WPI

Nova - Using Temporal Scaling for Latency Compensation in a Cloud-based Game

Written by Michael Bosik, Alex Hunt, Nina Taurich

A Major Qualifying Project report
Submitted to the faculty of
Worcester Polytechnic Institute
In partial fulfillment of the requirements for the
Degree of Bachelor of Science in Computer Science and the
Degree of Bachelor of Science in
Interactive Media and Game Development

Advised by Mark Claypool

Abstract

Cloud Gaming (CG) is gaining momentum as an alternative way to host and play games. However, cloud-based games are susceptible to latency because all player input must be sent to the cloud server, where all game logic is computed. In this project, we created a rhythm game hosted on Google Stadia that compensated for increases in latency by increasing the time window in which the player could complete actions. We evaluated our latency compensation technique by running a study where 27 users played through multiple rounds of our game with latency compensation on and off, and with random added latency. Analysis of the data from these users shows that our compensation method improves users' overall Quality of Experience and player performance compared to playing with latency compensation off.

Acknowledgements

This paper would not have been possible without the help of Mark Claypool, to whom we are extremely thankful. His knowledge, enthusiasm and industry experience provided key insights into the game development process and latency compensation techniques, as well as the structure of this paper.

Additionally, we would like to thank our sister project “Catalyst” and Xiaokun Xu for their help in research alongside ours. Finally, we would like to thank the team at Google Stadia for their support towards this project.

Table of Contents

1. Introduction

For over 50 years, the video game industry has been growing, taking on a large role in modern entertainment [1]. From the first Atari games, to the current Nintendo Switch, video games have become one of the world's biggest mediums of entertainment. According to Gaming Scan statistics, the revenue in video game sales this past decade was around 86.6 billion U.S. dollars [2], increasing at an exponential rate. As the demand for better quality in games increases, the technology used to play games has advanced to meet it.

Today, a new platform known as *Cloud Gaming* (CG) is gaining momentum as the way to host and play games. CG is a service in which games run on a cloud server, and data is transmitted to the client in the form of a rendered video stream [3]. This method of gameplay is beneficial to the user in that a client no longer has to own high-end graphics cards to render game graphics and compute physics, nor does the player need to install updates or data pertaining to the games.

Although cloud-based games are light on storage and graphics for the player, another issue arises: latency. Latency is the delay in response to player input. There are several possible sources of latency, both local and network, such as insufficient computational power, input delay, and network congestion. A higher latency in a game means more time from when the player provides input until that input is shown on the screen. Latency can make a game more difficult to play, to the point of frustration, thus lowering the player's *quality of experience* (QoE).

A significant source of latency for cloud-based game systems is the network. Until recently, games were all hosted locally, meaning the entirety of the game was loaded directly onto the player's console or computer. All computation and data was stored and processed as a whole within the same piece of hardware. When a game is played on a CG system, the input data provided by the player has to travel through the network, wait for computations done by the server, then travel back to the client. Over the internet, this round-trip time can increase latency by a lot.

In order to provide a high QoE, a developer on a CG platform must use latency compensation techniques to counteract the effects of latency on gameplay. Many

compensation techniques have been researched and are used in games today. However, not all of these techniques can be applied to CG. Techniques such as aim assistance and time warping have commonly been used in network games [3,7]. World alteration [4], the changing of the game world attributes based on latency, is a less studied compensation technique. World attribute scaling can include visible alterations such as target size, or invisible alterations such as hitbox size. The goal is to lower the precision and deadline required in the game, thus lowering the level of difficulty to be similar to the level with no network latency.

In this project, we created a game that adjusts to latency using world alteration and deployed on Google's CG service, Stadia [5]. We built a fast-paced rhythm shooter game called *Nova*, in which targets appear to the beat of the music at random points in the player's field of view. Players receive points by hitting the targets before they disappear. In rhythm games, correct timing of player actions is crucial for success. We used a world alteration technique, specifically temporal accuracy adjustment, for latency compensation to maintain game difficulty and QoE across different latencies. Temporal adjustments may be less visible and thus less likely to be noticed by the player than hitbox and target size adjustments. To scale temporal accuracy, we increased the amount of time a player has to hit a note proportionally to network latency.

To evaluate the effectiveness of our latency compensation on the player's QoE, we conducted a user study with Nova to evaluate player performance when artificial latency was applied. Users played through 16 different trials with varied amounts of latency and game difficulty, and with latency compensation on and off. Players completed a survey asking about their QoE after each trial. We had a total of 30 participants over the course of two weeks.

Analysis of the user study data shows that our latency compensation technique is effective in improving player accuracy, increasing the player's reaction percentage (the amount of time in a target's window it took for them to shoot the target), and marginally increasing the player's QoE. However, player performance and QoE still decreased as latency increased, so our latency compensation may need to be adjusted. A stronger latency compensation may have a more-constant QoE and player performance as latency increases.

The rest of the paper is organized as follows: Section 2 introduces cloud gaming, discusses cloud gaming's sensitivity to latency, and the latency compensation technique we implement. Section 3 outlines our game, which designed and developed from scratch for this project. Section 4 outlines our user study evaluation, including our methods, results, discussion, and limitations. Section 5 presents the conclusions from our data. Section 6 outlines possible future work.

2. Background

2.1 Summary

This chapter provides an introduction to cloud gaming, the effect of latency on player experience in cloud gaming, latency compensation techniques, and on rhythm games. Cloud gaming is a lightweight platform for gaming, especially effective for users that do not own powerful hardware. However, cloud gaming suffers from an increased sensitivity to latency. Several methods exist to compensate for latency, such as world alteration and deadline adjustment. For our purposes, we used the method of deadline adjustment in our rhythm shooting game.

2.2 Cloud Gaming Explained

Cloud gaming is a gaming platform in which the player uses a *thin client* to play a game hosted on a server [12]. The client sends all input to the server, and then receives a video and audio feed from the server. This offers a lightweight gaming experience, at the cost of increased sensitivity to latency [6].

2.2.1 Cloud Gaming vs. Conventional Online Gaming

Conventional online games require the player to install the game's files on their device, whether it is by sliding a disc or a game cartridge into a drive or by downloading the game's files onto a computer or console. This installs files onto the player's device that contains many of the game's files, such as compiled code and art and audio assets.

Then, during online play, the client exchanges data with a single authoritative server that handles the game logic [9]. The client sends player input data to the server and then receives information about the game's state, which it uses to render the game world.

Cloud-based games work differently to online games. Compared to the traditional "fat" client that keeps the game's state and does all the rendering work, cloud-based games rely on a *thin client* on the player's device. Thin clients do not do most of the rendering or computational work that fat clients would do. They only collect input and play audio and video from a feed, while the server performs all game state computation and rendering.The client sends input data to the server, and then the server then sends the client an audio/video feed to display [12]. Compared to other online games, which rely on the player's client to make computations and render frames, cloud-based games require little computational power from the player's device [3]. All computation and rendering is done on the cloud server. In theory, this allows for players to enjoy games without paying for expensive hardware.

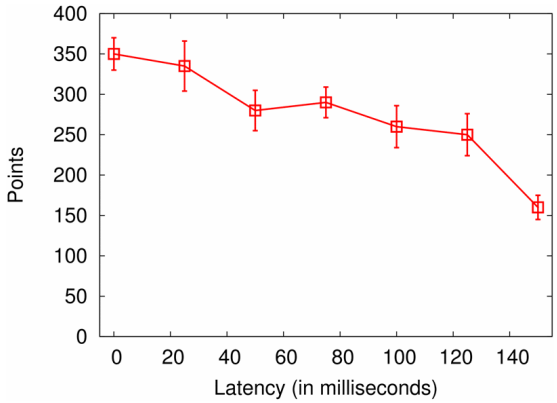


Figure 1. Player performance (in points) for Crazy Taxi vs. Latency on OnLive cloud gaming system.

Source: [6]

However, cloud gaming's QoE relies heavily on network quality. Since all player input is transferred to the server and computed on the server, cloud games are especially sensitive to network latency. Claypool and Finkel found that in cloud-based games, player performance degraded up to 25% with every 100 milliseconds of latency,

as illustrated in Figure 1 [6]. Since cloud-based games are so sensitive to latency, it is important to develop effective latency compensation methods for CG platforms, such as the technique we implemented and evaluated for Google Stadia.

2.2.2 Google Stadia

Google Stadia (Stadia) is Google's cloud gaming platform. Stadia allows players to access any games they own on a variety of devices, from computers to mobile phones and television screens with Chromecast [8]. In our project, we developed a game with latency compensation and deployed it to Stadia to test the effectiveness of our latency compensation techniques.

In order to use Stadia, Google requires that the player have an Internet connection speed of at least 10 megabits per second (Mbps) and either a recent version of the Google Chrome browser, or the latest version of the Google Stadia app installed.

2.3 Latency in games

Latency becomes more of a factor that deters QoE, for there are multiple characteristics of a game that can be affected by latency. Temporal accuracy, the time that a player has to perform an action, can be decreased due to slow response from a server or a lower feedback frequency [13]. Feedback frequency is how often the game provides visual, auditory or haptic feedback to the player. With a higher latency, feedback frequency is reduced. In a first person shooter (FPS) for example, the player has a small temporal accuracy, as two players targeting each other must react first in order to shoot the other. With a higher latency, the feedback frequency declines, which in turn, creates an even smaller temporal accuracy for the player. Other characteristics such as spatial accuracy and predictability can also be affected in the same way.

A study performed on both inexperienced and experienced game players tested the relationship between game performance and delay on QoE [14]. Participants played two games: "Need for Speed" and "Table Tennis". For each segment they played, a varying level of delay was added to the experience. They were asked to answer questions after each segment in terms of QoE, such as challenge, flow, and immersion. The study concluded in accepting the hypothesis: there is a relationship between

performance and QoE. The participant data showed that lower QoE ratings correlated to a worse performance.

2.4 Latency Compensation

2.4.1 World Alteration

World alteration is a way of changing the difficulty of a game based on the latency [4]. In a game, the optimal state for the user to be in is called *flow* [3]. A player is in a flow state when they are engaged and not too challenged, but not too bored. Latency can affect this balance because it increases the difficulty of a game. Compensation through world alteration tries to maintain flow by decreasing the difficulty. The more sensitive a game is to latency, the more difficult it becomes under a high delay. The precision-deadline model proposed by Claypool and Claypool indicates two causes for a game to be sensitive to latency: spatial accuracy (precision), and temporal accuracy (deadline) [15]. Modifying the size of game objects will decrease the precision required while changing the speed of the game will lessen the deadline.

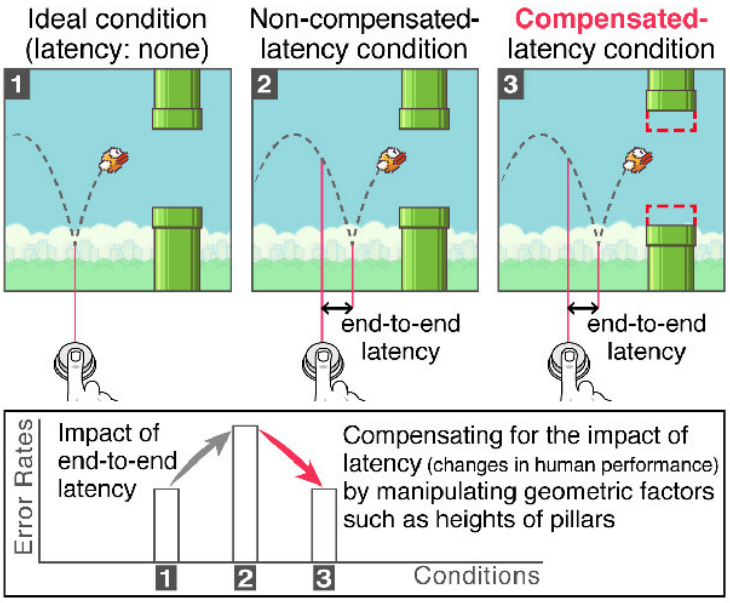


Figure 2. Overview of latency compensation through geometric scaling.

Source: [10]

Geometric scaling, as proposed by Lee et al., is a model of world alteration that changes the precision required to succeed in a game [10]. It geometrically transforms the shape of the game scene to compensate for latency, as seen in Figure 2. For

example, in a shooter game that requires the player to aim and shoot at targets, latency could make it more difficult for the player to aim, and thus lower their precision. In order to compensate for this, the size of the target's hitbox can be increased depending on the amount of latency experienced. This would make it easier for the player to hit the target with high latency and thus make up for the difficulty latency adds to aiming.

The effectiveness of geometric scaling has not been ascertained yet, and such is outside the scope of our project. However, geometric scaling may be an effective method of latency compensation and may particularly be useful in games with actions that rely on precision, such as shooters. Though our game's actions involve both precision and deadline, we have chosen to focus on altering the deadline of our game's actions to compensate for latency.

2.4.2 Temporal Accuracy Adjustment

As previously mentioned, temporal accuracy, or deadline, refers to the amount of time given to complete an action. The more latency there is, the harder it becomes to complete an action in the required time frame. There are several ways to adjust the required deadline in a game. In a study by Sabet et al. comparing the effect of delay on QoE by increasing the deadline, participants played four games with varied levels of latency [3]. They played two shooting games and two dodging games. To adjust the deadline, the speed of the targets was decreased. The result of the compensation was measured using input quality, calculated from a questionnaire about responsiveness and controllability. They found an increase in input quality after adaptation for all the games, but the shooting games had a significantly larger difference. Thus, increasing the required deadline for input actions can be an effective method of latency compensation for games that require short deadlines. This includes our rhythm game, as shooting at the targets to the beat has a very specific deadline.

2.5 Rhythm Games

A rhythm game is a genre of music-themed action video game that challenges a player's sense of rhythm. Games of this genre mostly involve following the beat of a song in the form of dancing, strumming a digital instrument, or accurately hitting a

target. One of the most famous rhythm games is “Guitar Hero” (RedOctane, 2005), in which the player strums a digital guitar to the beat of a song as notes scroll towards them. This game uses a customized guitar shaped controller that brings a new level of immersion into play. On Stadia, rhythm games, such as “Just Shapes & Beats” (Berzerk Studio, 2018) and “Thumper” (Drool, 2016), have the player move their character to the beat of the music playing by using certain controller inputs. Other rhythm games such as “Beat Saber” (Beat Games, 2018) and “Audica” (Harmonix, 2019), which are made for Virtual Reality (VR), use physical human motion as controller input such as slicing a block, or pointing a gun at a virtual target.

2.6 Summary

Cloud gaming is an emerging game platform that allows users to play games through a *thin client* that sends input data to a server, which does all the computation and rendering work before sending the client a video and audio feed. Cloud gaming platforms such as Google Stadia allow users to play games on a variety of devices without needing to acquire expensive gaming hardware. However, cloud gaming is even more sensitive to latency than conventional online gaming [6].

The presence of latency in a game, such as from a poor network connection, affects several parts of gameplay. Latency can affect the temporal accuracy, thus decreasing the amount of time the player has to execute a certain action. Latency lowers the feedback frequency, resulting in skipped frames and visible lag [13]. Latency can negatively impact player performance, and thus negatively impact the player’s QoE [11]. Thus, to maintain a high QoE for the player in cloud gaming, it is important to implement effective latency compensation techniques.

Several latency compensation techniques have been proposed. Geometric scaling compensates for latency by altering the physical geometry of the game world, such as hitboxes. This makes it easier for the player to complete actions that require precision, such as shooting a moving target in a first-person shooter. However, for our rhythm game, we have chosen to focus on an alternate method of latency compensation: loosening the deadline for game targets in response to higher latency.

3. Nova

3.1 Summary

To implement latency compensation and evaluate it on a cloud-based gaming platform, we created Nova, a first-person shooter rhythm game in Unreal Engine 4. In Nova, the player uses the mouse to look around and shoot targets that spawn to the rhythm of a song. We implemented latency compensation through adjusting the time the player has to shoot a target before it disappears. We spent a total of nine weeks developing Nova, including time spent porting Nova to Google Stadia's cloud gaming servers.

3.2 Development

3.2.1 Platform

We started developing Nova in Unreal Engine 4 (UE4) version 4.25, since we were most familiar with developing games in UE4. In order to spawn targets based on a song, we also chose to use one of UE4's plugins, Synesthesia, to read through a sound file and provide an API for us to work with in syncing events to audio waveforms [16]. Later on, as we ported the game to Google Stadia's cloud servers, we upgraded to UE 4.26.

3.2.2 Summary of Sprints

We split up Nova's development into three seven-week terms: A Term, B Term, and C term, with each term further split up into several one to two-week long development sprints. In A Term, we planned out Nova's features, including its latency compensation, and wrote out a proposal for the game. We researched several latency compensation techniques and decided on altering the targets' lifetimes. We also

developed a basic prototype in UE 4.25, which included the game's basic features and a basic latency compensation model.

In B Term, we developed a pre-alpha build of Nova, suitable for basic latency testing. For this build we created artificial latency within our game that could be adjusted. We gave this build to another group of students and allowed them to use it in testing the effect of latency on players' performance in Nova [17]. Afterwards, we completed an alpha build with all of the required features such as target spawning, shooting, and music. We presented Nova in WPI's yearly AlphaFest, an event for students and faculty to showcase and test games that they have developed. After AlphaFest we used the feedback from our playtesters to further refine Nova, thus completing our beta build.

Nova was nearly finished at the beginning of C Term, and we spent the first two weeks of C Term implementing final polish, implementing updated latency compensation, and preparing Nova for our evaluation. We planned out our evaluation, prepared survey questions and IRB protocol, and ported Nova to Google Stadia's platform. We partnered with the Stadia team and used the Stadia SDK to host an instance of our game on Stadia's cloud servers. This allowed us to easily host our game on an existing cloud gaming system and provided us with an API to emulate various network conditions, including varying network latency conditions with added jitter.

3.3 Latency Compensation

There were several ways we could have adjusted our game to latency. We chose to alter the lifetime of the targets because the change is not visible to the player. Figure 3 displays the timeline of a target's lifetime. In Nova, a target spawns and moves to a random location on the screen. Then a colored circle appears and closes until the music note associated with the target is played. Next, the circle changes color indicating that it is time to shoot it. This window, or the amount of time between the music note and the target's death, is the time that we are lengthening based on the latency. The higher the latency, the longer the player will have to shoot the targets. The idea is that players will

have more time to catch up if they fall behind due to a latency spike. This also means that players will be less likely to be shooting on the beat.

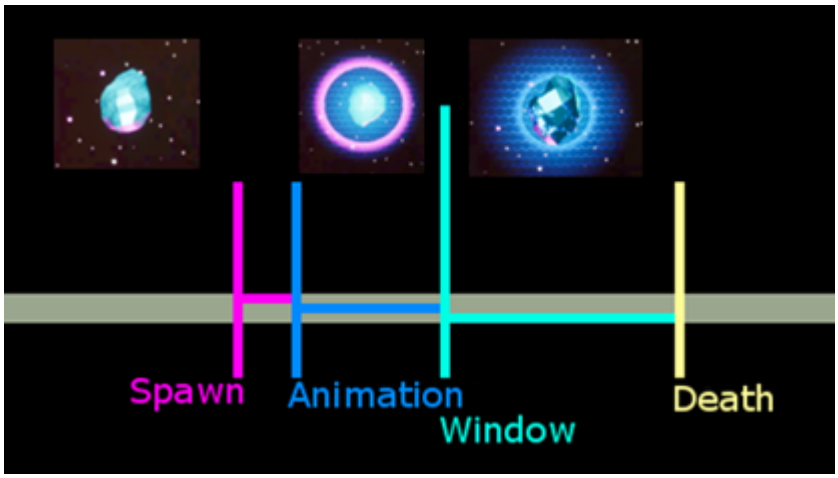


Figure 3. A visual representation of a target's lifetime, from spawn to death.

During this project, we collaborated with another team of WPI students doing an Interactive Qualifying Project (IQP) so that we could understand how to scale the window based on latency [17]. They were studying the effects of world alteration on accuracy in games. They used the pre-alpha build of our game in a user study to measure the effects of our latency adjustment technique on accuracy. In the study, participants played through several rounds of the same song with varying levels of latency, difficulty, and window time. To adjust the difficulty of the song they adjusted the cooldown between targets spawning. Cooldown is the minimum amount of time in seconds allowed between targets spawning. The shorter the cooldown, is the more targets will be spawned, thus making the song more difficult. As an outcome of their study, they gave us an equation which can be seen below. In this equation, latency is measured in milliseconds, and both window and cooldown are measured in seconds. Accuracy is on a scale from zero to one.

$$Accuracy = -0.00085351 * Latency + 0.09624587 * Window + 0.09650269 * Cooldown + 0.6349996613850233$$

We were able to use this equation when scaling the window size. The IQP team's study used cooldown values from 0.5-1.0. We made several adjustments to our game

since their study, such as adding another type of target. We altered the cooldown values for the songs to be between 1.0-1.5. This meant we were extrapolating beyond the space measured by the IQP team. With a latency of 0, we would have gotten values as low as 0.2 seconds. We adjusted the formula to return the types of values we expected that the player would have time to hit. Our final equation is shown below in Figure 4 For the hard songs with a cooldown of 1, this equation gives us values from 0.96-2.29 when latency is from 0 -150. For a cooldown of 1.5, the equation gives values from 0.46-1.79.

Final Equation:

$$Window = (Average\ Latency * 0.008868 + Accuracy / 0.09624) – ((Cooldown -0.25)* 1.003) - 6.598,$$

Where $Accuracy = 0.8.$

Figure 4. The final equation for a target's window.

In the equation latency is measured in milliseconds and window and cooldown are both in seconds.To implement the equation in our game we got the latency, calculated the average, and then entered that into the equation for window length. To get the latency between the client and the Google Stadia servers we used the API provided by Google Stadia and created an Unreal Blueprint node. Almost all of our game was written in Blueprints, so creating a Blueprint node allowed us to easily integrate the Stadia API with the rest of our code. The code shown below in Figure 5 gets latency in microseconds and converts it to a float so it is readable by Blueprints. We used conditional-compilation directives to declare two versions of code. One that compiles when launching to Google Stadia and the other when compiling for the local machine. Having two versions allowed us to continue to test locally.

```
#include "LatencyNodes.h"
#if PLATFORM_STREAMING
#include <chrono>
#endif

float UStadiaLatencyNodes::GetStadiaLatency()
{
#if PLATFORM_STREAMING
    std::chrono::microseconds latency = GetInputDelayForInput(GetPlayerId(), 0);
    float lag = latency.count()/1000;
    return lag;
#endif
    return -1.0;
}
```

Figure 5. Pseudocode to retrieve latency from the Stadia servers and make it accessible via Blueprints.

We have to account for jitter in the latency values because we are using real latency from Google Stadia. To do this, we calculated the average latency. To calculate the average we used a weighted moving average where Average = (Previous Average * Weight) + (Current Latency * (1 – Weight)) and with a weight value of 0.9. The average is updated every 0.5 seconds. Figure 6 shows an example of our measure of average latency compared to the instant latency during a trial. Figure 7 displays the implementation of calculating the weighted moving average. It gets the current latency using the blueprint node we created called Get Stadia Latency and inputs this into the equation. The variable storing the latency average is set with the result. Each target has its own window and decides its window length while it spawns by putting the current average latency into the equation defined above in Figure 4. This can be seen in Figure 8 which shows the function each note calls at spawn time to determine its window length.

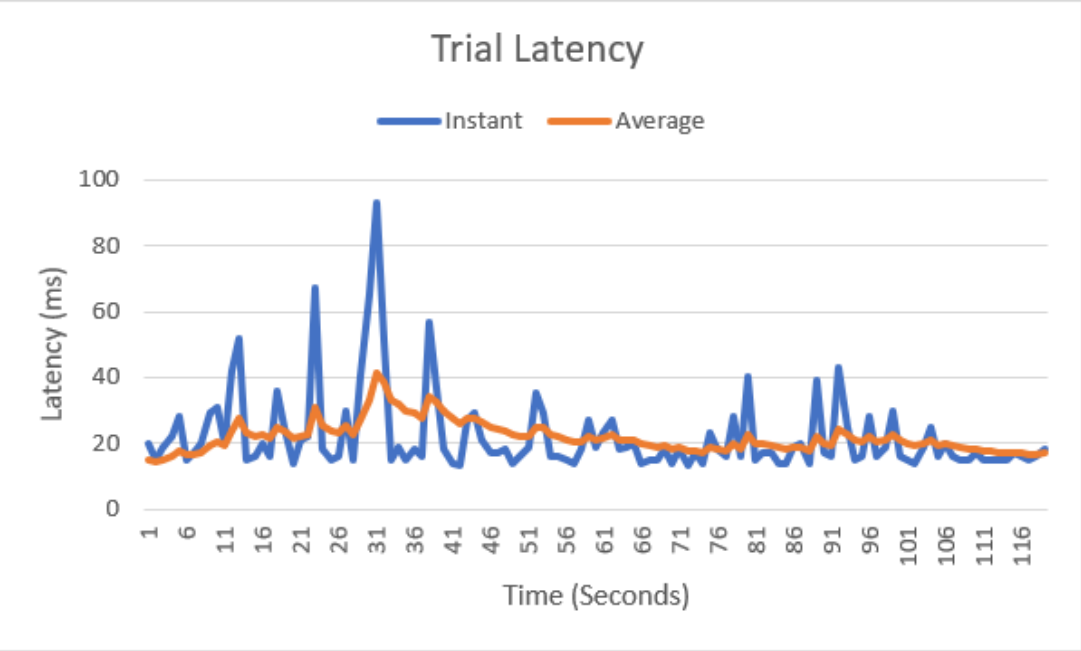


Figure 6. Network latency adjusted by Google Stadia network emulation tools (blue) and average latency (orange) vs. time.

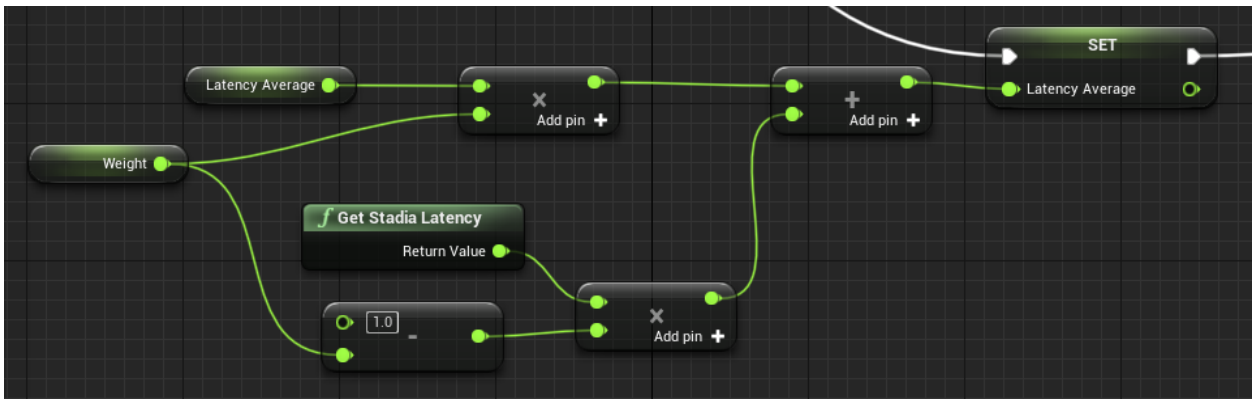


Figure 7. Blueprint code calculating latency average

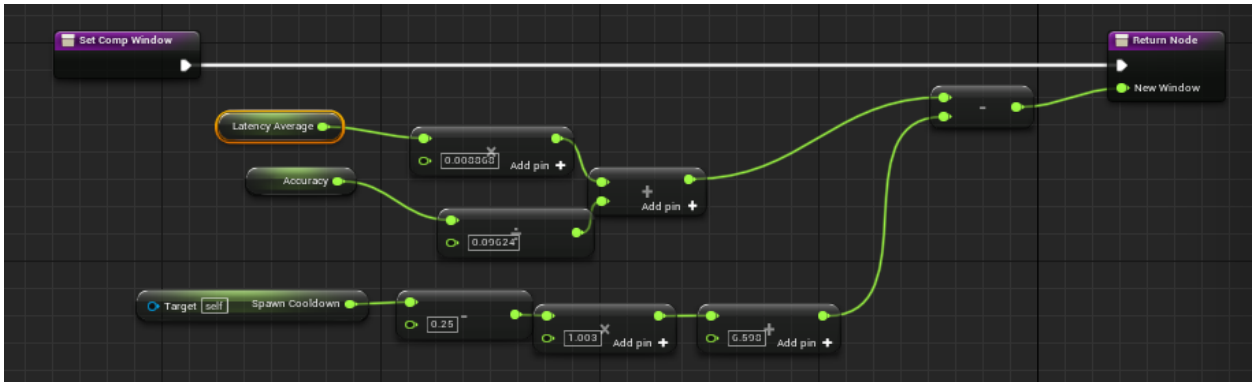


Figure 8. Blueprint code scaling the window length.

3.4 Features

Nova has many features, from procedural target generation to health, song selection, and a scoring system. In this section, we will outline them all.

3.4.1 Song Selection

Like many rhythm games, Nova includes several different songs. We chose three: Mysterious Green Fluid, Pandemic, and Fist of Fury. All songs had either Creative Commons licenses, or licenses that we purchased. In the main menu, the player can view a short tutorial of the game, edit options such as mouse sensitivity, or proceed to play the game, as seen in Figure 9.

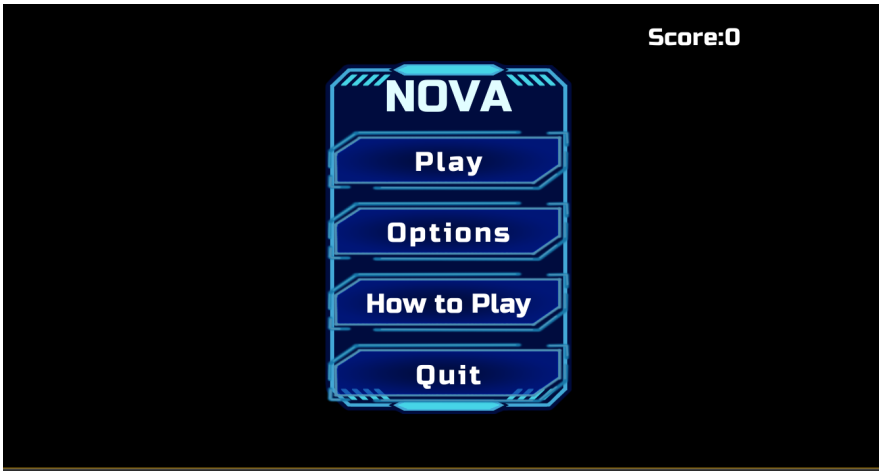


Figure 9. Main Menu Screen.

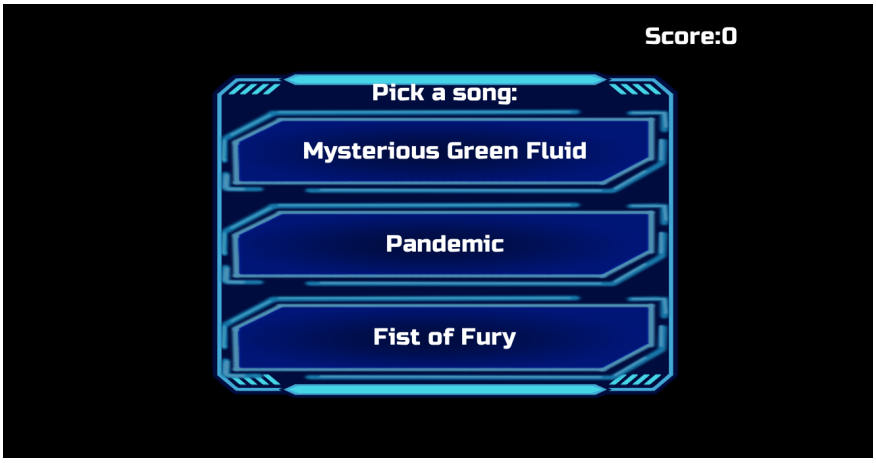


Figure 10. Song Choices Screen

Once the player proceeds, the player is able to select from any of the three songs and play through that song in its entirety, then go back and play through another, as seen in Figure 10. This constitutes the main game loop. Once the player misses five or more targets in a row, the game ends, as seen in Figure 11.

Figure 11. Game Over Screen

3.4.2 Procedural Generation spawning & clustering

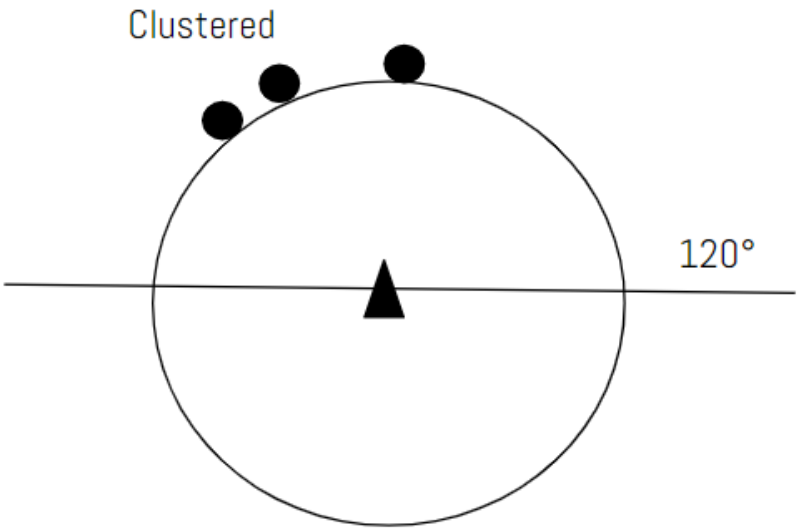


Figure 12. Illustration of targets clustering in front of the player.

For each song in Nova, the targets are spawned procedurally. By using an Unreal Engine plugin called Synesthesia, we are able to parse through the waveforms of a given audio file, and retrieve the loudness of the wave at any given time within the audio [16]. By doing this, we can then determine points in time relative to the start of the song that have a loudness reading of above a predetermined threshold. This threshold is a value that we manually set for each song in our game. For the songs implemented in our user study, we used thresholds ranging between 0.5s and 0.75s. When a point in time is identified that is above the threshold, we add it to an array of spawn times with an adjustment that accounts for the time it takes for a target to travel to its position, as well as the time it takes for the identifier circle to close around the target. This adjusted timestamp is now exactly when a target should spawn during playback of a song. This song preprocessing is done each time a level is loaded, and as the song plays, a target is spawned at the exact time that is saved in the array of timestamps.

Another variable that can be altered per playback is the song cooldown. This value, as mentioned previously, is the minimum amount of time in between two targets spawning. As we preprocess the song for timestamps, we measure whether or not the

next timestamp would be within the cooldown time. If it is, we disregard the timestamp so that the target will not spawn.

When a target spawns, we take the location of the previous target that was created, and select a new location relatively at a predefined distance. This causes targets to cluster nearer to each other (as seen in Figure 12) instead of appearing at random points around the player. The reasoning for this is to reduce the amount of time it takes for a player to locate their next target. When a target appears closeby to the previous one, it is easier to locate. Targets are also confined to a 120° radius around the player, in front of their view.

In addition to procedural target spawning, our game features two separate types of targets. During preprocessing of a song, any time a timestamp is selected, we determine if the target should be of type A (left click) or type B (right click). The two target types can be seen in Figure 13 and Figure 14. A type B note is determined if the audio loudness in the song's sound wave continues to be over the threshold for the entirety of time it takes for the cooldown to run. All other notes would be of type A. We had intended for this type B target to allow the player to click and hold for a more drawn out firing effect, although we were not able to implement this feature in time. However, it still stands that type A targets can only be destroyed via left click, and type B targets can only be destroyed with right click.

Left Click Target:

Right Click Target:

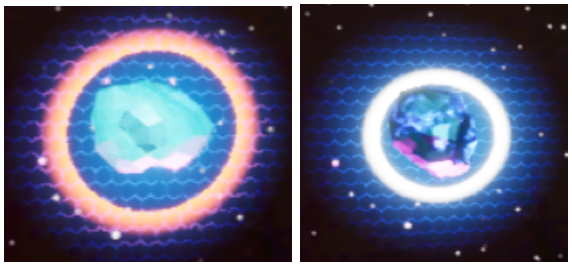


Figure 13. Comparison of both types of targets

Figure 14. Screenshot displaying both target types

3.4.3 Health

In order to add consequences for missing targets, we added a health feature for the player. Every time the player misses a target (does not shoot the target in its lifetime), the player loses some health, and the screen becomes more red and cracked. If the player does not miss any more targets, their health will slowly regenerate. After five misses in a row, the game ends. The screenshots below in Figure 15 show the progression from full health to no health.

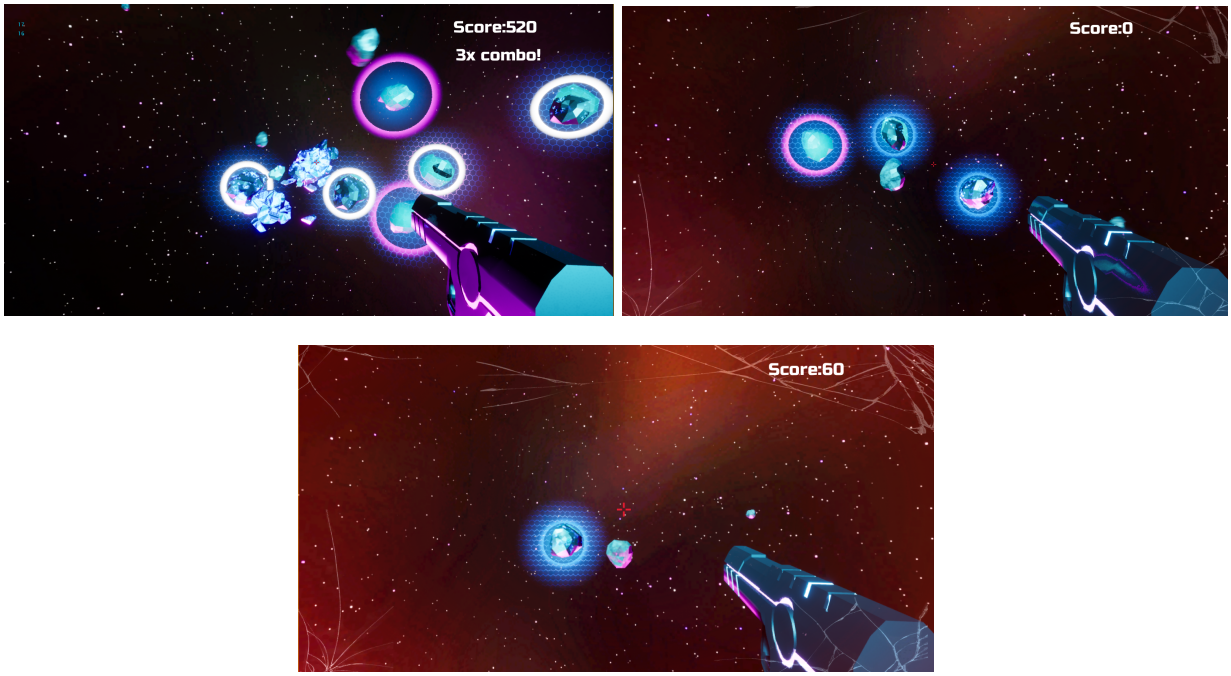


Figure 15. Illustration of health decreasing in the game.

3.4.4 Score & Combos

For Nova, we implemented a simple scoring system. Each time the player successfully shoots a target within the window time, 20 points are added to their score. If the player hits 10 targets in a row, their score gains are multiplied by two. At 20, their score gains are multiplied by four, and at 30, their score gains are multiplied by the maximum factor of eight.

3.4.5. Art assets

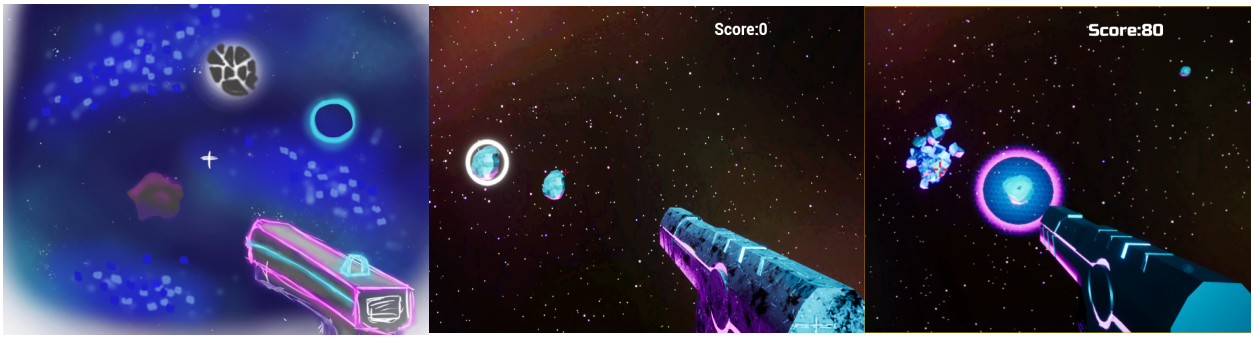


Figure 16. Nova concept art (left), the Nova beta build (middle), and Nova final build (right).

All of the game's visual art assets were custom-made. As mentioned in Section 3.4.1, all songs were downloaded from the Internet with either the Creative Commons License or a purchased license. Sound effects were downloaded from freesound.org [18] under the Creative Commons License.

4. Evaluation

4.1 Methods

4.1.1 Summary

To evaluate the effectiveness of latency compensation in a cloud-based gaming platform, we ran a user study. Participants played several rounds of a build of Nova deployed to Google Stadia's cloud servers, both with and without latency compensation and at four set latency levels. Participants then answered survey questions after every round about their quality of experience.

4.1.2 Setup

4.1.2.1 Hardware specifications

We ran the study on a WPI-owned laptop. The laptop's specifications are as follows:

Processor: Intel(R) Core(TM) i7-5600U CPU @ 2.60GHz (4 CPUs), ~2.6GHz
Memory: 8192MB RAM
DirectX Version: DirectX 12
Graphics Card: Intel(R) HD Graphics 5500
Display Memory: 4169 MB
Native Display Resolution: 1920x1080
Average Local Latency: 39.6ms (with a standard deviation of 5.86ms)

Participants played a build of Nova deployed to Google Stadia on Google Chrome version 88.0.4324. Nova ran in full screen mode on the native resolution of 1920x1080.

We measured local latency by recording laptop use with a high-speed camera recording at 1000 frames per second, analyzing the resulting video, and determining the difference in time between the user clicking and the result of that click displaying on the screen. After a total of five clicks, we averaged the results together to find an average local latency of 39.6ms, with a standard deviation of 5.86ms.

4.1.2.2 Lab setup

For the user study we conducted, we had participants sit at the laptop with our game running on its Stadia instance. The laptop had a USB mouse and a wired ethernet connection. A picture of the lab setup can be seen in Figure 17. A speed test using *fast.com* [19] measured an average upload speed of 840 Mbps and average download speed of 580 Mbps using the Ethernet connection for this laptop. The participants played trials of Nova on the laptop, and alternated to another desktop computer nearby to answer survey questions pertaining to quality of experience after each trial. A packet management software called Clumsy [20] was running in the background during trials to alter the amount of latency the user would experience. As participants progressed through trials, we changed parameters for this latency.

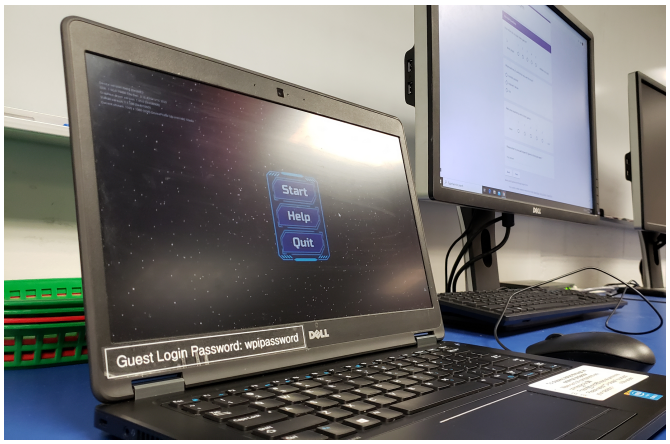


Figure 17. Picture of lab setup.

4.1.3 Pilot Study

Before we began our study, we ran a couple of pilot trials to test certain aspects of how our study would be run– more or less a practice round. From these pilot trials, we found some important details that needed to be changed or implemented to our game in order to have a smoother study. For example, the aiming reticle in our game was too small to see during higher latencies when video quality dropped. Also, we changed our trial ordering to implement a randomness in having latency compensation on and off.

4.1.4 User Study

4.1.4.1 Trial Waves

Our study consisted of 17 different waves of Nova that participants played through. These waves were 45 seconds in length and were each tied to varying values of latency, difficulty, and latency compensation. We chose a trial duration of 45 seconds after a pilot study, finding that 45 seconds of each song we included was long enough to contain a substantial number of targets to shoot, while being short enough to allow for multiple tests.

The first wave of the study was a simple tutorial of the game with no added latency or compensation. Inthe following 16 waves, we altered the latency being experienced by the participant between values of 0ms, 50ms, 100ms, and 150ms, along with whether or not the game was compensated for latency, and the overall difficulty of the song being played. We incorporated an easy and a hard song to be played for the waves. Compared to the easy song, the hard song had a higher BPM (beats per minute): 110 BPM v.s. 60 BPM. Therefore, the hard song has many more targets spawning. The easy song contains a total of 21 targets while the hard song has a total of 38.

The sequence of altering variables was as follows: A random uplink/outbound latency of the four available options was selected and applied to the laptop's ethernet

connection. Four waves were played with a random combination of difficulty and compensation being one of four options:

- Easy difficulty, no compensation
- Easy difficulty, compensation
- Hard difficulty, no compensation
- Hard difficulty, compensation

A diagram of how the waves were randomized can be seen in Figure 18.

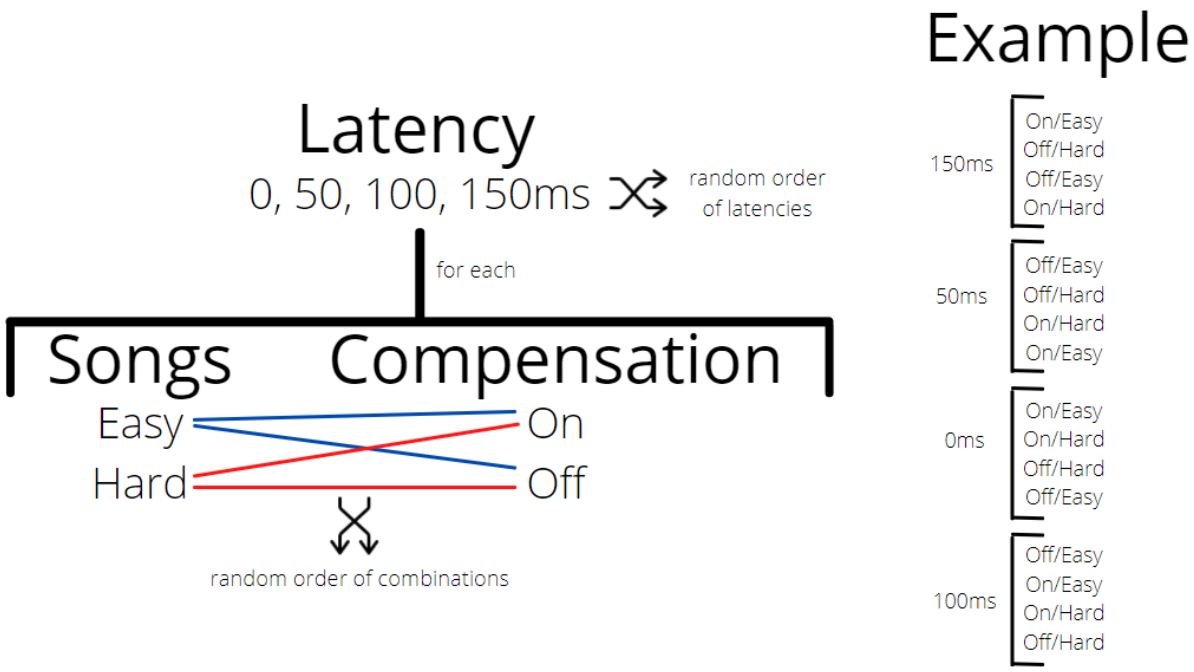


Figure 18. Every possible combination of latency compensation status, song difficulty, and latency levels.

4.1.4.2 Clumsy

In order to alter the latency the user experienced between trials, we used a third party software called Clumsy [20]. Clumsy is a packet management application that is able to filter incoming and outgoing traffic from a machine with whichever parameters are entered. A screenshot of Clumsy's interface can be seen in Figure 19. For our study, we filtered all outgoing packets from the laptop, and added a lag of X ms, X changing

between 0, 50, 100, and 150 in random order as necessary. A diagram of the network that this describes is shown in Figure 20. We automated this change in latency by creating a windows batch script that ran in the background during the study. Every four waves, the proctor of the study would continue the script which would re-launch Clumsy with new parameters. This would remove any potential human error in setting values of Clumsy incorrectly.

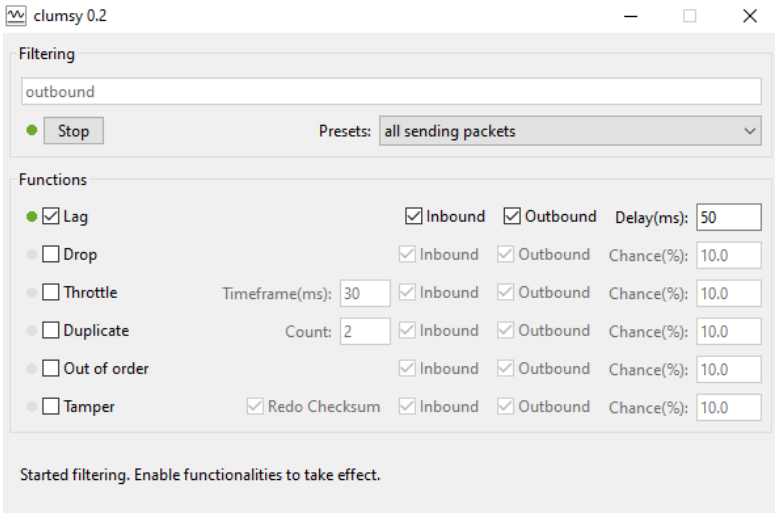


Figure 19. A screenshot of Clumsy's interface.

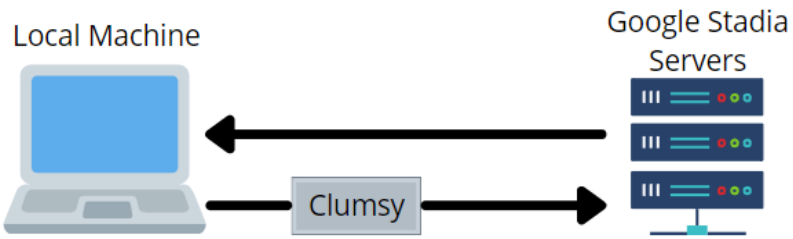


Figure 20. Illustration of Clumsy adding latency to our laptop's connection to the Google Stadia servers.

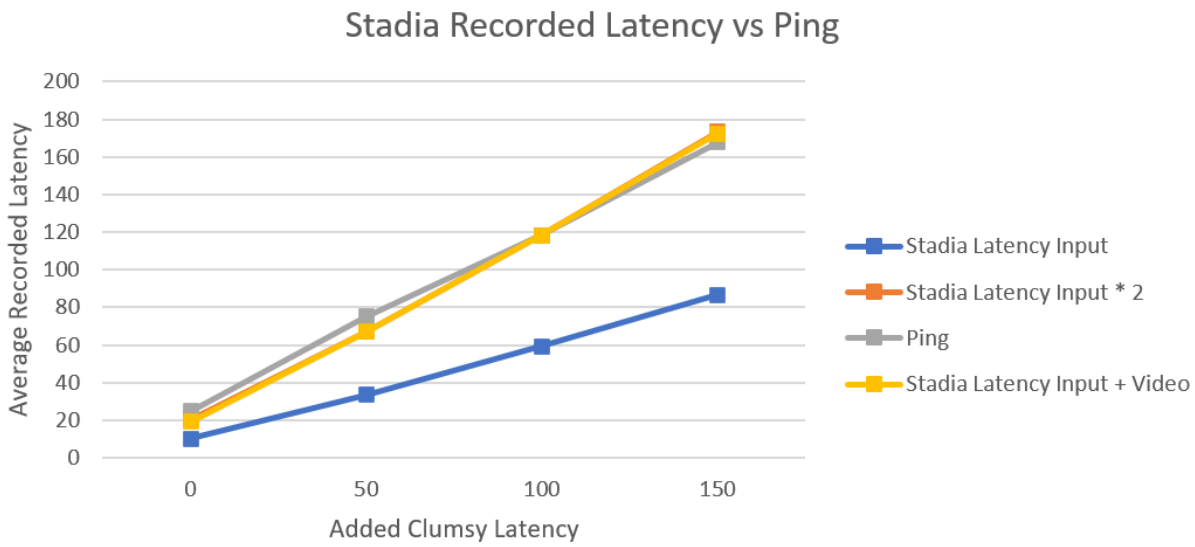


Figure 21. Graph comparing recorded latencies from Stadia's API to the recorded ping time to the Stadia server.

During our study, we had found that the values of latency being recorded from our game on Stadia when using just Stadia Latency Input were less than what should have been recorded. For example, with a Clumsy added latency of 150ms, we were reading roughly 80ms on Stadia. We then determined that the function we had been using to record latency was returning only half of the total round trip time.

To troubleshoot the issue, we began measuring ping to the instance IP address that our game was being hosted on. This was done using a Windows batch script that repetitively pinged the IP address and saved the output for analysis. We compared these ping time measurements to the Clumsy latency that was set at a given time and found that we were receiving a correct reading. We concluded that these ping values represent approximately the current latency being experienced by a player at a given time, but we realize that it may not be the most accurate representation.

Next, we compared the latency values we recorded from Stadia compared to the latency values retrieved by our pinging script as seen in Figure 21. Stadia provides two different latency records: network delay for input (blue line) and network delay for video. As seen in the graph, the Stadia Latency Input is much lower than the recorded ping

values. After multiplying the latency values recorded from Stadia by two (red), we found that the resulting values are nearly the same as the Stadia Latency Input + Video latency values and ping. Thus, for our purposes, the two are functionally the same, and in our analysis we can use the values gathered from Stadia Latency Input and multiply them by two.

4.1.4.3 Survey

As mentioned previously, participants were asked to complete a survey on Google Forms as they progressed through the study (See Appendix A). Before they began playing trials of Nova, they were asked to complete these questions:

1. How often do you play video games?
    a. On a 1-5 scale from never played to multiple hours a day
2. What method of controls do you use the most?
    a. Console controller
    b. Keyboard / Mouse
    c. N/A
3. Rate your experience with rhythm games
    a. On a 1-5 scale of None to A Lot

Each participant was also given a participant ID number at this stage for us to compare their answers to the gameplay data we collected. As they then proceeded through each trial of our game, they answered these questions immediately after a wave:

1. Rate your overall experience
    a. On a 1-5 scale from bad to excellent
2. Rate how much you agree with the following statements on a 1-5 scale from strongly disagree to strongly agree
    a. I was able to aim and shoot at the asteroids before they expired.
    b. The asteroids spawned in time to the music.
    c. There was a delay between when I clicked and the game responding.

3. Did you experience any difficulties? If so, please describe them here
    a. Short answer

Finally, once all of the waves were complete, the participants answered these questions as a way for us to collect their contact if they needed playtesting credit for a course or wanted to be considered for our raffle.

End Questions:

1. Do you require playtesting credit for an IMGD course?
    a. Yes
    b. No
2. Do you want to opt in to a raffle for a gift card?
    a. Yes
    b. No
3. If yes to either above question, what is your wpi email? (Will only be used to get you credit and/or for the raffle)

4.1.4.4 Data Collection

Aside from the data collected from the survey that participants filled out, we collected data based on technical information that was processed during each wave of a trial. We collected information including the wave number, current song, compensation on or off, amount of shots taken, amount of left and right clicks, amount of targets hit, amount of targets not hit, lowest health point and current latency being experienced. We also recorded information for each target that appeared including the target's ID number, window length, and the time taken for the player to shoot the target within its window. All of this information was recorded within the game during gameplay and saved to text (.txt) files. After each trial, we were able to download the output text files from the Stadia instance and save them for analysis.

4.2. Results

This section discusses the results of our user study. Figures in this section review the demographics, accuracy, window length, reaction time and quality of experience from participants.

4.2.1 Demographics

Our user study had 30 participants. Out of these 30 participants, 3 of the datasets we collected from their sessions were found to be invalid due to a technical error in Clumsy not working as intended. This brought our total to 27 valid participant sessions. From the demographic surveys for these 27, most of them play video games often, as seen in Figure 22 with median 4, mean 3.56, and standard deviation 1.25. The participants have little or no experience with rhythm games, as seen in Figure 23 with median 2, mean 2.59, and standard deviation 1.19.

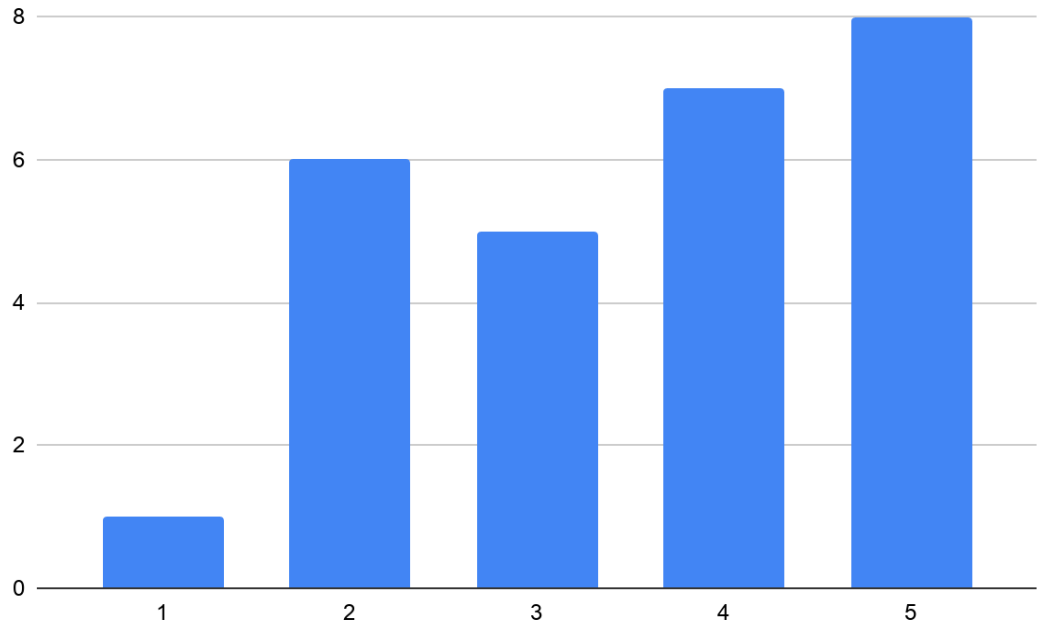


Figure 22. Histogram answering “How often do you play video games?”. 1 indicates the participant has never played and 5 indicates the participant plays multiple hours a day.

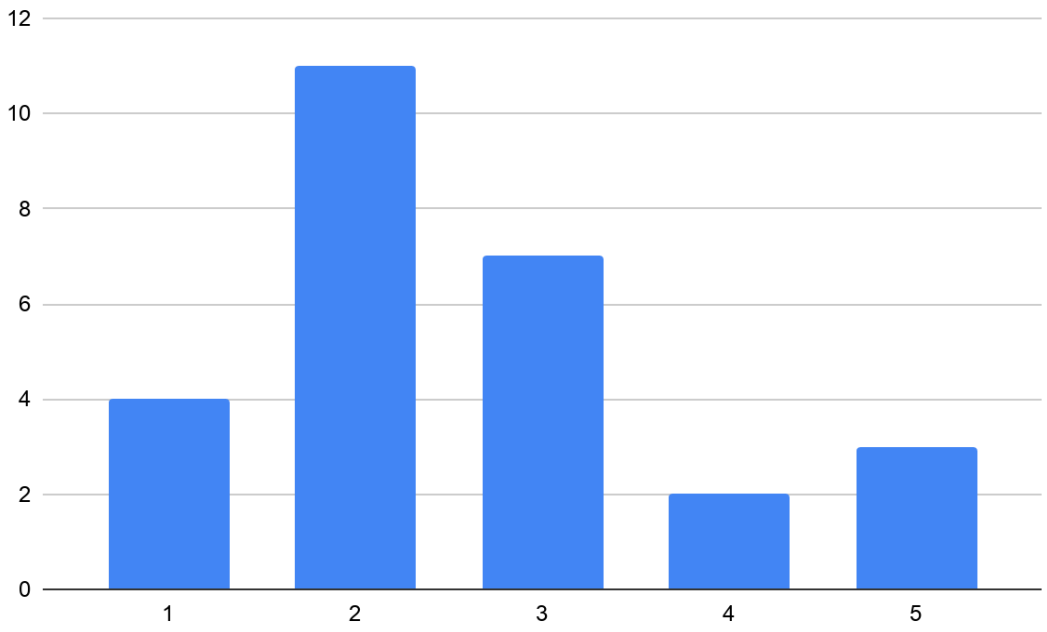


Figure 23. Histogram of user study participants answering “Rate your experience with rhythm games”. 1 indicates no experience and 5 indicates a lot of experience.

4.2.2 Window Length

As mentioned in Section 4.1.4.1, our study contained trials with varying difficulties, compensation, and latencies. Generally, our graphs show two data sets: compensation for latency off (blue), and compensation for latency on (red). Additionally, each variable comparison has two figures: one for easy difficulty (left) and one for hard difficulty (right).

Figures 24 and 25 show the calculated target window length in seconds versus measured latency in milliseconds. The blue non-compensated window lengths are consistent throughout the trials as expected. The red compensated window lengths show an increasing trend in both the easy and hard difficulties. This shows how Nova increases target window length with latency.

The trendline equations for both charts are as follows:

Easy, Compensation On: $y = 0.0044x + 0.46$

$R^2 = 1$

Hard, Compensation On: $y = 0.0044x + 0.96$

$R^2 = 1$

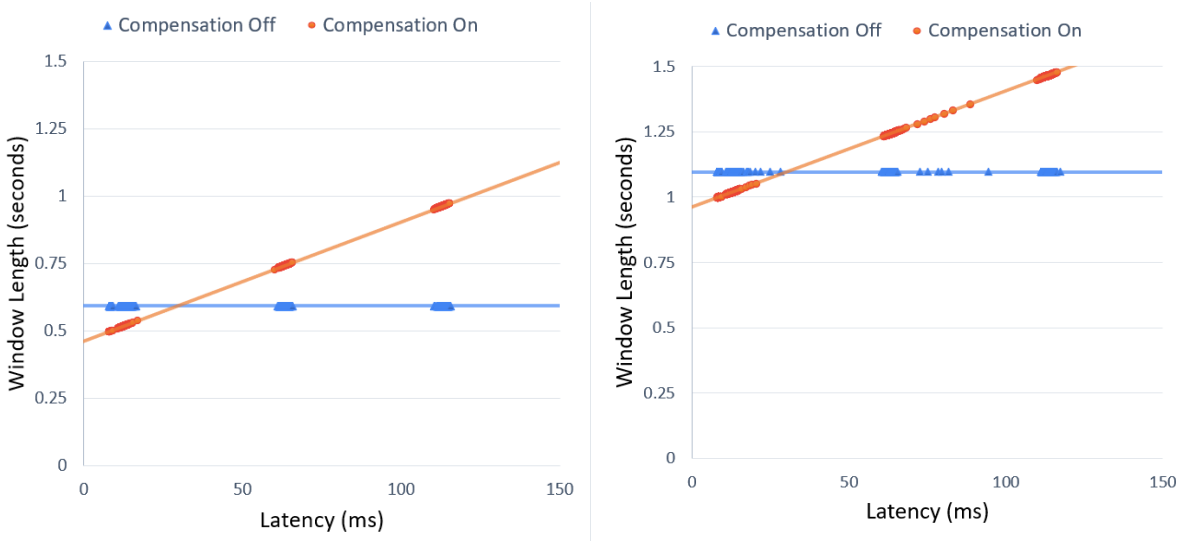


Figure 24 and 25. Calculated window length (time a target is available to be shot by the player) vs recorded latency compensation on and compensation off. Easy difficulty (left) and hard difficulty (right)

4.2.3 Accuracy

Figures 26 and 27 display the average change in the player's target accuracy compared to added Clumsy latency. The points show the mean values for all participants at that latency with the bars showing 95% confidence intervals. Accuracy is calculated as the number of targets hit within a trial over the total number of targets spawned in that trial, multiplied by 100. The change in accuracy is relative to the measured accuracy with 0 milliseconds of added latency. For the easy difficulty, the change in latency values had no significant difference in accuracy. This could be because the challenge presented by our "easy" difficulty was not enough for players to be unable to hit the targets. For the hard difficulty, there is a decline in change in accuracy as latency increases. This shows that as latency increases, accuracy decreases. In addition, when latency compensation is off, accuracy decreases at a more severe rate. When latency compensation is on, there is a smaller change in accuracy. This shows that our latency compensation is helping players maintain a higher accuracy in the presence of latency.

The trendline equations for both charts are as follows:

Easy, Compensation Off: $y = -0.0055x + -0.2$

$R^2 = 0.615$

Easy, Compensation On: $y = -0.0011x + 1.04$

$R^2 = 0.004$

Hard, Compensation Off: $y = -0.12x + 1.63$

$R^2 = 0.876$

Hard, Compensation On: $y = -0.07x + 1.44$

$R^2 = 0.874$

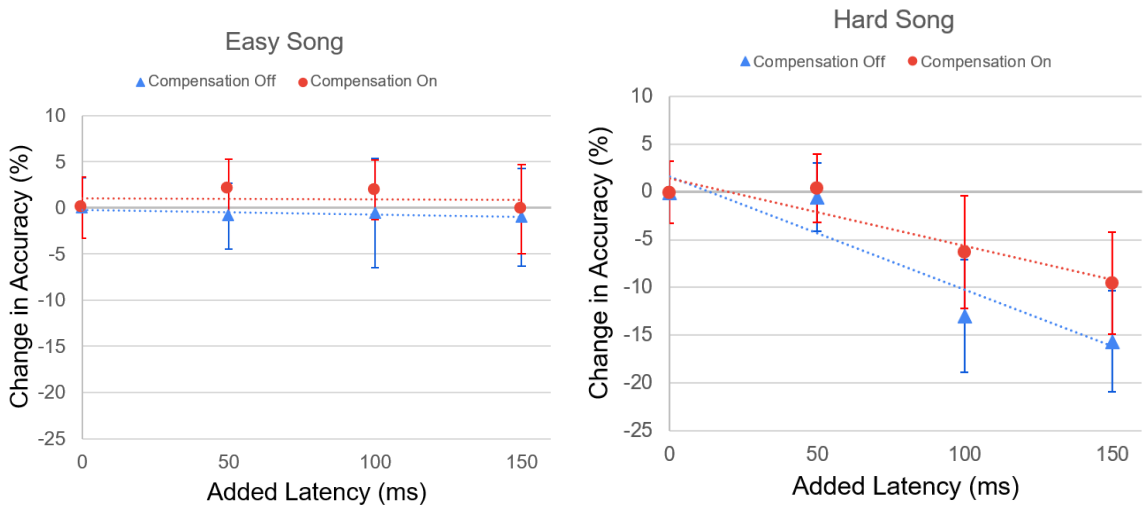


Figure 26 and 27. Change in accuracy compared to added latency (milliseconds). Change is relative to the average accuracy with 0 ms of latency. Easy difficulty (left) and hard difficulty (right).

4.2.3.1 Statistical Significance

Since our dataset includes multiple trials from the same subjects, with compensation on and off, we used a paired t-test to determine if the difference in accuracy between the two groups are statistically significant. We compared the difference in accuracy values between the trials with compensation on and with compensation off for both song difficulties. For the “easy” song, we determined there

was no significant effect on the difference in the trials with compensation on (M= 97.6, SD=4.1) and without compensation (M=96.4, SD 5.2), t(98) = 1.86, p = .0693.

For the hard song, there was a significant difference in accuracy for trials using compensation (M=93.5, SD=9.9) and no compensation (M= 89.1, SD= 14.6), t(98) = 3.03, p = 0.00313. This could be because the “easy” song was too easy. Players might have been able to hit the targets in high latencies because they were given a lot of time to aim at the next target. The easy song had a higher cooldown, creating longer pauses between targets. The majority of our playtesters played video games regularly which could have been why our “easy” song was not challenging enough to provide results.

To assess the effect of latency on player accuracy with both compensation off and on, we ran ANOVA tests on the four latency groups. The results of these ANOVA tests are summarized in Tables 2-3 for compensation off, and Tables 4-5 for compensation on. In both tests, the F-value calculated (6.909 for compensation off, 6.093 for compensation on) was greater than the critical F-value (2.652 for compensation off, 2.651 for compensation on), indicating that our latency compensation did not mitigate the effect of latency on player accuracy enough, possibly because there was not enough compensation (ie. the windows were not made large enough).

Added Latency (ms)	**P-value ($p < 0.05$ for significance, significant values bolded)**
0	0.207
50	**0.020**
100	**0.0004**
150	**0.036**

Table 1. Resulting p values from t-test on player accuracy for the hard song, compensation on v.s. compensation off.

Table 1 shows the p values of paired t-tests at each latency value for the hard song, comparing player accuracy with compensation off and compensation on. From

the table, three of our latency values (50ms, 100ms, and 150ms), there was a statistically significant difference between compensation on and compensation off. This suggests that our latency compensation was effective in increasing accuracy for higher latency game conditions.

SUMMARY - Accuracy, Compensation off		
Groups	**Average (percent)**	**Standard Deviation**
0ms	96.6	24.0
50ms	95.9	35.9
100ms	89.9	234.7
150ms	88.5	190.6

Table 2. A summary of average and standard deviation for each latency in terms of accuracy with compensation off for the hard song.

ANOVA - Accuracy, Compensation off						
Source of Variation	**SS**	**df**	**MS**	**F**	**P-value**	**F crit**
Between Groups	2538.325	3	846.108	6.909	0.00019	2.652
Within Groups	23514.10	192	122.469			
Total	26052.42	195				

Table 3. An ANOVA test for player accuracy for the hard song for all four latency groups, with compensation off.

SUMMARY - Accuracy, Compensation on		
Groups	**Average (percentage)**	**Standard Deviation**
150ms	92.2	117.0
100ms	94.6	76.8
50ms	98.1	13.0
0ms	96.8	19.6

Table 4. A summary of average and standard deviation for each latency in terms of accuracy with compensation on for the hard song.

ANOVA - Accuracy, Compensation on						
Source of Variation	**SS**	**df**	**MS**	**F**	**P-value**	**F crit**
Between Groups	1024.973	3	341.658	6.034	0.00060	2.651
Within Groups	10985.510	194	56.626			
Total	12010.483	197				

Table 5. An ANOVA test for player accuracy for the hard song for all four latency groups, with compensation on.

4.2.4 Reaction Time

Figures 28 and 29 show the average reaction time percentage for each trial compared to latency added by Clumsy. Reaction time percentage is calculated as the time taken for the player to shoot a target within its window, over the target's total window length multiplied by 100. With compensation off for both difficulties, reaction

time increases as latency increases. This is because the window length maintains the same value and the increase in latency makes it more difficult to aim. With compensation on, there is a smaller overall reaction percentage for each value over 0 ms since the larger time window makes targets easier to hit. This shows that players were given a better chance to succeed at hitting targets when latency compensation is on.

The trendline equations for both charts are as follows:

Easy, Compensation Off: $y = 0.05x + 31.6$

$R^2 = 0.8105$

Easy, Compensation On: $y = -0.07x + 34.3$

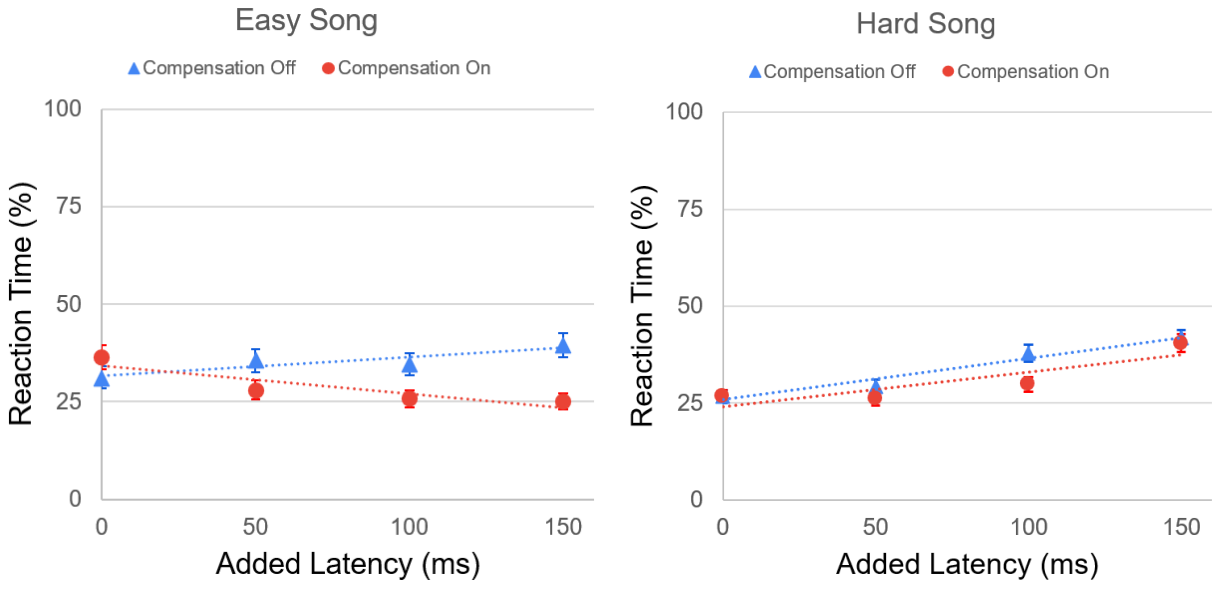
$R^2 = 0.808$

Hard, Compensation Off: $y = 0.11x + 25.8$

$R^2 = 0.9642$

Hard, Compensation On: $y = 0.09x + 24.1$

$R^2 = 0.752$



Figures 28 and 29. Average Reaction Percentage ((Time to shoot target within window / target window length) * 100) vs added latency (milliseconds). Easy difficulty (left) and hard difficulty (right)

Players' increased performance when compensation is on can also be seen in the change in reaction time. The reaction time is the time the player hit a target inside its window divided by the amount of time they had to hit the target (window length). We expected that with more latency, players would hit the target later in the window. The graphs in Figure 28 and Figure 29 show our results. When compensation is off, the trendline for reaction percentage increases with latency. For the trials that do use compensation, we expected the reaction percentage would be lower because the window size had increased. For both the "easy" song and the "hard" song, the trials using compensation had a slightly lower trendline. We intended the reaction time to be constant across all latencies, but there was still an increased reaction time for the hard song. This suggests that more compensation (i.e. larger windows) may be needed.

4.2.5 Quality of Experience

After each round, participants were asked a question similar to Google Stadia's QoE question: "Rate your overall experience". Possible responses are: "5 - Excellent", "4 - Good", "3 - Fair", "2 - Poor", "1 - Bad". Figure 30 shows the effects of added Clumsy latency on average quality of experience (QoE) for the hard difficulty. As latency increased, QoE decreased overall. However, with compensation on, QoE had a higher average than when compensation was off.

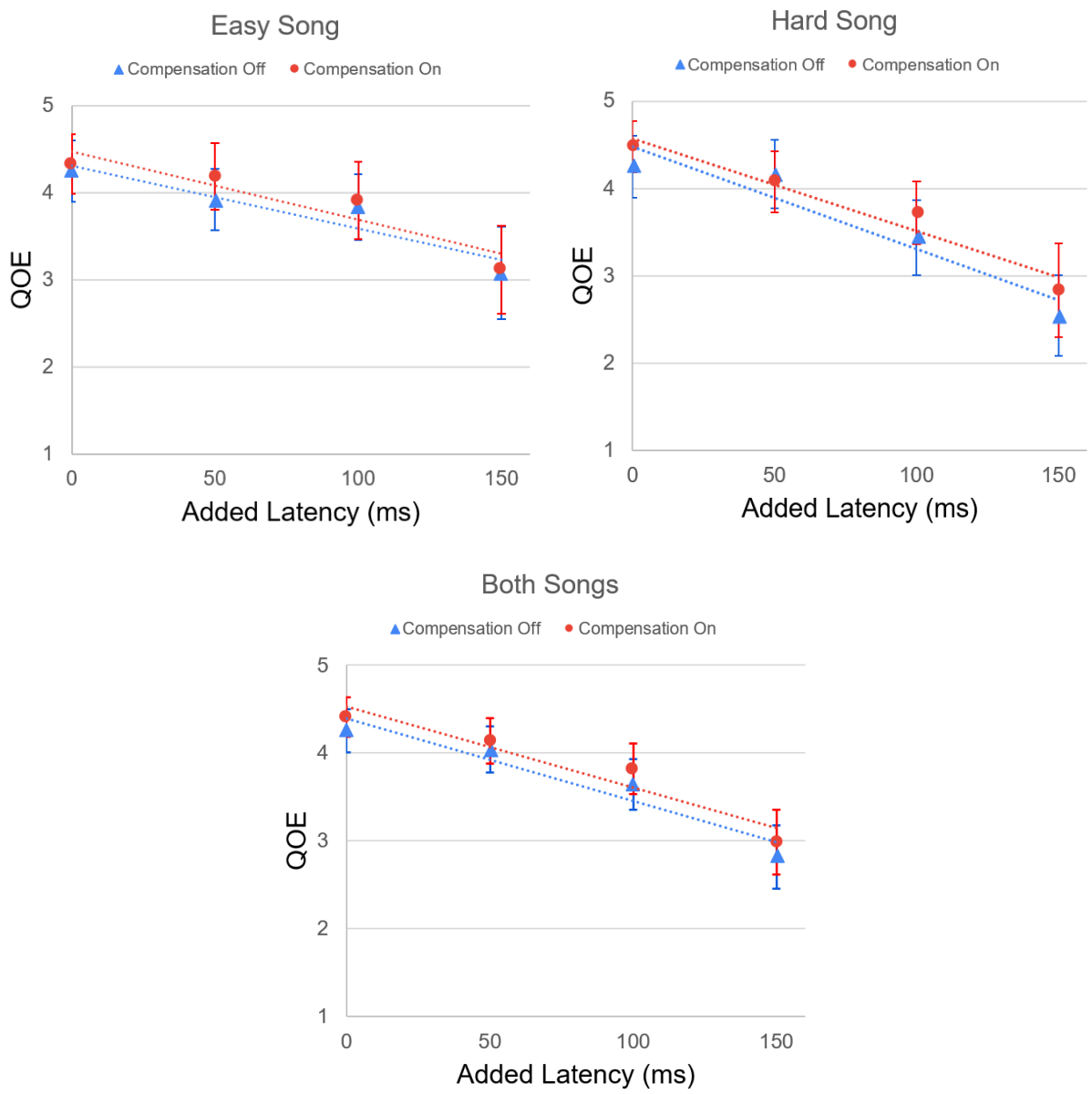


Figure 30. Quality of Experience to added latency (milliseconds)

4.2.5.1 Statistical Significance

We tested if users' Quality of Experience (QoE) was higher with compensation on. We used a paired t-test to determine if our latency compensation on resulted in a statistically significant difference in a player's QoE compared to latency compensation off. For the "hard" song there was a significant effect on the difference in QoE for

compensation on (M=3.85, SD=1.2) and compensation off (M=3.7, SD=1.2), t(197)= 1.87, p = 0.0325. Note, participants played with the same latency across four successive trials for each latency. This could have caused their QoE response to be similar across the four latencies.

Added Latency (ms)	P-value (p < 0.05 for significance, significant values bolded)
0	0.168
50	0.342
100	**0.035**
150	0.178

Table 6. Resulting p values from T-test on player QoE (both Easy and Hard songs), compensation on vs. compensation off.

To assess the effects of latency on QoE with both compensation off and on, we took the same approach that we used for player accuracy: We ran ANOVA tests on all four latency groups. The results of these ANOVA tests are summarized in Tables 7-8 for compensation off, and Tables 9-10 for compensation on. In both tests, the F-value we calculated (17.151 for compensation off, 17.00924 for compensation on) was greater than the critical F-value (2.651 for compensation off, 2.651 for compensation on).

This suggests that regardless of latency compensation, there was a significant difference in QoE between all four added latency values. Thus, our latency compensation did not mitigate the effect of latency on QoE enough. Similar to accuracy, this may be because we did not apply enough latency compensation to have a constant QoE regardless of latency.

SUMMARY - QoE, compensation off		
Groups	**Average (percentage)**	**Standard Deviation**
0ms	4.2	0.719
50ms	4.0	0.937
100ms	3.6	1.133
150ms	2.8	1.695

Table 7. A summary of average and standard deviation for the hard song for each latency in terms of QoE with compensation off.

ANOVA - QoE, compensation off						
Source of Variation	**SS**	**df**	**MS**	**F**	**P-value**	**F crit**
Between Groups	57.63699443	3	19.212	17.151	6.4E-10	2.651
Within Groups	217.318	194	1.120			
Total	274.955	197				

Table 8. An ANOVA test for QoE for the hard song for all four latency groups, with compensation off.

SUMMARY - QoE, compensation on		
Groups	**Average (percentage)**	**Standard Deviation**
0ms	4.408	0.663
50ms	4.135	0.942
100ms	3.816	1.0697
150ms	2.98	1.816

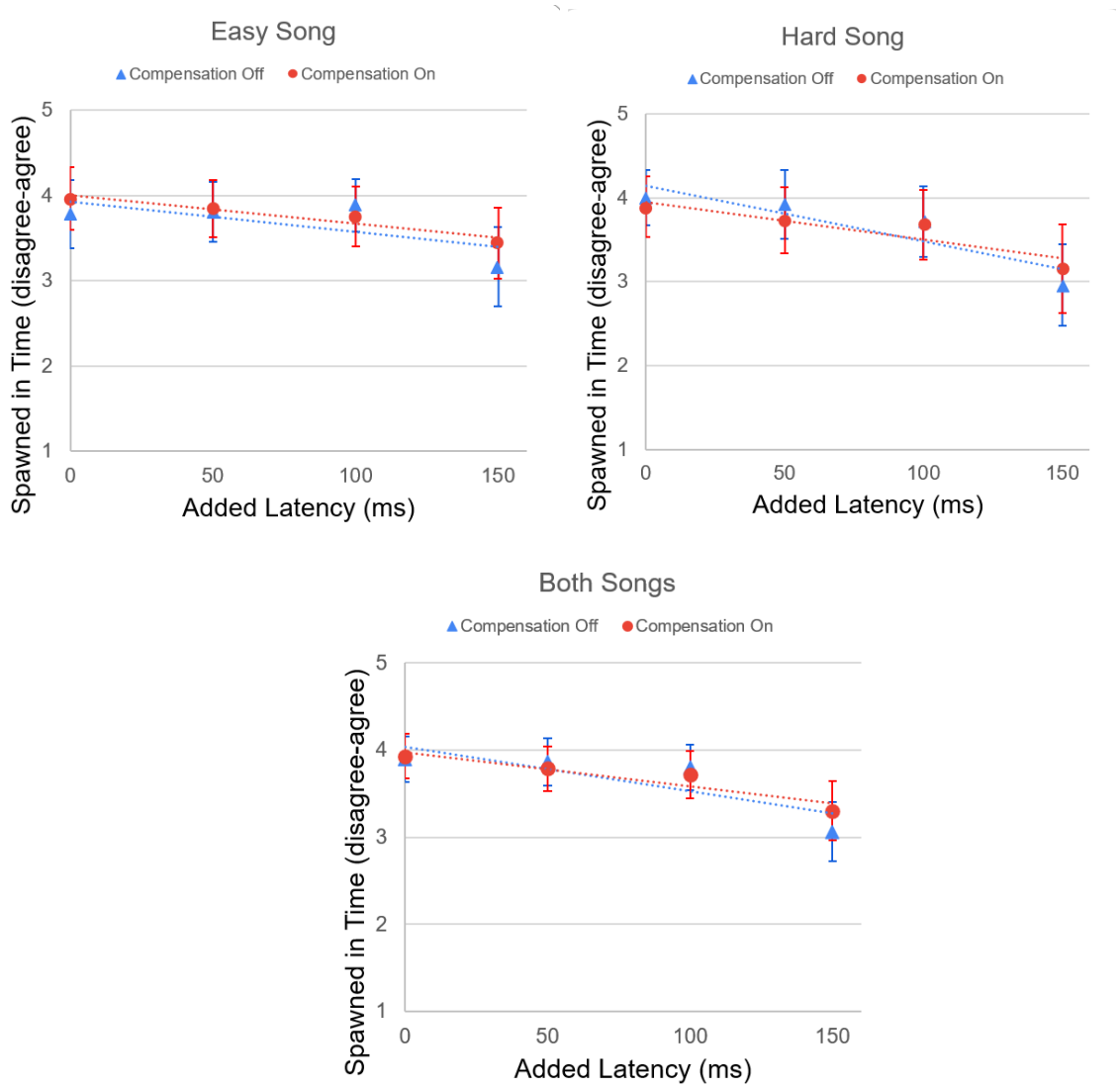
Table 9. A summary of average and standard deviation for the hard song for each latency in terms of QoE with compensation on.

ANOVA - QoE, compensation on						
Source of Variation	**SS**	**df**	**MS**	**F**	**P-value**	**F crit**
Between Groups	57.334	3	19.111	17.00924	7.37E-10	2.651
Within Groups	220.221	196	1.124			
Total	277.555	199				

Table 10. An ANOVA test for QoE for the hard song for all four latency groups, with compensation on.

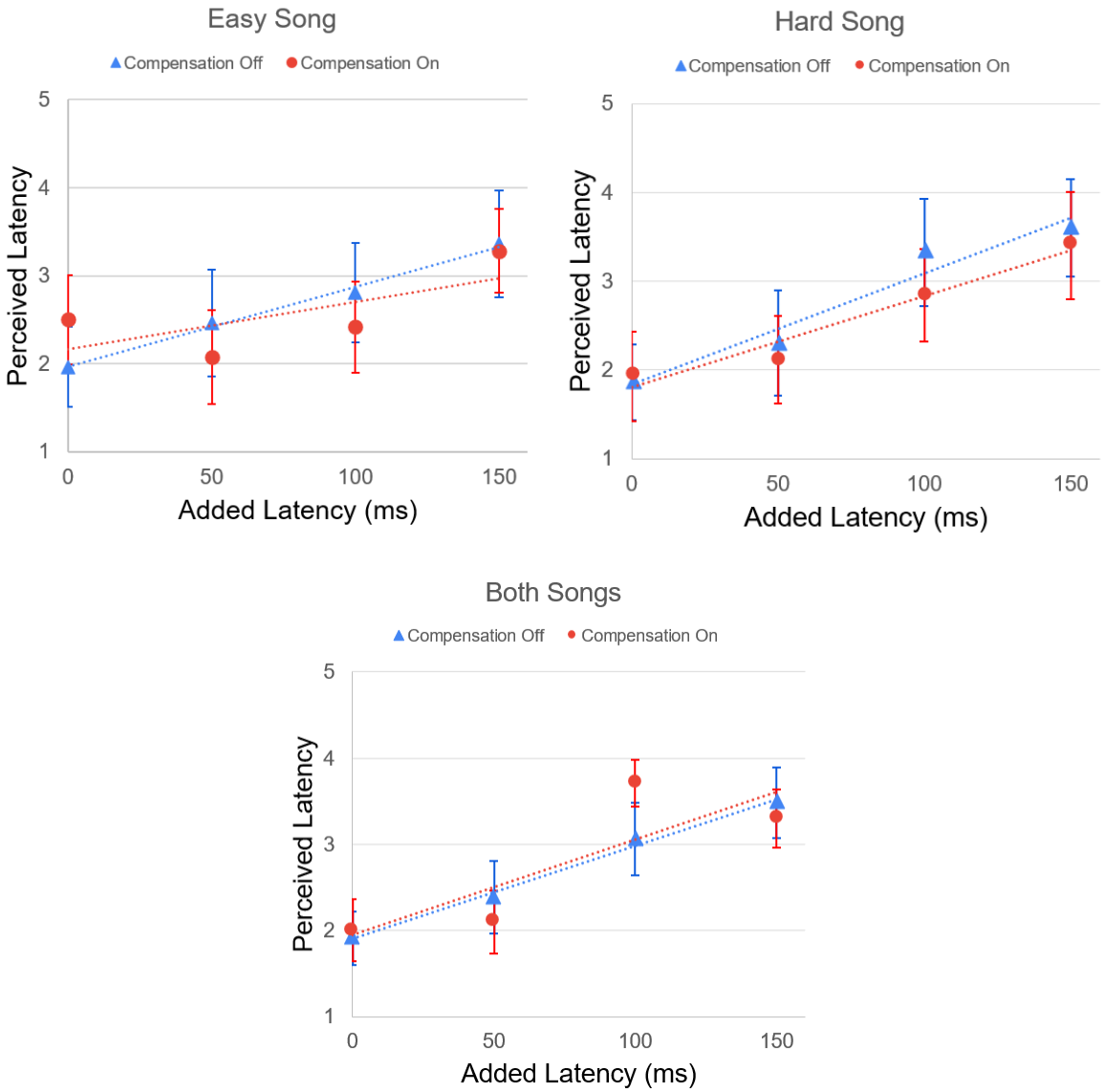
4.2.6 Survey Responses

Our survey results for the question “The asteroids spawned in time to the music.” are shown in Figures 31, 32 and 33. On a scale from 1 to 5, 1 being “Strongly Disagree” and 5 being “Strongly Agree”, participants answered lower on the scale as latency increased for both the hard and the easy difficulties. For the easy difficulty, when compensation was on, participants answered overall higher for this question. For the hard difficulty, participants answered lower than with compensation off for latencies 0 and 50 ms and more than compensation off for 100 and 150ms.



Figures 31, 32 and 33. Survey responses to the question “The asteroids spawned in time to the music.” with responses on a scale of 1-5 with 1 being “Strongly Disagree” and 5 being “Strongly Agree” compared to added latency in milliseconds.

Figures 34, 35 and 36 illustrate participants' answers to the question “There was delay between when I clicked and the game responding.” with answers on a scale of 1-5 with 1 with 1 being “Strongly Disagree” and 5 being “Strongly Agree”. As latency increased for both difficulties, perceived delay increased. For the easy difficulty, responses showed that perceived delay was greater with compensation off than with compensation on for 100 and 150ms. For the hard difficulty, perceived delay was consistently lower while compensation was on as actual latency increased.



Figures 34, 35 and 36. Answers to the question “There was delay between when I clicked and the game responding.” with answers on a scale of 1-5 with 1 with 1 being “Strongly Disagree” and 5 being “Strongly Agree”.

4.2.7 Limitations

In our user study we only compensated for half the actual latency. This was because the function used to get the latency only returned latency in one direction (outbound/uplink). We did not account for this in our formula adjusting the window length.

Players only played each trial for 45 seconds. This means they were not playing under normal gameplay conditions of a full song and may not have had enough time to feel a difference between rounds. Finally, the beginning of the session was the first time the participants had ever played our game. Throughout the trials, their accuracy could have increased as they gained more experience with the game.

4.2.8 Summary

Based on our analysis, players perform better, react faster, and have a slightly higher quality of experience when playing our game with compensation on. There is a downward slope in accuracy as latency increases, but with compensation on, the slope is shallower than with compensation off.

5. Conclusion

In order to determine if the latency compensation technique of attribute scaling could be effective in increasing players' Quality of Experience (QoE) and performance during periods of high latency in cloud gaming, we developed Nova, a rhythm shooter game. The objective of the game is to aim and shoot at as many targets or "notes" as they appear to the beat of a playing song. Nova included latency compensation in the form of longer deadlines, thus giving the player more time to complete this action as latency increased.

To assess the effectiveness of our latency compensation model, we performed a user study on a total of 30 participants, who played through several randomized rounds of Nova with varying latencies and latency compensation on/off status.

We found that our latency compensation was effective in increasing player performance as latency increased, and in increasing player QoE as latency increased.

Our goal was to keep player performance and QoE constant as latency increased, but even with compensation, both saw a downward trend. Perhaps stronger latency compensation could achieve this goal.

6. Future Work

Future work includes running the study again with adjusted code that compensates for the full latency and with harder songs. Compensating for the full latency would help determine if our original formula provides the right level of compensation. Also, we found that playing a harder song resulted in a higher contrast in the data so testing with a wider range of song difficulties would allow us to understand if our technique works for different difficulties. To test these ideas we would add Stadia latency in both directions (uplink/outbound and downlink/inbound) together in the code and use this value when calculating the average latency. This new average latency would then be used in our equation for window length. In our study we used cooldown values of 1(hard) and 1.5 (easy). If running the study again we would instead use cooldown values lower than 1. The same process as we described in our methodology could be used for an additional study with these parameters.

Due to our short deadline for analyzing our data we were unable to analyze all of it. If we had more time we would look at our other survey questions and how those relate to compensation and latency. For example testing if perceived latency was similar to actual latency and if compensation affected the players perception of the targets spawning with the rhythm of the song. We would plot graphs comparing the answers on a scale from 1-5 and latency. We would split up the results per song to see if there was a difference in answers after playing the easy rounds and difficult rounds.

Nova at its current form uses procedural generation to determine at which points in a song to place targets. We do not know the effect this has on QOE as players might feel targets are not spawning with the rhythm. Future work could include creating a version without procedural generation and running a user study to compare note spawning techniques. This would help determine the effect of procedural generation on our results.

Another future project could include testing different methods of attribute scaling with Nova. Other scales could be target size, hitbox size, or even scaling multiple attributes at the same time. Testing multiple techniques on the same game would allow us to know if different techniques used on the same game can have the same or better results. Also, because our game uses procedural generation, the difficulty can be scaled based on latency during the runtime. To test this idea, we would run the study again, only changing the technique used.

References

[1] A. McAloon, “Breaking down nearly 50 years of video game revenue.” /view/news/335555/Breaking_down_nearly_50_years_of_video_game_revenue.php (accessed Oct. 15, 2020).

[2] “2020 Gaming Industry Statistics, Trends & Data (Biggest Study),” *GamingScan*. https://www.gamingscan.com/gaming-statistics/ (accessed Oct. 07, 2020).

[3] S. S. Sabet, S. Schmidt, S. Zadtootaghaj, B. Naderi, C. Griwodz, and S. Möller, “A latency compensation technique based on game characteristics to mitigate the influence of delay on cloud gaming quality of experience,” in *Proceedings of the 11th ACM Multimedia Systems Conference*, Istanbul Turkey, May 2020, pp. 15–25, doi: 10.1145/3339825.3391855.

[4] Robert Salay. A Comparison of Automatic versus Manual World Adjustment for Network Game Latency Compensation, *M.S. Thesis*, Interactive Media and Game Development, Worcester Polytechnic Institute, Summer 2020.

[5] “Store - Stadia.” https://stadia.google.com/store (accessed Oct. 10, 2020).

[6] M. Claypool and K. Claypool, “Latency and Player Actions in Online Games,” *Communications of the ACM*, vol. 49, no. 11, pp. 40–45, Nov. 2006, doi: https://doi-org.ezpxy-web-p-u01.wpi.edu/10.1145/1167838.1167860.

[7] DevinDTV, *How It Works: Lag compensation and Interp in CS:GO*. 2015. https://www.youtube.com/watch?v=6EwaW2iz4iA&ab_channel=DevinDTV

[8] “Stadia-compatible gamepads and screens,” *Stadia Help*. https://support.google.com/stadia/answer/9578631 (accessed Oct. 12, 2020).

[9] Y. W. Bernier, “Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization,” p. 13.

[10] I. Lee, S. Kim, and B. Lee, “Geometrically Compensating Effect of End-to-End Latency in Moving-Target Selection Games,” in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems - CHI ’19*, Glasgow, Scotland Uk, 2019, pp. 1–12, doi: 10.1145/3290605.3300790.

[11] M. Long and C. Gutwin, “Effects of Local Latency on Game Pointing Devices and Game Pointing Tasks,” in *Proceedings of the 2019 CHI Conference on Human*

Factors in Computing Systems, New York, NY, USA, May 2019, pp. 1–12, doi: 10.1145/3290605.3300438.

[12] K. A. Rahman, R. McCool, and G. Somadder, *Gaming in the Cloud: A Technical Deep Dive*. May 15, 2019. Accessed on: Sep 29, 2020. [Video file]. Available: https://youtu.be/K33gctpveuk.

[13] S. S. Sabet, S. Schmidt, S. Zadtootaghaj, C. Griwodz, and S. Moller, “Delay Sensitivity Classification: Towards a Deeper Understanding of the Influence of Delay on Cloud Gaming QoE,” p. 6, Apr. 2020, doi: 10.1145/3386293.3397116.

[14] S. Shafiee Sabet, S. Schmidt, S. Zadtootaghaj, C. Griwodz, and S. Moller, “Towards Applying Game Adaptation to Decrease the Impact of Delay on Quality of Experience,” in *2018 IEEE International Symposium on Multimedia (ISM)*, Taichung, Dec. 2018, pp. 114–121, doi: 10.1109/ISM.2018.00028.

[15] M. Claypool and K. Claypool, “Latency can kill: precision and deadline in online games,” in *Proceedings of the first annual ACM SIGMM conference on Multimedia systems - MMSys ’10*, Phoenix, Arizona, USA, 2010, p. 215, doi: 10.1145/1730836.1730863.

[16] “Audio Synesthesia.” https://docs.unrealengine.com/en-US/Engine/Audio/Synesthesia/index.html (accessed Oct. 16, 2020).

[17] E. Carlson, T. Y. Fan, and Z. Guan, “Towards Usable Attribute Scaling for Latency Compensation in Cloud-based Games.” Interactive Qualifying Project. Worcester Polytechnic Institute.

[18] “Freesound.” *Freesound.* https://freesound.org/ (accessed Mar. 14, 2021).

[19] "Internet Speed Test. | Fast.com" Fast.com. https://fast.com (accessed Mar. 10, 2021)

[20] “clumsy, an utility for simulating broken network for Windows Vista / Windows 7 and above” GitHub. https://jagt.github.io/clumsy/ (accessed Mar. 10, 2021)

List of Figures and Tables

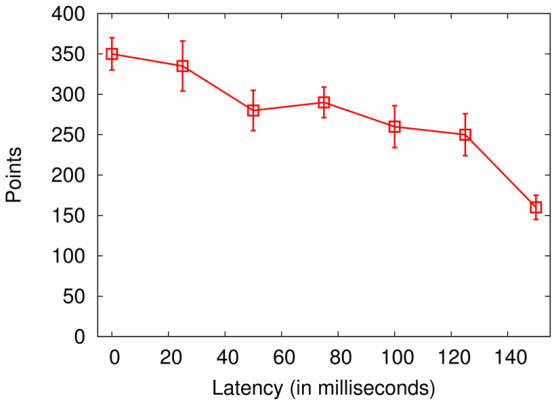


Figure 1. Player performance (in points) for Crazy Taxi vs. Latency on OnLive cloud gaming system.

Source: [6]

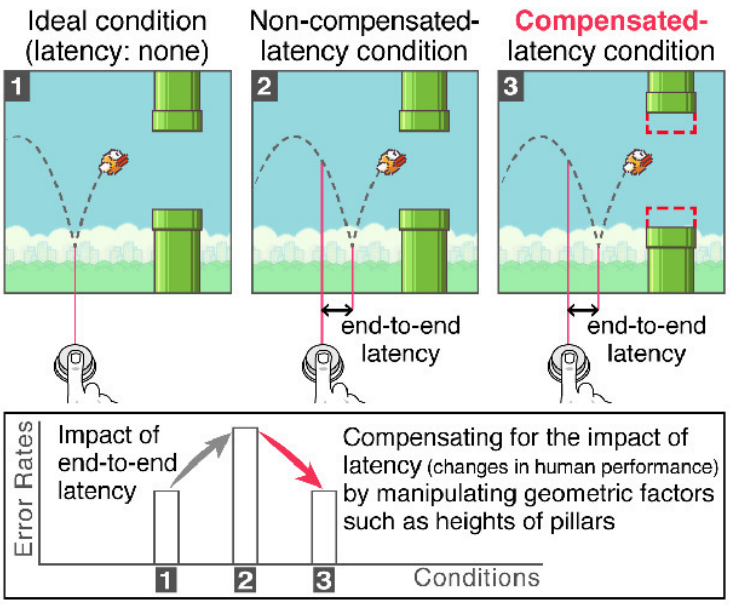


Figure 2. Overview of latency compensation through geometric scaling.

Source: [10]

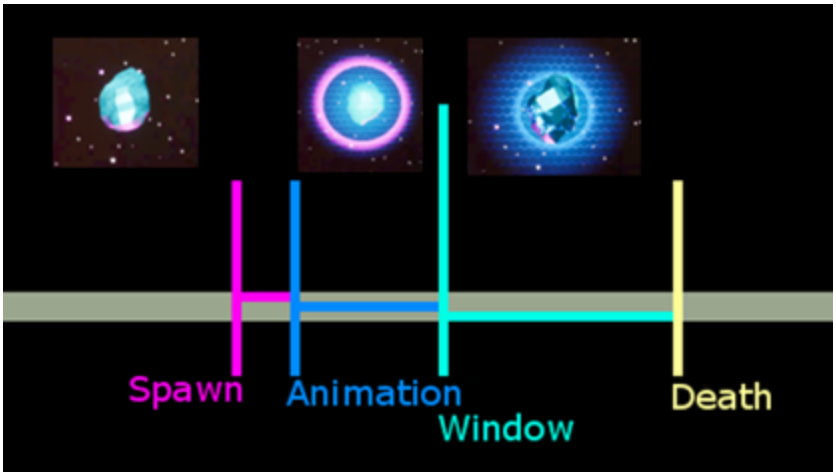


Figure 3. A visual representation of a target's lifetime, from spawn to death.

Final Equation:

*Window = (Average Latency * 0.008868 + Accuracy / 0.09624) – ((Cooldown -0.25)* 1.003) - 6.598,*

Where *Accuracy = 0.8.*

Figure 4. The final equation for a target's window.

```
#include "StadiaLatencyNodes.h"
#if PLATFORM_STADIA
#include <chrono>
#include "ggp/ggp.h"
#endif

float UStadiaLatencyNodes::GetStadiaLatency()
{
#if PLATFORM_STADIA
    std::chrono::microseconds latency = ggp::GetNetworkDelayForInput(ggp::GetPrimaryPlayerId(), 0);
    float lag = latency.count()/1000;
    return lag;
#endif
    return -1.0;
}
```

Figure 5. Code to retrieve latency from the Stadia servers and make it accessible via Blueprints.

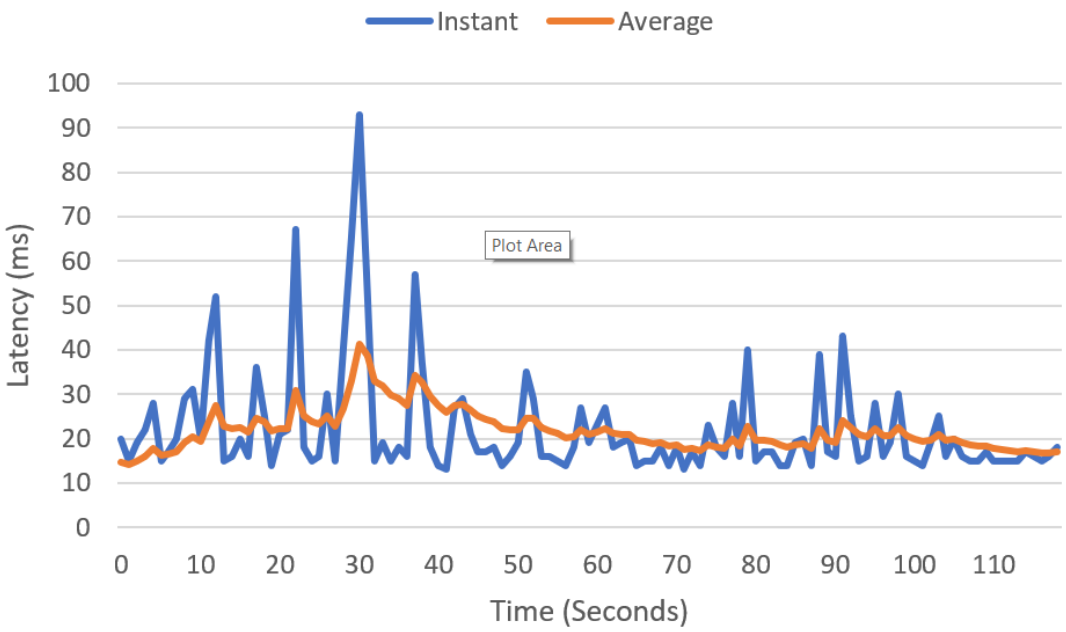


Figure 6. Google Stadia instant latency (blue) and average latency (orange) vs. time.

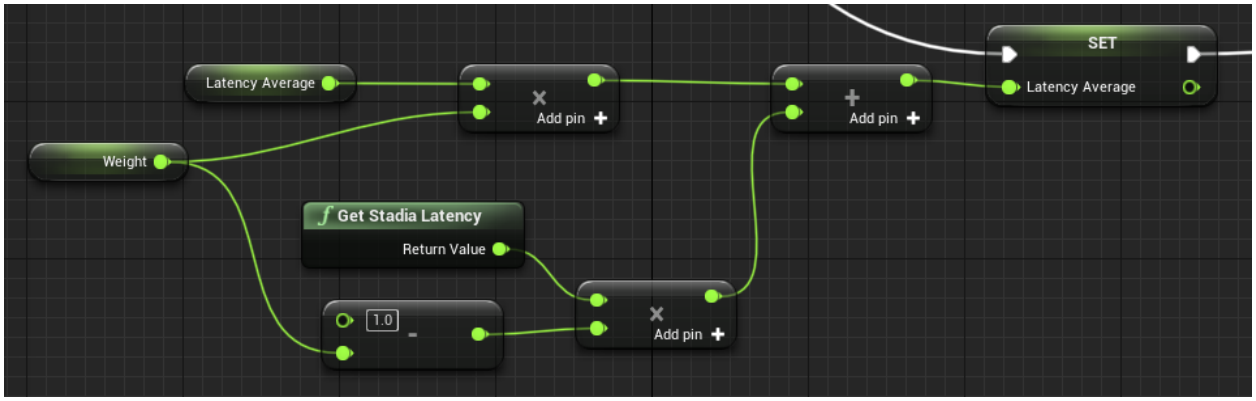


Figure 7. Blueprint code calculating latency average

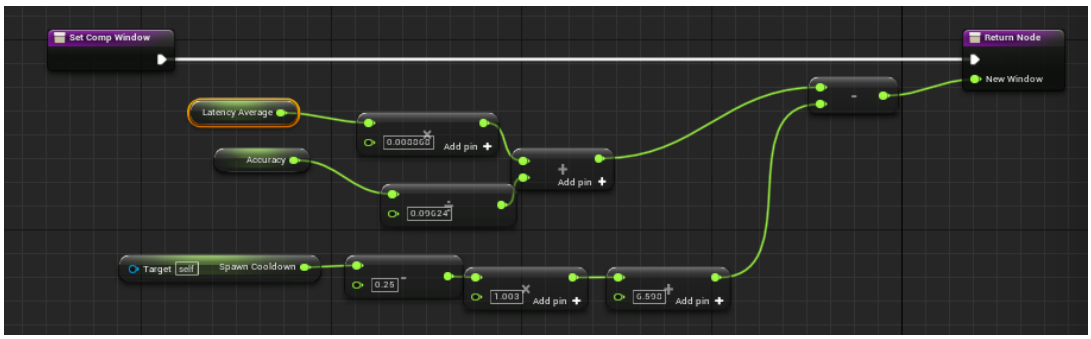


Figure 8. Blueprint code scaling the window length.

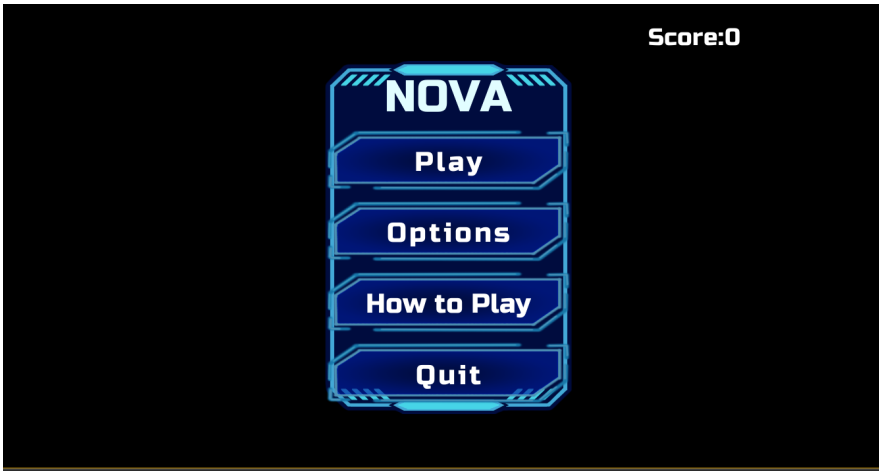


Figure 9. Main Menu Screen.

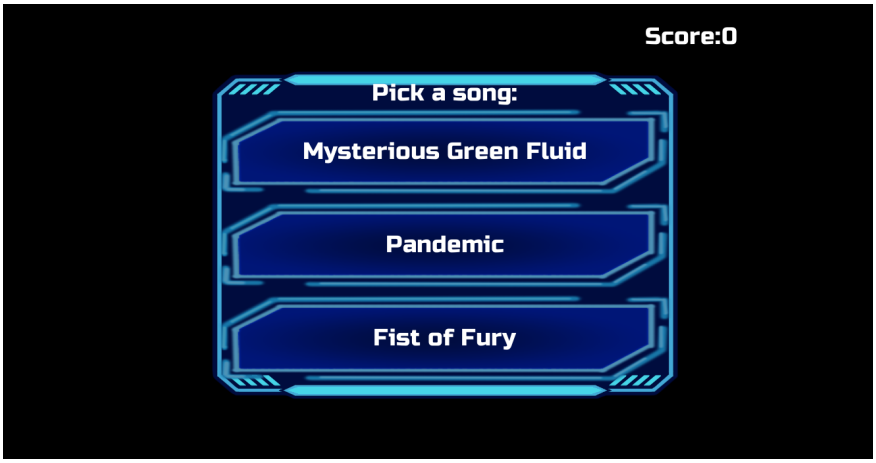


Figure 10. Song Choices Screen

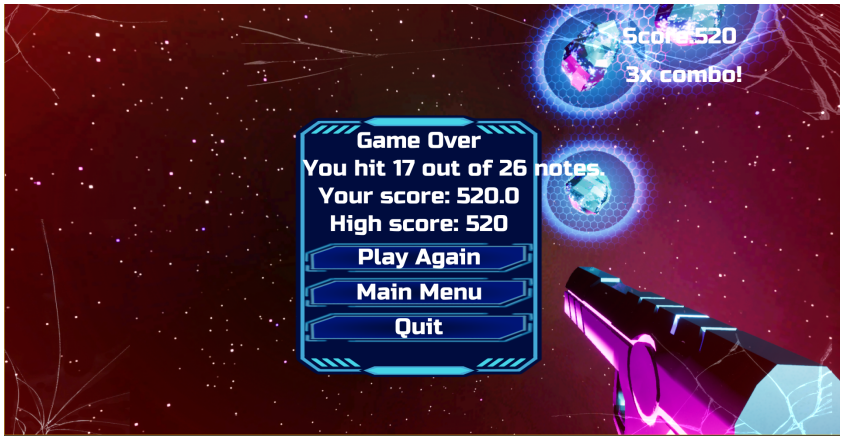


Figure 11. Game Over Screen

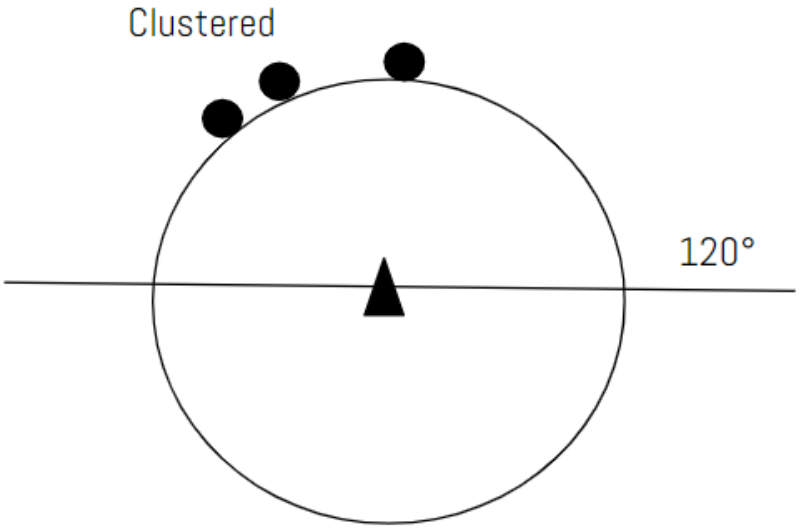


Figure 12. Illustration of targets clustering in front of the player.

Left Click Target:

Right Click Target:

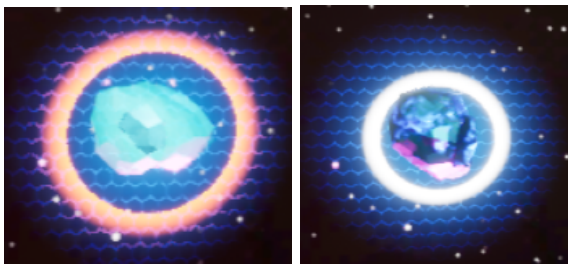


Figure 13. Comparison of both types of targets

Figure 14. Screenshot displaying both target types

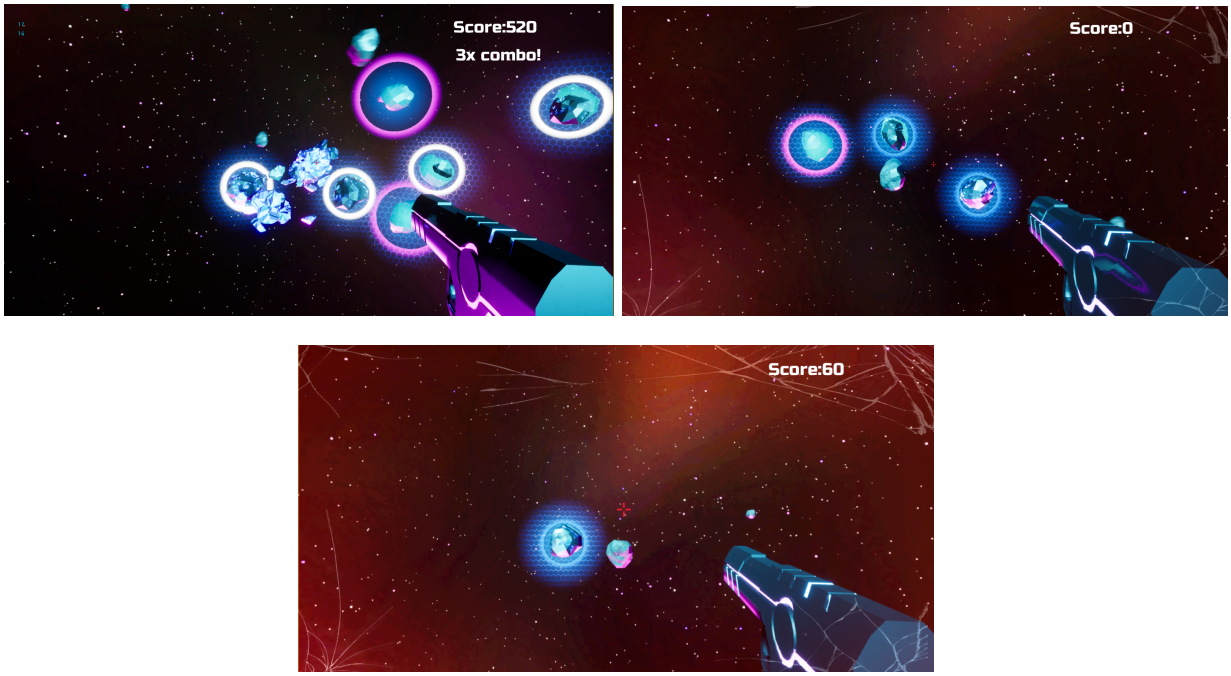


Figure 15. Illustration of health decreasing in the game.

Figure 16. Nova concept art (left), the Nova beta build (middle), and Nova final build (right).

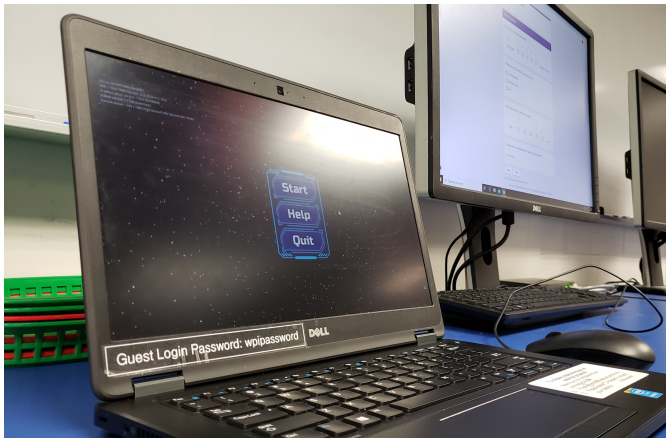


Figure 17. Picture of lab setup.

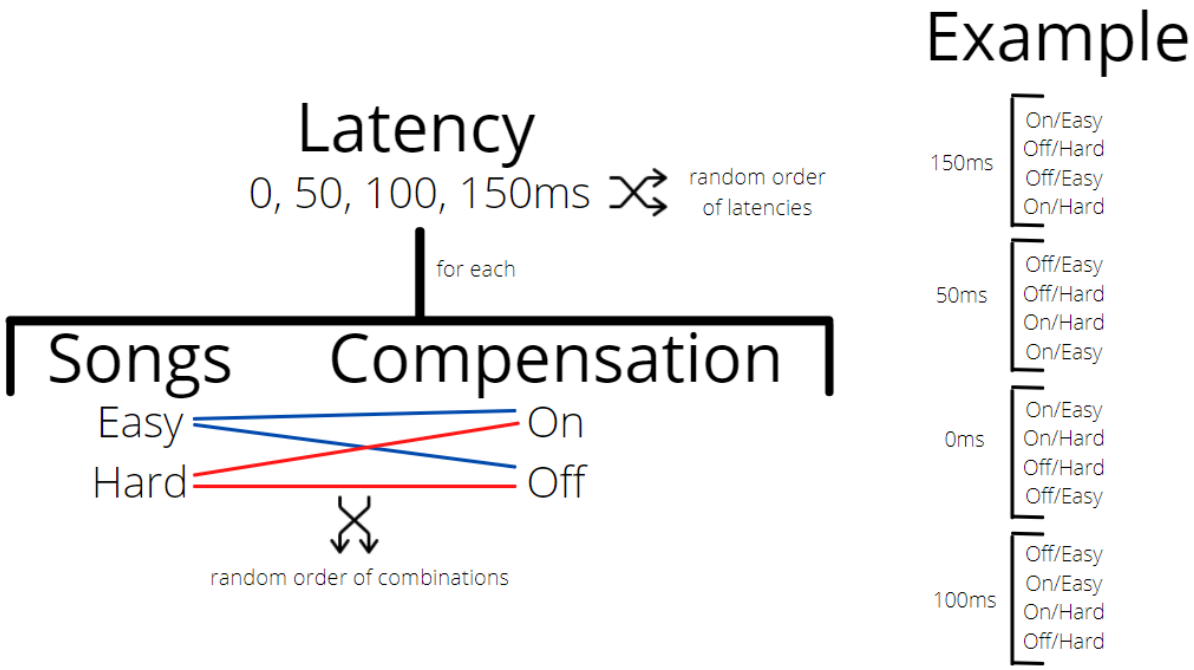


Figure 18. Every possible combination of latency compensation status, song difficulty, and latency levels.

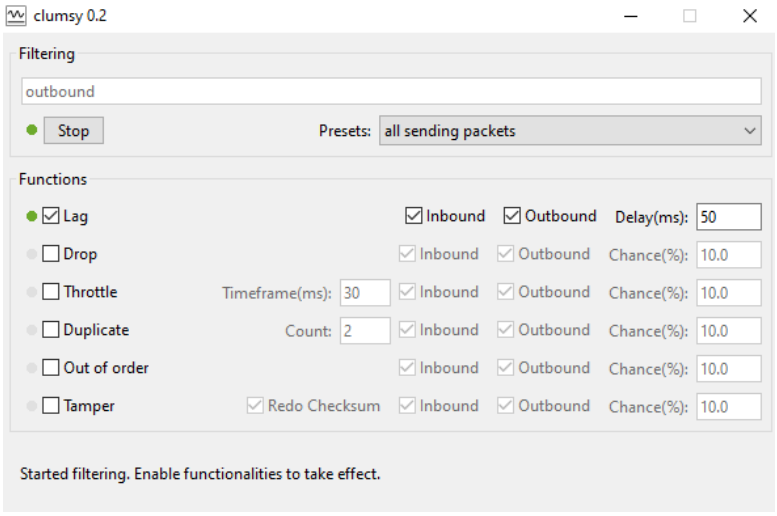


Figure 19. A screenshot of Clumsy's interface.

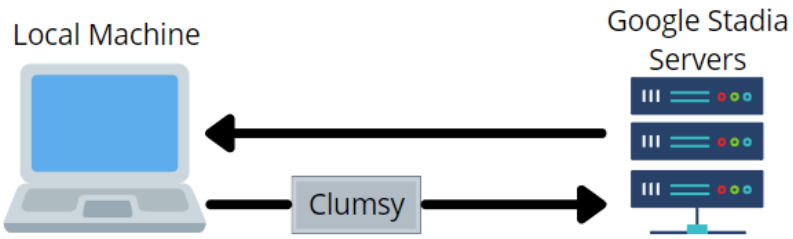


Figure 20. Illustration of Clumsy adding latency to our laptop's connection to the Google Stadia servers.

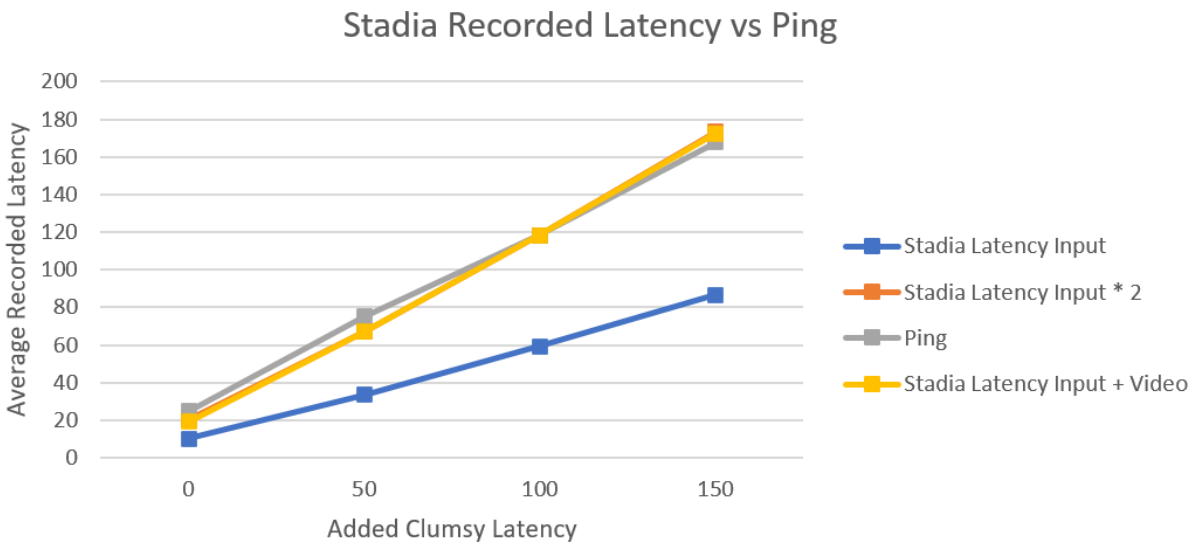


Figure 21. Graph of recorded latency from Stadia's API v.s. recorded ping time to the Stadia server.

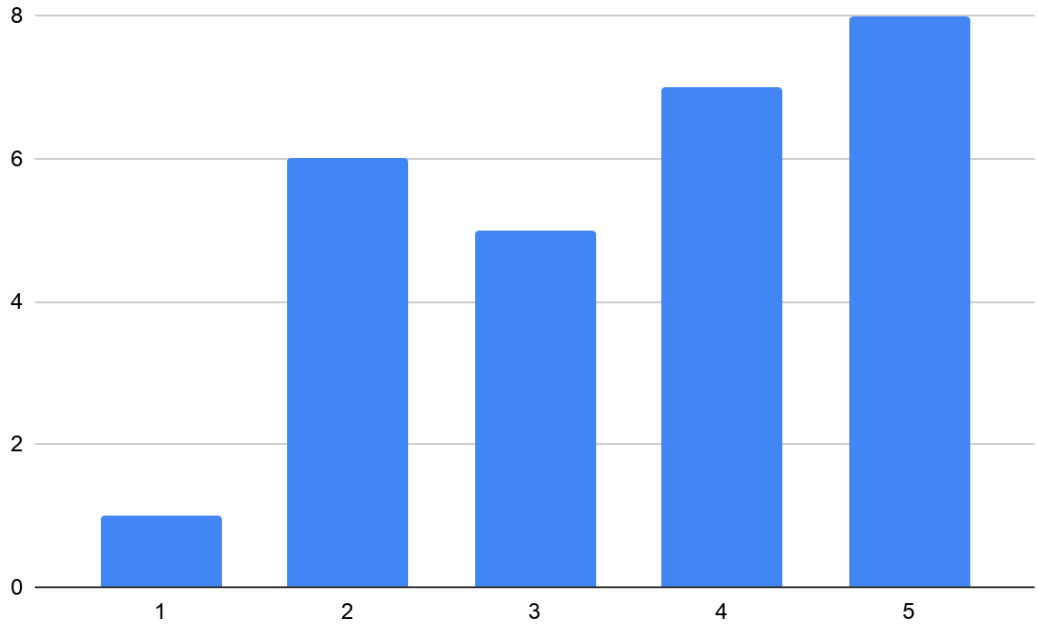


Figure 22. Histogram answering "How often do you play video games?". 1 indicates the participant has never played and 5 indicates the participant plays multiple hours a day.

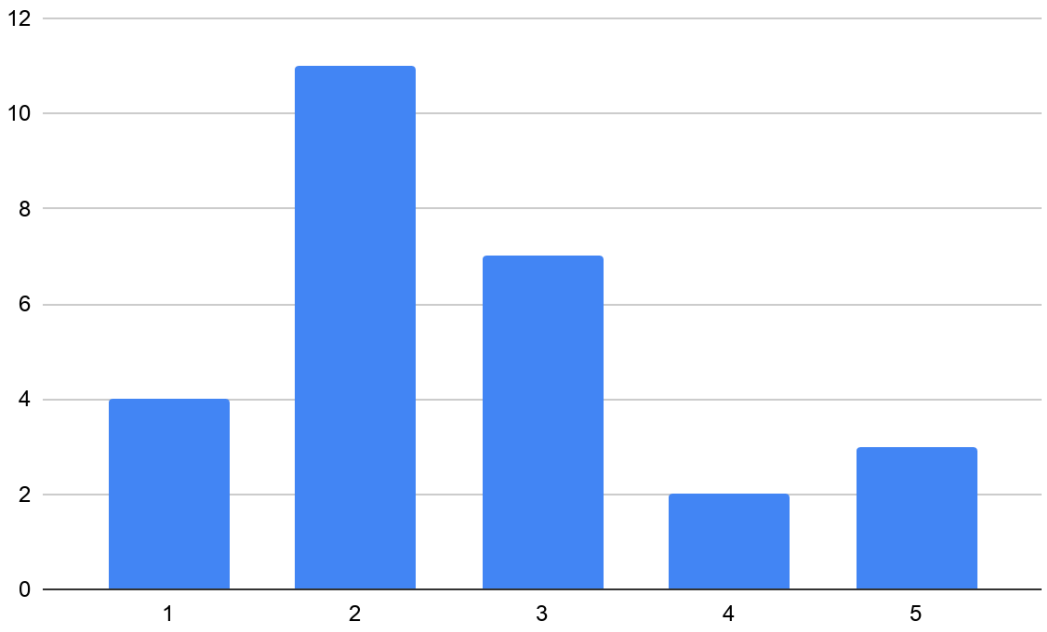
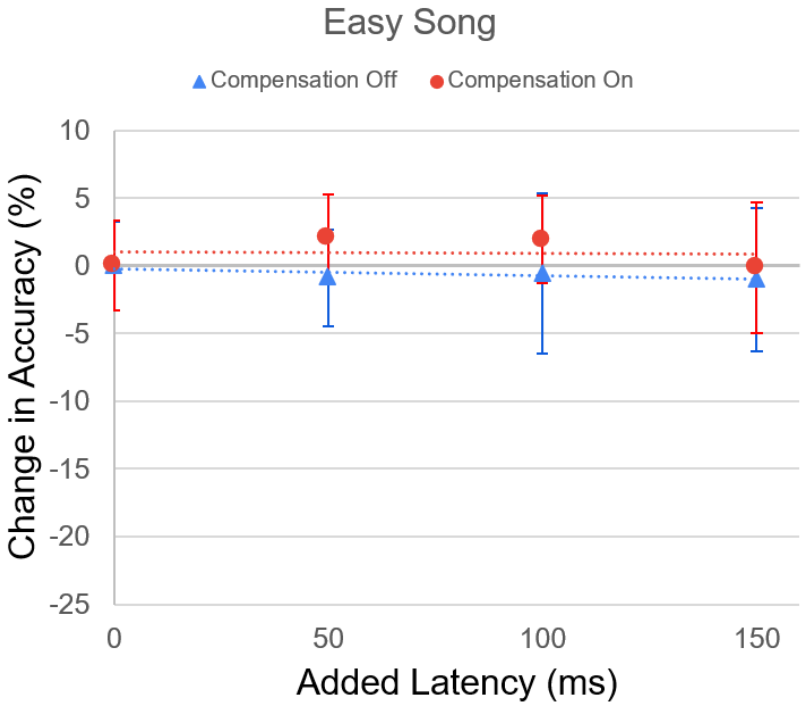


Figure 23. Histogram of user study participants answering “Rate your experience with rhythm games”. 1 indicates no experience and 5 indicates a lot of experience.



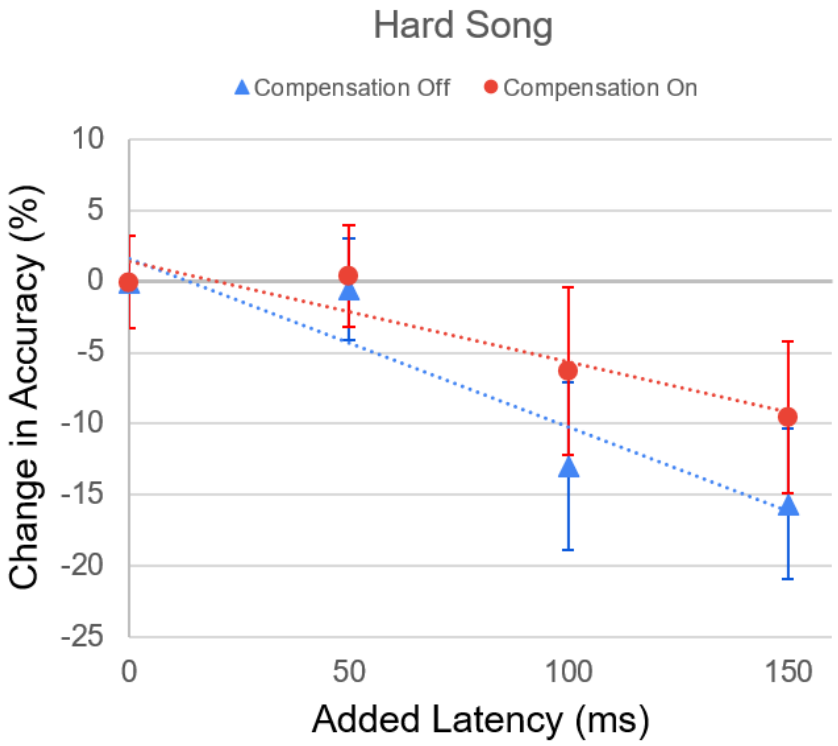


Figure 26 and 27. Change in accuracy compared to added latency (milliseconds). Change is relative to the average accuracy with 0 ms of latency. Easy difficulty (left) and hard difficulty (right).

Added Latency (ms)	**P-value ($p < 0.05$ for significance, significant values bolded)**
0	0.207
50	**0.020**
100	**0.0004**
150	**0.036**

Table 1. Resulting p values from t-test on player accuracy for the hard song, compensation on v.s. compensation off.

SUMMARY - Accuracy, Compensation off		
Groups	**Average (percent)**	**Standard Deviation**
0ms	96.6	24.0
50ms	95.9	35.9
100ms	89.9	234.7
150ms	88.5	190.6

Table 2. A summary of average and standard deviation for each latency in terms of accuracy with compensation off for the hard song.

ANOVA - Accuracy, Compensation off						
Source of Variation	**SS**	**df**	**MS**	**F**	**P-value**	**F crit**
Between Groups	2538.325	3	846.108	6.909	0.00019	2.652
Within Groups	23514.10	192	122.469			
Total	26052.42	195				

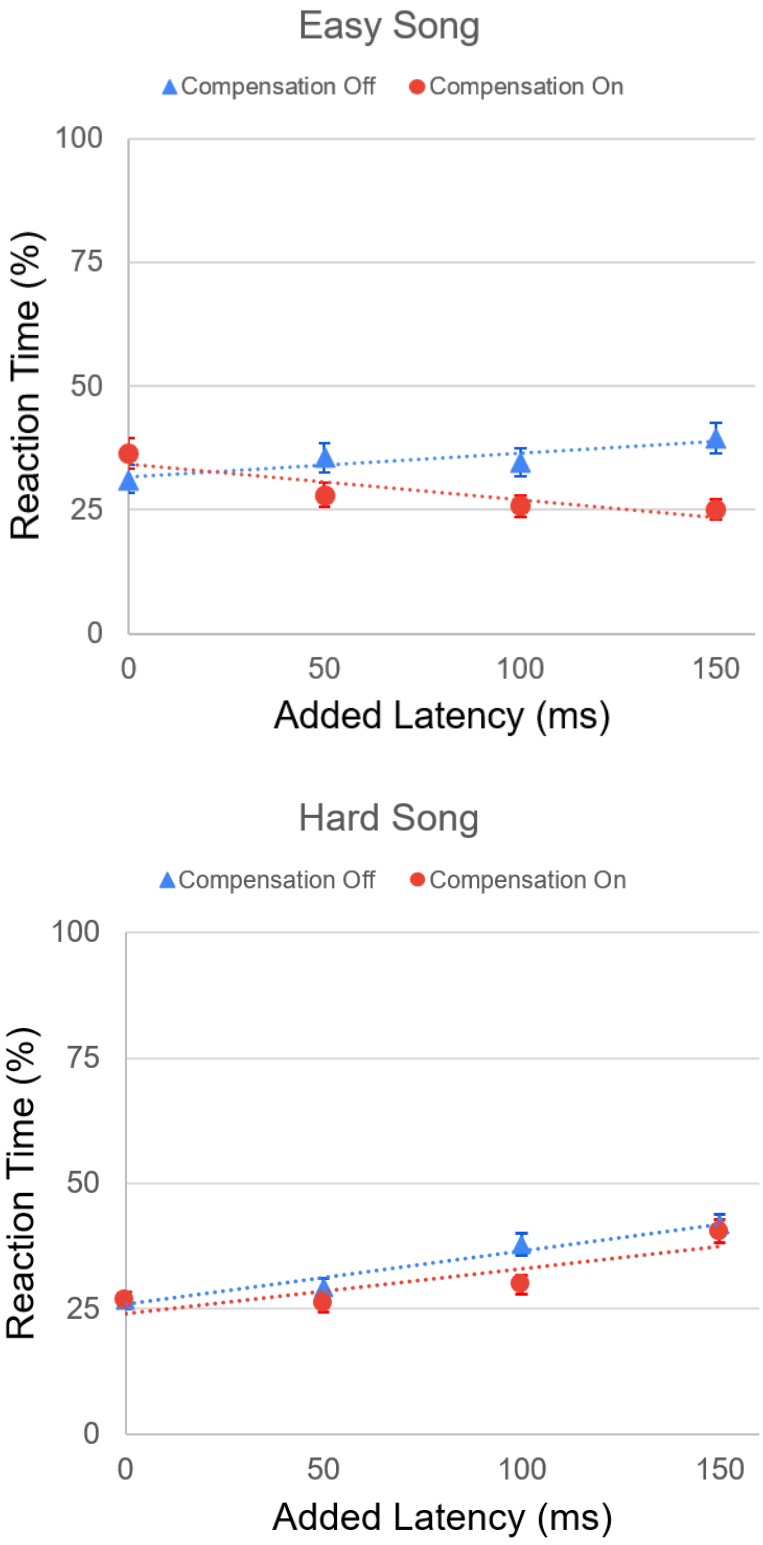
Table 3. An ANOVA test for player accuracy for the hard song for all four latency groups, with compensation off.

SUMMARY - Accuracy, Compensation on		
Groups	**Average (percentage)**	**Standard Deviation**
150ms	92.2	117.0
100ms	94.6	76.8
50ms	98.1	13.0
0ms	96.8	19.6

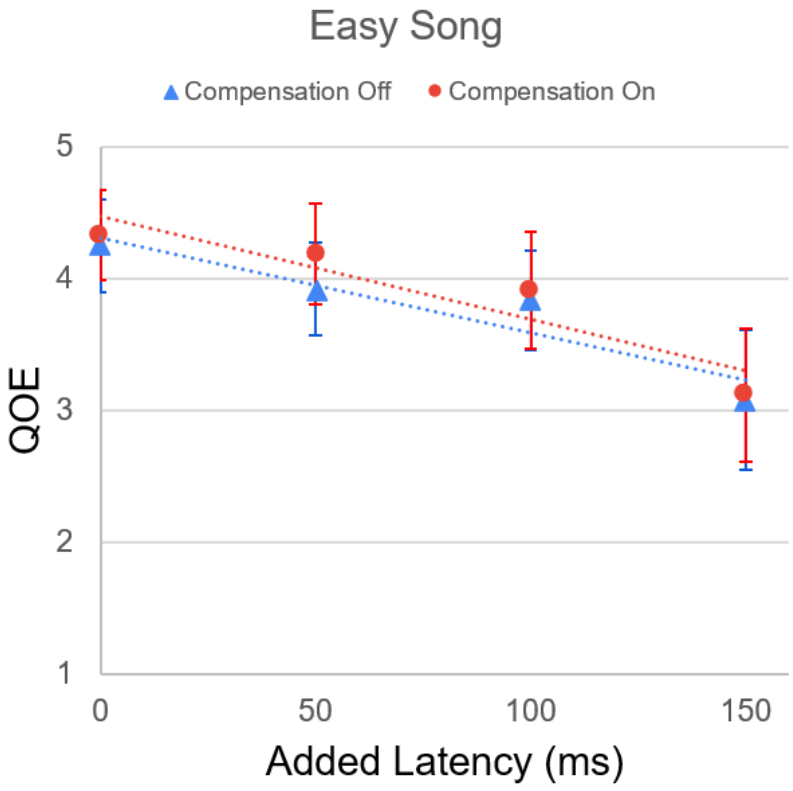
Table 4. A summary of average and standard deviation for each latency in terms of accuracy with compensation on for the hard song.

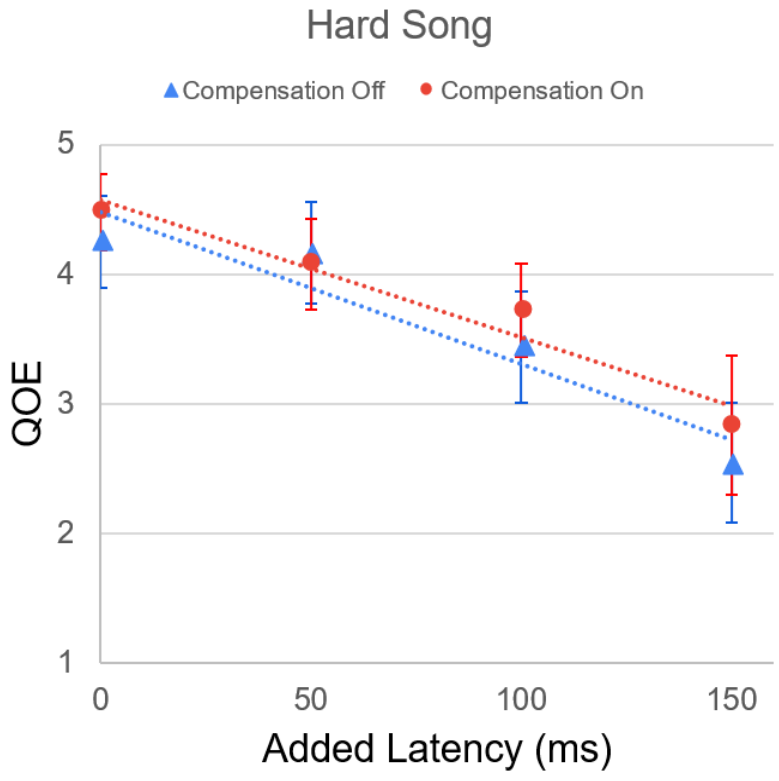
ANOVA - Accuracy, Compensation on						
Source of Variation	**SS**	**df**	**MS**	**F**	**P-value**	**F crit**
Between Groups	1024.973	3	341.658	6.034	0.00060	2.651
Within Groups	10985.510	194	56.626			
Total	12010.483	197				

Table 5. An ANOVA test for player accuracy for the hard song for all four latency groups, with compensation on.



Figures 28 and 29. Average Reaction Percentage ((Time to shoot target within window / target window length) * 100) vs added latency (milliseconds). Easy difficulty (left) and hard difficulty (right)


Easy Song
Compensation Off
Compensation On
QOE
5
4
3
2
1
0
50
100
150
Added Latency (ms)


Hard Song
Compensation Off
Compensation On
QOE
5
4
3
2
1
0
50
100
150
Added Latency (ms)

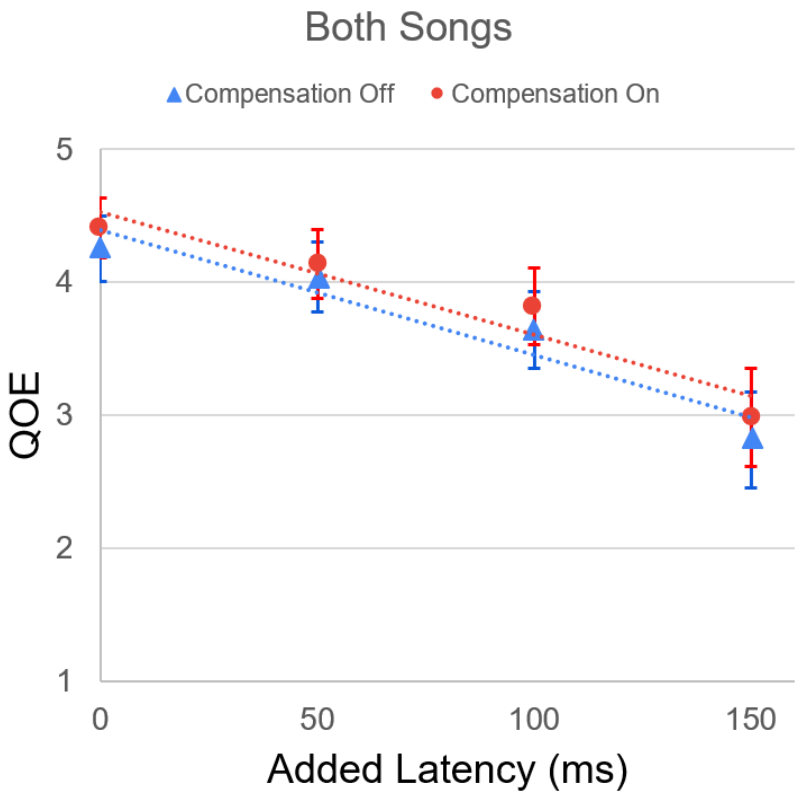


Figure 30. Quality of Experience to added latency (milliseconds)

Added Latency (ms)	P-value (p < 0.05 for significance, significant values bolded)
0	0.168
50	0.342
100	**0.035**
150	0.178

Table 6. Resulting p values from T-test on player QoE (both Easy and Hard songs), compensation on vs. compensation off.

SUMMARY - QoE, compensation off		
Groups	**Average (percentage)**	**Standard Deviation**
0ms	4.2	0.719
50ms	4.0	0.937
100ms	3.6	1.133
150ms	2.8	1.695

Table 7. A summary of average and standard deviation for the hard song for each latency in terms of QoE with compensation off.

ANOVA - QoE, compensation off						
Source of Variation	**SS**	**df**	**MS**	**F**	**P-value**	**F crit**
Between Groups	57.63699443	3	19.212	17.151	6.4E-10	2.651
Within Groups	217.318	194	1.120			
Total	274.955	197				

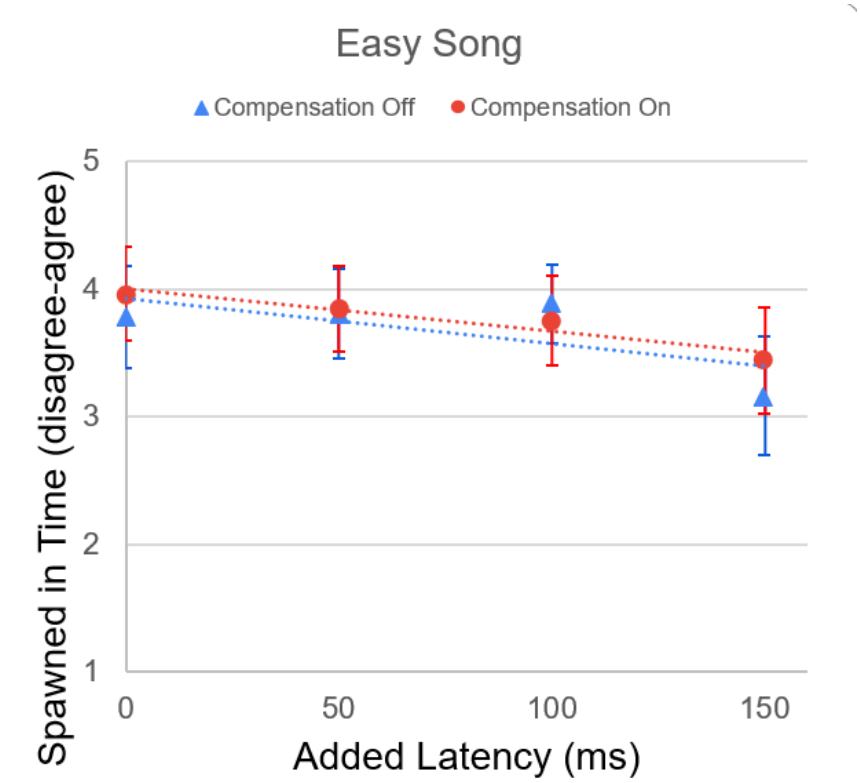
Table 8. An ANOVA test for QoE for the hard song for all four latency groups, with compensation off.

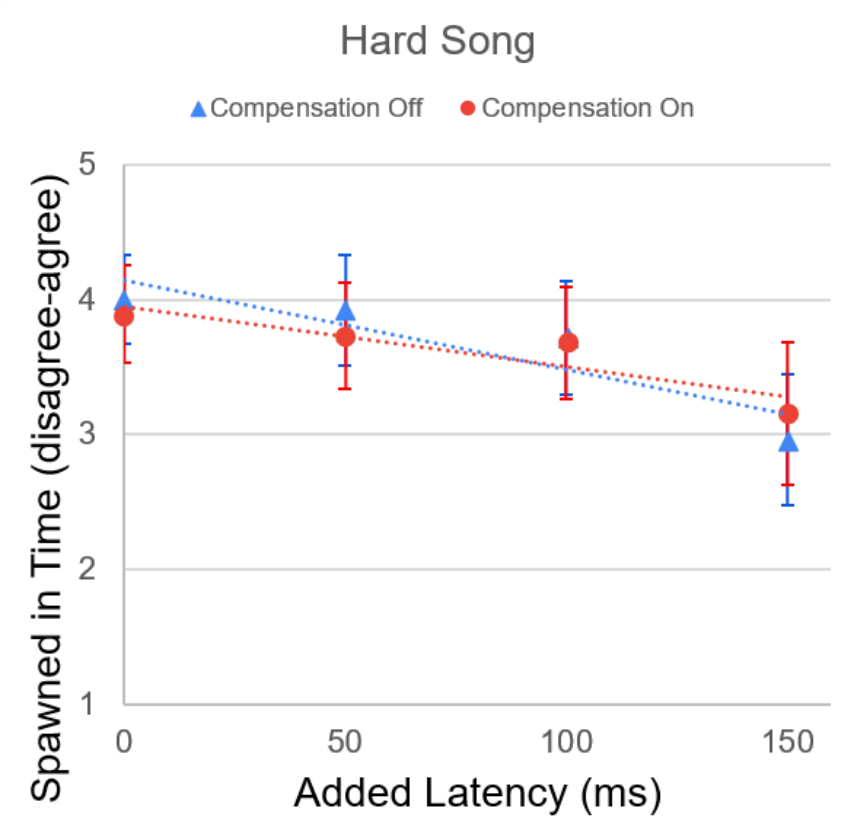
SUMMARY - QoE, compensation on		
Groups	**Average (percentage)**	**Standard Deviation**
0ms	4.408	0.663
50ms	4.135	0.942
100ms	3.816	1.0697
150ms	2.98	1.816

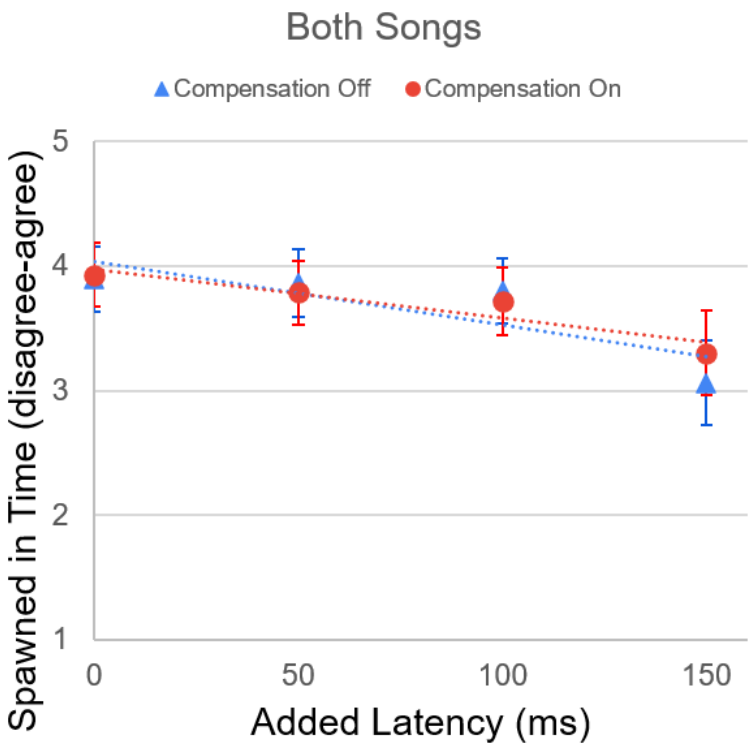
Table 9. A summary of average and standard deviation for the hard song for each latency in terms of QoE with compensation on.

ANOVA - QoE, compensation on						
Source of Variation	**SS**	**df**	**MS**	**F**	**P-value**	**F crit**
Between Groups	57.334	3	19.111	17.00924	7.37E-10	2.651
Within Groups	220.221	196	1.124			
Total	277.555	199				

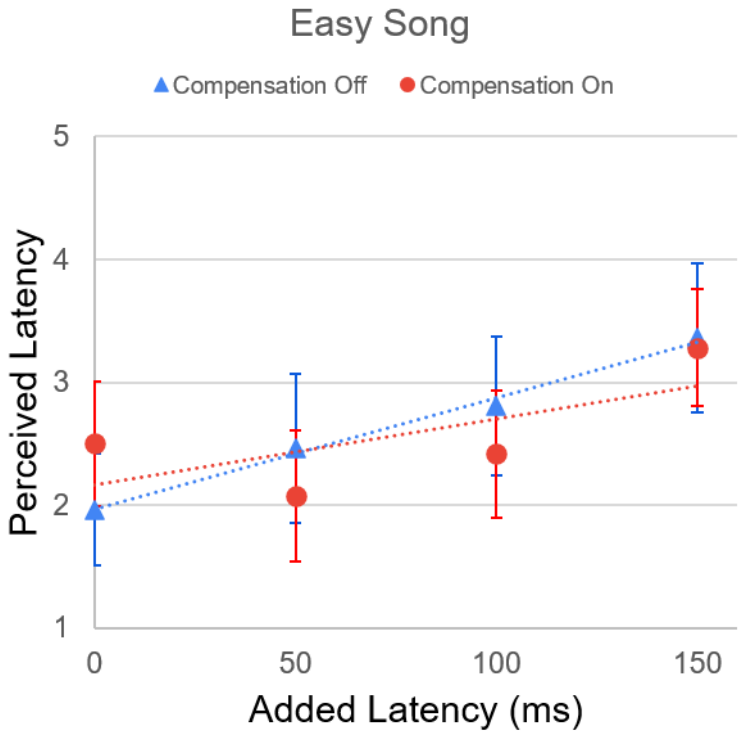
Table 10. An ANOVA test for QoE for the hard song for all four latency groups, with compensation on.

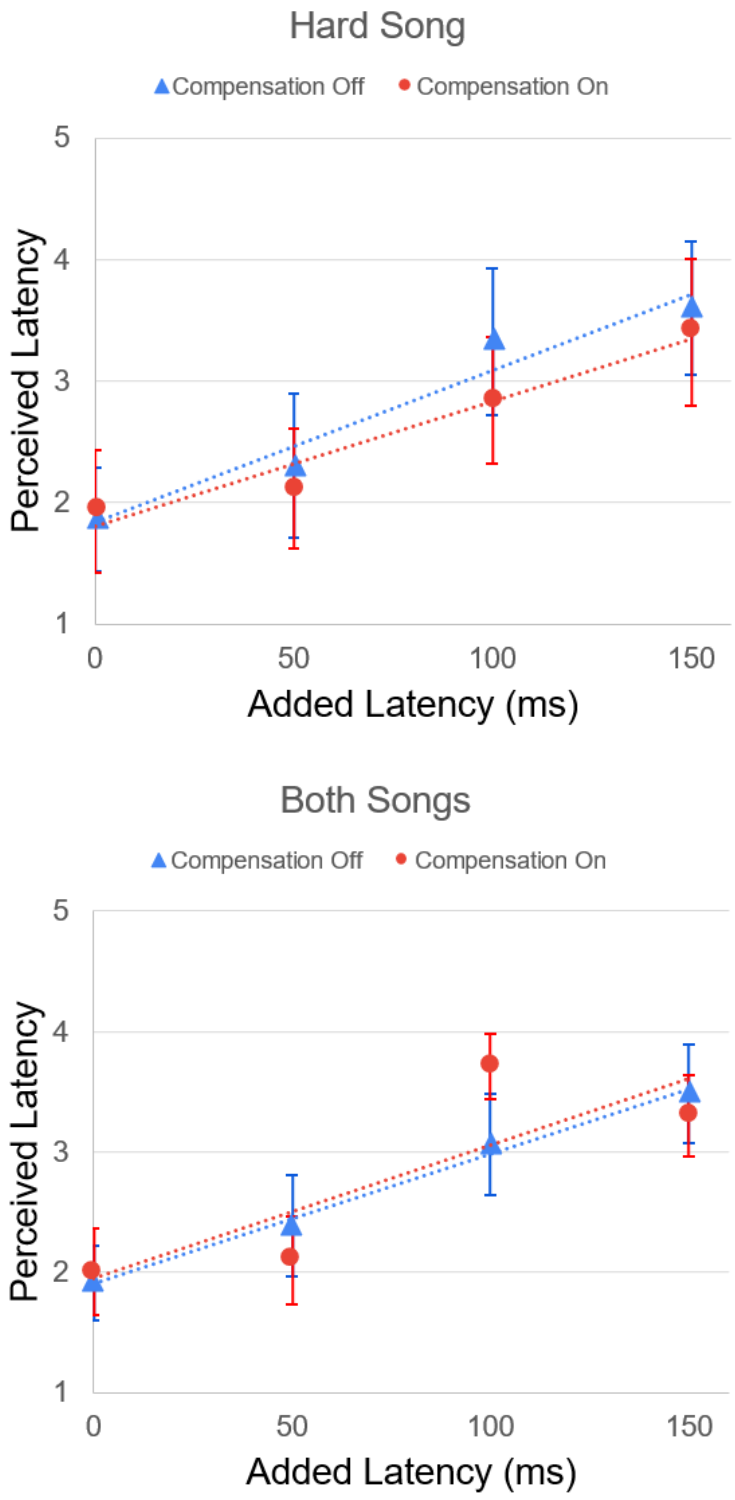

Easy Song
Compensation Off
Compensation On
Spawned in Time (disagree-agree)
5
4
3
2
1
0
50
100
150
Added Latency (ms)


Hard Song
Compensation Off
Compensation On
Spawned in Time (disagree-agree)
5
4
3
2
1
0
50
100
150
Added Latency (ms)



Figures 31, 32 and 33. Survey responses to the question “The asteroids spawned in time to the music.” with responses on a scale of 1-5 with 1 being “Strongly Disagree” and 5 being “Strongly Agree” compared to added latency in milliseconds.

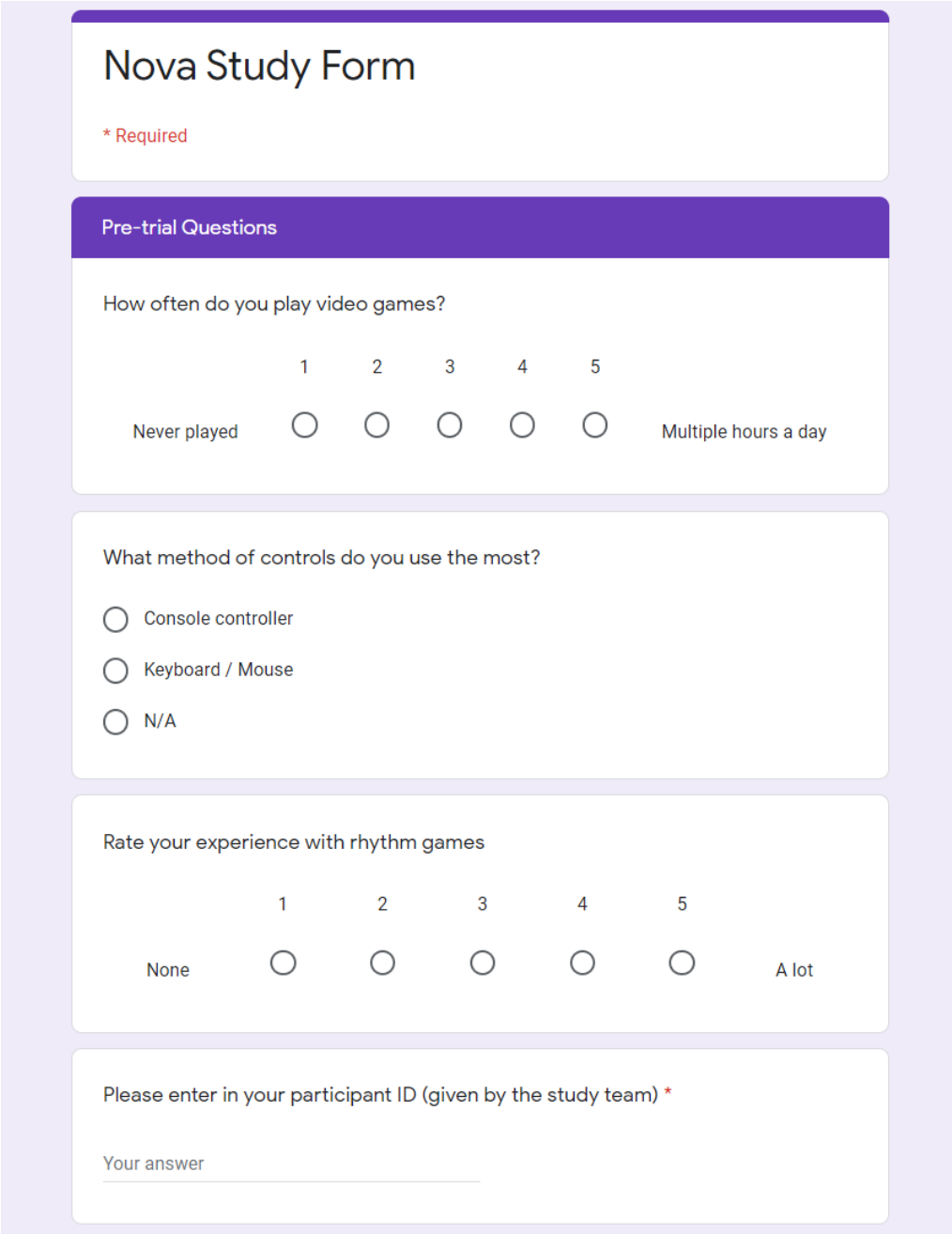




Figures 34, 35 and 36. Answers to the question “There was delay between when I clicked and the game responding.” with answers on a scale of 1-5 with 1 with 1 being “Strongly Disagree” and 5 being “Strongly Agree”.

Appendices

Appendix A: Survey Questions



Nova Study Form

* Required

Pre-trial Questions

How often do you play video games?

	1	2	3	4	5	
Never played	○	○	○	○	○	Multiple hours a day

What method of controls do you use the most?

- ○ Console controller
- ○ Keyboard / Mouse
- ○ N/A

Rate your experience with rhythm games

	1	2	3	4	5	
None	○	○	○	○	○	A lot

Please enter in your participant ID (given by the study team) *

Your answer

Wave #2

Only answer once you're instructed to.

Rate your overall experience:

- ◯ 5. Excellent
- ◯ 4. Good
- ◯ 3. Fair
- ◯ 2. Poor
- ◯ 1. Bad

For the next three questions, rate how much you agree with the following statements. 1 = Strongly Disagree, 5 = Strongly Agree.

The asteroids spawned in time to the music.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

There was delay between when I clicked and the game responding.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

Did you experience any difficulties? If so, please describe them here (optional):

Your answer

Final Questions

Do you require playtesting credit for an IMGD course?

- ◯ Yes
- ◯ No

Would you like to opt in to a raffle for a $25 Amazon gift card?

- ◯ Yes
- ◯ No

If you answered "yes" to either of the above questions, please enter your WPI email address below. (It will only be used to contact you with playtesting credit and/or for the raffle.)

Your answer

If you have any feedback on the game itself, please put it here. (optional)

Your answer

Back Submit