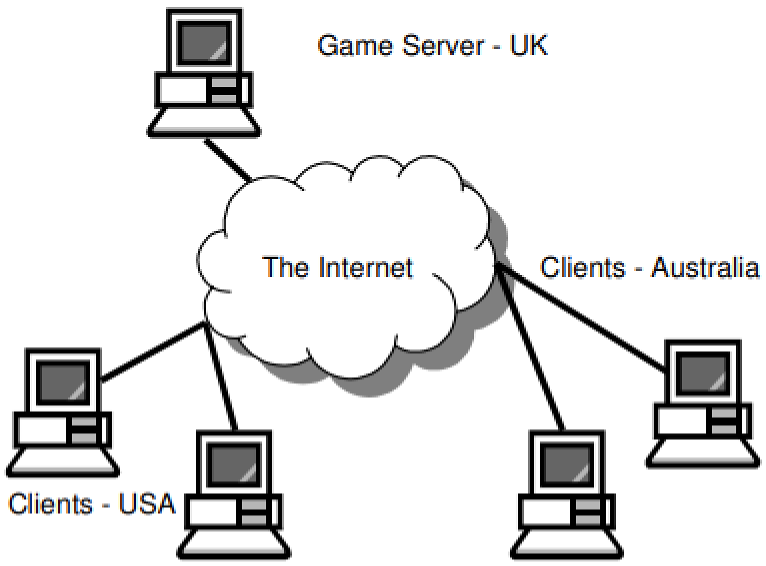


Figure 1: Clients all across the world can experience varying amounts of latency depending on where the server is located [4].

Latency is defined as the time it takes from when a player provides input until a new game state is rendered on a client [6]. In multiplayer games, latency is divided into two different categories: local latency and network latency. Local latency is the delay introduced by an individual's hardware and software, including graphics cards, CPU, keyboards, and mice. Network latency is the time it takes for client and server communications to relay to each other. This is due to the fact that the server must communicate an updated game state to connected clients based on the clients' inputs. Since network latency makes up the majority of total latency, this research focuses on network latency and will refer to network latency as just latency for the rest of the paper. Latency can occur in all parts of the client-server communication sequence, from transmitting and encoding packets to waiting in a router queue during congestion [8]. Due to many factors such as a user's physical location, type of connection, and hardware, latencies can vary from user to user, creating a situation where some users with lower latency receive the true game state quicker from the server than users with higher latency. For example, in Figure 1, game players in the United States may experience less latency than players in Australia if they are both connected to a game server located in the UK.

This imbalance affects many core components of a multiplayer FPS game, including responsiveness, fairness, player, and quality of experience (QoE). The average latency tolerance threshold for FPS games ranges between 120 ms and 150 ms [9]. Rates beyond this usually result in much poorer playing conditions that can affect the core FPS game components above, with the chance of players leaving due to network conditions increasing above 180 ms [10].

Section 2.3: Impact of Latency on Fairness, Responsiveness, and Performance

In the context of competitive online multiplayer games, the ideal scenario for all players is for each one to have equal latency when communicating with the server as described by Zander et al. (2005) [4]. Another interpretation for an even playing field for online multiplayer games emphasizes fairness as each client having network latency equal to the average network latency of all clients [11]. When clients receive different amounts of latency, there are issues of fairness. For the purposes of this research, fairness is defined as players having equal foundations on which they can act on to win the game. Fairness could be measured subjectively or objectively. An objective measurement of fairness is differences in player performance like scores and elimination rate, while a subjective measurement of fairness is player perception of how fair they think a game is.

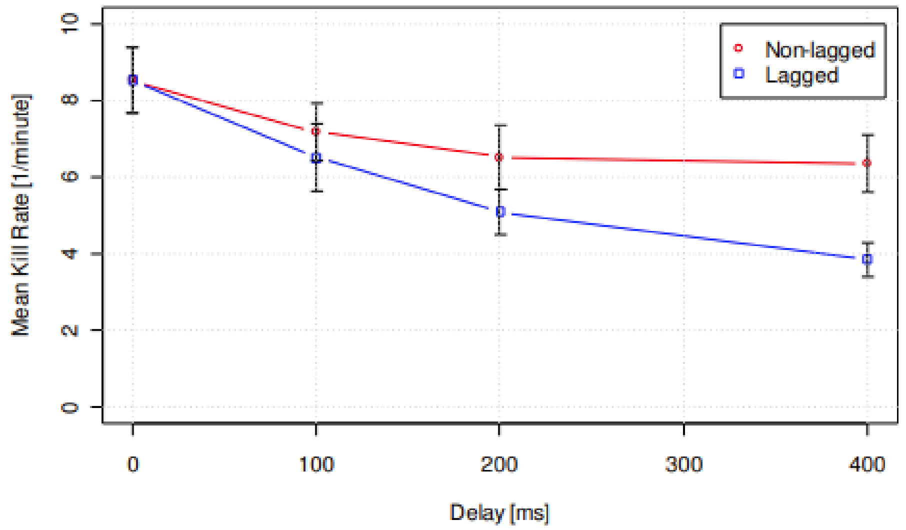


Figure 2: Effects of increased latency on fairness [4]

Figure 2 shows a graph of delay vs fairness, which is measured as Mean Kill Rate. As latency goes up for a particular user, their kill rate goes down faster than the non-lagged user [4]. This creates an unfair situation for the higher latency user, who has a more difficult time getting kills than the non-lagged user. In a hypothetical situation where client A has low network latency while client B has high network latency, client B will be at an unfair disadvantage because of a major negative effect of high latency: poor responsiveness.

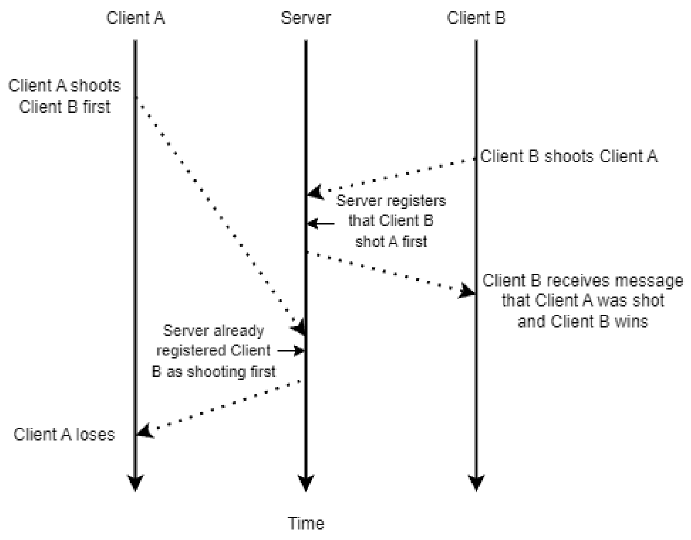

Client A
Server
Client B
Client A shoots Client B first
Client B shoots Client A
Server registers that Client B shot A first
Client B receives message that Client A was shot and Client B wins
Server already registered Client B as shooting first
Client A loses
Time

Figure 3: Unfair scenario due to reduced responsiveness

Generally, responsiveness in games is the speed in which players receive feedback from their inputs [6]. The example in Figure 3 shows how reduced responsiveness can create an unfair situation for a client with high latency. In the figure,

1. Client A has higher latency than Client B.
2. A shoots B first.
3. B shoots A sometime after.
4. However, because of A's higher latency, the message to tell the server that A shot B was received by the server after B's message that B shot A was received.
5. From the server's perspective, B shot A first.

Hence, this shows an unfair scenario for Client A as the game state in their point of view is not reflected by their inputs due to delayed feedback.

Section 2.4: Time Delay as a Latency Compensation Technique

To fix the problems due to high latency, software algorithms called latency compensation techniques are applied. Each latency compensation technique has specific target factors like responsiveness, performance, fairness, etc. that it aims to improve, usually at the expense of other factors. One technique intended to address unfairness due to latency is called time delay.

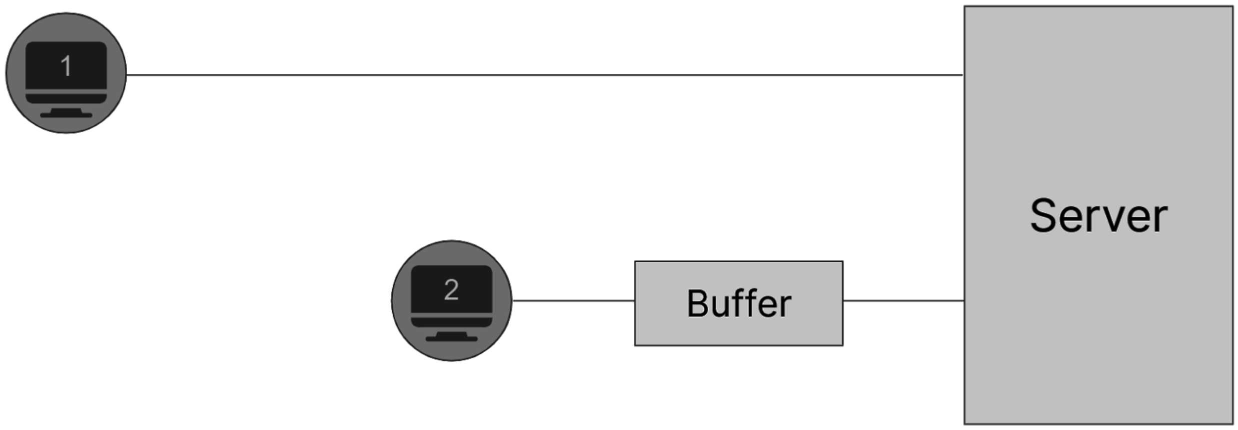


Figure 4: Time delay functionality

Time delay consists of two separate techniques that involve adding delays to sent and received messages [6]. Incoming delay is a technique that can be executed by both the client and the server. When executing an action, either the performing client or the server will introduce a buffer alongside the message. By doing so, the server is able to receive all client messages simultaneously. Outgoing delay is a server-only technique. Whenever the server plans to send a message to clients, the server adds a delay before sending the message. By adding a buffer to messages sent to lower latency clients, as in Figure 4, each client receives updates at roughly the same time.

By synchronizing the game states of each client, the server can help reduce the sense of unfairness [4]. However, time delay can introduce other problems, mainly poor responsiveness. Longer delays increase the gap between an action and feedback, which can create unfair situations for low latency players [6].

Deciding the length of each delay is a contentious point in network design. Designers must find a balance in designing around fairness and responsiveness [12]. There are several methods that are used to determine the length of the delay. One method is basing the delay on the latency of individuals with high lag. While this can create a sense of fairness for those users, basing the buffer on the highest latency or the average of latencies can result in outliers negatively impacting the experience. Alternatively, developers may opt to go for an algorithm to determine the artificial latency a player should experience. With an algorithm, designers can fine tune the experience and can account for outliers. Furthermore, to offset these issues, sometimes games introduce limits to the delays in order to strike a balance between fairness and responsiveness.

To limit the negative effects that time delay has on player responsiveness, we propose Adaptive Time Delay, a modification to time delay that only activates time delay when players interact with each other. The goal of this modification is to preserve the increased fairness afforded by time delay while also mitigating the average reduced responsiveness of time delay.

Section 3: Methodology

Since adaptive time delay is a novel technology with minimal literature from prior research, the team chose to test the efficacy of adaptive time delay by following three main steps:

1. Design a 2-player online multiplayer first person shooter game that provides an environment suitable for testing time delay and latency.
2. Implement adaptive time delay with field of view and proximity as the conditions that describe player interaction.
3. Conduct a user study to determine the efficacy of adaptive time delay on improving responsiveness while maintaining the fairness aspect from regular time delay.

Section 3.1: Color Clash

The team built the game ‘Color Clash’ with Unity using the NetCode for Game Objects (NGO) library to test adaptive time delay as a viable latency compensation technique. *Color Clash* is a client-server first person shooter game following a one versus one deathmatch format.

We designed a 60x60 map, shown in Figure 5, that consists of obstacles, tiles, ramps, and platforms. WASD controls are used for moving and the spacebar is used for jumping. Players can move the mouse to adjust their view, as well as use left click to shoot. If one player’s shots hit the opponent, the opponent is eliminated and the player gains a point. Once eliminated, the opponent’s screen displays ‘Respawning’ (Figure 6), and the opponent respawns in a random location away from the other player after three seconds.

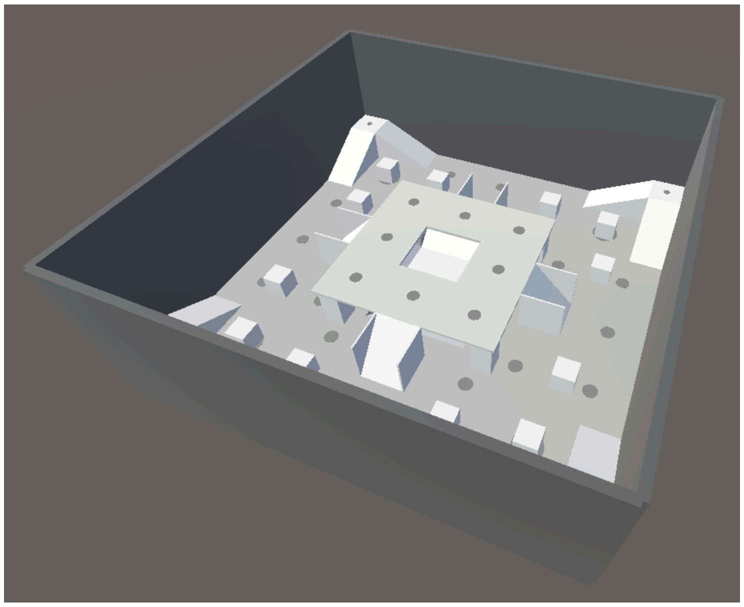


Figure 5: An overview of Color Clash's map.

You have been shot!
Respawning...

Figure 6: A respawn screen that appears when players are eliminated

In addition to shooting, the game includes a painting mechanic to test players' reactions to latency while completing directives apart from one another. Players can access the painting mechanic by right clicking on the mouse. Players are assigned a color, red or blue, when the session starts and that is the color they paint with. In order to paint, each tile contains a dark gray circular target. If a player shoots the target, the whole tile is painted their color (Figure 7). The ramps do not have targets. Six tiles are painted of each color (twelve total) randomly before each round begins. When players move across a tile of their color, their speed increases by 30%, and when they move across tiles of their opponent's color, their speed decreases by 20%.

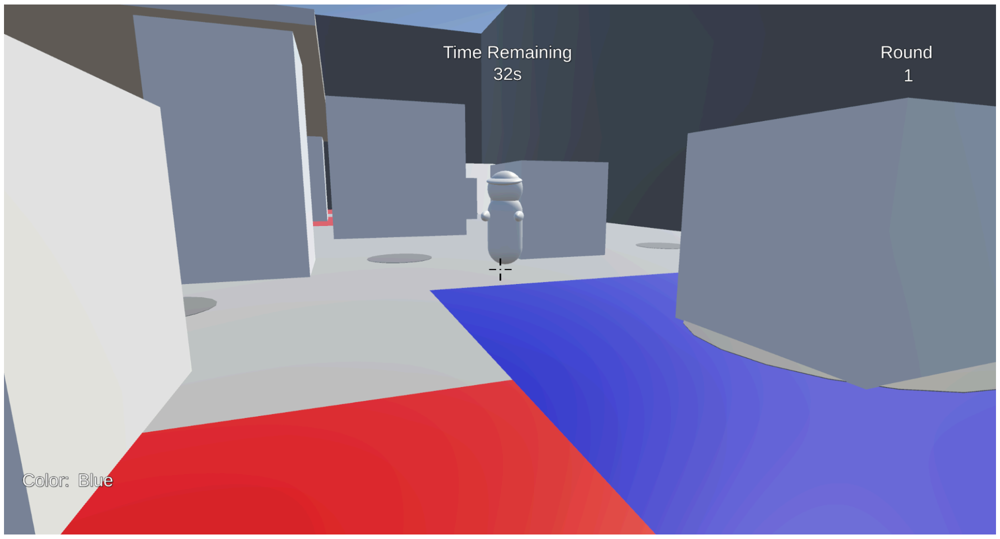


Figure 7: Painted tiles in the map

Players have one crosshair on their screens to show their aim. This aim applies to both painting and shooting. There is no limit on paint or ammo and thus no need to reload. Players have a delay of 250ms between shots.

Before beginning, each player must press a ready button on their screens as shown in Figure 8. When one player presses it, they proceed to a waiting screen. After both players press the button, a countdown from three shows up on each screen simultaneously. The game has a time limit of 50 seconds. Each player has a time limit at the top of their screen counting down. Whichever player has the most number of eliminations after time runs out wins, and the scores are shown at the end.

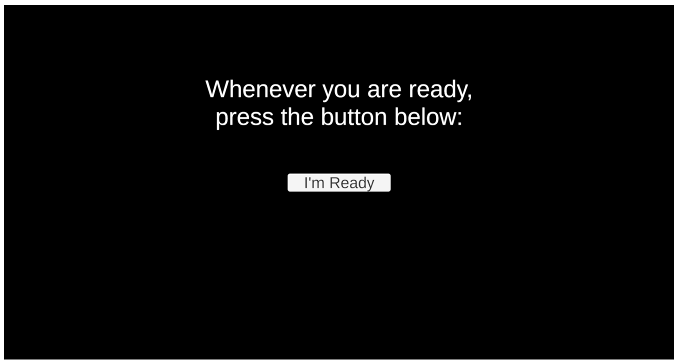


Figure 8: The ready screen that players must interact with to start a round.

Section 3.2: Adaptive Time Delay Implementation

3.2.1: Implementing Regular Time Delay

The goal of time delay is to ensure that all client messages are processed by the server around the same time and that all server updates are received by the clients around the same time. This was implemented for this study by adding a buffer before client messages get sent to the server, delaying those inputs for a specified latency. Time delay was applied only to the player with the lower latency in order to match the other player's higher latency.

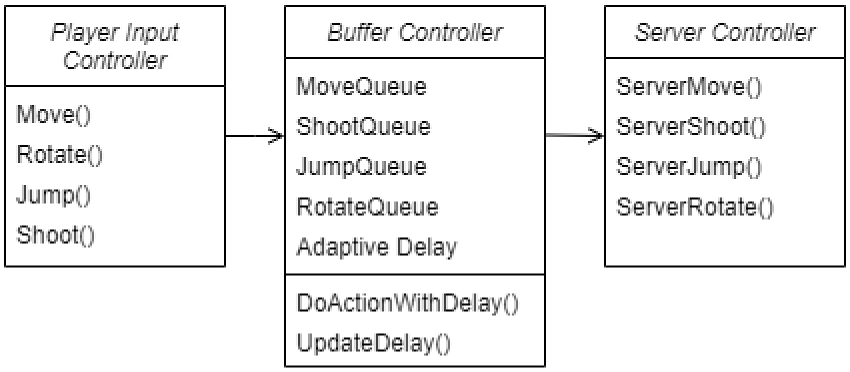


Figure 9: Buffer implementation for time delay

As shown in Figure 9, the player can do four types of actions: moving, rotating their camera, jumping, and shooting/painting. Once the player inputs a command, each action is sent to their respective queue in the buffer controller as shown above. For example, if the client wants to move forwards, this request is enqueued to the MoveQueue in the buffer controller.
The buffer controller and the input controller's update loops both run at 60 frames per second. The buffer controller then dequeues the contents of each queue after a specified latency and sends the inputs as remote procedure calls (RPCs) to the server where the command is executed. The latency is calculated by finding the difference between the two players' latencies. This ensures that all client messages arrive at the server at roughly the same time, satisfying the first

requirement for time delay. The setup used in this study involves two clients connected to the server through a LAN, and both clients have approximately the same latencies. This satisfies the second requirement that all server updates received by the clients arrive at approximately the same time.

In addition to the queues, three conditions were identified that would impact the amount of artificial latency that should be added to each queue.

1. Input command after a duration of no inputs greater than or equal to the artificial latency.
2. Continuous input commands.
3. Non-continuous input command within the previous input artificial latency value.

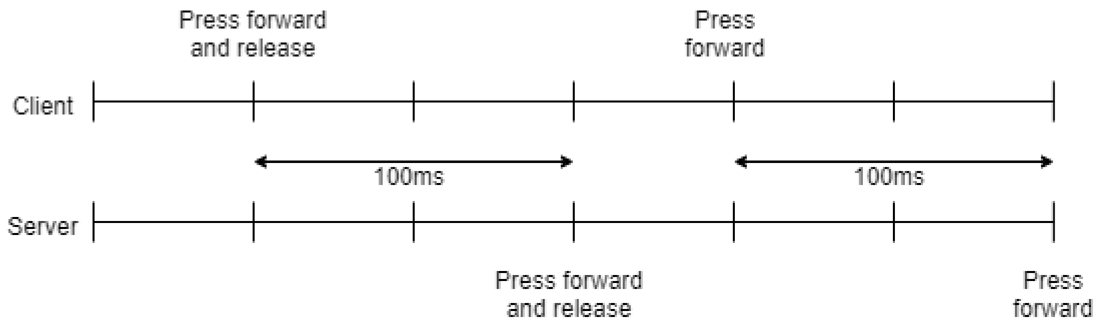


Figure 10: Delay calculation when input is the first input in a while (≥ 100ms)

For the first condition, the input command is a transition from an idle state to a busy state as shown in Figure 10. For example, if the artificial latency is 100ms, a player that has not moved for more than 100ms experiences the full latency duration when they input the next movement command.

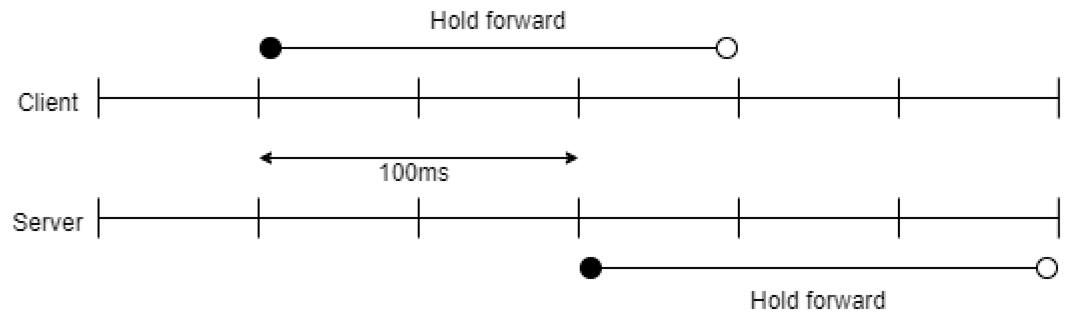


Figure 11: Delay calculation when input is continuous

In contrast, the second condition is when the input command is continuing a busy action. As depicted in Figure 11, an example of a continuous stream of input commands is holding down the W key, which makes the player move forward. If the artificial latency is 100ms, only the initial input command that starts the stream gets the full 100ms applied to it. All consecutive input commands follow with 0ms latency applied to them.

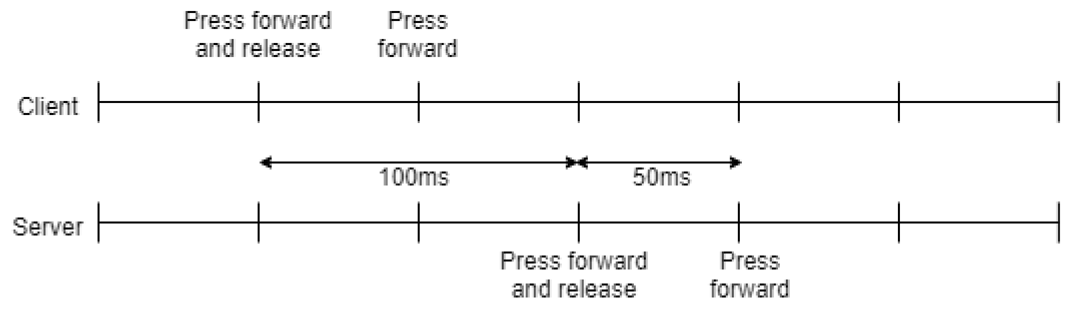


Figure 12: Delay calculation when input is non-continuous but within a short time (≤ 100ms) after the previous input

The last condition represents a situation where a non-continuous input command is made within the previous input command's artificial latency value. As depicted in figure 12, a player with 100ms of artificial latency makes a movement command, releases the key, then immediately presses the key again to move forward; the latency applied to the second press should not be the full 100ms of latency provided that the player pressed the key again within 100ms. Instead, the latency applied should be the time between the current and previous inputs, which is 50ms.

3.2.2: Adaptive Time Delay Conditions

For this study, the conditions that describe direct player interaction are proximity and field of view (FOV). Adaptive time delay only activates when there is direct player interaction, and vice versa.

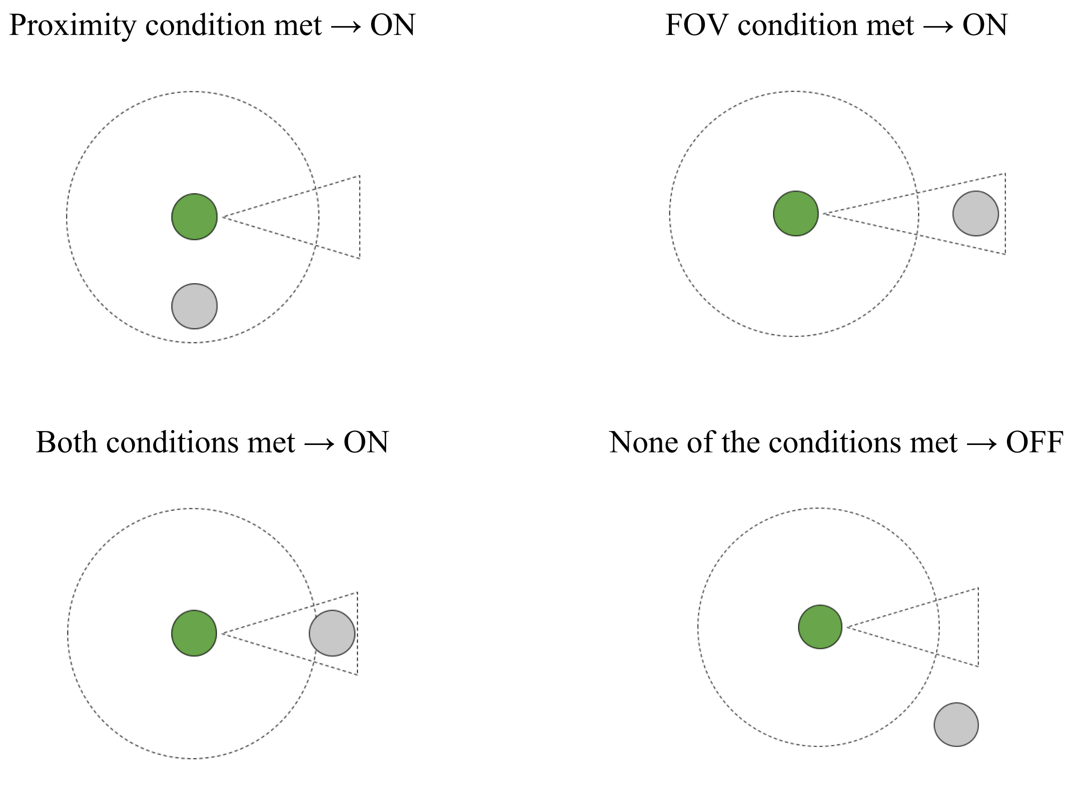

Proximity condition met → ON
FOV condition met → ON
Both conditions met → ON
None of the conditions met → OFF

Figure 13: Adaptive conditions for activating time delay

Figure 13 depicts the conditions that determine whether adaptive time delay is active. When any of the conditions are met, the player with the lower latency has time delay activated to match the other player's higher latency. The proximity condition is fulfilled when two players stay within 20 Unity units of each other. Adaptive time delay turns off when the players are 25

Unity units away to offset potential issues with time delay turning on and off. Even though there might be obstacles separating the players, this condition will be met as long as they are close enough. The FOV condition is fulfilled when at least one player is able to see another player on their screen; the other player does not need to see the player in return to activate time delay.

3.2.3: Adaptive Latency Smoothing Algorithm

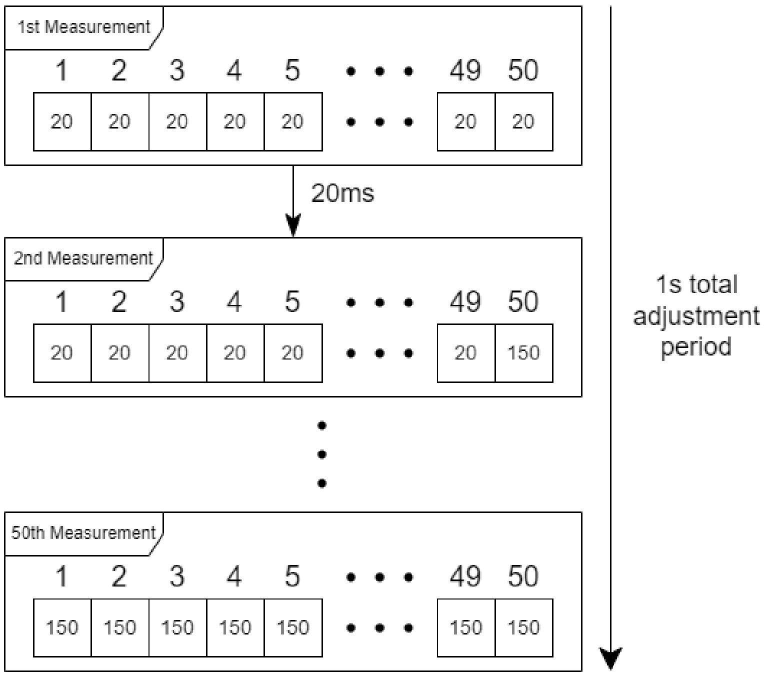


Figure 14: Adaptive Latency Smoothing Implementation

When adaptive time delay is activated for the lower latency player, the additional latency should be gradually increasing to match the higher latency player. Figure 14 shows the implementation for gradually adapting the lower latency player. The latency smoothing algorithm uses the average of the past 50 most recent readings. These readings are measured 50 times per second, which means that there is 20ms between each measurement. As a result, the lower latency player experiences an adaptation time of one second in order to fully adjust to the high latency player. The reasoning behind this implementation is that taking the average minimizes the effect of random latency spikes and fluctuations.

Section 3.3: User Study

Our user study was conducted in Fuller Labs 316 at WPI. The study equipment consisted of two client PCs, one server PC, and associated peripherals. All PCs were connected to a LAN network via an Ethernet connection. The client PCs ran Windows 11 and were equipped with Intel Core i7-8700K CPUs @3.7GHz with 6 Cores, Nvidia GeForce GTX 1080 graphics cards, and 64 GB of RAM. Each client PC had a Lenovo Legion Y25-25 monitor with 1920x1080 resolution, 240Hz refresh rate, and 1ms response time, a Logitech g502 Hero gaming mouse with 1200 DPI and 1000Hz polling rate, and a Dell KB216 keyboard for peripherals.

3.3.1: Preparing the Study

To recruit users for our study, we sent out a solicitation email that directed interested users to sign up for an appointment through Slottr. Participants received this email through student mailing lists. As incentives to sign up, all participants were offered relevant coursework credit where applicable as well as an entry into a raffle to win a $20 Amazon gift card.

Prior to users arriving, we prepared each client PC by opening the game, connecting to the game server, and leaving the game on the “Ready” screen. Upon arrival, each player sat at one of the two designated client PCs and signed a consent form. From there, they filled out a demographics survey (Appendix A) and completed a reaction time test to obtain a baseline experience level for the user. The survey included questions about keyboard-based FPS game experience as well as overall gaming experience.

3.3.2: Round System and Delay Configurations

To test the efficacy of adaptive time delay versus regular time delay, we developed a variety of different latency configurations to simulate real world scenarios that can occur when playing first person shooter games. A latency configuration is a combination of three factors that alter the latency users experience as they play through a particular round. These factors include:

1. Latency Value
2. Time Delay Setting
3. Adaptation Time

For the purposes of this study, we designated one client PC to always be our control player and the other to always be our experimental player. The control PC had different fixed latency values applied to it, while the experimental PC had different time delay settings applied to it. When the setting for a particular round was Adaptive Time Delay, the experimental player also had a different adaptation time applied to it. This is accomplished by changing the number of most recent latency readings in the adaptive latency smoothing algorithm in Section 4.2.3. We combined latency values with time delay settings and adaptation times to generate a list of possible latency configurations that would occur during the session, with a couple of exceptions. The exceptions are as follows:

- The adaptation times only combine with the adaptive time delay mode.
- The adaptation times of 0s and 2s only occur when the control player's fixed latency value is 150ms.
- There is no time delay setting applied when the control player has 0ms of fixed latency. This generates a list of 12 possible latency configurations.

A summary of the fixed latency values, time delay settings, and adaptation times can be found in Table 1.

To improve upon the size and significance of our data, each latency configuration is played twice during a session, meaning that users played a total of 24 rounds in a session. These configurations are predetermined when the game is started, and the order they appear in is randomized to eliminate any potential bias, as being able to predict the next round's configuration could influence the user's gameplay and questionnaire responses. We also gave users a practice round at the beginning of the session, after they completed the necessary forms upon arrival. This allowed users to get accustomed to the mechanics and controls of the game. This practice round did not have its data collected.

Control Player Fixed Latency Value (ms)	Time Delay Setting	Adaptation Time (s)
0	N/A	N/A
50	No latency compensation	0 (most recent reading)
100	Regular time delay	1 (50 most recent readings)
150	Adaptive time delay	2 (100 most recent readings)

Table 1: Latency values, time delay modes, and adaptation times combine to create a list of possible latency configurations that will occur during a session.

3.3.3: Running the Study

Once users arrived, signed the consent form, completed the demographics survey and reaction time test, and received a gameplay explanation, they were ready to begin playing. Users would begin by pressing the "Ready" button in the game window. Once both players have readied up, the round starts after a three second countdown. Each round in the 25 round sequence consists of the following:

1. Users will ready up by pressing the ready button
2. Once the countdown finishes, the latency configuration for that round is applied, and users play for 50 seconds.
3. When the round concludes, the player scores are shown.
4. After a brief pause, the in-game questionnaire is displayed.
5. Once the questionnaire is completed, the ready screen is displayed once more.

The game automatically moves on to the next round after the previous one concludes. This design choice reduced the chances of user error while loading as well as researcher error when setting up rounds, as they only needed to be set up once per session. After submitting the questionnaire for the last round, players were guided to a “Thank You” screen with the team’s information to let them know the session was over. Each user study session ran for about 30 minutes total.

3.3.4: Data Collection

Color Clash collects game data from the user as they play. This allowed us to evaluate the efficacy of adaptive time delay on responsiveness, fairness, and player performance. Data is collected using a custom built logging system embedded into the game. The log manager writes messages as well as important game values produced by a given client or server to one of six different log files. Each log file contains all of the data collected during a session that pertains to a particular action or game feature. The list of log files are as followed:

1. Shoot Log
2. Movement Log
3. Jump Log

4. Questionnaire Answers
5. Latency Configurations Log
6. Round Results Log

As participants play the game and perform in-game actions, messages are written to the client's log file when the action is requested. When the action is executed on the server, a message is written to the server log file. Table 2 shows the data that is written with each log message:

Data Name	**Contents**
System Time	The system time of the PC
Session Time	The cumulative time elapsed since the practice round started
Round Time	The amount of time elapsed since the beginning of this round
Round Number	The current round number
Adaptive Latency	The amount of time delay the client is experiencing. For the control player, this value is 0. For the experimental player, it depends on the time delay mode. If the mode is No Compensation, this value is 0. If it is Regular Time Delay, this value is equal to the control player's fixed latency value. If it is Adaptive Time Delay, this value is equal to the client's current additional latency, as long as adaptive conditions are met.
Movement Speed	The movement speed of the player, classified as one of 3 categories: ● Slow - The player is on a tile matching the opponent's color ● Normal - The player is on an uncolored tile ● Fast - The player is on a tile matching their color
Message	Stores data unique to the given action.

Table 2: The information attached to each log message.

In addition to this information, each log message contains additional data relative to the action performed. Table 3 shows the types of player actions and the data that is attached to each action's log message.

Action	Attached Data
Movement	● Player character's destination position
Jump	● No additional attached data
Shoot	● Shooter's color ● If an opponent was hit, a "player killed" message along with the opponent's color ● If a target was hit, the color of the corresponding tile before and after the shot was fired
End Round (Non-Player Action)	● Each player's scores

Table 3: The information attached to each log message.

The schemata shown in Table 2 and Table 3 apply to four of the six log files. These log files contain our data that we then used to analyze player performance. We used an additional log file which exists on the server that contains the generated list of latency configurations that are played during the session (See Table 1 for a list of possible configuration values).

In addition to performance data, an in-game questionnaire is shown to participants at the end of each round. This questionnaire serves as a method to gather each participant's subjective feedback to the latency conditions of each round. The questionnaire contains two questions. Question 1 asks the participant to rate how much lag they felt in the most recent round, while Question 2 asks the participant to rank how fair they thought the most recent round felt. The questionnaire as it appears in-game is shown in Figures 15 and 16. Both of the answers are written to a comma separated log file, shown in Table 4, which contains this subjective data.

Column	Description
Round Number	The current round number
Fixed Latency	The amount of fixed latency applied to the control player
Time Delay Mode	The time delay mode applied to the experimental player
Question 1 (Responsiveness)	A numerical answer between 1 and 5. 1 being no latency felt, while 5 is extreme latency felt.
Question 2 (Fairness)	A numerical representation of the answer between -5 and 5. Negative values represent the game being unfair in favor of the opponent, positive values represent the game being unfair towards the person answering the question, and 0 being a perfectly fair game.

Table 4: An explanation of how the questionnaire answers are stored for analysis.

The player performance data combined with the responses to the in-game questionnaire provided the team with objective and subjective data that helps us assess how adaptive time delay affects the responsiveness, fairness, and performance of the players.

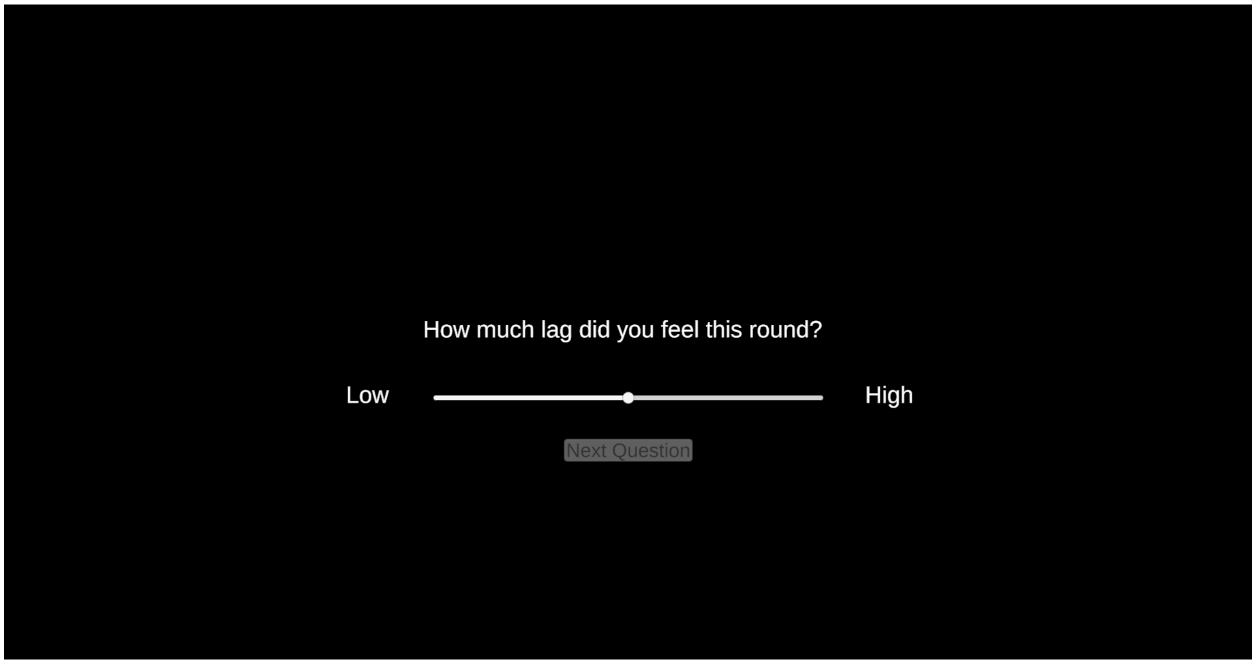


Figure 15: Questionnaire question 1 as it appears in game.

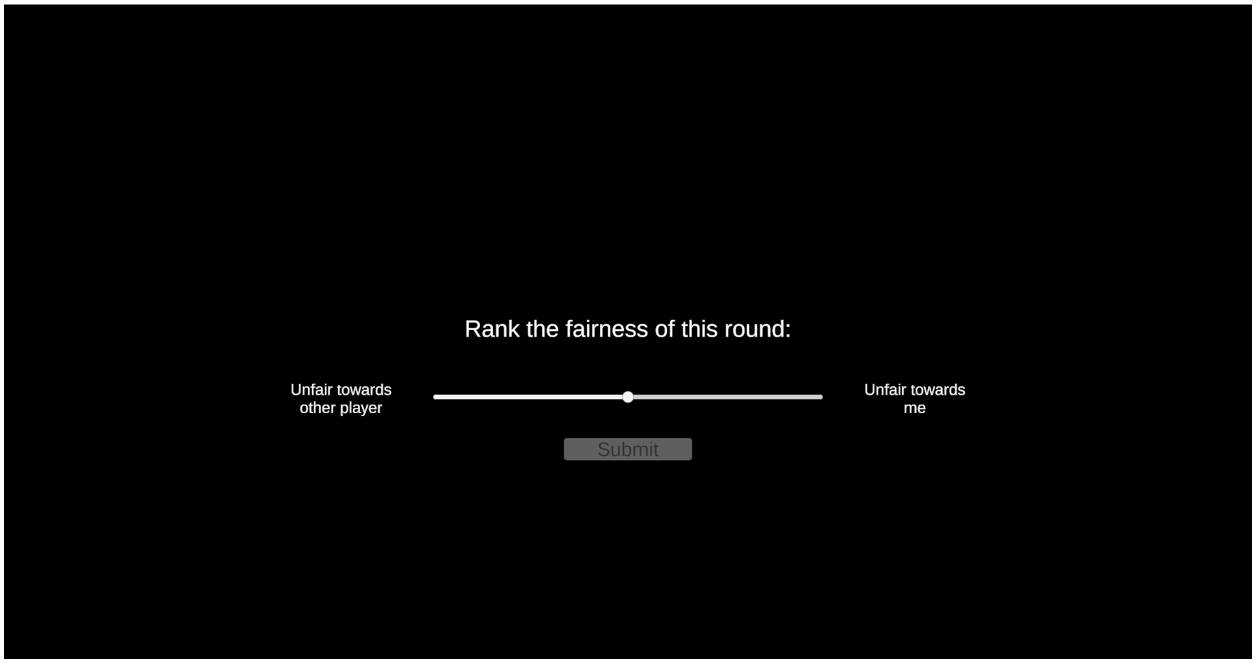


Figure 16: Questionnaire question 2 as it appears in game.

Section 4: Analysis

We performed an analysis on the performance and questionnaire data obtained through our user studies using the Pandas and Seaborn libraries for Python and Microsoft Excel. We analyzed player performance, game responsiveness, and game fairness. When one participant was able to attend a user study session, we had one of the study proctors play as the control player. The performance and questionnaire answers from the experimental player still reflected actual values.

Section 4.1: Data Cleaning

Before analyzing our data, we filtered out potentially problematic data points. The filtered out data points were not included in our analysis, as seen in Table 5. We filtered out a user study session where a proctor filled in as the experimental player instead of the control player. Additionally, we filtered out session number 9, where 21 of the 25 rounds played resulted in at least one player getting zero eliminations, and 14 of those 21 rounds played resulted in both players having zero eliminations–there was a lack of performance data. We filtered out the control player data during the sessions that involved a study proctor playing as the control player.

Beginning Total Players	Control Players Removed	Experimental Players Removed	Control Players Analyzed	Experimental Players Analyzed	End Total Players
46	6	2	17	21	38

Table 5: Result of Session Data Cleaning

There were additional sessions that were considered for filtering, with the main cause of the consideration being a significant difference in demographics, particularly gaming skill and experience. However, we decided to keep these sessions due to these types of scenarios occurring regularly in first person shooter games.

An interesting scenario to consider is when a player with adaptive time delay interacts with the opponent for the whole duration of the round. This essentially creates a situation where adaptive time delay acts like regular time delay.

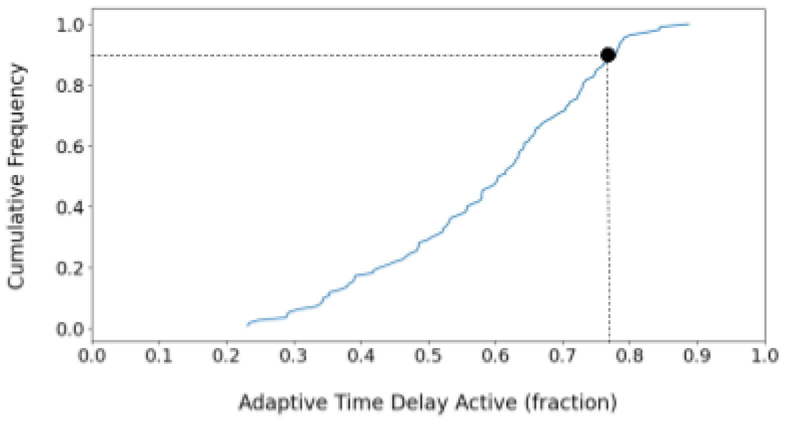


Figure 17: Cumulative frequency of the time adaptive delay is active for all adaptive rounds

We computed the fraction of time adaptive time delay was on for each round. In Figure 17, the x-axis is the fraction that represents how long the adaptive delay is active during the round. The y-axis is the cumulative frequency of these rounds. In total, there are 210 rounds with adaptive time delay. This figure shows that approximately 10% of these adaptive rounds have adaptive time delay active for at least 75% of the total round time. Hence, these rounds were moved to fixed latency rounds and removed from the adaptive time delay rounds. Table 6 shows the final number of rounds for each latency configuration for the user study.

Number of Rounds for Zero Condition	Number of Rounds for No Compensation	Number of Fixed Latency Rounds	Number of Adaptive Latency Rounds
42	126	147	189

Table 6: Results of Round Data Cleaning on Experimental Players

Section 4.2: Demographics

Table 7 summarizes the demographics in the study. Typical genres played and preferred gaming systems were given as multiple choice questions with an ‘other’ write-in option. The mode was used to answer the ‘average’ genre and gaming system. Amount of hours played per week in general and in regards to FPS games was open-ended. When users put in a range of time, such as 5-7 hours per week, we took the average. Finally, experience and lag effect were both asked on a five point scale, with one representing low experience/low effect on gameplay, and five representing high experience/highly affected by lag.

There were 38 participants and 23 sessions played. When a participant did not have a partner, one of the team substituted in for the control (blue) player. The team’s demographic and reaction test data were excluded from our analysis. The most preferred gaming system was by far keyboard and mouse. The most common genre played was Role Playing Games (RPGs), followed closely by First Person Shooter games. The average hours of video games in general played per week was 12.74 hours, and the average hours of First Person Shooter games played per week was 3.86 hours. Users rated their experience and impact from lag both between 3 and 4, representing that they had general knowledge of First Person Shooter games and they could typically recognize lag. Figure 18a and Figure 18b show the average reaction times for both control and experimental players, both of which are around 189ms.

Users	Preferred gaming system	Most common genre played	Hours of video games played per week	Hours of FPS played per week	Experience with FPS	How much lag affects gameplay	Average Reaction Time for control players (ms)	Average Reaction Time for experimental players (ms)
38	Keyboard and Mouse	RPG	12.74	3.86	3.19	3.51	189.34	188.64

Table 7: Distribution of demographics.

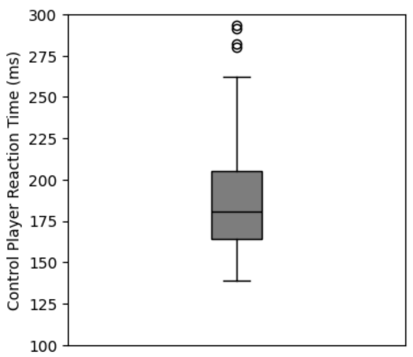


Figure 18a: Reaction Times for Control Players

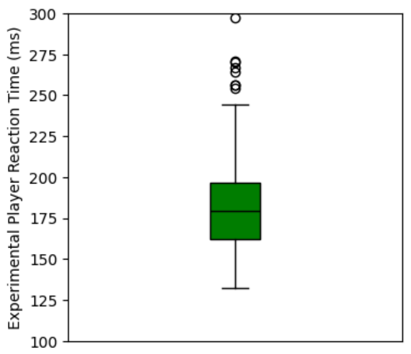


Figure 18b: Reaction Times for Experimental Players

Section 4.3: Performance Results

Performance is defined as how well a player does in the tasks necessary to win the game. In this study, one measure of performance is how accurate a player is with their shots. *Color Clash*'s built-in log manager automatically recorded every time a client requested to shoot and every time the shot was executed on the server. It also recorded every time a player was eliminated through a successfully aimed shot. Using these two unique messages about shots and eliminations, we were able to extract an accuracy value for any given round. Obtaining the accuracy data from each round allowed us to organize the data based on the delay configuration it came from. We visualized trends in our data based on any combination of fixed delay value, time delay mode, and adaptation time.

First, we analyzed the effects of different fixed latency values on performance, separated by time delay mode. Figures 19a, 19b, 19c, and 19d show the mean accuracy for both the control player and the experimental player under different latency configurations. In Figures 19a, 19b, and 19c, the ticks on the x-axis represent a latency configuration, while the y-axis represents the mean accuracy for all players over all sessions. In Figure 19d, the ticks on the x-axis represent adaptation time in seconds, while the y-axis represents mean accuracy. In all figures, the gray data points are the control player data, while the green points are the experimental player data. The latency configuration graphs are all based on 1s adaptation time. Two rounds each session had a 0s and 2s adaptation time. All means are bounded by 95% confidence intervals.

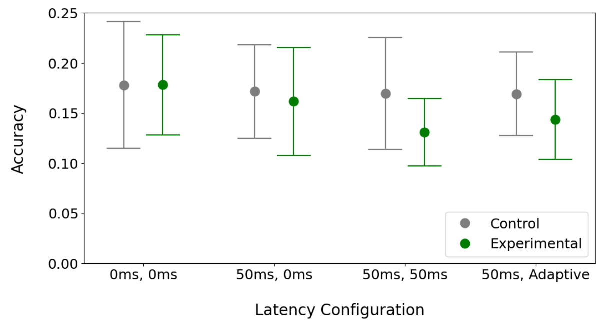


Figure 19a: Mean accuracy with 50ms fixed latency

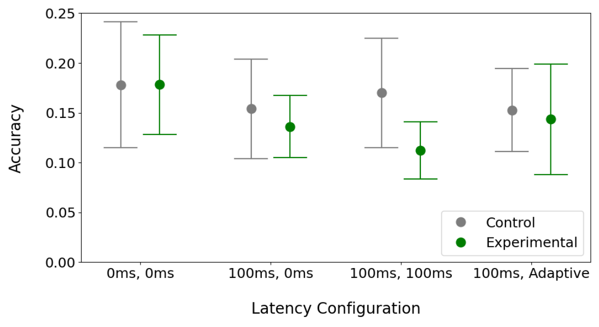


Figure 19b: Mean accuracy with 100ms fixed latency

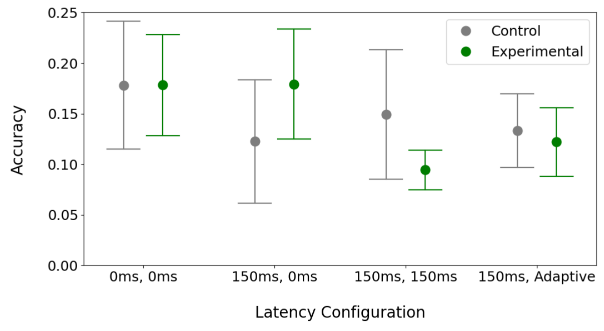


Figure 19c: Mean accuracy with 150ms fixed latency

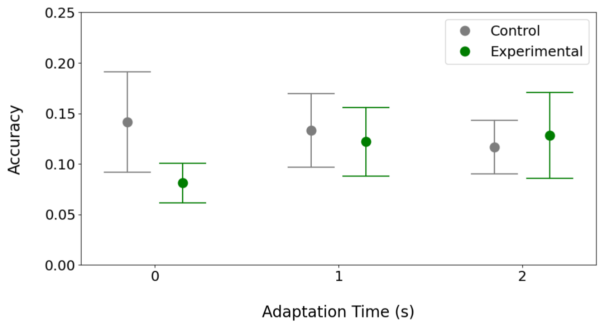


Figure 19d: Mean accuracy with different adaptation times for 150ms fixed latency

In Figure 19a, the additional 50ms of latency for the control group does not significantly affect their mean accuracy, but does slightly lower the accuracy of the experimental group. In Figures 19b and 19c, the additional latency now lowers the accuracy of the control group by a significant amount, while the experimental group's accuracy only lowers significantly when they receive regular time delay or adaptive time delay. Figure 19c shows a situation where the control group has 150ms of fixed latency applied to them. This significantly reduces their accuracy, putting the experimental group at an advantage when they do not receive any latency compensation. When they receive adaptive time delay, their accuracy now lowers to a value similar to the control group. This shows that in extreme conditions where one player is experiencing high amounts of latency and the other is not, adaptive time delay can be introduced to preserve the fairness experienced when both players have similar latency values.

Figure 19d shows the effects of different adaptation times for the latency smoothing algorithm on players' mean accuracy. The first column shows a scenario where the adaptation time is instant, meaning as soon as the conditions for adaptive time delay are met, the experimental player's latency goes up from their base value to their base value plus 150ms. According to the first column, this instant adaptation time significantly reduces their mean accuracy. The second and third column have adaptation times of 1s and 2s respectively, and the means of the control group and experimental group are much closer in these two columns. This shows that a non-instant adaptation time plays an important role in maintaining fairness for both players.

Section 4.4: Perceived Latency Results

Responsiveness refers to feedback from player inputs. The study focuses on qualitative responsiveness, and the metric to measure this is the player's perceived latency. During the questionnaire portion of the user study, participants were asked how much lag they felt throughout each round of a session. The results from the questionnaire are then organized by the delay configuration.

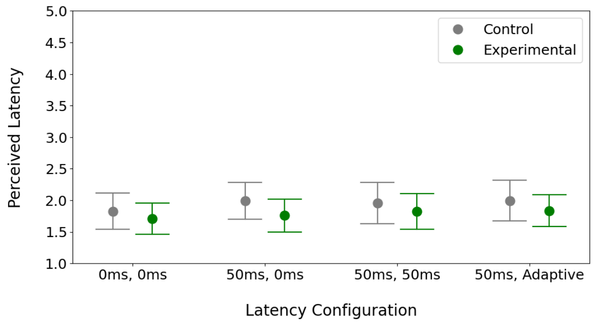


Figure 20a: Mean perceived latency with 50ms fixed latency

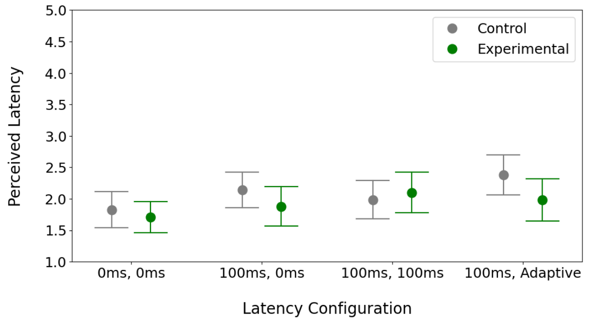


Figure 20b: Mean perceived latency with 100ms fixed latency

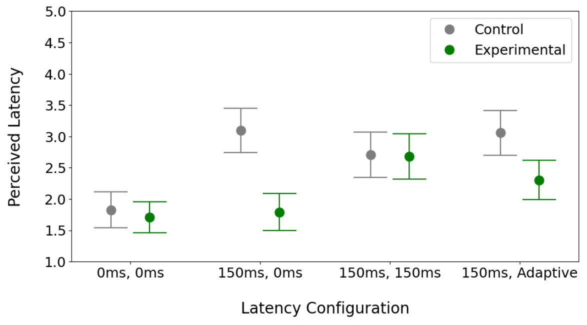


Figure 20c: Mean perceived latency with 150ms fixed latency

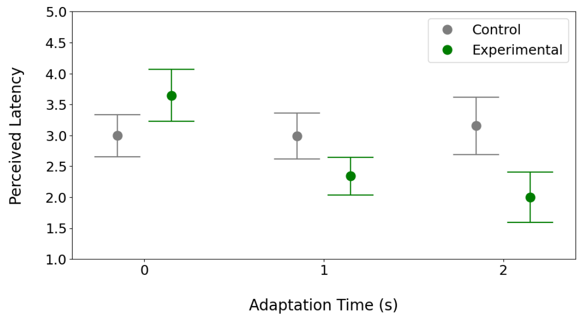


Figure 20d: Mean perceived latency with different adaptation times for 150ms fixed latency

In Figures 20a, 20b, 20c, and 20d, the control group is represented by the gray data points, while the experimental group is represented by the green data point. The y-axis represents how much latency players are able to perceive. 1 means that the player did not perceive noticeable latency, while 5 means that the player perceived extreme noticeable latency. For Figures 20a, 20b, and 20c, the x-axis represents the different latency configurations. For each x-axis tick, the left number is the control's latency, while the right number is the experimental's latency. For Figure 20d, the x-axis are the different adaptation times in seconds for the experimental group during adaptive rounds. In all figures, each data point is the mean perceived latency with a 95% confidence interval.

Figures 20a, 20b, and 20c show the change in players' perceived latency when subject to different latency configurations. When subject to 0ms of latency, the control group does not perceive noticeable latency; but, when the control group is given its respective fixed latency value, there is an increase in perceived latency, which means that the control players notice an increase in latency. The experimental group in these figures also show that players did not notice the latency when subject to 0ms, but they were able to perceive a similar latency level as the control group when the experimental players were given the same fixed latency as the control players. This suggests that both the control and experimental groups have a similar perception of latency. When the experimental players were given adaptive time delay, there is a decrease in perceived latency compared to the control group's perceived latency as well as the experimental group's perceived latency when subject to the control's fixed latency.

However, this trend for Figures 20a and 20b is not significant for latency values of 50ms and 100ms since the confidence intervals were noticeably overlapping. This suggests that the values of 50ms and 100ms might not be large enough for our sample to perceive latency accurately. Contrary to this, the results in Figure 20c are significant as the confidence intervals have minimal overlap. This shows that adaptive time delay can significantly improve responsiveness especially for high latency values since the experimental players do not notice as much latency.

Figure 20d shows the impact of adaptation time on player's perceived latency during adaptive time delay rounds. The control group's means are similar as adaptation time does not impact them. An interesting find is that when the adaptation time is instant, the experimental players' perceived latency is higher than the control groups' perceived latency, suggesting that abruptly adapting in and out of time delay creates a worse user experience and responsiveness. However, when the experimental players were given a 1s or 2s adaptation time, their perceived latency is lower than the control players' perceived latency as well as the experimental's perceived latency during instant adaptation. These observations are significant as there is either minimal or no overlap between the control and experimental group perceived latency means. This shows that adaptation time significantly affects the responsiveness and perceived latency for players with adaptive time delay.

Section 4.5: Perceived Fairness

Qualitative fairness is another focus of this study, and it is measured by the player's perceived fairness. Perceived fairness was measured by asking players how fair they thought each round was. The results from the questionnaire are then organized by latency configuration.

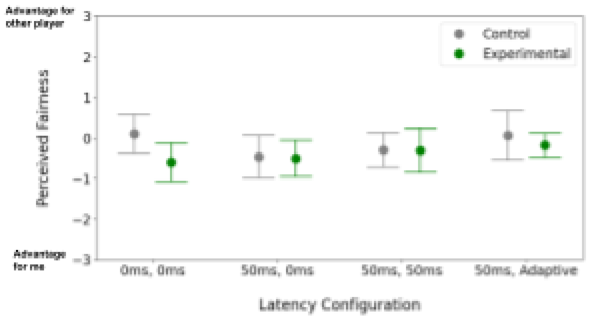


Figure 21a: Mean perceived fairness with 50ms fixed latency

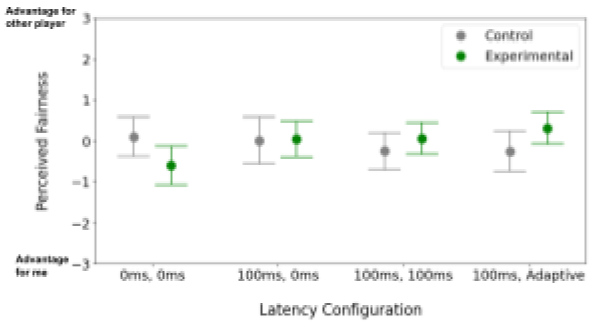


Figure 21b: Mean perceived fairness with 100ms fixed latency

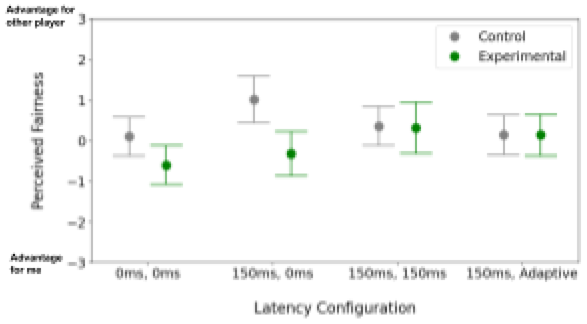


Figure 21c: Mean perceived fairness with 150ms fixed latency

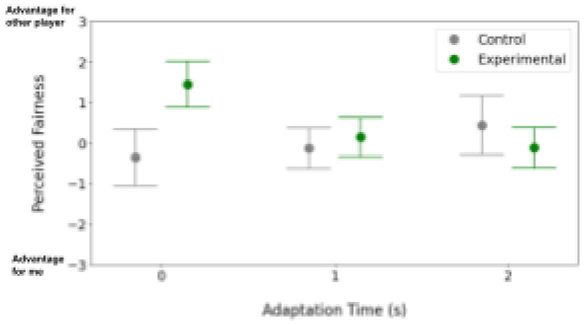


Figure 21d: Mean perceived fairness with different adaptation times for 150ms fixed latency

In Figures 21a, 21b, 21c, and 21d, the control group is represented by the gray data points, while the experimental group is represented by the green data point. The y-axis represents how "fair" a round users are able to perceive. The scale was from -5 to 5, with -5 being "Unfair towards other player", meaning that the user believed they had an advantage over their opponent, 0 for "Fair", meaning the players were on a level playing field or 5 meaning "Unfair towards me", where players thought that their opponent had an advantage over them. For Figures 21a, 21b, and 21c, the x-axis represents the different latency configurations. For each x-axis tick, the left number is the control's latency, while the right number is the experimental's latency. For Figure 21d, the x-axis are the different adaptation times in seconds for the experimental group during adaptive rounds. In all figures, each data point is the mean perceived latency with a 95% confidence interval.

Figures 21a, 21b, and 21c show the change in players' perceived fairness when subject to different latency configurations. When exposed to 50ms and 100ms of latency, neither the control nor the experimental player experiences significant differences in fairness. However, in figure 21c, a noticeable difference arises when players experience different configurations under 150ms of latency. When both players experience no latency and 150ms of fixed latency, the experimental player still feels as though they have a slight advantage, however that advantage is insignificant. When the control player experiences 150ms of latency while the experimental player experiences no latency, a difference in answers is apparent. This suggests that players are able to perceive the difference in fairness for higher latencies. An interesting finding from these graphs is that the control and the experimental players report the rounds with 150ms and adaptive conditions, respectively, perceive the same level of fairness. Not only that, but they both feel that the round is fair, suggesting that adaptive time delay under high latency conditions gives both players the perception that the round is fair.

Figure 21d shows the impact of adaptation time on player's perceived fairness during adaptive time delay rounds. The control group's means are similar as adaptation time does not impact them. An interesting finding is that players perceive 1s of smoothing to be more fair than both 0s (instantaneous) and 2s. However, more testing is necessary to see if 2s adaptation times have a significant impact compared to 1s adaptation times as there is some overlap between the control and experimental players' confidence intervals. Overall, the data shows that adaptation time affects players' perception of fairness.

Section 4.6: Summary of Results

To determine the effect of adaptive time delay on fairness, the study looks at quantitative and qualitative fairness. Quantitative fairness is measured through player performance, while qualitative fairness is measured through players' perceived fairness. For 150ms latency, the trend for player performance follows the trend for perceived fairness. This is a strong indicator that these two measurements can be used to measure overall fairness. Refer to Appendix B for the relationship between accuracy and perceived fairness.

Players were very adept at perceiving latency conditions. They correctly reported feeling more latency as more was added to individual players, and both the control and experimental tended to report the same perception under the same conditions. For 150ms, the trend for perceived latency follows the inverse of the accuracy graph, strongly indicating that accuracy is a method players use when gauging the effect of lag on their gameplay. Refer to Appendix C for the relationship between perceived latency and accuracy. Furthermore, players are more likely to perceive an advantage towards themselves when they feel low latency, while they are more likely to perceive an advantage towards their opponents when they feel high latency. Refer to Appendix D for the relationship between perceived fairness and perceived latency.

Thus, the results of performance, perceived latency, and perceived fairness show that adaptive time delay maintains fairness and increases responsiveness for high latencies (i.e. 150ms). For all three factors, the latency adaptation time for the lower latency player has a significant impact. If the adaptation time is instant, adaptive time delay actually creates more unfairness and less responsiveness when compared to not having the latency compensation technique active.

Section 5: Limitations

While our analysis was able to produce promising results for the efficacy of adaptive time delay, our work has some limitations. Our game implementation contained design decisions that might have impacted our results, and the details of our user study might have done the same. This section describes the limitations that arose during our implementation and user studies.

Section 5.1: Game Implementation

Color Clash was designed in a way that promoted easy to use controls and simple graphical design. The intention was to make it as easy as possible for user study participants to learn the game's features so they could be quickly prepared to play through the session and produce data. However, the design decisions made could have impacted the effectiveness of adaptive time delay in other scenarios.

Color Clash has limited graphics, sacrificing graphical quality for improved framerate and lower base latency. While this is helpful for our study, most commercial FPS games contain much more complex graphics, including lighting, particle effects, and complex object models. Higher quality graphics can decrease performance due to potentially lower framerate and increase base latency. In addition, as shown in Figure 22, our character model consisted of only one hitbox, extending from the bottom of the body to the top of the head in a capsule shape. In commercial FPS games, character models often have several hitboxes that have different effects depending on which part of the model was shot at. Our game also only had a single map available and a single game mode available, while commercial FPS games commonly have multiple maps and game modes that aim to diversify the playing field.

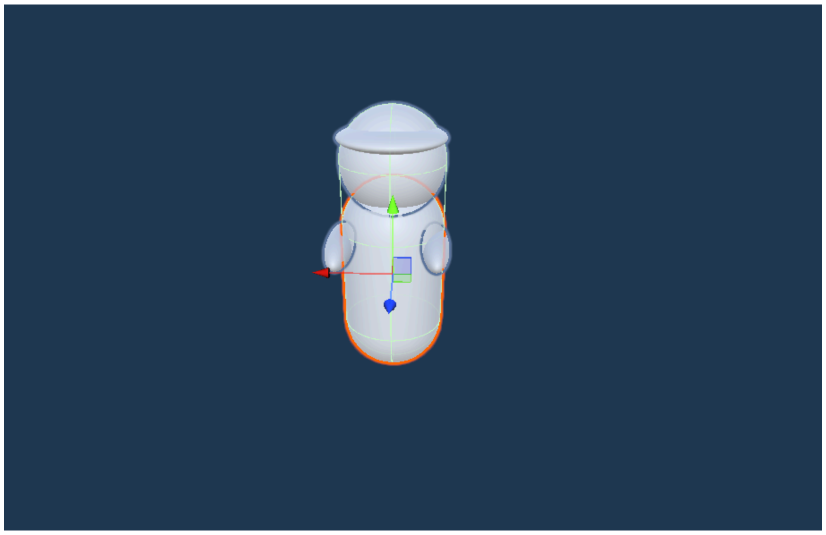


Figure 22: Our player character's hitbox, shown as a green capsule.

The player character had access to unlimited ammunition, with the only thing restraining their shots being the 0.25s cooldown before being able to shoot again. It is common for weapons in FPS games to have a limited magazine size, where players must reload their weapon after expending all the ammunition in the magazine. This additional time spent reloading often plays an important role in a player's playstyle, as players typically will take cover or move to a safer location to reload. In commercial FPS games, there are often a wide variety of weapons that deal different amounts of damage to players. In *Color Clash*, our weapon instantly eliminates the opponent after landing a successful hit. Including weapons that have a longer time to kill could cause players to meet the field of view condition for adaptive time delay for longer periods of time, which could influence performance as well as participant responses to our questionnaire question on how much lag they felt. In addition, our weapon was a hitscan shot, meaning the shot fired has an instant travel time. This instant travel time is not always the case in some FPS games such as *Quake* or *Halo*, where shots primarily follow a projectile pattern.

Our game focused on a one vs. one deathmatch scenario where there are only two players playing at once and the only objective is to obtain more points than the opposing player. While adaptive time delay produced promising results in this scenario, many commercial FPS games follow a team-based format. Shooters such as *Overwatch*, *Valorant*, and *Counter-Strike 2* all involve teams of more than a single player.

Section 5.2: User Study

Our user study was run over the course of three weeks, and each session involved two participants playing 25 rounds of *Color Clash*. Due to the nature of this study, however, there are a few key details that might have impacted our results.

41 players participated in our user study over 23 sessions, with 22 experimental players and 19 control players. Due to this relatively small sample size, our confidence intervals for our mean performance, fairness, and responsiveness measurements are larger than they would be with more players. In addition, we filtered two sessions out of our analysis completely and control player data during sessions in which a study proctor played as the control player.

We did not exclude players due to their video game and first person shooter experience, and as a result there were a few sessions that produced potential outlier data because of a large gap in experience. Although the data from these sessions was included in our analysis, some of our overall results could have been impacted by sessions with large experience / skill gaps. Performance data might have been affected due to the higher experienced player netting significantly more eliminations than their opponent. Responsiveness and fairness data might have been affected due to the possibility that lower experienced players might not fully understand the questions or recognize when a round was unfair or had poor latency conditions.

Section 6: Conclusion

Online multiplayer FPS games are heavily affected by latency as they involve fast-paced action and player competition. High latency leads to a multitude of problems including lack of responsiveness, which then creates unfairness and affects player performance. To solve these problems, time delay is a latency compensation technique that ensures fairness for FPS players at the expense of responsiveness since the lower latency player's input requests and responses to and from the server are delayed to match the higher latency player.

To address time delay's weaknesses, this study explores adaptive time delay, a novel modification that activates time delay only when players have direct interaction with each other. Thus, an FPS game designed to test adaptive time delay is created, and a user study is conducted to test the viability and efficacy of adaptive time delay for online multiplayer FPS games. The objectives that determine the viability of adaptive time delay is if adaptive time delay (1) maintains fairness and (2) improves responsiveness for the player with adaptive time delay.

In conclusion, the results of the user study suggest that adaptive time delay succeeds in maintaining the fairness from regular time delay but also improves the responsiveness for the player subject to the latency compensation technique. However, the improved responsiveness is only significant for high latencies like 150ms. Adaptation time is also another crucial aspect to consider for adaptive time delay. When a player with adaptive time delay adjusts to the other player's high latency instantly, it actually creates a more negative effect on fairness and responsiveness compared to when the latency compensation technique is not used at all. However, having an adaptation time of at least 1s will achieve the two objectives of adaptive time delay.

Section 7: Future Work

Although adaptive time delay shows promising results from our analysis, there is still more to evaluate about its effectiveness in other scenarios. This section describes potential improvements, adjustments, and situations that adaptive time delay can be tested with in future projects. Future work is split up into three categories:

- Short Term - Additional project work that could be done immediately succeeding this one.
- Medium Term - An additional project that expands upon the work done by this project.
- Long Term - Extensive work building off of the work done by this project that could take a significant amount of time to complete.

Section 7.1: Short Term

The latency smoothing algorithm showed to have a significant impact on player performance and responsiveness, as shown in Figures 19d and 20d. The impact was noticeable when results for player accuracy and perceived latency for adaptation times of zero seconds and one second are placed next to each other. Future work that could improve upon the smooth algorithm would be to introduce new, potentially more effective algorithms. Our algorithm is relatively simple, taking the average of the past N latency readings from the player with higher latency and applying that average as fixed latency to the lower latency player. Introducing a more complex algorithm could improve player performance and increase responsiveness even further than our own implementation. This could be done by creating a smoothing algorithm that follows a nonlinear function to increase a player's latency.

Another simple idea would be to conduct our user study with the control player having a fluctuating amount of delay. Doing this would test adaptive time delay in a more typical real world scenario where someone's latency is constantly changing. It would also test how effective the smoothing algorithm is at making adjustments based on a changing delay value. This could be achieved by creating a program that fluctuates a PC's latency, or by using our time delay implementation and implementing an algorithm to constantly change the control player's latency over a round.

Section 7.2: Medium Term

Our game implementation, *Color Clash*, follows a 1v1 deathmatch format. However, most commercial FPS games typically follow a team based format. Modifying adaptive time delay to account for a team situation where there are many more than 2 players playing would be interesting, as the current implementation only applies to a 1v1 scenario. One would need to start by determining the following:

- A new set of conditions a player must meet to activate time delay
- Which latency values to use and when. Possibilities include the team's average or only the ones that are within the conditions a player is fulfilling.
- An adaptive latency smoothing algorithm that takes into account multiple players.

Color Clash has fairly basic first person shooter mechanics, featuring only 5 actions the player can take: Moving, Looking, Jumping, Shooting, and Painting. Many commercial FPS games feature a variety of additional mechanics to allow players to develop additional skills. Introducing modern mechanics such as crouching, leaning, and aiming down a sight could impact the adaptive time delay conditions as well as cause the amount of latency applied to require a change. Game objectives such as pushing a payload, holding a capture point, or planting a bomb could require additional conditions to be met to activate adaptive time delay as well.

Section 7.3: Long Term

Although the latency smoothing algorithm can be altered to fit the conditions experienced in any FPS game, this algorithm would remain the same throughout the entire game. Implementing “adaptive adjustment”, where the adaptation rate when adaptive time delay is on also depends on certain conditions, could produce a more robust implementation of adaptive time delay that can be better suited for a particular game. For example, the two conditions for enabling adaptive latency implemented in a FPS game could be line of sight and proximity. If the player is meeting the proximity condition, the adaptation rate is over one second. When they are meeting both conditions, the adaptation rate is a half of a second. This could also be implemented with more complex smoothing algorithms and more complex game mechanics to create a specialized version of adaptive time delay that works well with a given FPS game.

A common practice in commercial FPS games is to combine latency compensation techniques together. Half-life combines the interpolation and time warp techniques to compensate for latency [13]. Implementing a first person shooter game with adaptive time delay as well as other compensation techniques could further improve the responsiveness and fairness of the game.

References

[1] 2022 essential facts about the video game industry. *Entertainment Software Association*. (2022, June 10). https://www.theesa.com/resource/2022-essential-facts-about-the-video-game-industry/

[2] Long, M., & Gutwin, C. (2018, October). Characterizing and modeling the effects of local latency on game performance and experience. In *Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play*, 285-297. SIGCHI Conference Paper Format

[3] Liu, S., Claypool, M., Kuwahara, A., Sherman, J., & Shovel, J. J. (2021, May). Lower is better? The effects of local latencies on competitive first-person shooter game players. *ACM Digital Library*. https://doi.org/10.1145/3411764.3445245

[4] Zander, S., Leeder, I., & Armitage, G. (2005, June). Achieving fairness in multiplayer network games through automated latency balancing. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, 117-124. https://doi.org/10.1145/1178477.1178493

[5] Wikstrand, G., Schedin, L., & Elg, F. (2006). High and low ping and the game of pong effects of delay and feedback. *Dept. of Computing Science Umeå University*, ISSN 0348–0542.

[6] Liu, S., Xu, X., & Claypool, M. (2022, September). A survey and taxonomy of latency compensation techniques for network computer games, *ACM Computing Surveys, Article 243, Volume 54, Issue 11S*. https://doi.org/10.1145/3519023

[7] Oluwatosin, H. S. (2014). Client-Server Model. *IOSR Journal of Computer Engineering. 16*. 57-71. https://doi.org/10.9790/0661-16195771

[8] Claypool, M. & Claypool, K. (2006, November). Latency and player actions in online games. *Commun. ACM 49, 11*, 40–45. https://doi.org/10.1145/1167838.1167860

[9] Dick, M., Wellnitz, O., & Wolf, L. (2005, October). Analysis of factors affecting players' performance and perception in multiplayer games. In *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, 1-7. https://doi.org/10.1145/1103599.1103624

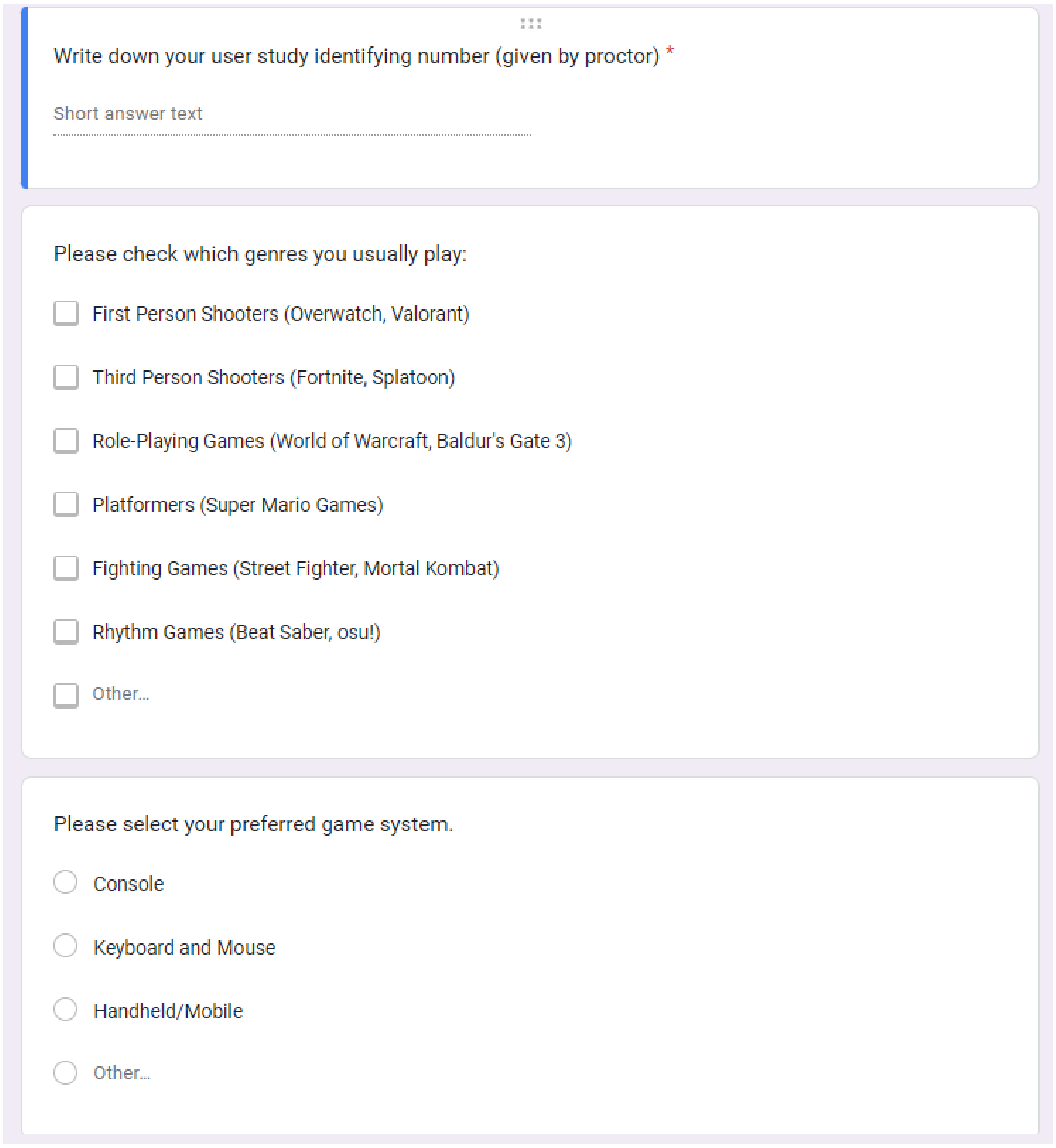
[10] Chen, K. T., Huang, P., & Lei, C. L. (2008). Effect of network quality on player departure behavior in online games. *IEEE Transactions on Parallel and Distributed Systems*, *20*(5), 593-606. https://doi.org/10.1109/tpds.2008.148

[11] Le, A. & Liu, Y. E. (2007). Fairness in multi-player online games on deadline-based networks. *2007 4th IEEE Consumer Communications and Networking Conference*, 670-675, https://doi.org/10.1109/ccnc.2007.137.

[12] Paik D., Yun, C., & Hwang, J. (2008). Effective message synchronization methods for multiplayer online games with maps. *Computers in Human Behavior, Volume 24, Issue 6*, 2477-2485. https://doi.org/10.1016/j.chb.2008.03.004.

[13] Bernier, Y. W. (2024). Latency compensating methods in client/server in-game protocol design and optimization. *Valve Developer Community*. https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization

Appendix A: Demographics Survey



Write down your user study identifying number (given by proctor) *

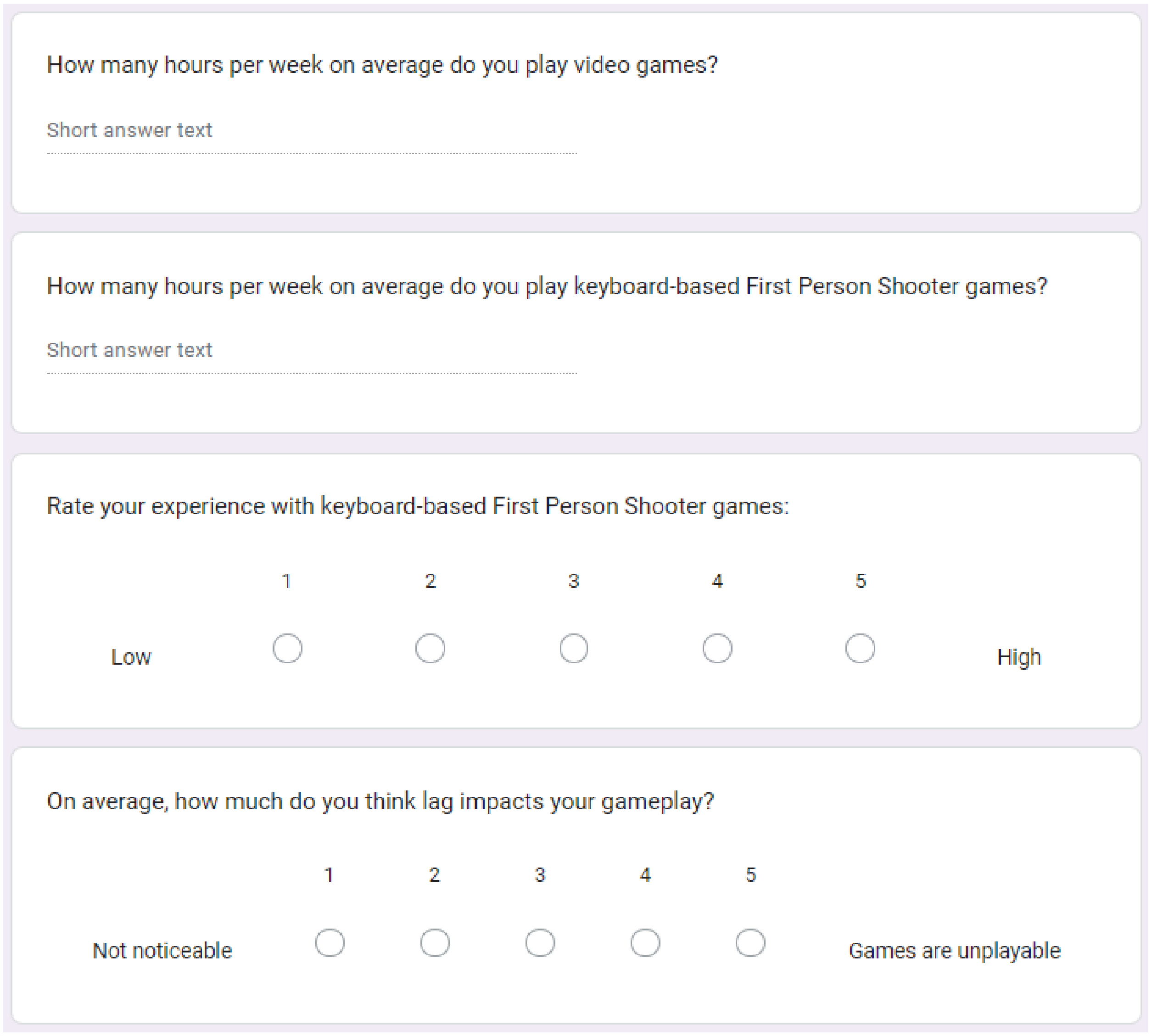
Short answer text

Please check which genres you usually play:

- ☐ First Person Shooters (Overwatch, Valorant)
- ☐ Third Person Shooters (Fortnite, Splatoon)
- ☐ Role-Playing Games (World of Warcraft, Baldur's Gate 3)
- ☐ Platformers (Super Mario Games)
- ☐ Fighting Games (Street Fighter, Mortal Kombat)
- ☐ Rhythm Games (Beat Saber, osu!)
- ☐ Other...

Please select your preferred game system.

- ○ Console
- ○ Keyboard and Mouse
- ○ Handheld/Mobile
- ○ Other...



How many hours per week on average do you play video games?

Short answer text

How many hours per week on average do you play keyboard-based First Person Shooter games?

Short answer text

Rate your experience with keyboard-based First Person Shooter games:

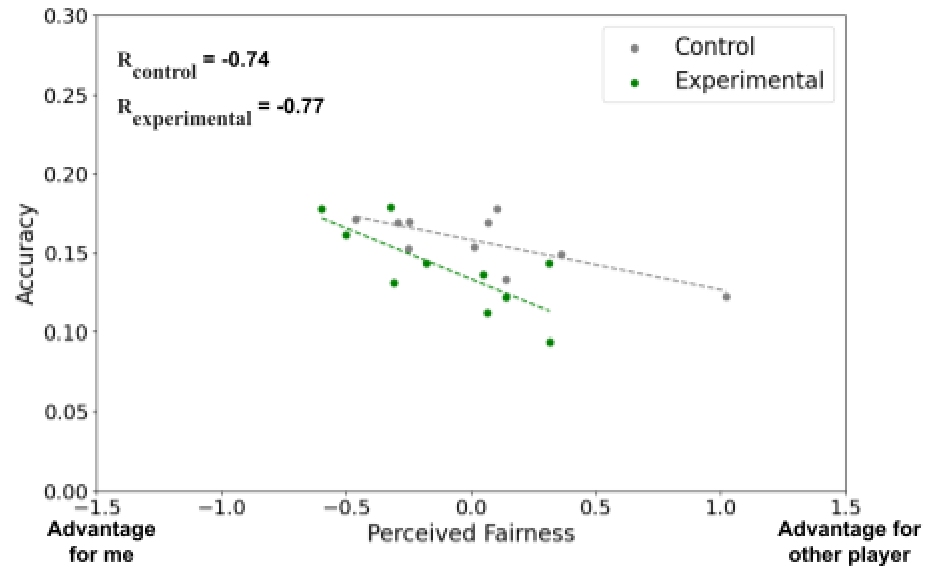
	1	2	3	4	5	
Low	○	○	○	○	○	High

On average, how much do you think lag impacts your gameplay?

	1	2	3	4	5	
Not noticeable	○	○	○	○	○	Games are unplayable

Our demographics survey was used to provide us with an idea of the skill range and experience in FPS games of our study participants. The user study identifying number allowed us to associate a participant's responses with in-game performance and questionnaire answers while preserving anonymity. Short answer questions allowed participants to provide more than a numerical value in case their answers can vary.

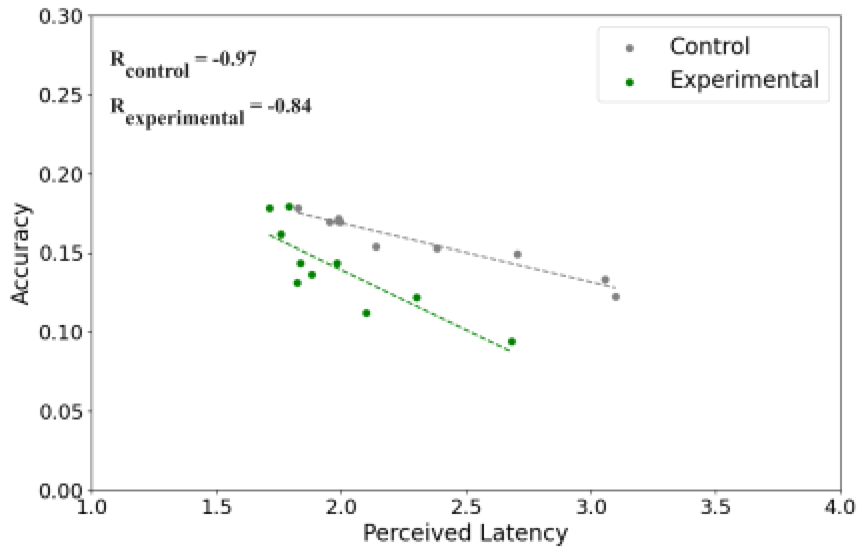
Appendix B: Perceived Fairness vs. Accuracy



This graph shows the means of perceived fairness compared to the means of player accuracy. Perceived fairness is on a scale from -1.5 to 1.5, where -1.5 means that the player thought that they were at an advantage, 0 meant they thought it was fair, and 1.5 meant they thought their opponent had an advantage. Player accuracy is how many of their shots collided with their opponent divided by their total shots taken. The gray data points refer to the control group, while the green data points refer to the experimental group.

As player accuracies decrease, players tend to see their opponent as holding the advantage in the game. This data suggest a strong negative relationship between accuracy and perceived advantage as shown by a Pearson R score less than -0.7 for both control and experimental groups.

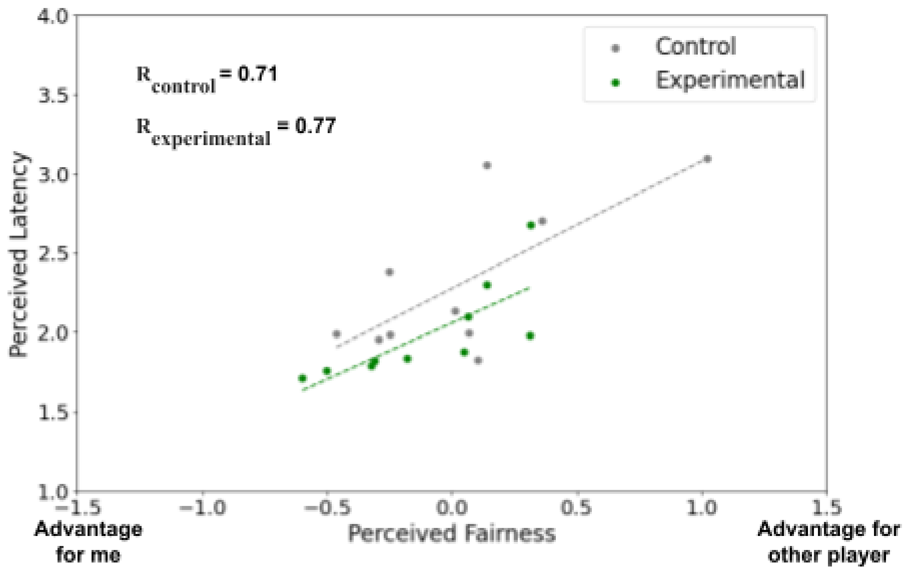
Appendix C: Perceived Latency vs. Accuracy



This chart depicts the relationship between perceived latency and player accuracy. The x-axis is the player's perceived latency, where a lower number means less latency is felt while a high number means a high latency is felt. The y-axis is the accuracy calculated by dividing the number of shots landed by the number of total shots taken. The gray data points refer to the control group, while the green data points refer to the experimental group.

Both control and experimental groups have a high accuracy score when the perceived latency is low. Likewise, a low accuracy score is accompanied by a high perceived latency. This shows a strong negative linear trend with a Pearson R score of less than -0.7 for both control and experimental groups, suggesting that players determine their perceived latency using their accuracy score.

Appendix D: Perceived Fairness vs. Perceived Latency



This chart shows the means of perceived fairness compared to the means of perceived latency. Perceived fairness is on a scale from -1.5 to 1.5, where -1.5 means that the player thought that they were at an advantage, 0 meant they thought it was fair, and 1.5 meant they thought their opponent had an advantage. Perceived latency is on a scale from 1 to 4, with 1 meaning the player felt little, if any, latency, and 4 was the maximum amount of latency they could feel. The gray data points refer to the control group, while the green data points refer to the experimental group.

This chart shows that the more latency a player felt in the game, the more they felt their opponent had an advantage. This relationship is a strong positive linear trend with a Pearson R of greater 0.7 for both control and experimental groups.