Analyzing the Effects of Time Warp Latency Compensation and the "Shot-Around-the-Corner" Problem on a First-Person Shooter Game

An Interactive Qualifying Project Report

By:

Mattheus Faria
Miles Gregg
Jonathan Hsu
Pari Nguyen

Project Advisor:

Mark Claypool

Graduate Mentor:

Samin Shahriar Tokey

Sponsored by:

Abstract

When playing multiplayer First-Person Shooter (FPS) games, players often encounter latency that can degrade the gaming experience. Game developers typically implement latency compensation techniques such as Time Warp so players can perform actions as if there was no latency at all. However, there have been few attempts to quantify the effect that these latency compensation techniques have on player performance and experience. Our project explores how players are impacted by Time Warp, specifically the accuracy impact and the shot-around-the-corner problem. We have human users play multiple rounds of an FPS test environment against a custom bot at varying latencies with Time Warp alternating between on and off. Based upon the study results, Time Warp increases the prevalence of shots-around-the-corner as the latency between the shooter and server increases. When told the network conditions, human players report an improved quality of experience when Time Warp is on, and an increased sense of "fairness." Accuracy decreases as latency increases, with Time Warp having minimal effect on accuracy regardless of if it is on or off.

Table of Contents

1) Introduction

For two people to play together on separate systems, most games use a central server that manages the game and allows players to connect to that server using a client, sharing information back and forth through the server. The delay between sending information between clients and servers is called latency.

More specifically, latency is how long it takes information (in the form of data packets) to travel from one network node to another, such as a client to a server. High values of latency can be caused by network congestion, low-capacity links, and the distance between the client and server leading to increased time needed for packets to travel between these nodes.

High latency values are generally associated with poor gameplay experiences, as this delay can cause a player's game state to be behind the game state of another player. Take for example a First-Person Shooter (FPS) multiplayer game, with player A experiencing a higher latency than their opponent, player B. Both players A and B can be moving into a corridor, but because player A is experiencing higher latency, player B will be able to see and shoot player A before player A's client can display player B on their screen. While game designers and network conditions overall have brought this delay down to generally a few hundred milliseconds at its maximum, an average FPS player can detect and shoot another player in under half a second, making this smaller-than-seconds time crucial to both the player's performance and gameplay experience.

Hardware upgrades can serve to reduce latency for end users, however, it is not feasible for developers to expect users to procure better equipment. Thus, various software methods can be used to account for differences in latency values between each client and the game server, called latency compensation techniques.

There are a variety of latency compensation techniques that are designed to minimize the amount of perceived latency. Time Warp is a widely implemented technique that is used to mitigate the impact of differences in network latencies between game clients. In a game with an authoritative server, an implementation of Time Warp would have the server create a separate game state that takes into account the latency between itself and the clients, and place player positions as if they had no latency. After this, game events such as shots fired from a weapon would be registered in this game state, and the effects would be applied and sent to the clients.With Time Warp, players do not have to "lead" their shots ahead of their opponent's velocity vector in order to hit them at higher latencies. However, Time Warp can lead to a phenomenon known as the "shot-around-the-corner". When the authoritative server registers a shot on a player even after the player that has been shot has just moved behind cover. Both the

benefits to aiming and shooting and the “shot-around-the-corner” problem are well-understood, however, quantifying the impacts and user's perception of Time Warp has yet to be thoroughly explored.

The results of our study show that the implementation of Time Warp leads to more instances of shot-around-the-corner. There was also a positive relationship between Time Warp being on and participants perceiving an increase in game fairness. Participants also noted that lag decreased with Time Warp on, meaning that the implementation of Time Warp was beneficial towards assisting perceived hit rate. In terms of recorded accuracy, there was an overall decrease in accuracy as latency increased. Time Warp had minimal-to-no effects on accuracy when turned on compared to when turned off.

In this study, we analyze the effects of Time Warp on player gameplay experience and the shot-around-the-corner problem using data gathered from users while they played through multiple sessions of the FPS game FPSci under various network conditions. Chapter 2 provides a thorough background to the terms presented in our report, as well as introduces FPSci. Chapter 3 goes into more detail surrounding the environment and methods we use to carry out the experiment, including the map, experiment and the pre-programmed bot, which controlled one of the clients connected to the server. We analyze our results in Chapter 4, focusing on accuracy, shot-around-the-corner, and player perception. Chapter 5 includes possible extensions to our work should it be continued in the future, and Chapter 6 concludes our report.

2) Background

This chapter focuses on the details of our modified FPSci game, along with our latency compensation techniques. The first section describes FPSci and the features that have been implemented or used for our study. The next section describes our latency compensation techniques and what they actually do. The last section provides details on ‘the shot around the corner’ problem that arrives from one of our latency compensation techniques, Time Warp.

2.1 FPSci

FPSci is an open-source tool initially created and developed by Nvidia in order to study FPS Esports aiming. The game contains many different customizations and can be changed for various purposes. The modified version of FPSci utilized for this study has a server-authoritative multiplayer server-client setup and an implementation of Time Warp, the latency compensation technique which we test.

Figure 1: In-game Screenshot of FPSci

In FPSci, clients connect to a separate server. Players then hit “TAB” in order to ready up and begin a round. For our version of FPSci, we chose to change the model of the character from the default sphere to a thin pill shape. The sphere shape was too large in radius and was an easy target to hit and by changing the model to a pill shape, we were able to make the game a bit more challenging so that players would not effortlessly hit all of their shots.

The purpose of changing the model is two-fold: to provide a greater potential for players to be impacted by latency and Time Warp, as well as have the mode more closely resemble those found in commercial games.

The default FPSci weapon is a semi-automatic rifle that has a refresh time after every shot. This refresh time is shown by a circular reloading icon in the center of the player's screen. The weapon was not visible on-screen in our testing environment.

Figure 2: Reloading Icon After Shooting

2.2 Time Warp and Latency Exposure

In our study, we analyze two well-known latency compensation techniques: Time Warp and Latency Exposure. These techniques are implemented to help players deal with latency at high round-trip values.

Time Warp, sometimes called rollback or even simply referred to as lag compensation, is a widely implemented technique where, before executing a player's action such as shooting, the server warps the world state back to when the player would have issued this command.

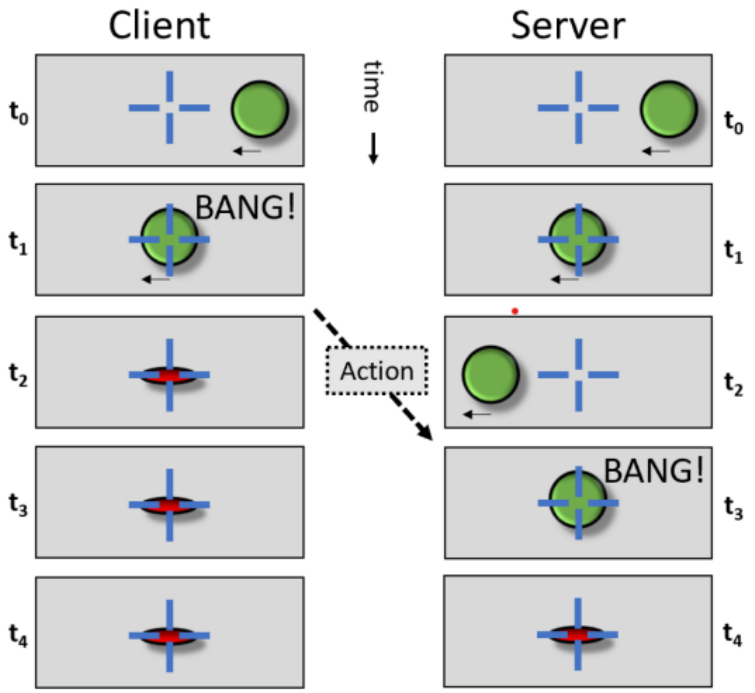

Figure 3: Example of Time Warp [1]

In Figure 4, we see a visual representation of the effects of Time Warp from both the perspective of the client and server. As the client on the left fires a shot at time t_1, the server accounts for the movement of the target player in the time it takes for the command to reach the server. Thus, it applies the effects of the shot when it receives the command at t_4, applying it on a "warped back" state where the target player's position more closely resembles that on the shooting client's screen at time t_1.

Latency exposure is an indicator to show the latency that the player is currently experiencing, or a value corresponding to its magnitude. This indicator was present in all trials that we ran for our user study.

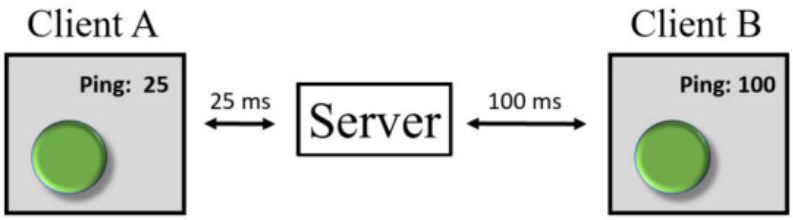

Figure 4: Example of Latency Exposure[1]

2.3 Shot Around the Corner Problem

The shot-around-the-corner problem is a well-known side-effect of Time Warp and has been observed in many FPS games. This problem occurs when two clients face off around a corner and there is a significant difference in latency between the two players. With Time Warp implemented, a player on client 1 may see the player on client 2 and shoot them in the open. For the player on client 2, they may believe that they have already made it behind cover before the player on client 1 shoots. Due to Time Warp's features, the server takes the action from the player on client 1 and applies it to a "warped back" game state where the player coordinates of the players are altered to account for latency. This may lead to some inconsistencies in FPS games, as this "warped back" state can result in a game state that benefits one player over another, but generally favors the shooter.

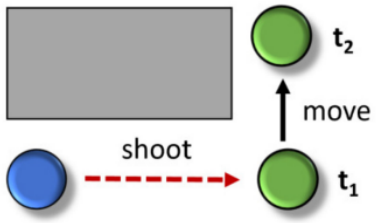

Figure 5: Example of the Shot-Around-The-Corner Problem. In this diagram, the blue player model represents client 2 in our experimental setup. At t=1, client 2 sees client 1 at the position labeled "t_1" and fires, however, client 1 has already moved to the position labeled "t_2" from their perspective[1]

This can result in a perceived shot-around-the-corner instance, as the altered positions to which the shot action is applied could bring the running player out of cover, causing that player to die while behind cover from their own point of view. By analyzing the times the movement in Figure 6 occurs in FPSci, we calculate the amount of instances of shot-around-the-corner as well as recording the participant's perception of shot-around-the-corner.

3) Methodology

This chapter goes over our experimental design and setup. We explain what experimental conditions the player was tested under, what the player did in our experiments, and the specifics of our experiments and independent/dependent variables. We explain the map that we have built for our version of the application and how our latency compensation technique, Time Warp, was tested with the different players on our map.

3.1 Hypotheses

One of the main reasons that FPS games implement Time Warp is to help compensate for latency when a player is shooting at a target.

> **Hypothesis 1:** With Time Warp enabled, the accuracy of a player will increase overall at higher latencies.
> **Hypothesis 2**: With Time Warp enabled, the accuracy of a player will decrease overall at low or negligible latencies.

Time Warp was expected to help players aim more accurately when there is latency as a factor. As we progressed to the higher latency values, we believed that Time Warp would help to a greater degree and that we would see a much larger increase in accuracy than at the lesser latency value.

Since Time Warp leads to instances of the shot-around-the-corner problem, we predicted that there would be more instances of the shot-around-the-corner problem with Time Warp enabled. We also believed that the value would likely increase proportionally with latency, due to the difference between the game state created for the Time Warp implementation on the server and the game state perceived by the clients being increasingly different as latency increases.

> **Hypothesis 3**: With Time Warp enabled, instances of the shot-around-the-corner problem will increase as latency increases.

3.2 Experiment

To run our experiment, we utilized a fork of NVidia's FPSci code modified by Samin Shahriar Tokey. This code consists of a server and a client for each of the two players participating. The code for adding "Time Warp" to Tokey's modified build of FPSci was provided by a separate MQP team who created and tested different latency compensation techniques.

3.2.1 Roles

To test shot-around-the-corner, we need a runner who is trying to run behind a corner, and a shooter to try and hit the runner as they run. Our study consisted of these two roles, where we planned to have each client play 20 rounds as one role, then swap for another 20 rounds as the other for a total of 40 rounds per participant. The starting role that our participants received depended on which client we connected with first. As the runner, the player was asked to run back and forth across the horizontal hallway without being shot by their opponent. As the shooter, the player was asked to shoot the runner opponent as they made their attempts to cross the hallway.

3.2.2 Map

We were able to create our own map and import this into FPSci to use for our studies. The map consists of a vertical hallway between the two players and a horizontal hallway for the runner player role. This vertical hallway had a length of 30 meters and a width of 10 meters. For the runner, there was also 30 meters of space to move left and right. The movement speed was set to be 7 meters per second, so the runner would take at least 1.4 seconds to cross the gap while visible to the shooter. We adopted this map design to better test shot-around-the-corner. With the shooter role being at the very end of the hallway, the runner would attempt to run back and forth between the horizontal hallway. The shooter would then shoot the runner as many times as possible to gather data on the effects of latency. We also test shot-around-the-corner by introducing two corners that the runner could pass with the hope of generating data regarding possibly getting shot around the corner.

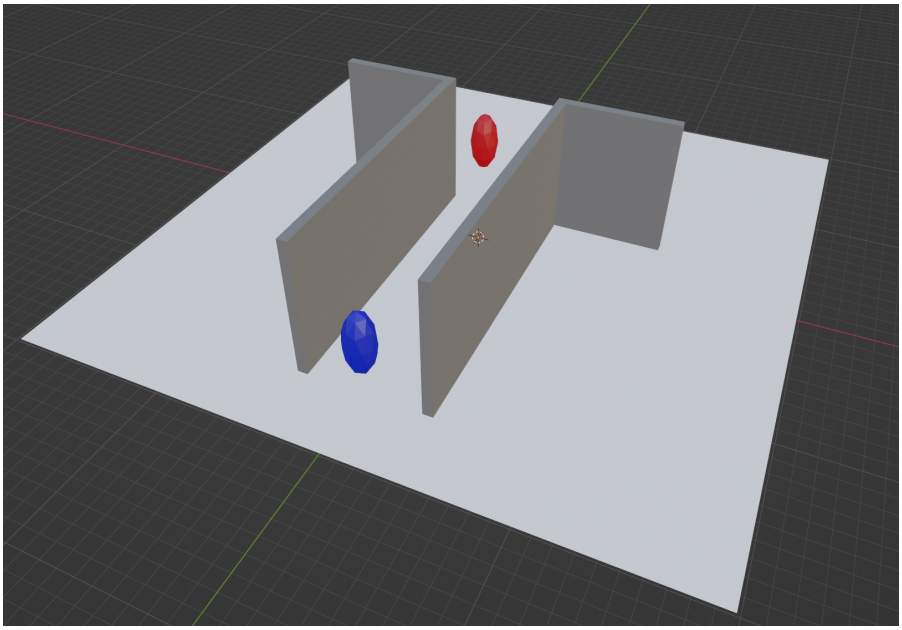

Figure 6: A diagram illustrating the layout of the experimental map. The blue icosphere marks the position of the shooter, while the red indicates one potential position of the runner as it moves along the X-axis (marked in red)

3.2.3 Experiment Design

The experimental setup allowed us to analyze the shot-around-the-corner problem. If a ray is cast from the shooter to the runner's server-side location at the same time that the shot is registered by the server, it allows us to see if the shot would have landed on the runner. With this experimental setup, we are able to take an objective approach to see if Time Warp latency compensation has a measurable effect on the shot-around-the-corner problem, and how many instances of the problem it introduces into the game environment.

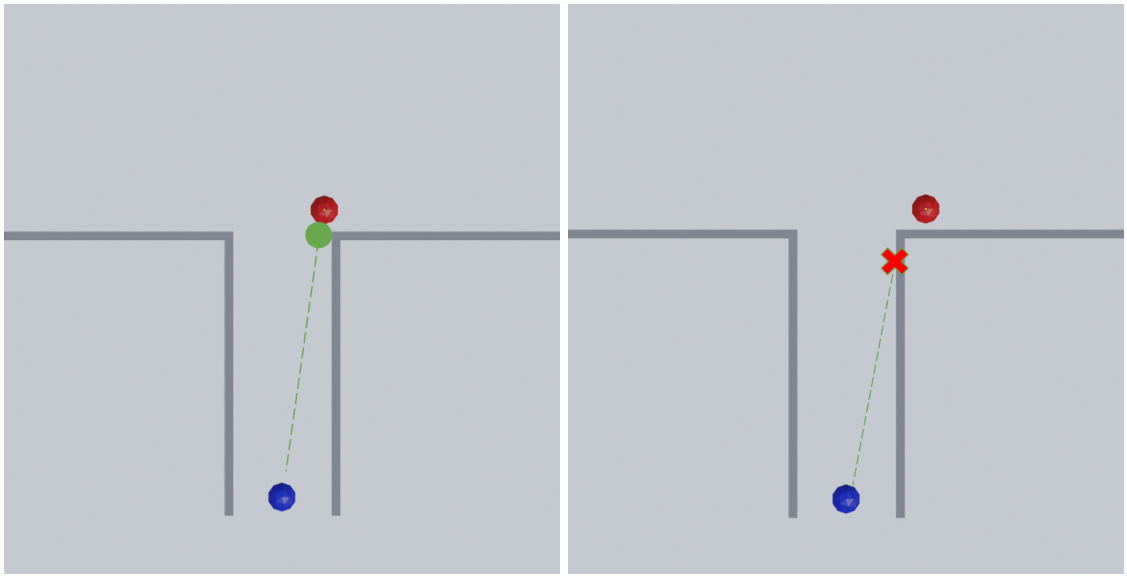

Figure 7: At the present gamestate, the two game clients have differing perspectives of the gameworld. On the left, the blue player is able to see the red player and expects to be able to shoot them with a clear line-of-sight. However, without latency compensation measures to benefit them, the blue player's shot will tend to miss at higher latencies, as the true position of the red player will be further to the right. The red player's perspective (right) places them as already having moved behind the cover of the wall, where they expect to be safe and out of the line-of-sight from the blue player.

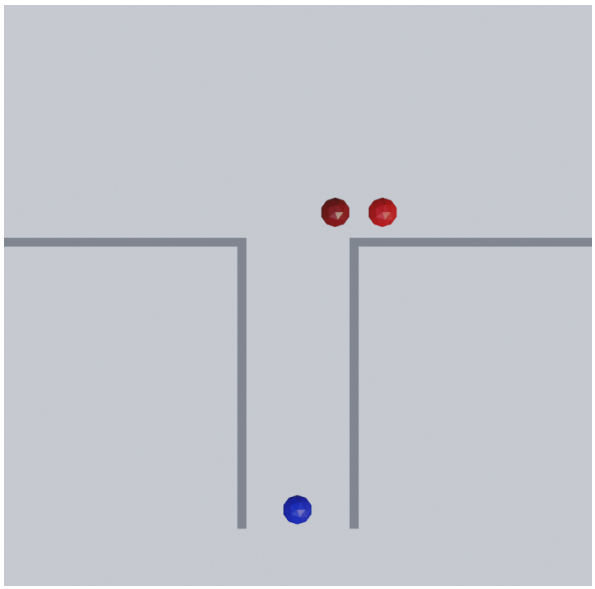

Figure 8: With Time Warp compensation enabled, a gamestate is created in which the player is still running across the hallway, and is visible to the blue player, despite the true location of the red player being behind the wall. The blue player will shoot and kill the red player, and these changes to the gamestate are then applied.

Data logging occurred in the form of the server and client computers recording the players' position and heading as well as the result of every shot fired. In-built tools allow us to log directly to the database file whether or not a resolved hit occurred as a shot-around-the-corner for a client. This can then be compared to the user's perception as indicated in their post-round survey, to see if what occurred within each round matched their perceived outcomes.

A single weapon was used throughout the study and was constant for every tester. The weapon type was explained to the tester and its semi-automatic nature was introduced, along with the visual refresh time indicator.

For our study, we chose to base our player speed on a current and popular first-person shooter, Valorant. The base Valorant movement sprint speed is about 7.0 m/s, and we used this for our character movement speed as well.

In our version of FPSci, we have visual indicators to show both latency exposure and that Time Warp is either on or off. In the top left corner of the game, the latency is shown in milliseconds to each player. The presence of a letter 'T' indicates when Time Warp is active for the player.

3.3 In-Game Bot Setup

Two bots were created in order to simulate the gameplay movement within the FPSci game to reduce the variability that might arise during gameplay to minimize inconsistencies in our analysis after the user studies. This was done in order to reduce the amount of human variability in our data gathering, eliminating factors that could detract from accurate data gathering such as the learning effect in which human users improve over time throughout the testing session. Running and shooting bots were made in order to recreate actual human movement during gameplay. Both bots are controlled using a master Python script that contains the code for both the bots, switching from shooter bot and runner bot depending on what role the client the bot is running on is currently fulfilling.

The first bot created was a "shooter" bot that autonomously shot at players as they neared the two corners of the hallway. This was done by scanning an array of 5 pixels under various conditions in order to determine where the bot should aim. In addition to this, the script would read a file to determine the network conditions currently being tested in order to take this into account and aim accordingly.

The second bot that we made was the "runner" bot that would read from an input JSON (JavaScript Object Notation) file and then perform those inputs by controlling the computer's keyboard and mouse movements. These random movements were made before the user studies

were run to ensure every user had the same opposing bot movements, this was very important to reduce inconsistencies between different users.

In order to account for the change in latency and the toggling of Time Warp being on or off, the bots need to be able to read the values of a round. While the initial instance of FPSci did not have a simple way to actively record this data for it to be used in real-time, we were able to implement an actively updating file that recorded the latency and state of Time Warp in a text file named “clientProperties.txt.” Our Python script could then read this updating text file to correctly set parameters that would change with different latencies, such as the offset that the shooter bot would use to lead their shots at varying lengths. The text file also records which role the clients are in, which is used by the overall master bot script to determine which bot code to use; when the text file states “RUNNER,” the bot would use its runner bot code and vice versa.

3.3.1 Shooter Bot

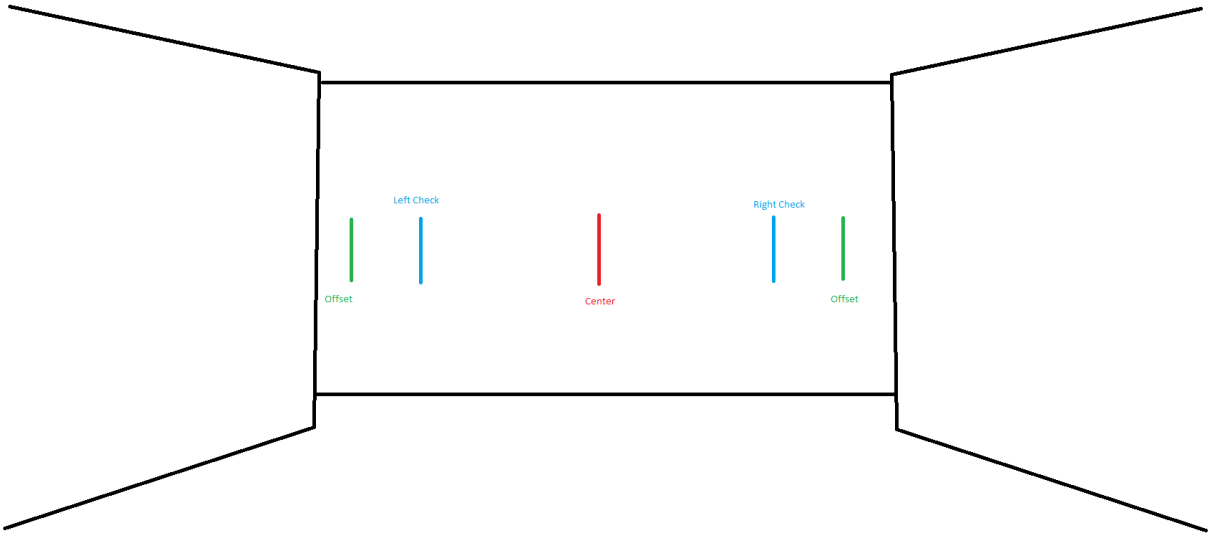

Figure 9: Diagram illustrating the points of interest for the shooter bot

As the runner passes the center point outlined in red, the script then begins to scan the two pixels labeled in blue. This allows the bot to determine whether the running player is committing to a run toward the left or right side of the hallway. After the runner passes through one of the pixels labeled in blue and is detected by the bot, the script then produces a mouse input that aims the crosshair at either the left or right pixels labeled in green, depending on which pixel in blue the runner has passed through.

As latency increases, the two “offset” points move further outwards due to the need for the bot to compensate for this increased latency. This mimics a human player’s learned approach of “leading” shots at higher latencies. With Time Warp on, this outwards movement offset was

decreased. In addition to this offset, a timed delay was introduced when Time Warp was active. This results in the shooter bot pivoting the cursor to the predetermined offset point and then waiting for a specified time before firing. This delay allows for the server-side orientation of the bot to fully reflect the shooter's client-side orientation before the shot is taken on the client.

3.3.2 Runner Bot

The goal of the runner bot was to set up random movement for the participants in the user study. These movements would simulate another person playing the game except the movements were the same for all of the participants throughout the whole user study. It was important to have the same movement for all of the participants to get equal results for our data analysis on the participants' movements.

Runner bot movement was controlled by a library in Python called pyautogui, this library controls human inputs on the computer which are mouse and keyboard controls. Using this library we were able to reenact human-like inputs from the mouse and keyboard in order to make the player model move within the game. This helped us achieve the overall goal of setting up a bot to simulate an actual human movement during user studies.

The runner bot was programmed so that in every round of the game the bot would be given a new set of command movements. These command movements were made beforehand by another Python script that we developed. From this Python script, it would choose random movements from the random.uniform(0, 1) function in Python, if the result was less than 0.5 then the bot would move left and if the result was above 0.5 then the bot would move right. From this, we were also able to choose random durations for the bot to move in those specified directions which ranged from 0 to 5 seconds long. We made each round have 15 movement commands in order to exceed the round times which we have set to 20 seconds. Then we made 100 possible match configurations for each of the 15 different movement commands. All of this data was stored in a JSON file that we would read from in the bot code during the live gameplay. Figure 13 below shows how the data was formatted inside of the JSON file, it was formatted this way in order to read easily from the Python script. When reading in the movements from the JSON file the command direction would be mapped to specific keyboard keys which are ("left" = a keyboard key) and ("right" = a keyboard key). The time is the variable after the dash which will be extrapolated from the string and then used within the handler thread to press that key for that specific duration.

```
{
    "tests":
    [
        {"test 1":
        [
                "right-2.38",
                "left-2.97",
                "right-2.88",
                "left-2.72",
                "right-3.48",
                "left-3.07",
                "right-3.7",
                "left-4.0",
                "right-3.16",
                "left-3.14",
                "right-3.2",
                "left-2.79",
                "right-3.98",
                "left-2.44",
                "right-3.93"
        ],
        "more...":
        [
                "more..."
        ]
        }
    ]
}
```

Figure 10: JSON Storage of Running Bots Movements

The runner bot was designed with multithreading in place to run the commands while looking for specific changes on the screen in order to see the current status of the bot. We iterated to a new set of bot movements once we saw that the user participant landed a hit on the bot. Since we could not get any input from the game directly, we had a colored wall to show our program when the player was shot and respawned. The first thread in the script would look at a pixel in the middle of the screen (floor(height/2), floor(width/2)) and would look at RGB (168, 175, 25) which is our yellow wall on the map. We chose yellow because of its distinct color compared to the rest of the map. Once the bot saw this yellow wall we used a queue pipeline to tell the other thread that the bot had been shot and it needed to continue to the next movement routine. The two threads are called the detector and handler in which the detector thread detects the pixel color change to yellow and the handler thread handles the current reading from the JSON file.

```python
def detector(out_queue):
    curr_img = 0
    last_color_seen = None
    yellow_seen = False
    prev_yellow_seen = False

    latchedBoolean = LatchedBoolean()

    while True:
        pixel_color = getpixel(1920//2, 1080//2)
        if latchedBoolean.update(pixel_color == (168, 175, 25)):
            out_queue.put('reset')
            out_queue.put('start running')

def handler(in_queue, config):
    global found
    found = False
    test_iteration_index = -1

    while True:
        data = in_queue.get()
        print("new command incoming: ", data)

        if data == 'reset':
            test_iteration_index += 1

        if data == 'start running':
            current_test = config['tests'][0]['test ' + str(test_iteration_index+1)]
            for i in range(len(list(current_test))):
                split = current_test[i].split('-')
                press_button(key_mapping[split[0]], float(split[1]))
```

Figure 11: Running Bot Thread Functions

The two functions are run concurrently by using the built-in threading library in Python. Once the functions are running together in sync they communicate all commands with one another through the queue pipeline in order to keep fluid control over the bot's movements.

```python
# setup threads
q = Queue()
t1 = Thread(target = handler, args =(q, config, ))
t2 = Thread(target = detector, args =(q, ))
t1.start()
t2.start()
```

Figure 12: Running Bot Thread Startup Concurrently with Queue Pipeline

3.4 Testing Environment

The hardware used for this user study allowed us to run the FPSci server and clients at a monitor refresh rate of 144Hz. Given that one of the connected clients would be controlled via a Python script, measures were taken to ensure that the program was structured in a way that allowed for the threads to also run at least as quickly as each frame was processed. All other factors between the two clients were kept identical, including computer specifications, and the models of peripherals (mouse, keyboard, and monitor) provided for each client. Test participants were not informed of the nature of their opponent, and were generally unaware that they were playing against a bot client. The two client and one server computers all ran on the same specifications: a Windows 11 installation alongside an Intel Core i7-8700K Processor, NVidia GTX 1080 GPU, 64GB of RAM (Random Access Memory), and Lenovo 240Hz monitor. In addition to this, the participants used a Logitech G502 mouse set at 800 DPI.

3.5 Experimental Parameters

Before the start of our user study, we isolated the independent and dependent variables that we wanted to control and analyze. To study the effects of Time Warp and latency in a variety of conditions, we varied these two parameters, with players participating in trials with 0, 60, and 120-millisecond round-trip-times and with Time Warp enabled or disabled. In addition to this, the role of the player is varied between shooting and running, in order to allow for analysis of their perception of the quality of their experience.

Independent Variables	Dependent Variables
Introduction of Latency Compensation	Player Accuracy
Latency	Shots Expected
Player Role (Shooting vs Running)	Quality of Experience

Table 1: Experimental Variables

3.6 Demographic and Quality of Experience Questions

Prior to the start of each user study session, participants were asked to perform a reaction time test, as well as a demographic survey. This demographic survey consisted of three questions. We asked participants to self-report their video game experience, as well as FPS game experiences. They were also asked to provide us with a list of FPS titles that they play or have had prior experience with. This allowed us to collect the average reaction times of our

participants along with their levels of experience that they have with FPS games. We wanted our study to have participants who have higher levels of experience so that players' capabilities would not play a huge role in our data collection.

To implement the experiment on the effects of "Time Warp" as a latency compensation technique, we designed our experiment to have randomly created experimental scenarios, varying the amounts of latency and Time Warp being on and off for each round. At the end of each round, our participants were asked a few questions. Questions asked at the end of the round consisted of three different questions that were related to fairness and gameplay of that recent round. These questions helped us observe whether the implementation of Time Warp helped compensate for the amount of latency that players had to deal with. It also helped us determine whether or not Time Warp increased the amount of shot-around-the-corner instances that players believed they were experiencing.

3.7 Recruiting Participants

Recruitment of study participants initially began with securing 40 participants for our study. Of those 40, 17 were able to successfully participate in and complete the tests. The reason for the low number of participants that were able to successfully complete a valid testing session is due to bugs that arose in the development of the Time Warp implementation as well as automated match-making. Due to these issues, the testing procedure had to be repeated twice. In terms of the actual recruitment process, we sent out an email across the computer science, data science, and electrical and computer engineering major general bodies. This initial email helped us gauge the amount of interest that potential participants had in our study. As an incentive, to increase interest, we let all recipients know that they would be receiving a $10 gift card of their choice for participating in our study. We received a large number of responses and decided to add a cap of around 40 participants. A second email provided this entire group of potential participants and had them sign up for a scheduled slot in our study. There was a limit of 40 slots that would help us cap the participant number. Again, due to the bug that was found, a third of our data collection was invalid. Overall, we were able to test 17 participants on our updated build of the game to collect valid data.

4) Analysis

This chapter analyzes all of our data from our user study to test our hypotheses. We present the participant demographics and the data that we have retrieved from the game's database files in various graphs to be analyzed, such as accuracy, shot-around-the-corner and quality of experience statistics.

4.1 Demographics

For our final dataset, in total, there were 17 participants that completed valid test sessions. Before we started each user study session, we had our participants fill out a demographic survey. We asked players to rate their FPS game experience on a scale of 1-5 with 5 representing the greatest amount of experience. We also asked our participants to list the different types of FPS games that they had played in the past. In figure 16, we can see that all of our participants had FPS experience with 37%, 19%, and 44%, reporting levels 3, 4, and 5 respectively. In addition to this, the provided lists allowed us to see which FPS games were most commonly played by the study participants. Notably, many of the participants indicated that they had played CSGO or Valorant, where shot-around-the-corner instances are perceived to be quite common. This also aligns well with our study, as the movement speed of the players was set to 7 meters per second, similar to the base movement speed in both of these titles.

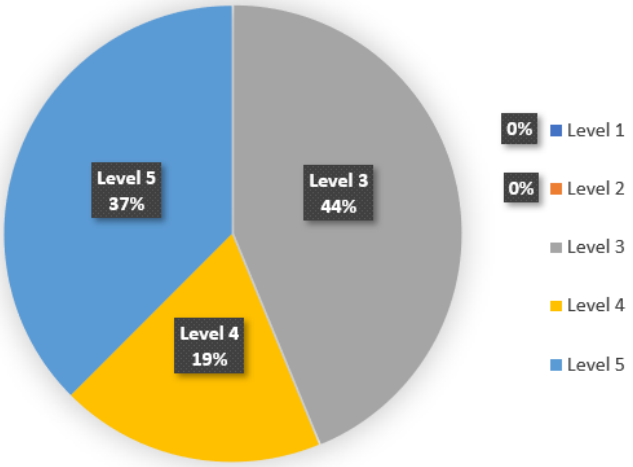

Figure 13: FPS Experience Information

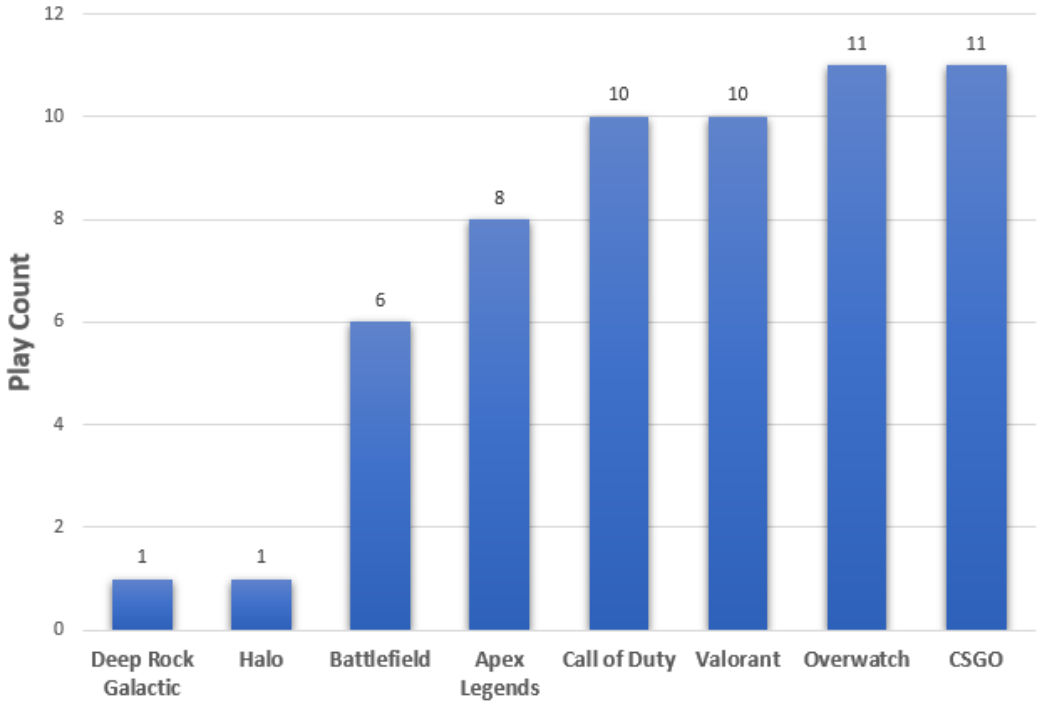

Figure 14: Played FPS Games

4.2 Experimental Data and Observations

4.2.1 Analyzing Player Accuracy

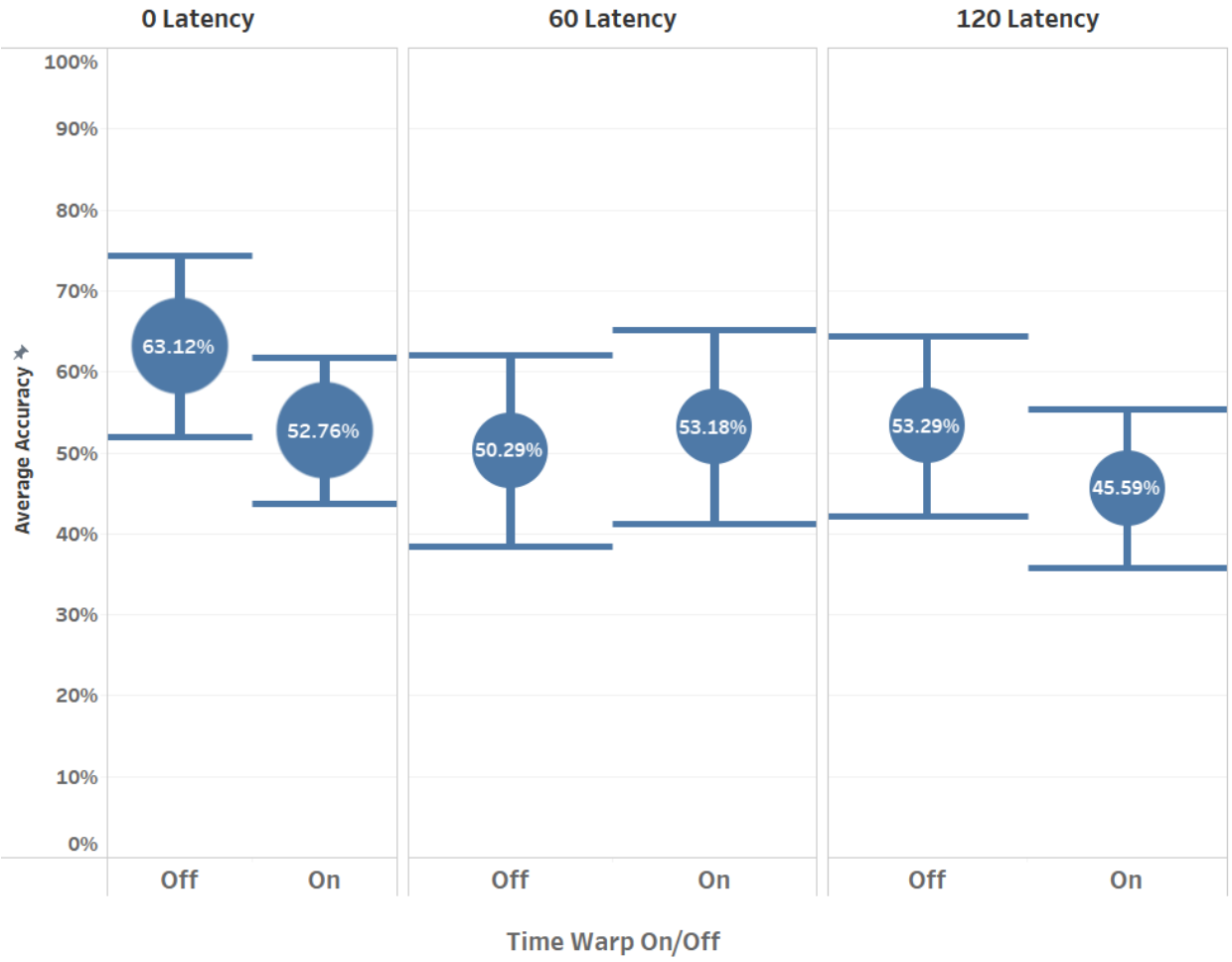

Figure 15: Accuracy versus Latency.

Before addressing the analysis of the data gathered, we must first take note of a discrepancy in the latency that was reported by each client. Due to a known bug in FPSci, on the very first instance of a shot occurring on either client, there is a 3-4 frame delay, resulting in roughly 50 ms of additional lag. Due to the implementation of Time Warp present in our build, this additional 50 ms offset results in the shooter's game state updating relative to the server's game state with an additional delay of the aforementioned 3-4 frames. The reported latency of the shooter client has an additional 50 ms offset above the latency values reported in our graphs.

There is a decrease in accuracy when Time Warp is on at a round-trip latency of 0 ms. This is to be expected, as there should be no effect from latency compensation for a 0 latency environment.

At a round-trip latency of 60 ms, there is a slight decrease in accuracy compared to latency 0, with the accuracy increasing with Time Warp on; both of which are expected with respect to our hypothesis. However, the difference in the average accuracy between the tests conducted at 60 ms latency with Time Warp on and off was much smaller than expected. It was small enough to be explained by a margin of error that could have occurred during testing, thus this increase in average accuracy at 60 ms with Time Warp active is negligible.

We believe these values could be due to the structure of our experiment design, as our experiment swaps Time Warp on and off after every round, meaning the participants potentially could not get acclimated to the shooting characteristics with Time Warp on or did not have adequate time to perceive a change at all. It could also be due to a flaw in the implementation of Time Warp that was used in testing.

Some unexpected values were observed at a round-trip latency of 120 ms, where the average accuracy is nearly equal to the average accuracy at a latency of 60 ms. In addition to this, the average accuracy at a latency of 120 ms for the shooter decreased once Time Warp was active, compared to the accuracy without Time Warp at the same latency condition. These outcomes did not match our initial expectations for our hypotheses.

4.2.2 Analyzing Shot-Around-the-Corner Data

For each instance of a hit occurrence in each session's server data, the database saved a variable displaying whether it detected a shot-around-the-corner on the hit or not. After processing this data and creating a visualization of each shot, we determined that the database calculation was often incorrect, as most of the hits deemed shot-around-the-corner were hits when the runner was in clear view of the shooter, sometimes even directly down the hallway when being shot.

We calculated each instance of shots-around-the-corner by casting two rays from the shooter to the sides of a runner for each hit instance. Using the hallway walls as two different line segments, we calculated the number of times that the shooter-to-runner rays intersected with the wall rays, which were called instances of shot-around-a-corner.

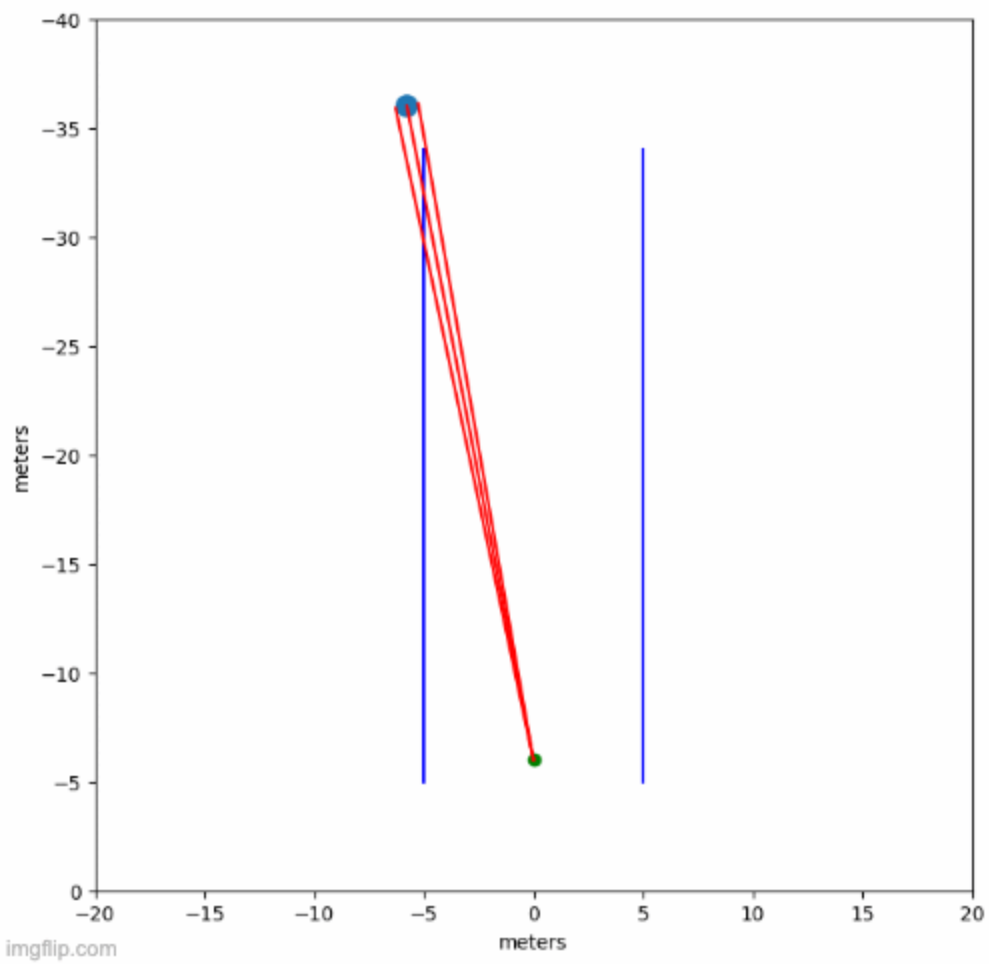

Figure 16: A visualization of the rays cast in order to determine shots-around-the-corner. This is a graphical representation of our map, with the green dot representing the shooter and the blue dot representing the runner.

Shots-around-the-corner should not occur when Time Warp is off, as no server-side calculations occur to apply shots to altered player locations. We confirmed that when analyzing the updated counts, instances of shots-around-the-corner never occurred when Time Warp was off, which is to be expected.

Using the number of shots-around-the-corner generated in Figure 19, we calculated the percentage of shot-around-the-corner instances per hit occurrence, and then took the average of each session, separated by latency values.

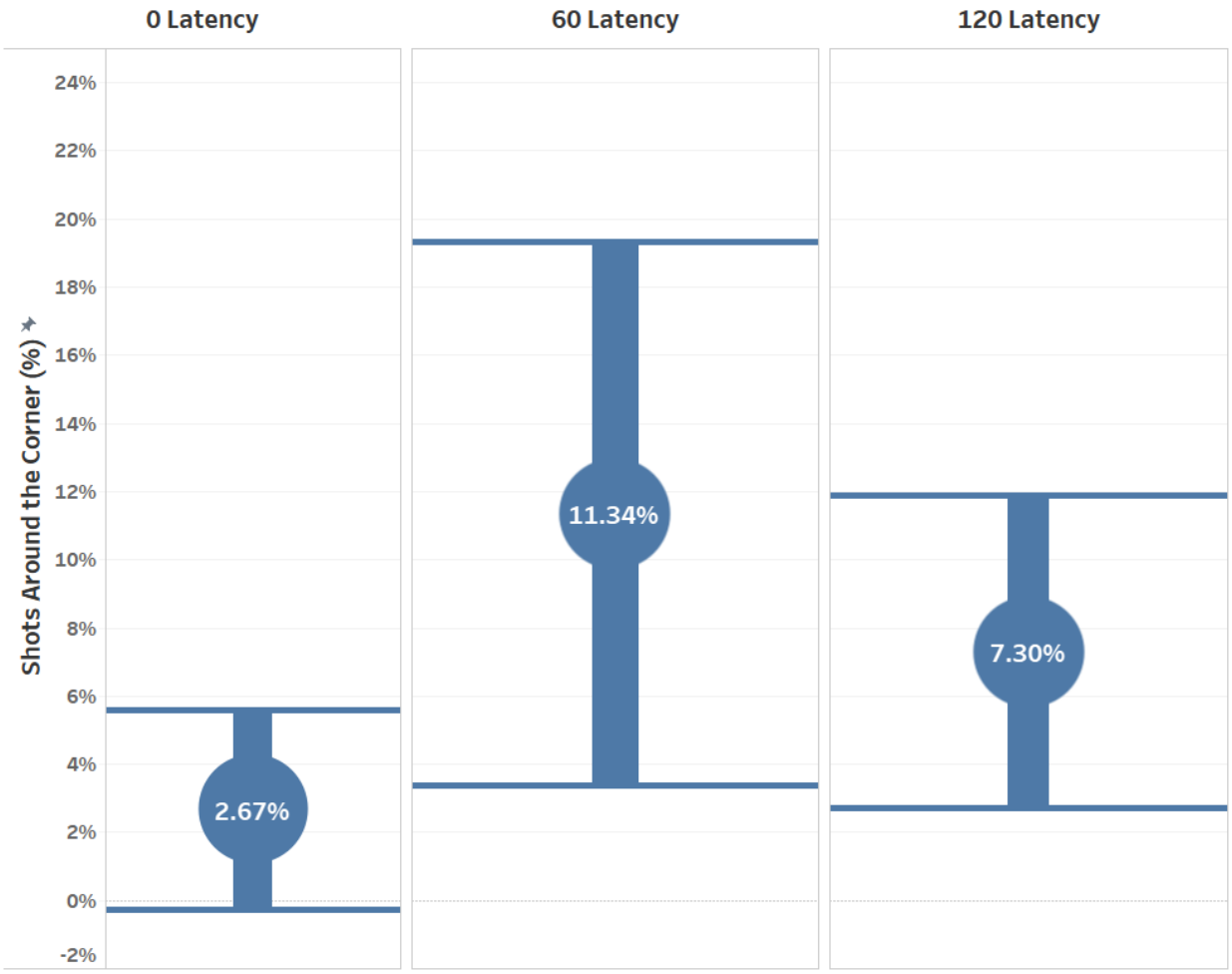

Figure 17: Shot-Around-the-Corner versus Latency. The values are a percentage of how many hits were labeled as "Shot-Around-the-Corner."

According to Figure 20, our values somewhat reflect the expected outcome, which was an increase in shots-around-a-corner when latency values increased while Time Warp is active. However, there is a large spike in shots-around-a-corner when Time Warp was on for a round-trip latency of 60 ms as opposed to when the latency was 120 ms, which was not expected. The expectation was for there to be the most instances of shots-around-a-corner at the highest latency with Time Warp turned on, as this would result in the greatest difference between the game state created server-side in order for Time Warp to function, and the game state that was currently being perceived by the running client.

One possible explanation for this discrepancy was observed in the initial testing of the shooter bot. At higher latencies and with Time Warp turned on, the bot had a tendency to be inconsistently accurate, missing shots during some rounds and hitting all of them in others. This was despite the movement of the shooter bot being absolutely consistent and these rounds occurring with the same network conditions. This can potentially be attributed to both the design

of the bot, as well as the implementation of Time Warp not allowing for rapid cursor inputs without introducing inaccuracies. However, this does not explain why the bot would hit shots at some rounds with 120 ms round-trip latency and Time Warp active, and miss shots in other rounds under these same conditions. We believe that this behavior is a result of a combination of factors, which exist in both the design of the shooter bot, as well as the implementation of Time Warp that was tested.

4.3 Post-Round Questionnaire Data

At the end of each round, players completed a set of questions pertaining to the round. The three questions can be found at the end of the paper in Appendix B. We found the average responses to each of these questions and organized them by latency value and Time Warp being on and off.

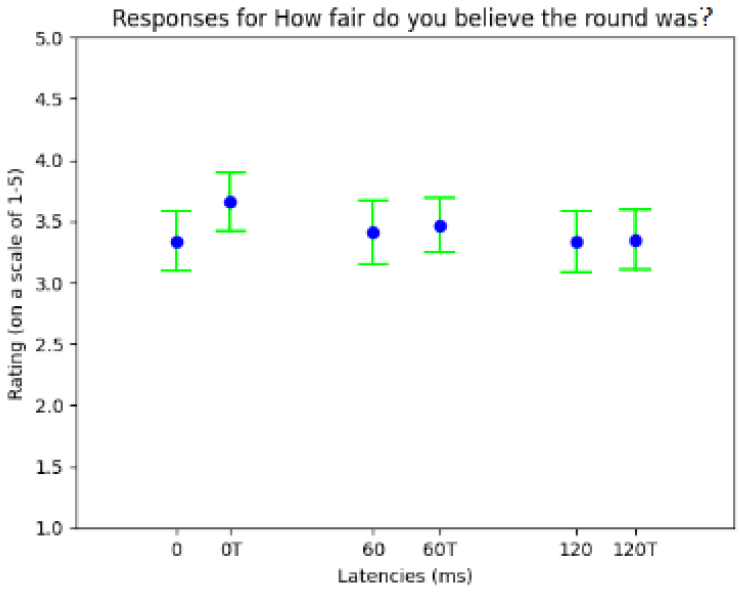

Figure 18: User perception of fairness

Our first question was “How fair do you believe the round was?” and was presented alongside a 1-5 scale, with 1 being the least fair and 5 being the most fair. Analysis of the responses to this first question shows that, on average, players believed that the rounds were fair or moderately fair, no matter the latency value. Upon closer examination, we can see that with Time Warp, the average response increases in value. This supports our hypothesis that Time Warp aided users in compensating for the latency that the players experienced.

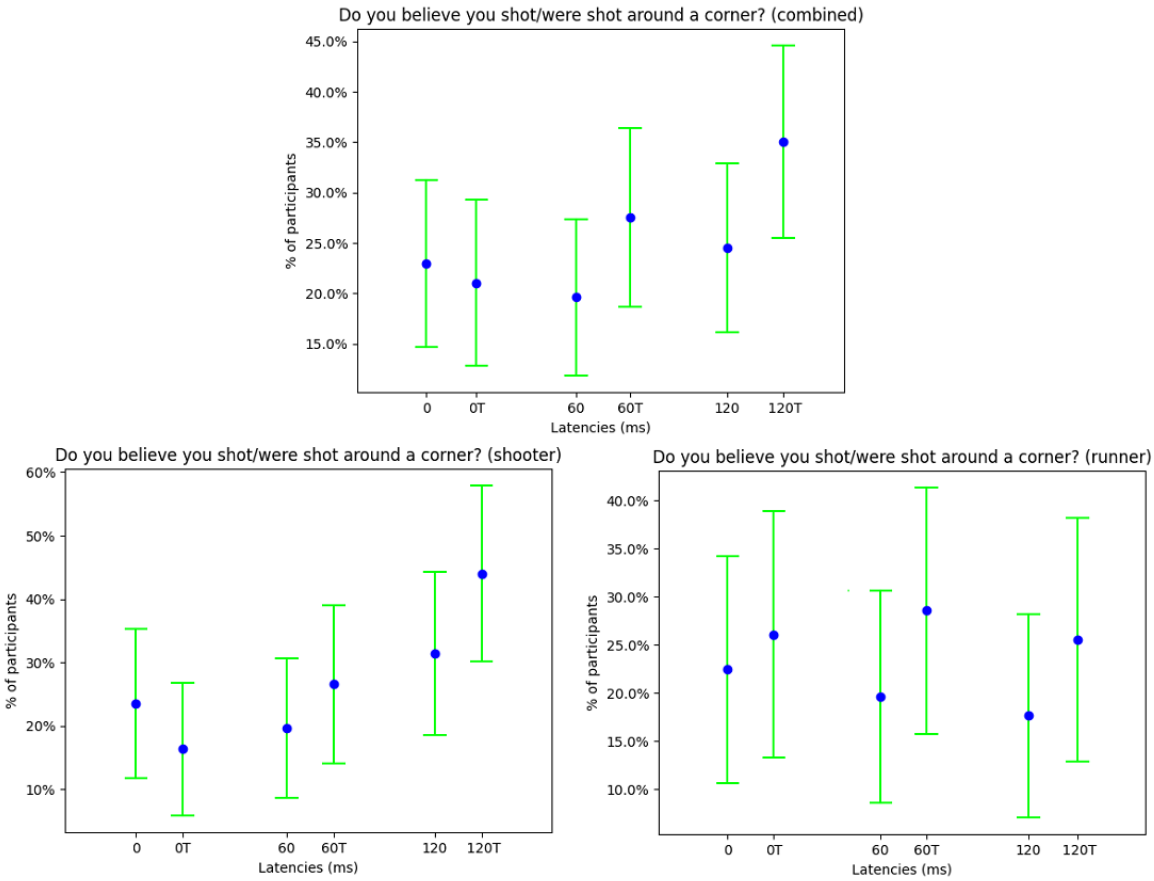

Figure 19: User perception of shot-around-the-corner

Our second question was “Do you believe you shot/were shot around a corner?” and was presented as a yes or no question. Analysis of our second question shows that in all of the latency and Time Warp scenarios, the majority of the players did not believe that they were shot around a corner or shot a player around a corner. In rounds with Time Warp turned on, there was an increase in how many players believed they were shot around the corner. This was true for latencies other than 0ms, where there was a decrease in the number of participants that reported shooting around the corner. Therefore, the activation of Time Warp supports our hypothesis that it causes an increase in shot-around-the-corner at non-zero latencies.

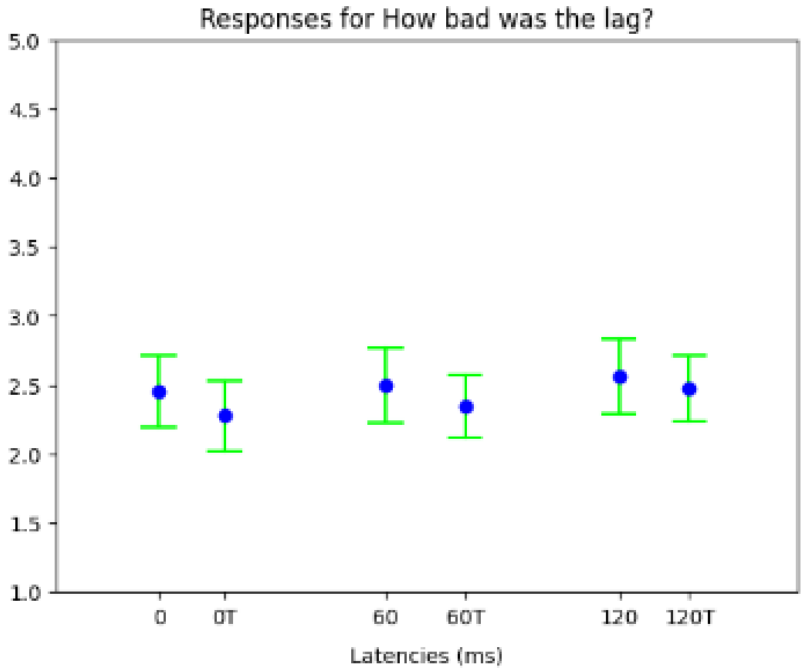

Figure 20: User perception of lag

Our third question was “How bad was the lag?” and was presented alongside a 1-5 scale, with 1 representing negligible latency and 5 representing the greatest amount of latency. Analysis of our third question shows that, on average, players believed that there was only a fair amount of lag. Higher latencies, such as 60 milliseconds and 120 milliseconds, show a higher average than 0 milliseconds, but there is no significant discrepancy between the averages of 60 milliseconds and 120 milliseconds. We are also able to see that, with the implementation of Time Warp, the average perceived lag decreased. Therefore, on average, Time Warp being on was able to decrease the amount of perceived lag that players were experiencing

5) Future Work

Including a larger participant pool would allow for more accurate data gathering. In the middle of conducting our study, a bug was discovered in the implementation of Time Warp that was used for our study. This required us to restart our user studies and limited us to a total of 17 final participants. Gathering a larger number of participants may provide a more accurate representation of how Time Warp affects gameplay and player experience.

Due to time constraints, we only tested 0, 60, and 120-millisecond round-trip latencies. From analysis, it seems that this range of latencies may not have been varied enough to show significant changes in player experiences when implementing Time Warp. Future work should conduct tests on a latency scale with a broader range, such as testing with 0, 50, 100, 150, 200, and 250-millisecond latencies.

One other factor that could show data differences is varying avatar movement speeds. This was not tested in our study. Testing different player speeds may help assess Time Warp on a broader range of FPS games, as many of these games such as Valorant and Overwatch contain characters that are provided with movement abilities.

Future work could also run experiments on a refined, bug-free environment, whether that be FPSci or another game environment. As stated above, bugs inhibited us from gathering a larger set of data and significantly restricted our timeframe for analysis. A large portion of our study was also spent drafting things that could possibly be added to FPSci that would provide us with the relevant data points needed to explore our hypotheses. As our suggestions were not originally part of the game, we had to wait for updates that often came at the cost of introducing more bugs to fix. Using a completed game environment would be beneficial in terms of time saved on planning, running, and analyzing experimental data.

6) Conclusion

The quality of experience of players of networked first-person shooter games relies heavily upon the network conditions between the client and the game servers. At high latencies, players may notice a decrease in accuracy and have to lead their shots in order to hit targets properly.

Latency compensation methods can be used such as Time Warp to improve the gameplay experience. Unfortunately, Time Warp can lead to instances of being “shot-around-the-corner”, where players that have run behind cover from their own point of view are still registered as having been shot on the server once the effects of Time Warp are applied. Prior to this, Time Warp and its effects on accuracy and shots-around-the-corner were unknown.

In our study, we explored the interaction between Time Warp and latency in order to determine how the implementation of Time Warp can lead to shots-around-the-corner at differing latencies. We achieved this by utilizing an autonomous bot program which we tested against human players at varying latencies while alternating between the use of Time Warp and its absence.

We observed a positive relationship between the use of Time Warp and an increase in the perceived fairness of a game session by the participants of our user study. In addition to this, there was a clear benefit in the use of Time Warp for the perceived lag that each player reported. There was also an increase in the number of perceived shots-around-the-corner with Time Warp, indicating that players notice the negative effects, too.

References

[1] Claypool, Mark, Xiaokun Xu, and Shengmei Liu. “A Survey and Taxonomy of Latency Compensation Techniques for Network Computer Games.” *ACM Computing Surveys*, Article 243, Volume 54, Issue 11S, Technical Report, Computer Science, Worcester Polytechnic Institute, January 2022. https://web.cs.wpi.edu/~claypool/papers/lag-taxonomy/

[2] DevinDTV. “How it Works: Lag Compensation and Interp in CS:GO.” YouTube, November 22, 2015. https://www.youtube.com/watch?v=6EwaW2iz4iA

[3] Liu, Shengmei, Atsuo Kuwahara, James Scovell, Jamie Sherman, and Mark Claypool. “The Effects of Network Latency on Competitive First-Person Shooter Game Players.” In Proceedings of the 13th International Conference on Quality of Multimedia Experience (QoMEX), Virtual Conference, June 14-17, 2021 http://www.cs.wpi.edu/~claypool/papers/csgo-net-21/

[4] Boudaoud, Ben, Josef Spjut, Joohwan Kim, Arjun Madhusudan, Pyarelal Knowles, and Zander Majercik. “NVlabs/FPSCI: Aim Training Experiments.” GitHub. Accessed May 3, 2023. https://github.com/NVlabs/FPSci

[5] Spjut, Josef, Ben Boudaoud, and Joohwan Kim. “A Case Study of First Person Aiming at Low Latency for Esports.” In EHPHCI: ACM CHI Workshop on Esports and High Performance Human Computer Interaction, May 08-13, 2021, Yokohama, Japan. ACM, New York, NY, USA. https://osf.io/nu9p3/

[6] Lee, Wai-Kiu, and Rocky K. C. Chang. "On ‘Shot Around a Corner’ in First-Person Shooter Games." In Proceedings of 15th ACM SIGCOMM Annual Workshop on Network and Systems Support for Games (NetGames), Taipei, Taiwan, 2017. https://ieeexplore.ieee.org/document/7991545/

[7] Armitage, Grenville. “An Experimental Estimation of Latency Sensitivity in Multiplayer Quake 3.” The 11th IEEE International Conference on Networks, ICON2003, Sydney, NSW, Australia, 2003. https://ieeexplore.ieee.org/document/1266180/

[8] Quax, Peter, Patrick Monsieurs, Wim Lamotte, Danny De Vleeschauwer, and Natalie Degrande. “Objective and Subjective Evaluation of the Influence of Small Amounts of Delay and Jitter on a Recent First Person Shooter Game.” In Proceedings of 3rd ACM SIGCOMM Annual Workshop on Network and System Support for Games (NetGames). Association for Computing Machinery, New York, NY, USA, 2004. https://dl.acm.org/doi/abs/10.1145/1016540.1016557

[9] Liang, Dingliang, and Paul Boustead. “Using Local Lag and Timewarp to Improve Performance for Real Life Multi-Player Online Games.” In Proceedings of 5th ACM SIGCOMM Annual Workshop on Network and System Support for Games (NetGames). Association for Computing Machinery, New York, NY, USA, 2006. https://dl.acm.org/doi/abs/10.1145/1230040.1230047

[10] Lee, Wai-Kiu, and Rocky K. C. Chang. "Evaluation of Lag-Related Configurations in First-Person Shooter Games" In Proceedings of 14th ACM SIGCOMM Annual Workshop on Network and Systems Support for Games (NetGames), Zagreb, Croatia, 2015. https://ieeexplore.ieee.org/document/7382997

[11] Xu, Xiaokun. Reaction time test. Accessed May 3, 2023. https://web.cs.wpi.edu/~xxu11/

Appendix A

IQP Demographic Survey

1) On a scale from 1-5, how much experience do you have with First Person Shooter video games, with 1 being completely inexperienced, and 5 being proficient?

1 2 3 4 5

---

2) On a scale from 1-5, how much experienced would you say you are with video games in general, with 1 being completely inexperienced, and 5 being proficient?

1 2 3 4 5

---

3) Which of these FPS games have you played?

- ☐ Valorant
- ☐ CSGO
- ☐ Overwatch
- ☐ Call of Duty
- ☐ Battlefield
- ☐ Apex Legends
- ☐ None of the Above
- ☐ Other… (Specify)

Appendix B

IQP Questions After Each Round

1) On a scale from 1-5, how fair do you believe the round was, with 1 being the least fair, and 5 being the most fair?

1 2 3 4 5

---

2) Do you believe that you shot the player/got shot by the player around a corner?

Yes No

---

3) On a scale from 1-5, how laggy do you believe the round was, with 1 being the least laggy, and 5 being the most laggy?

1 2 3 4 5

Appendix C

IRB Approval Letter

WORCESTER POLYTECHNIC INSTITUTE

100 INSTITUTE ROAD, WORCESTER MA 01609 USA

Institutional Review Board
FWA #00030698 - HHS #00007374

Notification of IRB Approval

Date:	13-Mar-2023
PI:	Mark L Claypool
Protocol Number:	IRB-23-0588
Protocol Title:	IQP Analyzing Shot Around the Corner Problem in FPS Games
Approved Study Personnel:	Faria, Mattheus~Claypool, Mark L~Nguyen, Pari~Hsu, Jonathan~Gregg, Miles~
Effective Date:	13-Mar-2023
Exemption Category:	3
Sponsor*:	

The WPI Institutional Review Board (IRB) has reviewed the materials submitted with regard to the above-mentioned protocol.We have determined that this research is exempt from further IRB review under 45 CFR § 46.104 (d). For a detailed description of the categories of exempt research, please refer to the IRB website.

The study is approved indefinitely unless terminated sooner (in writing) by yourself or the WPI IRB. Amendments or changes to the research that might alter this specific approval must be submitted to the WPI IRB for review and may require a full IRB application in order for the research to continue. You are also required to report any adverse events with regard to your study subjects or their data.

Changes to the research which might affect its exempt status must be submitted to the WPI IRB for review and approval before such changes are put into practice. A full IRB application may be required in order for the research to continue.

Please contact the IRB at irb@wpi.edu if you have any questions.

Appendix D

User Study Bot Code

Random Generator: https://github.com/MilesGregg/iqp-bot/blob/master/testing/makerandom.py

Both Bots:
https://github.com/MilesGregg/iqp-bot/blob/master/masterbot.py