Latency and Jitter in Cloud Game Streaming

A Major Qualifying Project

submitted to the Faculty of

Worcester Polytechnic Institute

in partial fulfillment of the requirements for the

degree of Bachelor of Science

Written By:

Ryan Darcy

Sean O'Connor

Wenjie Zhang

Submitted on:

1 May 2022

Report Submitted To:

Professor Mark Claypool

Worcester Polytechnic Institute

Abstract

Cloud game streaming service allows users to play games without downloading locally by hosting the game on a server and streaming the game through the network to the user's client. Network conditions such as jitter and latency can negatively impact users' quality of experience (QoE) when using cloud streaming services. To determine the effect of latency and jitter on the user's cloud steaming QoE, we developed a custom game that allowed us to test different combinations of latency and jitter with different game perspectives. A separate buffer for the cloud streaming client Moonlight was developed to smooth out jitter and latency and improve the user's QoE. For the study, the buffer is simulated through different levels of jitter and latency. After running the study, which included 35 participants, we concluded that perspective had a significant effect on QoE, score, and damage taken, while difficulty and texture quality were less conclusive.

Acknowledgements

We would like to acknowledge the following individuals for their contributions during this study. First, we would like to thank Professor Mark Claypool for his feedback and support as our project advisor. Additionally, we would like to thank Xiaokun Xu for his support in creating the testbed and scripts used to run the study, and for his assistance in analyzing the study results. We would also like to thank Botao Han for his contributions towards the introduction and background sections of this paper and the implementation of multi-threading in the client-side buffer.

Table of Contents

List of Figures

Section 1 Introduction

Cloud-based game streaming continues to grow in popularity as the technology and Internet speeds powering it improve (Roach & Parrish, 2021). Through services such as NVidia GeForce Now and Sony PlayStation Now, cloud gaming has become more readily available for consumers. Compared to normal computer gaming where the user's local computer handles executing the game, cloud-based gaming utilizes servers to stream the game to a user-side client machine. Simply put, a user remotely controls the game that is being played on the server-side computer.

Although cloud-based game streaming provides convenience for the users since they do not need to download games locally, cloud gaming has encountered several issues. Adverse network conditions such as latency and jitter present problems for cloud-based game systems (Jarschel et al., 2011). Latency, also labeled as delay, refers to the time it takes for a packet to be sent and received. Jitter refers to the variance in time between the arrival of packets. Users' quality of experience (QoE) is greatly affected by latency and jitter. There is a considerable amount of research towards improving QoE in cloud gaming by reducing the effect of delay, but there is minimal research towards QoE and jitter specifically in cloud gaming.

In this paper, we analyzed the effects of latency and jitter on the quality of experience of cloud game streaming. We researched implementing our own client-side buffer into an existing cloud-streaming platform and developed a game that switches between multiple parameters, notably three different perspectives: first-person, third-person, and overhead. The game was then used within a user study through an open-sourced cloud-streaming platform, switching between multiple combinations of game parameters and network conditions.

Through the user study consisting of 35 participants, the final results of the study indicate that perspective, difficulty, and texture quality QoE ratings were all more affected by delay than jitter. Perspective related QoE ratings displayed significant changes because of delay and jitter, while QoE ratings for difficulties and texture qualities showcased less noticeable changes.

The rest of the paper is organized as: Chapter 2 explains cloud-based streaming and buffers for cloud streaming, Chapter 3 presents similar works done by other researchers, Chapter 4 introduces the setup of the research and user study, Chapter 5 covers the results of the user study, Chapter 6 concludes the findings, and Chapter 7 explores potential future work for the study.

Section 2 Background

Cloud game streaming is a method for playing games with advantages and challenges compared to traditional game playing methods. By sending packets to and from a server, a client can play a game over the Internet without the need to utilize local hardware to run the game. While this provides underpowered devices a way to play demanding games, cloud game streaming also introduces complications for network conditions like jitter and delay.

Section 2.1. Cloud-based Game Streaming

Cloud gaming allows a client to play games using an Internet connection to connect to remote servers (Roach & Parrish, 2021). Rather than the traditional method of playing games where users install games onto their machine, cloud gaming has servers handle running the game. The client then sends and receives packets to and from the server for game interaction and video/audio output respectively. The model has been adapted into various products and systems and offers a way for consumers to play games on various devices without needing to install the game locally.

Section 2.1.1. How it Works

Cloud gaming works through its use of a server and client. From the client-side, the game is streamed to the machine, allowing for player interaction. Inputs from the client machine are then sent back to the server, which is then processed by the server and used by the game. The server renders and plays the game, handling the work that a user's local machine would typically do. The rendered frames are sent as a video stream to a client computer, where the game can be seen and interacted with as shown in Figure 1. While video streaming services are similar to cloud-based game streaming, video streaming services do not require the as frequent interactions that cloud gaming needs to work properly.

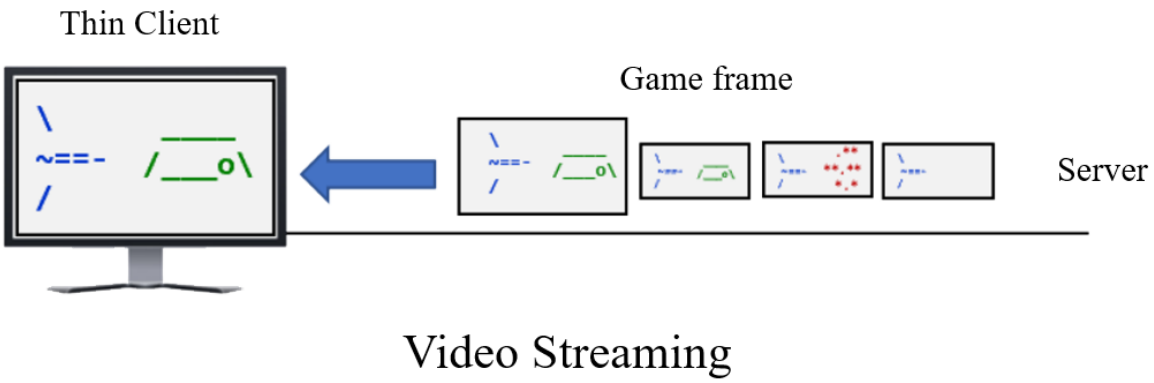


Figure 1. Diagram Depicting the Process of a Cloud Game Streaming Service

Streaming from a server allows the client machine to be one of many types of devices, such as a desktop PC, a console, or a mobile device. This gives cloud gaming a unique advantage over traditional game playing methods, allowing for devices that do not have the hardware specifications required to run the game effectively an alternative to deliver the game. However, a stable Internet connection with high bandwidth and low latency is needed for the games to play effectively. Otherwise, game resolutions and responsiveness can suffer, diminishing the user's quality of experience.

Section 2.1.2. History

Commercial cloud game streaming systems began with the founding of *G-Cluster* in 2000 ("G-cluster", 2022). However, cloud gaming products at the time were limited by the high bitrate required, given the infrastructure and Internet connections of that time. In 2010 and 2011, OnLive and Gaikai were released respectively (Mangalindan, 2020). OnLive allowed gamers to play games on demand with a subscription fee, with more costs for renting or purchasing games. The service struggled to build a catalog of games, as publishers did not embrace the subscription model. Gaikai instead focused on delivering game demos through cloud gaming, making the service more appealing to publishers. In June 2012, Sony purchased Gaikai and used the

service's technology as the foundation for the PlayStation Now service. Sony purchased OnLive in April 2015, shutting down the service after.

Cloud gaming services offered currently to consumers include Amazon Luna, Microsoft XCloud, NVidia GeForce Now, and Sony PlayStation Now. These platforms allow users to stream games at higher graphical quality and frame rates than past services. GeForce Now is notable for allowing users to stream games they already own on other existing PC storefronts.

Section 2.2. Existing Cloud-Streaming Platforms

To evaluate the available cloud-streaming platforms, the video quality between the client and server-side computers while playing were compared using screenshots. This was done to effectively compare bitrates and visual quality for Parsec, Moonlight/Sunshine, and OpenStream.

Parsec is a commercial cloud-gaming platform that supports both remote desktop and cloud-gaming similar to Stadia. Parsec has different products aimed for a wide range of purposes such as Parsec for Work, Parsec for Gaming, and Parsec Pros (gives better visual quality and more settings) ("Connect to Work or Games From Anywhere | Parsec", n.d.). We did not choose Parsec as our testing platform because the resolution degraded when jitter was added. The ideal testing cloud-streaming platform for us should not perform any type of adaptation to jitters or delay.

Moonlight is an open-source cloud-streaming client that implements NVIDIA's GameStream protocol. Moonlight supports multiple client platforms such as Android, iOS, PC, Apple TV, Mac, Chromebook, PS Vita, Wii U, Raspberry Pi, LG, and webOS TV ("Moonlight Game Streaming: Play Your PC Games Remotely", n.d.). Sunshine is an open-sourced cloud-streaming host for Moonlight, supporting AMD, Intel and Nvidia GPUs ("GitHub - LizardByte/Sunshine: Self-hosted game stream host for Moonlight", 2019). To pair Sunshine

with Moonlight, Sunshine uses a web UI to generate a password to be entered on Moonlight. Sunshine also uses the web UI to configure streaming settings. Additionally, Moonlight and Sunshine are open-sourced so we could modify and recompile them as needed. We initially chose Moonlight and Sunshine as our cloud-streaming platform because visually they did not degrade resolution when adding jitter, showing no noticeable bitrate reduction. However, because Sunshine could not be found on the Moonlight client on the day we tested the user study test bed setup, we chose OpenStream as the server of the test bed.

OpenStream is another low latency cloud-streaming host based on other open-sourced streaming platforms such as Moonlight and Sunshine. It currently does not have its own client platform released ("Open-Stream – The first completely integrated open-source game streaming platform", n.d.). As explained above, we chose OpenStream as the server for the study. As for the purpose of testing a custom buffer and running the study, the choice of platform for the server does not affect the result significantly.

Section 2.3. Challenges with Cloud-Based Game Streaming

While cloud-based game streaming has its benefits, two main challenges negatively impact a user's quality of experience: latency and jitter.

Section 2.3.1. Latency

Generally, latency refers to the time between a stimulus and the beginning of a response. There are many different specific causes of latency, especially within a system such as cloud-based game streaming (as shown in Figure 2).

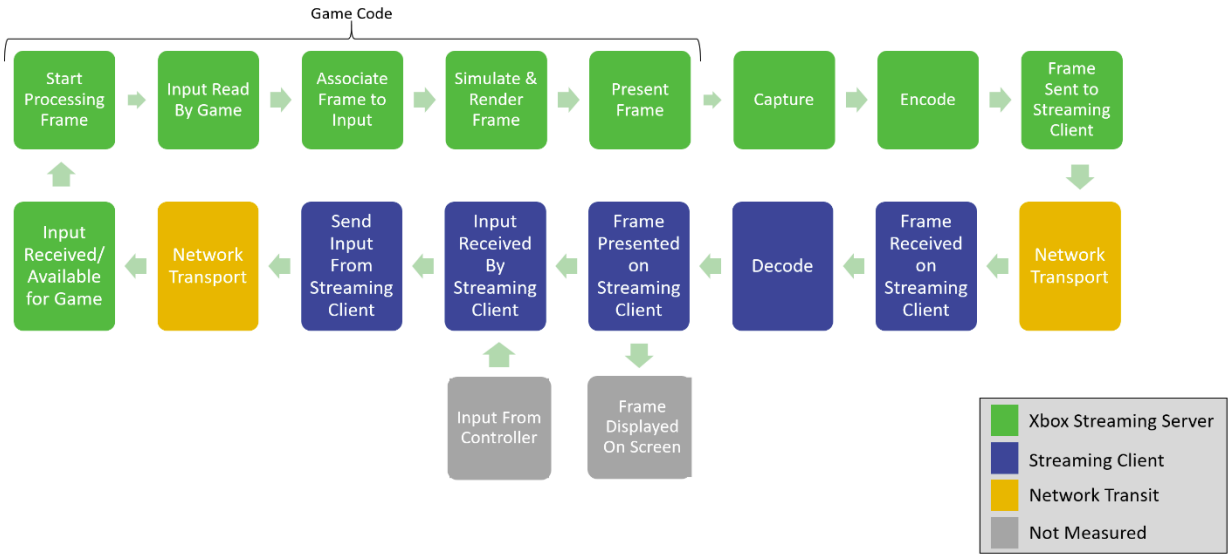


Figure 2. Diagram of the Steps in Xbox Game Streaming (M-Stahl et al., 2023)

Figure 2 specifically refers to Xbox Game Streaming, but the process is similar for any cloud-based game streaming system. Each step takes some amount of time, and when combined, will result in a certain amount of overall latency from when the player provides input to when they receive the frame where that input first takes effect.

This paper focuses on the effects of network latency, represented by the yellow boxes in the diagram, and the delay caused by a client-side frame buffer. The client buffer is not pictured in Figure 2 but would be placed between the "Decode" and "Frame presented on streaming client" steps.

Network delay is always present to some degree due to the laws of physics for light in fiber-optic cables (typically about 5 to 5.5 microseconds per kilometer) ("Fiber-optic cable", 2023). Client-side buffer delay, however, is entirely dependent on the buffering policy, as discussed in Section 2.4.

Section 2.3.2. Jitter

Jitter refers to the variance in latency of a network connection. When transferring data, packets do not take a consistent amount of time to get from the server to the client. As shown in Figure 3, the time interval d3 between packet 1 and packet 2 does not equal the time interval d4 between packet 2 and packet 3 on the client, where the time interval d1 equals interval d2 on the server. This can be due to several factors, but as this paper is focusing on network jitter, the typical cause is network congestion and variable bandwidth (IR Team, n.d.).

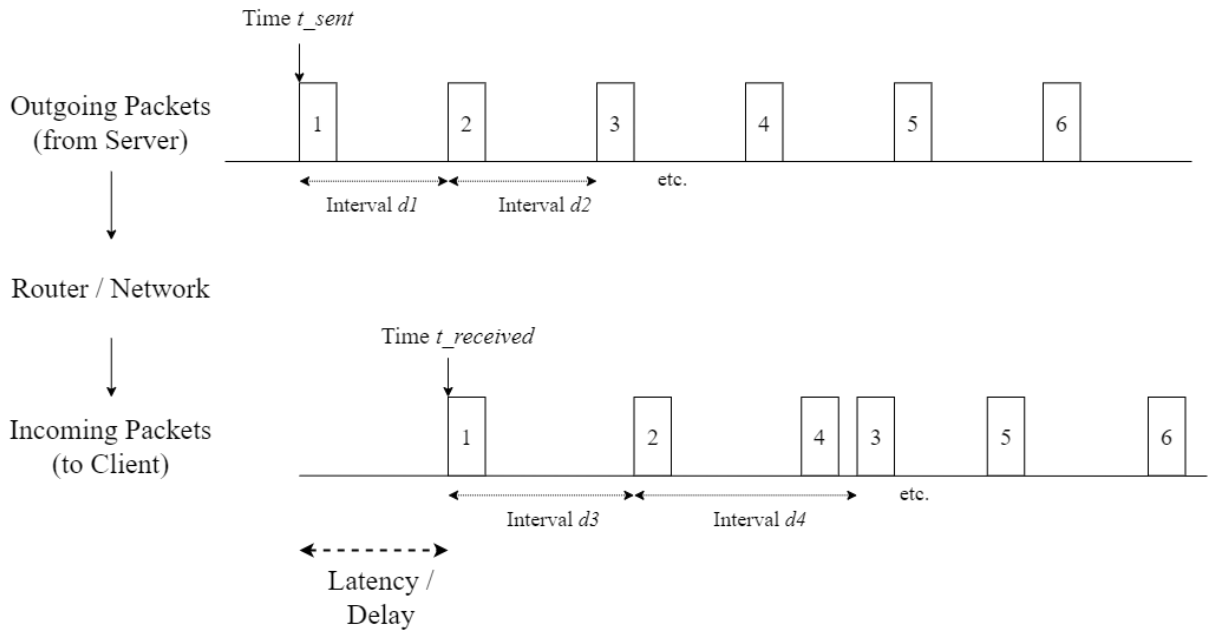


Figure 3. A Visualization of Jitter and Latency within a Client-Server Connection

High jitter negatively affects a user's quality of experience while playing a game if the frames are displayed as soon as possible by the client. This causes the video playback to appear choppy rather than smooth. In video streaming, this problem is solved by having a playback buffer, but video games are interactive and the user's quality of experience suffers from the delay caused by a buffer.

Section 2.4 Client-Side Buffer

To reduce jitter's negative impact on the user's quality of experience (QoE), buffers are implemented on the client. Instead of outputting frames as soon as the client receives a frame from the stream, a buffer receives frames from the stream and stores them in a container, then outputs the frames at a specific rate, as illustrated in Figure 4. If jitter takes place without a buffer, the effect of the jitter is obvious to the user since frames are being streamed as soon as they arrive at the client-side. With a buffer, jitter can be moderated by outputting the frames stored previously in the buffer at the default streaming rate while waiting for the next frame to arrive at the buffer so the stream would look continuous on the client.

The size of the buffer can affect users' QoE. A large buffer is able to smooth out greater jitter spikes since it can store more frames and output frames from the buffer for a longer time to wait for the jitter to end. However, having a large buffer size and holding more frames might cause greater latency on the client-side.

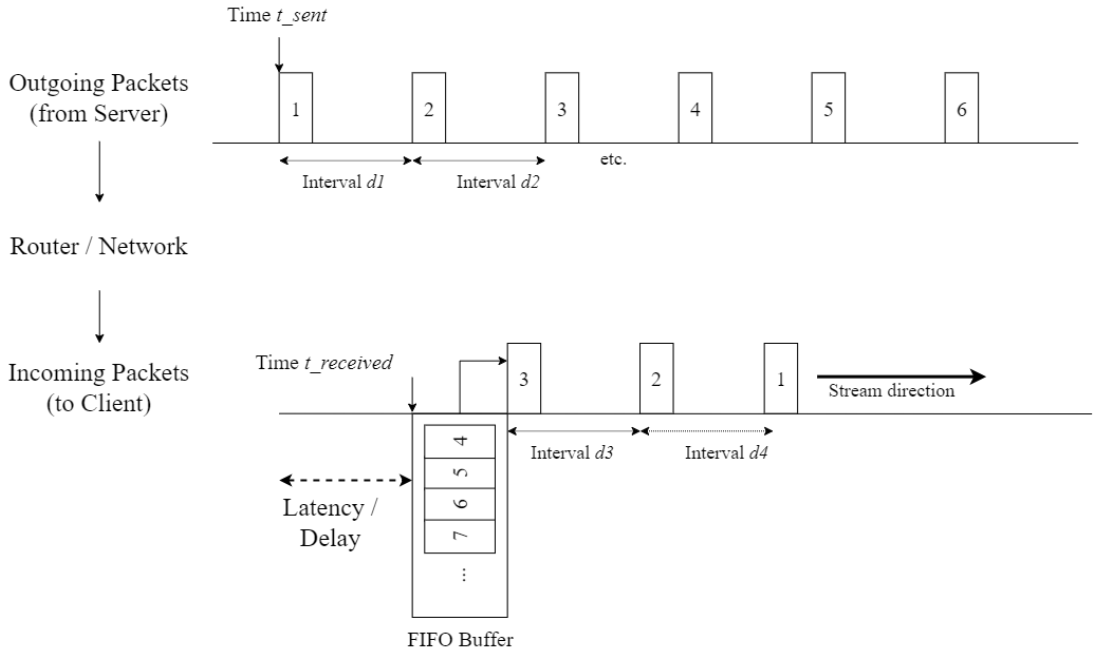


Figure 4. Visualization of a Buffer within a Client-Server Connection

Section 3 Related Work

This chapter gives an overview of research related to cloud game streaming delay, jitter, and frame buffers. More research has been conducted on game sensitivity to delay as opposed to jitter and buffer sizes for cloud game streaming in particular. However, additional research has been done for jitter and buffering for typical video streaming instead. The following research influenced the design of the custom game and the user study.

Section 3.1. Game Sensitivity to Latency

Perspectives, Frame Rates, and Resolutions: It's all in the Game, written by Mark Claypool and Kajal Claypool, discussed the effects of frame rate and resolution on game performance (2009). The paper researched these effects with a custom game, utilizing three perspectives: first-person linear, third-person linear, and third-person isometric. Although this research was focused on frame rates and resolutions, it indicates the performance of players may be impacted differently by frame rate based on the perspective of the game, and that the effects of frame rate may be similar to that of delay. These conclusions influenced the game designed for our study (as detailed in Section 4.1), particularly the three perspectives to evaluate the effects of delay and jitter.

Delay Sensitivity Classification of Cloud Gaming Content, written by Saeed Shafiee Sabet, Steven Schmidt, Saman Zadtootaghaj, Carsten Griwodz, and Sebastian Möller, classified nine game characteristics influencing the sensitivity of delay for games on cloud game streaming (2020). The paper indicated that the number of input directions (capability to move back and forward, left and right, up and down, rotation around the x and y axis) proved to be significantly delay sensitive. It influenced our game design decision of allowing users to move in all directions and move along the y axis by jumping in the first-person and third-person perspective.

Feedback frequency (amount of visual, auditive, or haptic feedback) also proved to be a delay-sensitive characteristic, which influenced our game design of adding visual feedback such as sparks and audio feedback such as gunshot sounds.

Section 3.2. Game Sensitivity to Jitter

Quality of Experience Evaluation for Buffer Sizing of Cloud-Based Game Streaming, by Brennan Aubuchon and Morgan Langstaff, discusses the effects of latency, jitter, and buffer sizes on user's experience playing games (2022). They found that games with lots of camera movement, complex animations, and/or many particle effects are more negatively affected by jitter and smaller frame buffer sizes compared to other types of games. This influenced our initial game design by pushing us toward a game genre that could easily incorporate different types and amounts of animations, and varying amounts of camera movement. It also gave us the idea to incorporate and vary the intensity of particle effects.

Section 3.3. Effects of Buffer Sizes on QoE

Joshua Allard and Andrew Roskuski studied quality defects in video streaming in their paper *Measuring the Annoyance in Streaming Media Caused by Buffers and Interrupts* (2015). By having participants watch a series of videos with differing buffer or interrupt counts, they attempted to find a relationship between annoyance levels and video streaming issues. Their results indicated that not only was overall annoyance towards jitter higher than for buffers, but videos with low motion had the highest average annoyance score for jitter related results. While not specifically related to cloud game streaming, these results indicate that there may be some acceptable amount of delay to users if it means smoothing out jitter spikes, though it is likely much lower than for video due to the interactive nature of games.

Section 4 Methodology

This chapter gives an overview of the custom game Robot Rampage, the client-side buffer, and the user study. Section 4.1 explains the planning stages of Robot Rampage. Section 4.2 focuses on the development of Robot Rampage. Section 4.3 presents an overview of the final build of Robot Rampage. Section 4.4 covers the design of the client-side buffer. Section 4.5 provides an outline for the user study structure and additional information related.

Section 4.1. Custom Game: Robot Rampage Planning

The development of our custom game, *Robot Rampage*, took place from October 2022 to March 2023. During the planning stages, the focus was on incorporating various parameters that were, based on our research, either sensitive to delay or sensitive to jitter. These parameters needed the ability to change throughout the course of a user study session. These primary requirements informed much of the development process and decisions made regarding many aspects of the game.

Section 4.1.1. Delay-sensitive Game Parameters

As discussed in Chapter 3, prior research in the areas of cloud game streaming and similar areas indicated certain elements of games that are sensitive to delay. These include:

- Perspectives (first-person, third-person, overhead)
- Enemy bullet accuracy
- Enemy movement speed
- Player bullet spread size/accuracy

From that list, the team narrowed down which elements we thought would a) be able to be implemented by us in our custom game and b) could be changed to various levels of sensitivity. For example, the accuracy of enemies could be decreased or increased, making it

easier or harder for players to avoid taking damage, especially as network delay increases. The resulting list was as follows:

- Perspectives (first-person, third-person, overhead)
- Enemy movement speed
- Player bullet spread size/accuracy

Section 4.1.2. Jitter-sensitive Game Parameters

As discussed in Section 3.2, while limited, the research on the effects of jitter to user's quality of experience in game streaming indicates that certain elements of games are more sensitive to jitter. These include:

- Number of animations (fewer/more animations)
- Level of visual effects (fewer/more visual effects)

Like with the delay-sensitive parameters in the previous sub-section, the team narrowed down this list to parameters that could be feasibly implemented and changed to varying degrees. As an example, enemy animations could vary from few or none (e.g., they slide around when moving, disappear when they die, etc.) to prevalent and elaborate. The resulting list was as follows:

- Number of animations (fewer/more animations)
- Level of visual effects (fewer/more visual effects)
- Quality of textures (less/more detailed)

Section 4.1.3. Early Game Design Decisions

The first game design decision was what genre/type of game we should make in order to best incorporate all the parameters we decided upon. To do this, the team went through all the widely accepted main genres of video games and gave them a rating based on feasibility. One of

the genres the group rated most highly was action, and when broken down by sub-genre, we rated shooter most highly. It was agreed upon that shooters can be flexible in terms of many of the parameters we discussed such as perspective and difficulty. It was also agreed upon that the genre would be easier to develop compared to other genre contenders such as platformers.

The second major decision was what game engine to use to make this game. We decided to make Robot Rampage in Unity as the team members working on it were already familiar with that game engine and knew the capabilities of it. While Unity may not be widely considered to be the best choice for shooters, it is a relatively simple and flexible engine. This was important given the timeline for development and due to the number of unknowns regarding game features.

An additional decision we had to make before development was how the game was going to be structured. Keeping in mind the context of the user study having many rounds, the team decided a 'boomer shooter'-like game with some amount of procedural generation in the levels would be a good way to go. 'Boomer shooter' as it is commonly used refers to "first-person shooters that intentionally harken back to the classic PC games of the late '90s like *Doom* and *Quake*" (Franzese, 2021). These and similar games served as inspiration for some of the mechanics as well as the mood of Robot Rampage.

Section 4.2. Custom Game: Robot Rampage Development

The bulk of development on Robot Rampage focused on implementing the design decisions made during the planning stages on-time and effectively. To ensure that work would be properly managed and completed, the team structured development loosely around the Agile development cycle. The intent of designing development around this method was to ensure tasks of the utmost importance were prioritized and assigned to a team member. Typically, a handful of tasks were targeted each week based on what had already been completed, what was most

important for the study, and how difficult the task was. These tasks were created and managed within the project management tool Trello.

To properly share the work done over Unity between team members and to have some form of version control, a GitHub repository was created specifically for the game. The team briefly considered utilizing Plastic SCM for version control given its built-in functionality with Unity. However, due to the potential costs of maintaining it and our unfamiliarity with using it compared to GitHub, the team pursued GitHub.

Early builds created during development were referred to as the MVP, or minimum viable product. This build of the game was created to get the core functionality of the game complete as early in the development cycle as possible. However, much of this version of the game was scrapped after feedback from players at Worcester Polytechnic Institute's Alphafest and due to the inclusion of the default Unity third-person controller.

Section 4.2.1. Game Structure

The initial game loop concept was for players to fight their way through several rooms before reaching the end of the level, thus concluding a single round. Each room would have contained robots to fight, preventing the player from progressing unless all robots had been defeated. However, as development progressed, the team found several issues with this structure within the context of the study. For the study, having one consistent level would become an issue when trying to analyze how the change in game parameters and network conditions affect QoE. As players would have to play this level multiple times, there was potential for players to memorize the level and adjust how they play based on the level layout rather than the change in conditions. Instead of making several levels and potentially running into this issue, the team

instead developed endlessly generating rooms for a level rather than having a specific end. More detailed technical and design information on the level generation can be found in Section 4.2.4.

Initially, the player had a limited set of health and ammo. Players would lose health and ammo with any hits from enemies or any shots respectively. To give players an opportunity to regain health and ammo, power-ups were created as seen in Figure 5 However, these two were scrapped near the end of development for the final build, as the two features introduced new challenges and issues with other game features and requirements. The current structure of the game gave players no goal beyond defeating as many enemies as they could within the given time. With limited time for rounds, players dying due to a lack of health would significantly increase the time spent not actively playing the game. The ammo caused a similar issue, since a player could run out of ammo and would need another way to fight. Giving players ammo would have made the need for an ammo count unnecessary, and waiting to give players ammo after enough time had passed would share a similar issue with the health. With the presence of the points system, these two were removed and their power-ups were reworked, giving players an endless amount of health and ammo.

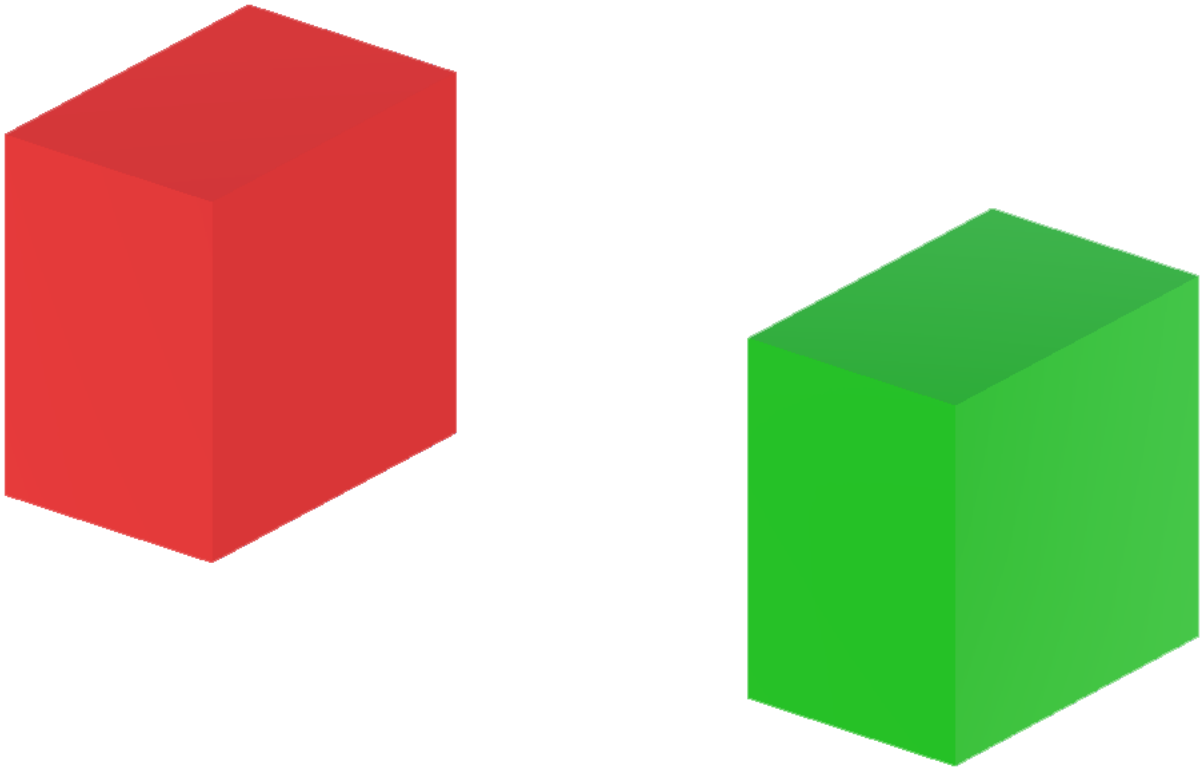


Figure 5: Initial Health and Ammo Power-up Models. The Red and Green Models are the Health and Ammo Respectively

The point system was introduced later into development to incentivize players to proceed forward through the level and defeat as many robots as possible. Points were designed to reward players for playing well and punish them for performing poorly, such as when they destroyed a robot or had been hit respectively. As a result, the points worked better alongside the structure of the game compared to the health and ammo, as a player could be rewarded or punished for their actions without slowing the pace of the game or study itself. Once the health and ammo were removed, the points became the primary way to gauge how successful a player was during the game.

Power-ups would drop randomly from enemies when defeated to assist players when they were running low on health and ammo. While the power-ups were originally designed for the

health and ammo systems, they would be reworked once those two systems were scrapped. Instead, the power-ups temporarily increase the stats of the player. Speed and damage increases were chosen, incentivizing players to defeat robots to collect them, thus allowing them to more efficiently and effectively defeat more robots. To prevent players from becoming too powerful, the power-ups were given time limits, encouraging players to strategically choose which power-up to grab in a given situation.

Section 4.2.2. Player Mechanics

The player character's controls and abilities remained consistent across development from a surface level view. Players controlled the character with the mouse and keyboard, walking, jumping, and shooting in the process. However, the player controller within Unity changed drastically during development in response to player feedback and the three cameras required for the study.

Initially, a custom player controller was created that contained the same player keyboard and mouse controls as in the final build. Both the MVP build and the final player controller used raycasts to create a hitscan gun. Rather than fire out a physical object, with hitscan the player can fire the gun into the direction of the target and so long as it falls within the set range of the raycast it will connect. To give players an indication of where they would shoot, a reticle was placed in the center of the screen and would change colors to indicate when the player could successfully shoot an enemy. This reticle feature stayed consistent throughout all builds of the game. What changed were the physics of the player controller, as early playtesters of the MVP build found the player character too floaty and slippery. This player controller was built with a first-person perspective, lacking the other two perspectives in the process, and only using a default Unity camera. The most significant difference between the MVP player controller and the

one found within the final build was the ability to aim in any direction, which would later change as a result of the overhead view. Figure 6 presents an early version of the MVP build within the Unity editor.

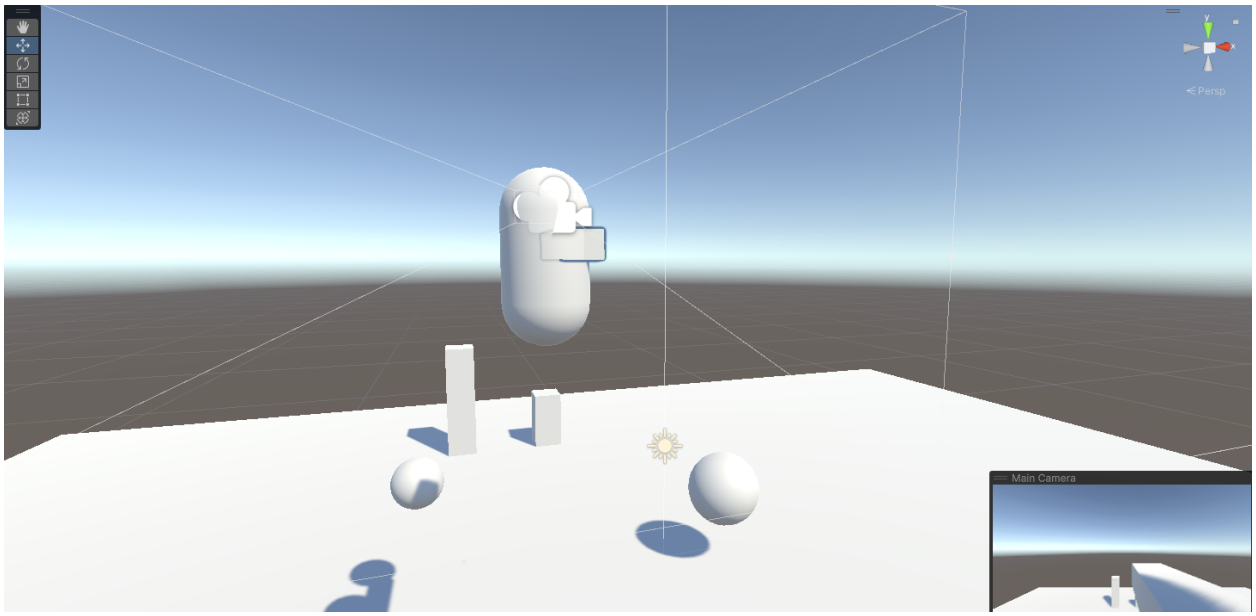


Figure 6: Early Version of the MVP Build, Containing the Basic Player Controller and Example Scene

In response to the feedback given by playtesters of the MVP build and to make adding additional camera perspectives easier, the team switched to Unity's default third-person controller. The controller was customizable, allowing for us to tweak it to get the desired feel from the game interaction. The physics of the controller were much more elaborate, giving a better sense of game feel as a result. The controller also utilized Unity's cinemachine camera system, providing a much more intricate and complex camera system than what was seen within the MVP build.

While the first-person and third-person perspectives were relatively similar and required no changes in player controls or mechanics, the overhead view significantly altered the view and how players controlled the game. With the overhead view, players were put at a significant

disadvantage when trying to aim along the y-axis. The ability to aim along the y-axis had to be removed to compensate. To make the controls as consistent as possible between each perspective, the same was done to the first-person and third-person views. Now the only axis that the game would consider when a player aimed with the mouse was the x-axis, meaning the game only took in left and right mouse movements. Further changes were made to how the player aimed and moved within the overhead view as detailed in Section 4.2.3.

For the purposes of the study, the game needed adjustable difficulty levels. To do this, the player shooting was modified to allow for changeable accuracy ranges. The SettingsManager handled the three difficulties, changing the range of values within the player attack script to match.

Section 4.2.3. Camera Functionality

In the initial MVP build, the game only contained a first-person view. The camera utilized within this build was a default Unity camera, lacking the complexity of other Unity camera systems such as cinemachine. With the switch to the Unity third-person controller, a cinemachine camera was utilized.

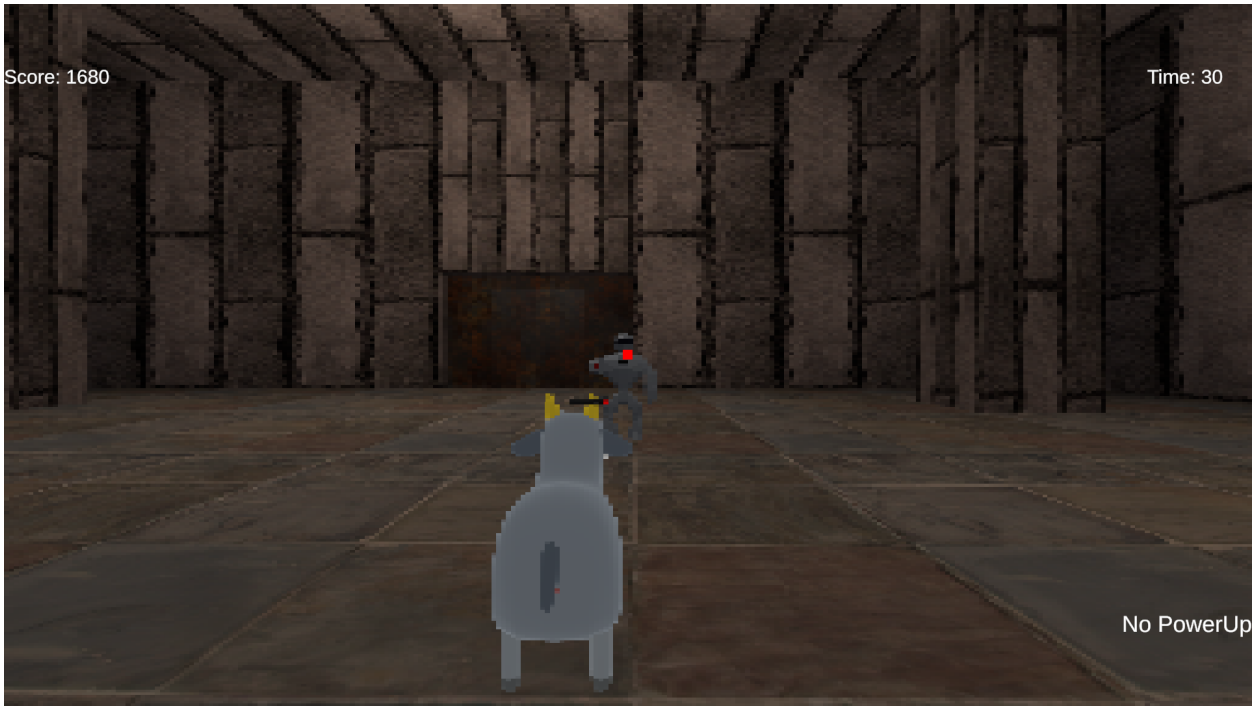


Figure 7: A screenshot of the final build's third-person perspective

Given that the third-person controller only contained a third-person camera, the first-person and overhead cameras had to be implemented. To manage switching between cameras for the three perspectives, a Unity game object called CameraMonitor was created to change the current camera and any other attributes. Using the Unity third-person controller, the third person perspective was already completed. To implement a first-person perspective, an additional cinemachine camera was added and placed in front of the player model. When switching to the first-person camera, CameraMonitor would turn off the player model's mesh, preventing the model from clipping into view.

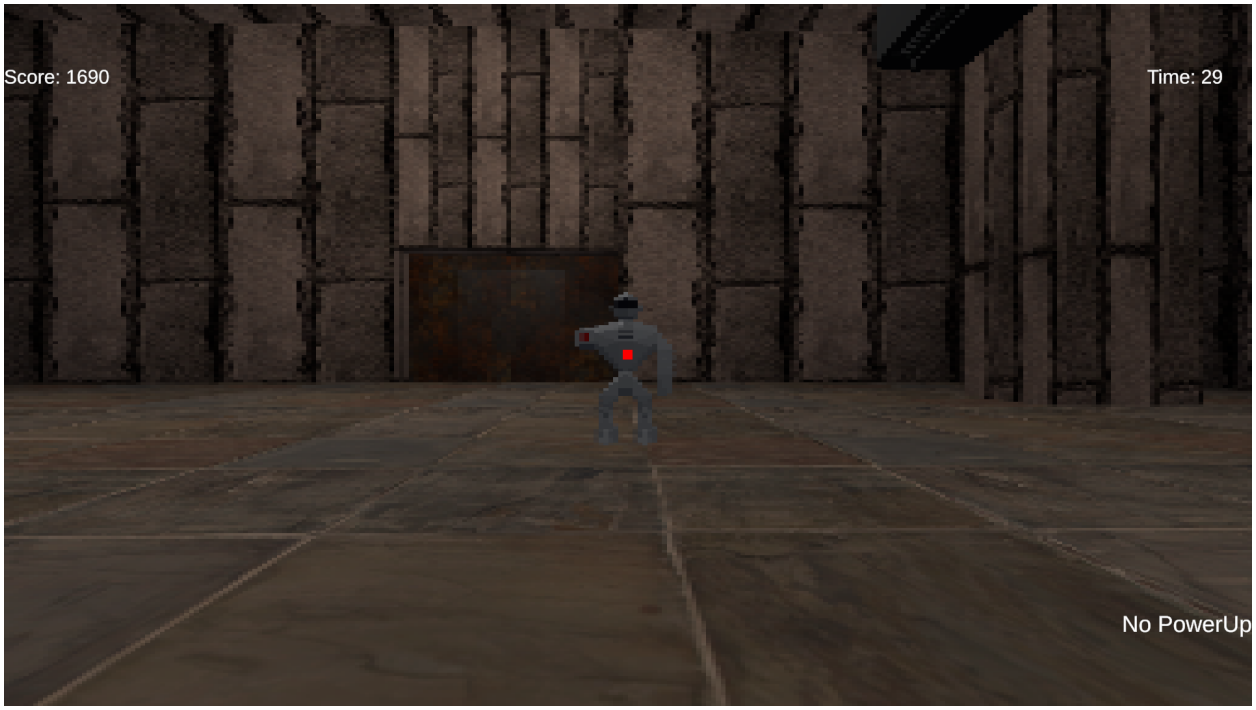


Figure 8: A screenshot of the final build's first-person perspective

The most difficult perspective to implement was the overhead view. To solve the issue of aiming within the overhead view, the team considered various ideas. At one point, the team tested the ability to aim with the arrow keys rather than the mouse. However, the team found that this control scheme felt uncomfortable in all three perspectives. The final implementation was to always keep the player aiming straight in front of them, only allowing the player to rotate the entire player in the overhead perspective and move up, down, left, and right relative to the screen. To give a proper indication of where the player was shooting, a Unity line renderer replaced the reticle in the overhead view. The line works similarly to the reticle by changing colors when the player could shoot an enemy. CameraMonitor manages turning on and off the reticle and render line based on the current camera.

Figure 9: A screenshot of the final build's overhead view

Section 4.2.4. Levels

From early on in development, the team planned to have the levels of Robot Rampage be procedurally generated in some way. We hoped the procedural aspect of the level generation would keep the game from getting too monotonous to play for the duration of the study while requiring minimal work in level design on the part of the team. There are several ways of going about procedurally generating levels, such as generating terrain maps using noise functions, using 2D dungeon-generating algorithms, generating full 3D cave systems, and various other more or less complex methods. Given the chosen setting of the tunnels under Atwater Kent and the other technical and scope limitations of the project, the team decided to make several preset rooms that would be placed procedurally to form a given level.

The actual algorithm governing the level generation is relatively simple. There is a LevelGenerator object in the level scene that starts by instantiating the start room and then picks

a random doorway from the most recently placed room, attempting to attach a random room to it. This repeats for as many rooms as wanted per level, checking if any rooms overlapped each time to avoid collisions.

The initial rooms were simply made of primitive blocks in the Unity editor to work on implementing the placement system. There were a few different kinds of rooms to provide variety such as a square room that would spawn enemies as seen in Figure 10, several types of hallways, and a taller room with some platforms to get to a second-story door.

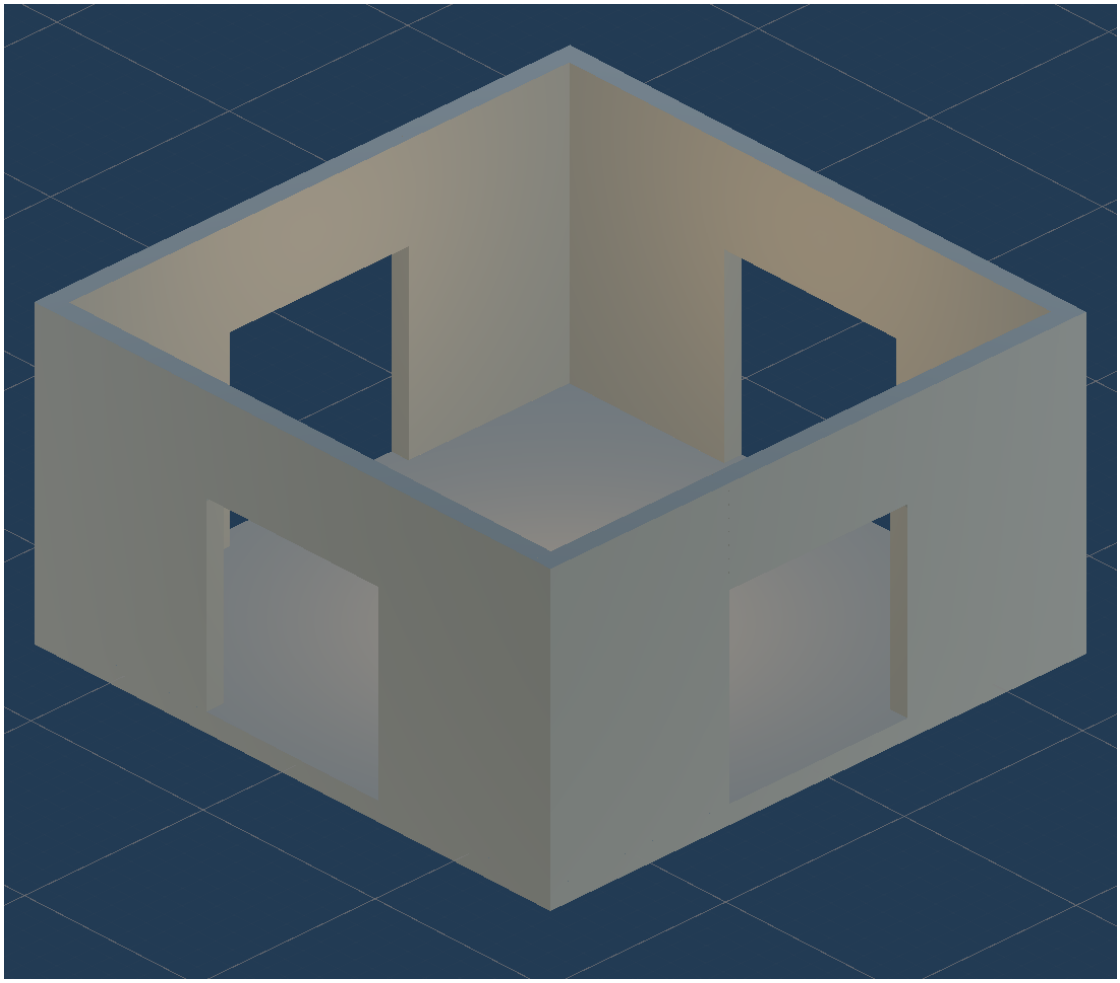


Figure 10: An Example of an Initial Room with an Isometric View

Once the team started finalizing the design of the user study, the initial concept of having individual levels for each round was changed. If there was a specific number of rooms in each

level, this may result in some players finishing the level before the expected time, while some players may not finish the level in time. Additionally, starting a new level for every round may not be very time efficient as beginning a level may be slightly disorienting as well as not contain any enemies. Therefore, the final level generation was changed such that it would infinitely spawn new rooms and destroy old rooms. Gameplay would simply be paused between rounds, and that way the user would pick up right where the previous round ended.

Additionally, the final rooms were redesigned. From feedback at Alphafest as well as our own feelings towards the rooms, we decided the rooms should be larger, and there should be fewer hallways. Additionally, we decided to remove most of the verticality from the final rooms, particularly in the way that all of the final rooms sit on the same plane (we removed the hallways that start and end at different heights, for example). One reason for this is the fact that we eliminated players' ability to aim in the vertical direction. With how the overhead camera ended up working (we included a black plane outside the rooms since the overhead camera could see beyond the walls of the room), there was the potential for verticality in the levels to cause issues with the overhead view.

The final rooms designs were made in Blender. No one on the team has much experience with 3D modeling or asset creation, and only some limited experience using Blender. While Blender is a powerful tool for model creation, it turned out that we likely did not require it for our purposes with Robot Rampage. Given the user study was the priority of the game, the actual room design and creating assets was a low priority, and it turned out to be easiest to use mostly primitive shapes in Blender to create the rooms. This likely resulted in a similar result to re-doing the room creation in Unity itself. Some of the final rooms are pictured below:

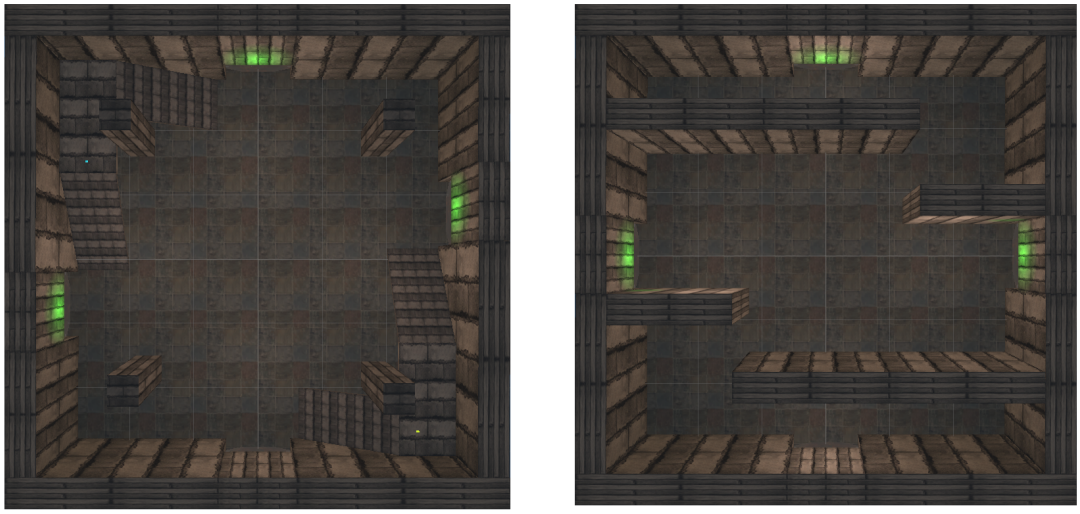


Figure 11: Two examples of the final room designs for Robot Rampage from a top-down view.

Section 4.2.5 Enemies

The addition of enemy robots gave players a proper goal: to destroy as many robots as possible. Robots were designed to stay still when they spawned in a room, only attacking when a player was nearby. State machines were used to achieve this, switching the robot from idle, walking, and attacking states. If a player was near, the robot would exit its idle state and begin walking towards the player. Once close enough, the robot would enter its attack state and fire at the player. The robots would enter and exit these states based on their distance from the player.

To create variety for users, three enemy types were created, each of which sporting a different color scheme, walking state distance, attack state distance, health, walk speed, and damage output. The intention was to have three difficulties of enemies with variance in the previously described values. The easy enemies would be unable to move and do the least amount of damage, the middle difficulty enemies would do considerable damage, and the hardest enemies would have the most damaging attacks. However, with the exclusion of health and

ammo from the final build of the game, many of the enemy stats went unused. This resulted in the medium difficulty enemies, who had a slightly higher walking range and speed than the hardest enemies, becoming the hardest enemies in the game due to the lack of damage values. The intended difficulties of the enemies appear below in Figure 12 from left to right: easy, medium, and hard.

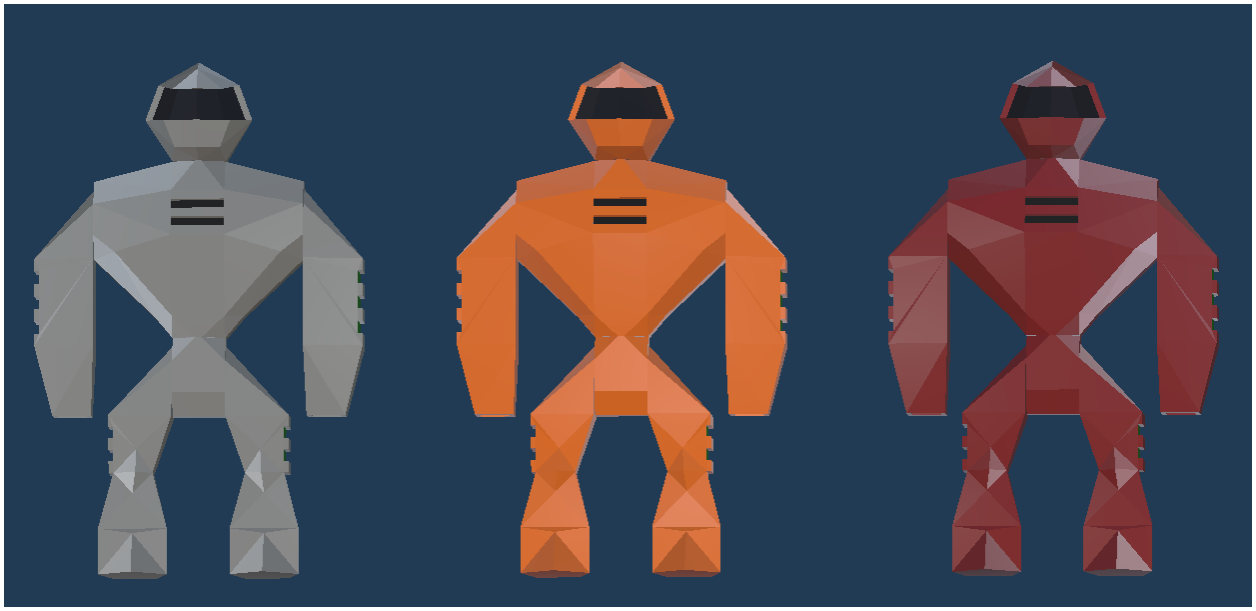


Figure 12: The Three Enemy Types

Section 4.2.6. Art and Sound

The development team lacked both a dedicated artist and sound designer. As a result, the game utilized existing assets from various sources to speed up the development process and to improve the user experience. Original assets and modifications to existing ones were also created by the team using Unity's built-in functionality.

For models, the team gathered three models from the Unity Asset store. The player model, Gompei the Goat, originated from the *Goat - Quirky Series* asset (Omabuarts Studio, 2022). The enemy robot model used was gathered from the *Robot JR-1 (animated) + mod1 & mod2* pack of assets, specifically robot JR-1 (Jope Anti-Studio, 2020). Additionally, the pistol

model used was the Pistol_05 prefab found within *PolyGun Sample Pack* (Out of Ram Studios, 2020). To modify the colors for the player, the team changed the colors in the texture used by the model's material. The process was simpler for robot models, as we could directly apply new materials created within Unity to any part of the robot such as the body. Additional models for the power-ups and enemy bullets were created by using Unity game objects and applying materials for the colors.

The majority of animations within the game originated from the included models, such as the player movement and robot actions. However, much of the player animations were modified using Unity's built-in animator to properly sync the player's gun position with the player model animations. The player shooting animation in particular was created by taking an existing animation for the goat and having the gun spin around the goat's horn during the time of the animation. Functionality to turn on and off all animations in real-time was added by using override controllers in Unity, but this functionality went unused in the final build.

Two textures were used for the floors and walls of the levels, both of which originated from *OpenGameArt*. The two textures were of rocky blocks, giving the level a rough, rugged look. A material was then applied to the level geometry, darkening the color of the textures.

Figure 13: The Original Textures Used for Rooms (qubodup, 2012; bart, 2012)

Basic particle effects were created to enhance the visual look of the game and provide proper feedback to players. The first particle effect appeared when a player shot their gun. The end of the gun produced a brief explosion to indicate that the player had fired the weapon. In addition, spark particle effects would shoot out of robot enemies when successfully shot. Both particle effects were created using Unity's built-in particle effect.

To give the game a distinct look and to blend the different art assets together more seamlessly, a pixel filter was created for the game. Using a Unity material and a shader script, the filter could be easily modified to change the size of the pixels in the filter. Initially, we explored modifying the filter as a parameter for the study, but instead kept it as an unchanging aspect of the game. A comparison between the game with the pixel filter off and on can be seen in Figure 14, showing the filter off and on in the top and bottom images respectively.

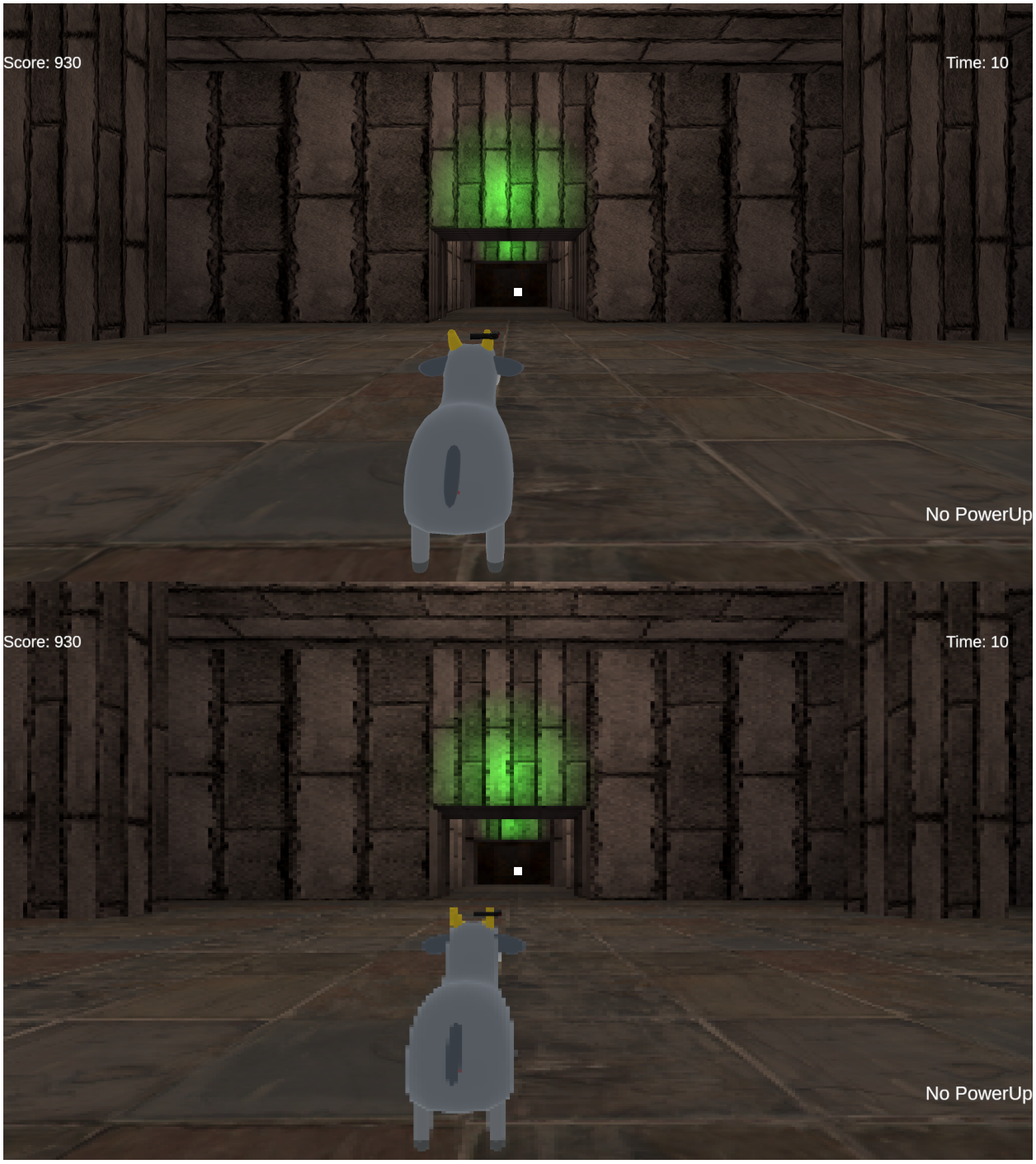


Figure 14: Comparison Between the Pixel Filter On and Off In-game

Various music and sound effects were included to not only enhance the atmosphere of the game but to provide enjoyment for players during the study. Music used within the game originated from the site *Soundimage.org* and sound effects were gathered from *FreeSound*. In

total, the game included seven songs and thirteen sound effects. Music played throughout the game, alternating between one another every thirty seconds. Sound effects would play after specific in-game events, such as a player attacking or walking. An audio manager would handle playing these sound effects, controlling their volume, and properly swapping between music and sound effects when needed. Tables containing the specific music and sound effects used in the game can be found in Appendix E.

Section 4.3. Custom Game: Robot Rampage Final Build

The final build of the *Robot Rampage* took the form of a shooter where players control Gompei as he traverses a variety of rooms and takes down robot enemies. In the story, robots from Worcester Polytechnic Institute's robotics department unleashed hundreds of deadly robots into the tunnels of the Atwater Kent building. It is up to Gompei to go down to the tunnels and stop the robot rampage. The final build aimed to provide a clear objective to players while effectively providing information for the study.

Section 4.3.1. Game Overview

In *Robot Rampage*, the player's objective was to score as many points as possible, which is accomplished by defeating robots, collecting power-ups, and taking as few hits as possible. The rooms and robots were generated endlessly, allowing for players to play for several rounds. Users utilized a mouse and keyboard to play. The basic controls for players include:

- W, S, A, and D Keys: Move Gompei up, down, left, and right respectively.
- Spacebar: Have Gompei jump.
- Mouse Left-click: Shoot Gompei's weapon.
- Mouse Movement: Rotate Gompei.

Enemies spawned within rooms in three types, each of which contains a different color. Attack range, health, and speed differ between enemy types. Gray robots had the longest attack range, but they also possessed the lowest health and did not have the ability to move toward players. Red and orange robots had shorter attack range but had the ability to walk towards the player if too far to attack and had higher health values.

Two power-ups had a chance to spawn once an enemy was defeated. The power-ups included a speed boost and a damage boost, both increasing the respective properties of the player for a set amount of time. Power-ups could not be combined, so players had to choose carefully when taking a dropped power-up.

Figure 15: Example of the Final Build Gameplay. Depicted is the Third-Person Perspective, a Dropped Speed-up Power-up, and Spark Particle Effects Originating from the Robot

Various actions within the game contributed to the player's score. A player could increase their total score by hitting a robot, destroying a robot, or collecting a power-up. However, a

player would lose points if hit by a robot or if they did not engage in combat for a certain period of time.

Section 4.3.2. Logging

Robot Rampage would log specific game-related actions into a csv file until shutdown. Each action was represented as a row within the file, with each column containing information related to the game during that specific action. From left to right, the columns included the time in seconds since the beginning of the game, the current round number, the current camera option, the current difficulty, the current texture quality, and the specified action.

The actions logged include:

- “Collected Damage Up”, logged when a damage power-up was collected by a player.
- “Collected Speed Up”, logged when a damage power-up was collected by a player.
- “New Round Started”, logged when a new round has begun.
- “Player Hit”, logged when a player was hit by a robot bullet.
- “Round Score”, logged when a round ended. The cumulative score up to the end of the round was included.
- “Shot Fired”, logged when a player shoots.
- “Shot Hit”, logged when a player successfully hits a robot.

Section 4.3.3. Parameters

The game changed three specific parameters in-between rounds: camera type, difficulty, and texture quality. These parameters were used based on the parameters related to delay and interrupts that were analyzed during development.

The camera type parameter had three options: first-person, third-person, and overhead. First-person took on the perspective of the player character, third-person depicted the game from behind the player character, and overhead aimed the camera at the player from above.

Difficulty switched between three options: easy, medium, and hard. As the difficulty increased, the required accuracy for the player to hit a robot increased as seen in Figure 16. The robot's accuracy for shots increased alongside the player's accuracy.

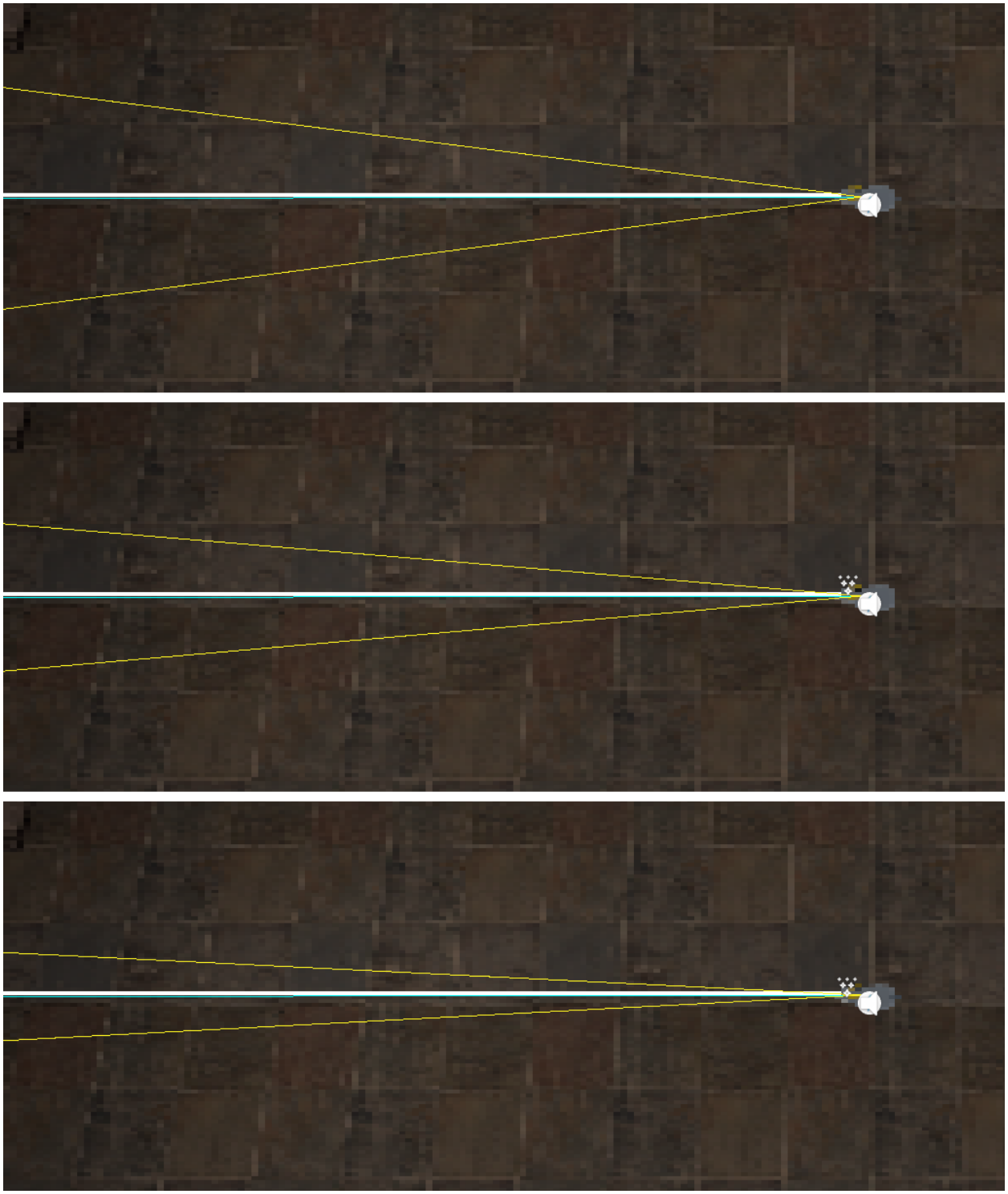


Figure 16: Visual Representation of the Changing Difficulties Using Unity's Debug Tools. From Top to Bottom: Easy Difficulty, Medium Difficulty, and Hard Difficulty

Textures also changed between low, medium, and high quality. High quality textures were the original textures, but as the quality decreased the more compressed the texture appeared as seen in Figure 17.

Figure 17: An Example of the Three Texture Quality Settings. From Left to Right: High Quality, Medium Quality, and Low Quality

Section 4.4 Client-Side Buffering Methods

A multi-threaded buffer written in C was implemented on a separate open-sourced Moonlight client from the study. The buffer was implemented with a queue with two threads for enqueue and dequeue. The initial buffer time determines the max size of the buffer. A separate function was written to perform all dequeuing actions. The code containing enqueuing actions was written in LiCompleteVideoFrame(), a Moonlight original function, where arriving frames were rendered.

In the dequeue thread, the function decided the two states of the buffer: FILL and PLAY. If the queue size was less than or equal to 0, the buffer entered the FILL state. The buffer entered the PLAY state when the queue size was greater than the max size set by the initial buffer time. Although the max queue size was set, it was still possible for the queue size to fluctuate around the max size due to the network condition for the given run. In the FILL state, the buffer did not

dequeue frames and allowed the enqueue thread to store frames. In the PLAY state, the buffer entered the critical region, dequeuing one frame item and one drstatus item (drstatus is part of Moonlight's original rendering code), leaving the critical region. The dequeued frame and drstatus item were used in the rest of the program to render the frame. The dequeue thread was sent to sleep at the end of the buffer function to match the average enqueue rate to avoid the buffer from being drained.

Section 4.4.1. Average Enqueue Rate Calculation Size

Two runs of buffer settings of 500 ms and 800 ms initial buffer time with a 30 ms jitter magnitude were performed in addition to a run of 0 ms buffer (no buffer) with no jitter for comparison. The average enqueue rate was calculated on the enqueue thread and was used to match the enqueue and dequeue rates. Interframe times for each frame were stored in an array for later average enqueue rate calculation. We performed three 30 seconds test runs with average enqueue rates of every window of 10 or 100 frames and logged the queue occupancy of the corresponding run. The queue occupancy graphs below allow us to see if the selected average calculation size can produce a relatively stable performance in queue occupancy. Since having a greater window size takes longer to produce an average, in Figure 19, the queue was continuously drained and filled for the first five seconds since the average still needs to be calculated. This type of behavior was not ideal for producing a relatively stable queue occupancy. In Figure 18, the queue was filled from the start and stayed relatively stable for the rest of the stream time. Thus, we chose to calculate the average enqueue rate every 10 frames.

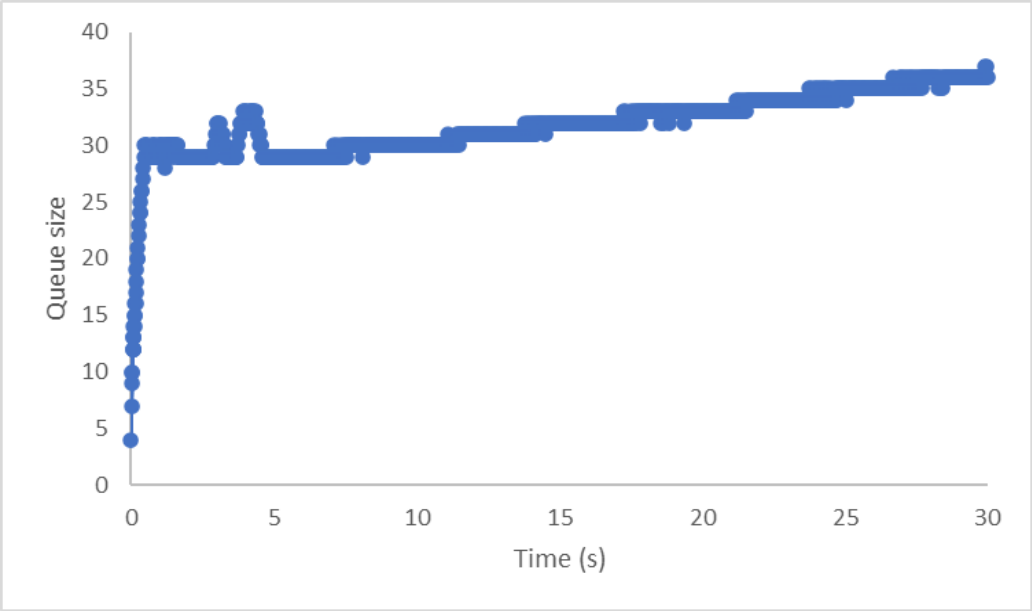


Figure 18: Queue Occupancy for a Buffer Using Average Enqueue Rate for Every 10 Frames

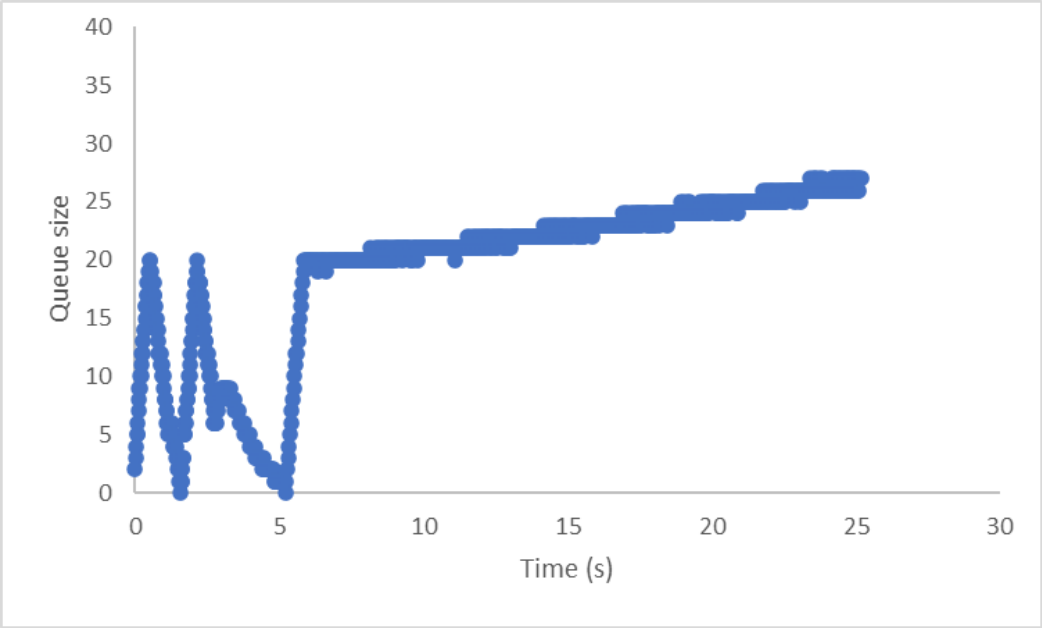


Figure 19: Queue Occupancy for a Buffer Using Average Enqueue Rate for Every 100 Frames

Section 4.4.2. Simulated Buffer

Due to the time and scope of the project, the buffer discussed above was not able to be implemented for the study. For the study, the buffer was simulated by changing jitter and latency

values through a router. Buffering can reduce jitter spikes but will result in a certain amount of additional latency depending on the size of the buffer. A larger buffer takes longer to fill and thus can cause greater latency. For the jitter settings we had none (0 ms), low (30 ms), and high (60 ms) magnitude values. A buffer was simulated by adding latency and reducing the jitter magnitude by 30ms. For example, the low jitter setting with a simulated buffer would become the no jitter and low latency setting.

Section 4.5. User Study

To properly collect data for the purposes of the study, the user study was designed to effectively gather data from users both inside and outside the playable game. The user study consisted of multiple steps:

1. The start of the study consisted of the user reading and signing the informed consent form, completing a reaction time test, and filling out a demographics survey.
2. The user played one practice round for each perspective for twenty seconds.
3. The user played a round of the game for thirty seconds.
4. Once the round finished, the user would fill out a post-round survey, answering two questions about their experience.
5. The user repeated steps 2 and 3 for forty-two total rounds excluding the initial practice rounds, with each round consisting of a different combination of game parameters and network conditions.
6. Once the user completes the required number of rounds, the study would conclude.

Section 4.5.1. Test bed

The user study utilized a setup of two computers and a Raspberry Pi that performed as the server, the client, and the router respectively. The streaming host OpenStream was installed on the server computer, hosting the custom game as well. The router computer was connected to a Raspberry Pi that received command lines and added different sets of delay and jitter to the internet connection. The streaming client Moonlight was used on the client computer, connecting to the server for players to play the game.

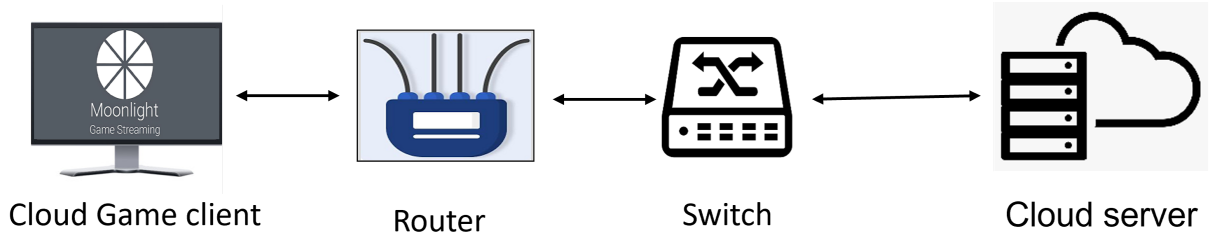


Figure 20: A Diagram Depicting the Test Bed Setup for the User Study

Section 4.5.2. Preliminary Tests

Before starting the study, the participant was tasked to complete three items before beginning the game. The first of which was an informed consent agreement, informing them of the study and any risks associated with it. The informed consent agreement is attached in Appendix A. Once completed, the participant was required to fill out a demographics survey, giving us insight into the experience with cloud gaming the users had prior to the study. More detail on the demographics survey can be found in Section 4.5.5.

The final requirement prior to the start of the game was a reaction time test. The test gathered information on the user's ability to react quickly to changes on the monitor. The reaction time test requires the user to click on the colored interface as soon as the interface changes color for ten rounds. In between each round, the test displayed the time the user took to

react and click on the interface once the color changed. After all ten rounds, the data was saved and the participant proceeded to the game.

Section 4.5.3. Game Round Structure

Once the participant completed the demographics survey, *Robot Rampage* was started for the user to play. The user played one practice round of 20 seconds for each perspective (third person, first person, top-down view). After filling out the practice round survey, the user plays multiple rounds, each round consisting of the user trying to score as high as they can by shooting robots and collecting power-ups in the given time before they fill the post-round survey. In between rounds, the game settings changed in addition to the network condition settings. There are six network condition combinations: no jitter no latency, low jitter with no latency, high jitter with no latency, no jitter with low latency, no jitter with high latency, and low jitter with low latency. For latency, a low setting was 50 millisecond and a high setting was 150 millisecond. As a result of the changes, the user's experience between rounds would change.

As the user played the game, specific information from each round was logged. This information indicated what actions the user took within a round and how well they did. Example information gathered includes the number of shots taken, the number of shots successfully hit, and the number of times hit by an enemy.

Section 4.5.4. Data Collection

Demographic survey data was collected through Google Forms and was stored in a Google Drive made for the study. In-between round survey data was collected by python scripts running alongside Moonlight on the client machine. Specific in-game actions were logged for each game round and stored together in a csv file. Once a participant finished answering all round surveys, a script compiled the in-between round surveys and the in-game round data into a

single zip file. All user study data files were secured on WPI sharepoint until the end of the study where the faculty advisor, Professor Claypool, archived the data. No identifiable information was released, and only group members and advisors were able to access user study data files.

Section 4.5.5. Surveys

The user study started with a demographics survey. The survey acquired the user's age and gender and gained information about the user's experience with games. Different levels in gaming can influence the user's adaptation to different internet conditions and thus produces variances in data collected from the survey. For example, a more experienced gamer might adapt to internet delays better than a less experienced gamer and perform better. In the demographic survey, we asked if the user plays games and how often they play games on a weekly basis to help us determine the user's experience as a gamer. User demographics survey questions are provided in Appendix B.

The user would fill out a survey after each round of playtesting. The survey acquired information about users' quality of experience (QoE) of the round. Examples include how smooth was the game experience for the round. From the survey responses, we can interpret the effect of jitter and latency on users' QoE and how well can different buffer sizes reduce the negative impact of poor internet conditions. In-between round survey questions are provided in Appendix C.

Section 4.5.6 Advertisement

The user study was advertised through email and was sent to all IMGD and CS students. To incentivize participation, playtesters were given a $10 Amazon gift card and playtesting credit for any class at WPI that required it. The recruitment email is provided in Appendix D.

Section 5 Analysis

This chapter presents the analysis from data collected in the user study and client-side buffer log files. Section 5.1 provides an overview of the demographics of the user study. Section 5.2 analyzes the relationship between QoE for both delay and jitter with game perspective, difficulty, and texture quality. Section 5.3 discusses player performance in relation to delay and jitter, primarily by perspective. Section 5.4 presents data for the enqueue rate of the client side buffer. Section 5.5 shows data for client side buffer queue occupancy. Section 5.6 presents data for client side buffer dequeue time.

Section 5.1. Demographics

The demographics survey questions found in Appendix B were filled out by participants prior to the start of the game. In addition, the participants took a reaction time test after the demographics survey but before playing the game.

There were a total of 35 participants, with the average age was 20.8 with a mode of 20. The youngest participant was 18 years old, and the oldest was 44 years old. There were 26 male participants, 6 female participants, one nonbinary participant, one genderfluid participant, and one participant who was unsure.

A majority of 31 participants answered yes to the question that asked whether the participants play video games, and 4 participants answered no. Out of the 31 participants who answered yes, 16 of them play video games almost every day, 6 participants play 4-5 times a week, and 9 participants play 1-3 times a week.

Only 11 participants had used any cloud game streaming service prior to the study and 24 had no cloud gaming experience. When asked to select which cloud game streaming services participants specifically used, the most frequently used service was NVIDIA GeForce Now with

5 results. Google Stadia placed second with 3 results, Xbox Cloud Gaming had 2 results, and the remaining services only had 1 result. Beyond the services listed, 5 participants selected Other.

The average reaction time was 250.124 milliseconds to click on the screen after the color changed. The median reaction was 232 milliseconds, indicating the distribution is slightly skewed to the right by a few participants that had slower reaction times. Local system latency could have added to the results of participants' reaction time.

Although the majority of the participants had gaming experience, more than half of the participants did not have cloud gaming experience. Among those who had experience with cloud gaming, NVIDIA GeForce Now was the most popular out of the given selection, but just as many participants had used a service not specified. Due to this, many of the QoE ratings were from participants experiencing cloud gaming for the first time.

Section 5.2. Quality of Experience Analysis

Average QoE ratings were calculated from the first question in the in-between round survey and analyzed versus perspective, difficulty, and texture quality across three levels of added delay or jitter. The delay magnitudes include none (0 milliseconds), low (50 milliseconds), and high (150 milliseconds). The jitter magnitudes set include none (measured at 9.2 ms/s), low (measured at 152 ms/s), and high (measured at 334 ms/s). Because the magnitude might range from the intended added amount to two times the added amount during streaming, the actual average jitter magnitudes for each set magnitude were calculated.

Section 5.2.1. Quality of Experience for Perspective

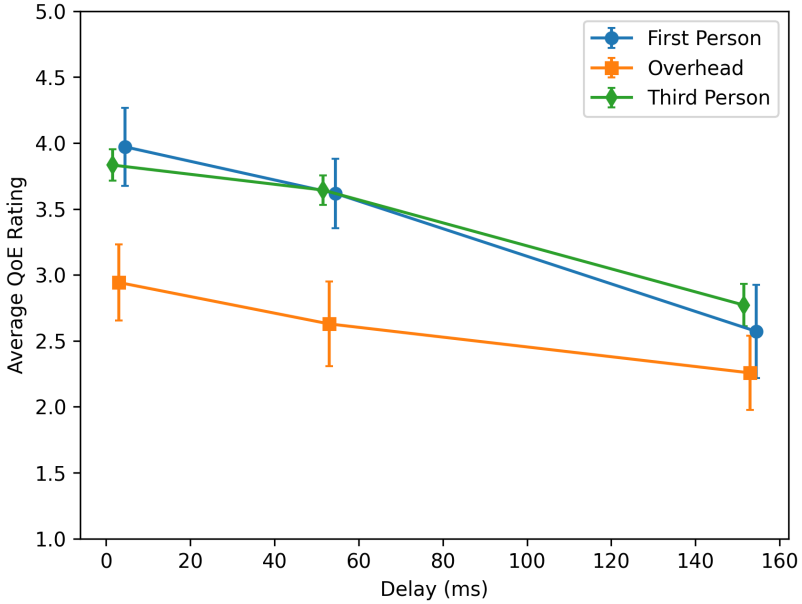


Figure 21: Average QoE Rating Versus Delay By Perspective

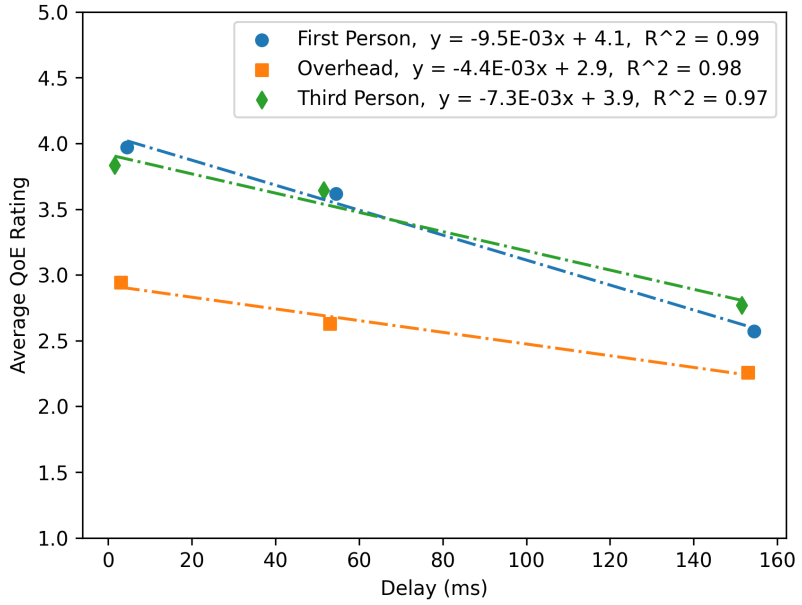


Figure 22: Average QoE Rating Versus Delay By Perspective with Trend Lines

To calculate average QoE ratings, combinations of delay magnitude and perspective were analyzed. As seen in Figure 21, the overhead view consistently ranked the lowest in QoE ratings, while the first-person and third-person perspectives ranked higher. However, while the first-person view ranked the highest with no delay, the increase in delay led to the perspective receiving worse QoE ratings compared to the third-person view. When taking a look at the regression lines of each perspective, the overhead perspective remained the most consistent for QoE ratings, while the first-person perspective had the sharpest decline in ratings, all with fairly high R^2 values.

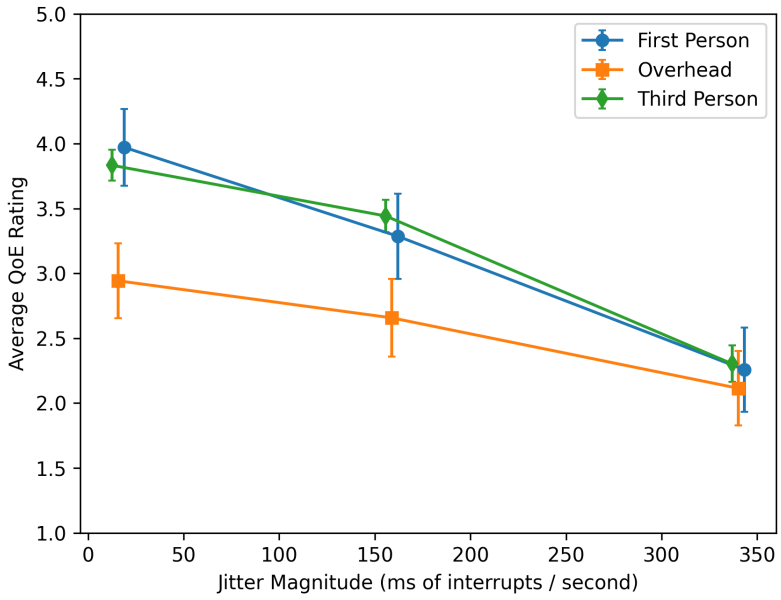


Figure 23: Average QoE Rating Versus Jitter Magnitude By Perspective

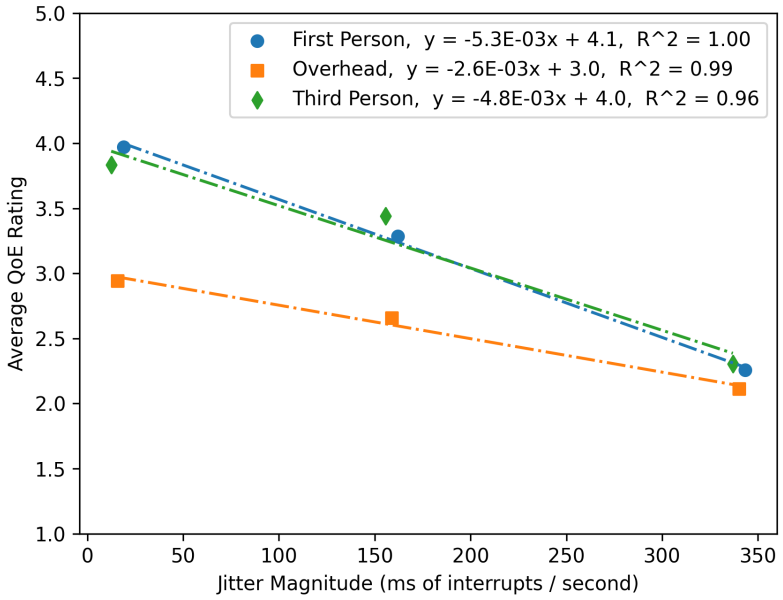


Figure 24: Average QoE Rating Versus Jitter Magnitude By Perspective with Trend Lines

Based on Figure 23, we noticed the overhead perspective had the lowest rating regardless of jitter magnitude. This could be because participants did not prefer the controls that the overhead view provided. Although the ratings for the overhead perspective were low, the slope of its trend line across the different levels of jitter magnitude was the smallest. The first person perspective had a steeper trend line than the third person perspective according to Figure 24. Compared to the first and third person, the overhead perspective was relatively more resilient to jitter, or participants disliked this perspective in general. The first person perspective had the highest QoE rating when no jitter was present, but its rating dropped below the QoE rating of the third person for low and high jitter magnitude settings. For high jitter magnitude, none of the three perspectives had a significant advantage over each other.

Section 5.2.2. Quality of Experience for Difficulty

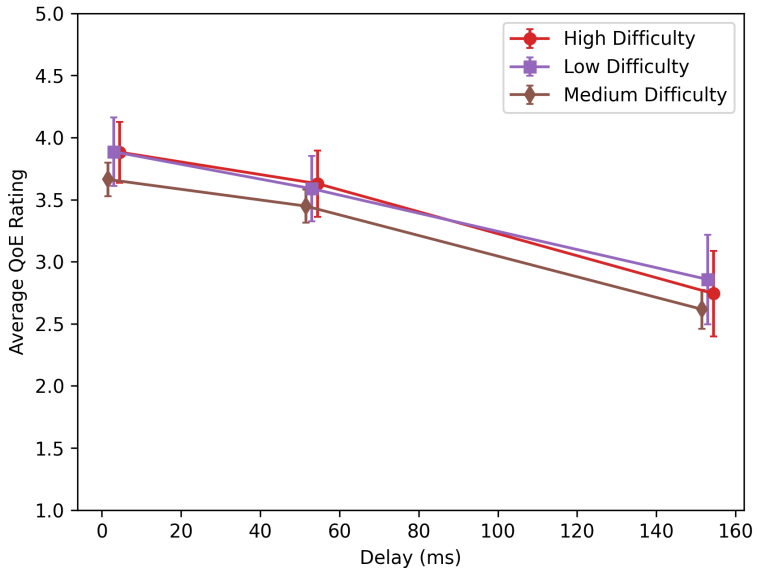


Figure 25: Average QoE Rating Versus Delay By Difficulty

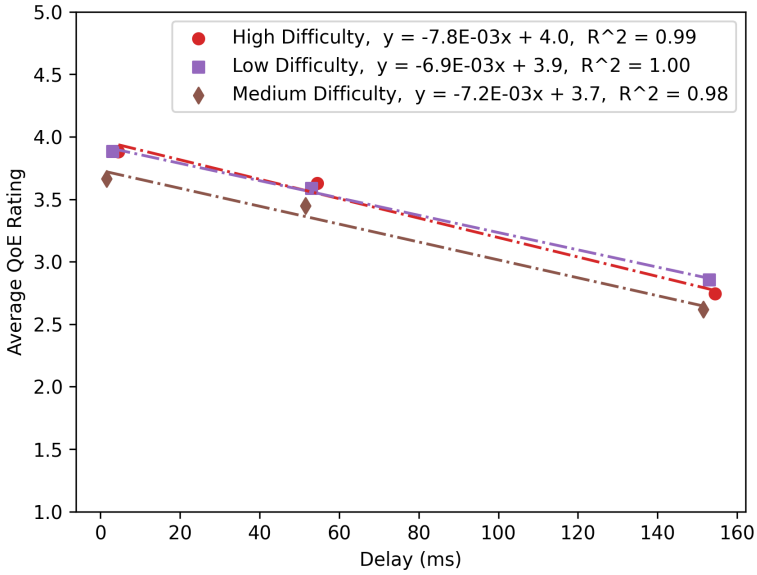


Figure 26: Average QoE Rating Versus Delay By Difficulty with Trend Lines

The average QoE ratings corresponding to the three levels of difficulty were compared with different levels of delay. According to Figure 25, three difficulty levels had similar trends,

and the data points were relatively close. Thus there was no statistical difference between each level of difficulty. Medium difficulty had the lowest rating for all delay settings, though not by any statistically significant margin. All three levels of difficulty's QoE ratings decreased at roughly the same rate as delay increased, and all with high R^2 values.

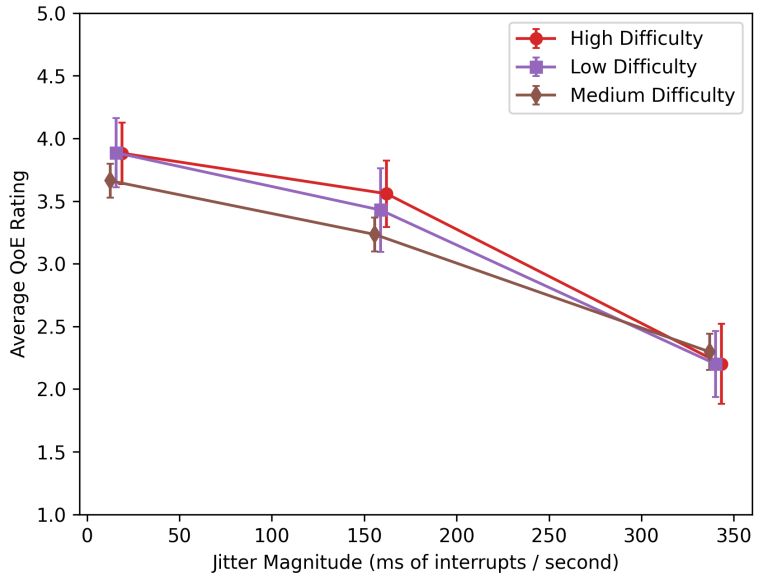


Figure 27: Average QoE Rating Versus Jitter Magnitude By Difficulty

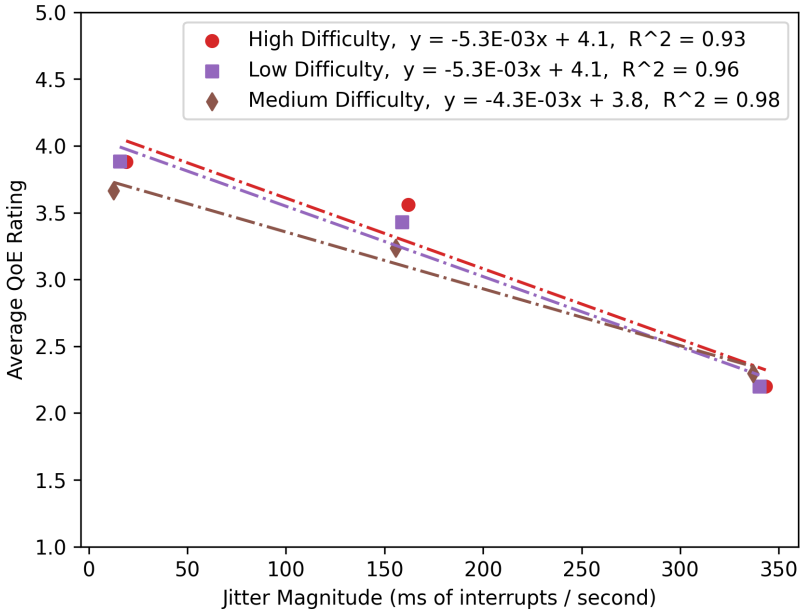


Figure 28: Average QoE Rating Versus Jitter Magnitude By Difficulty with Trend Lines

In Figure 27, combinations of jitter magnitude and difficulty were utilized to analyze average QoE ratings. The low and high difficulties scored the highest QoE ratings with no jitter, with both ratings rounding out to 3.88. With a low jitter magnitude, the high difficulty ranked with the higher rating over the low difficulty until both equaled out for QoE ratings with high jitter. The medium difficulty ranked the lowest in terms of QoE ratings for no jitter and low jitter, but ranked slightly higher than the other difficulties with the highest jitter magnitude. All three difficulties were similarly consistent in their drop in QoE ratings, indicating that difficulty had little effect on participants' experience when affected by jitter.

Section 5.2.3. Quality of Experience for Texture Quality

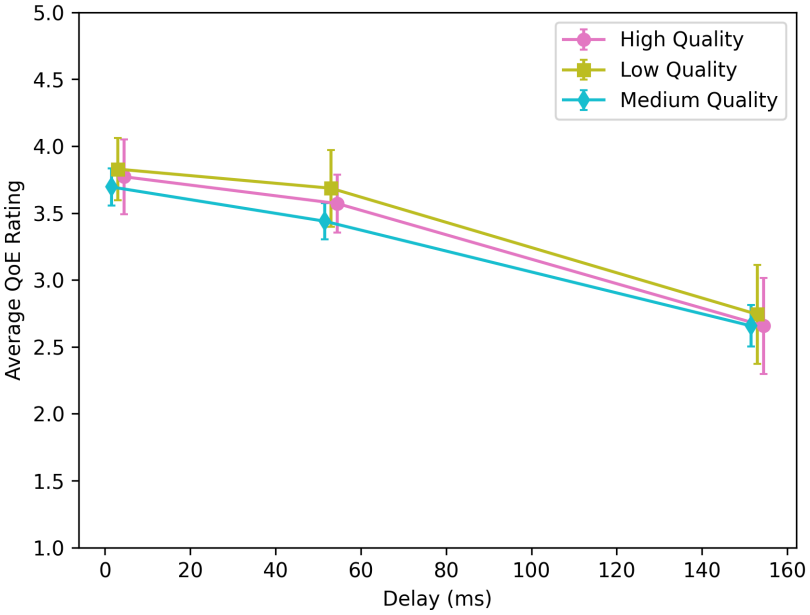


Figure 29: Average QoE Rating Versus Delay By Texture Quality

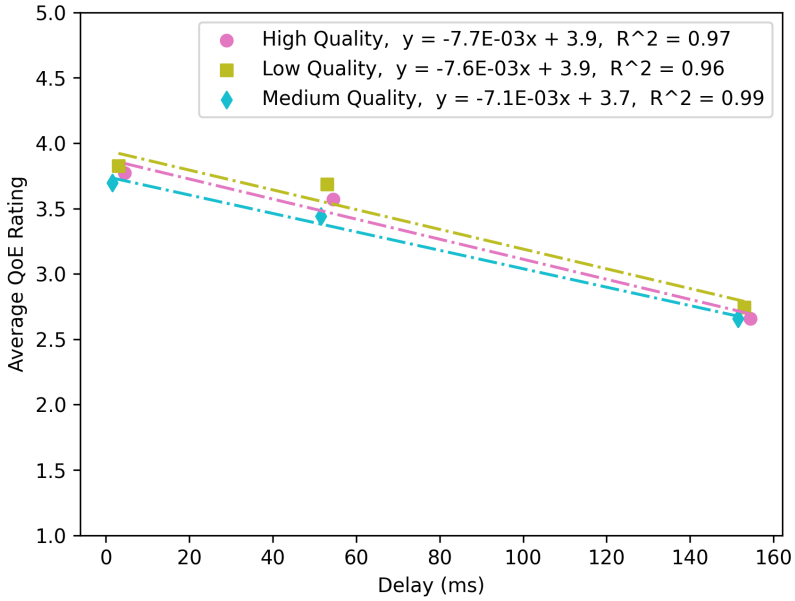


Figure 30: Average QoE Rating Versus Delay By Texture Quality with Trendlines

The average QoE rating was calculated for combinations of different delay levels and texture quality. There were three levels of texture quality: low, medium, high. Low texture quality had the highest QoE ratings across different levels of delay. As shown in Figure 29, the average QoE ratings for the three levels of texture quality were very similar to each other across different levels of delay. All three levels of texture quality's QoE ratings decreased as the delay level increased at roughly the same rate as shown in Figure 30.

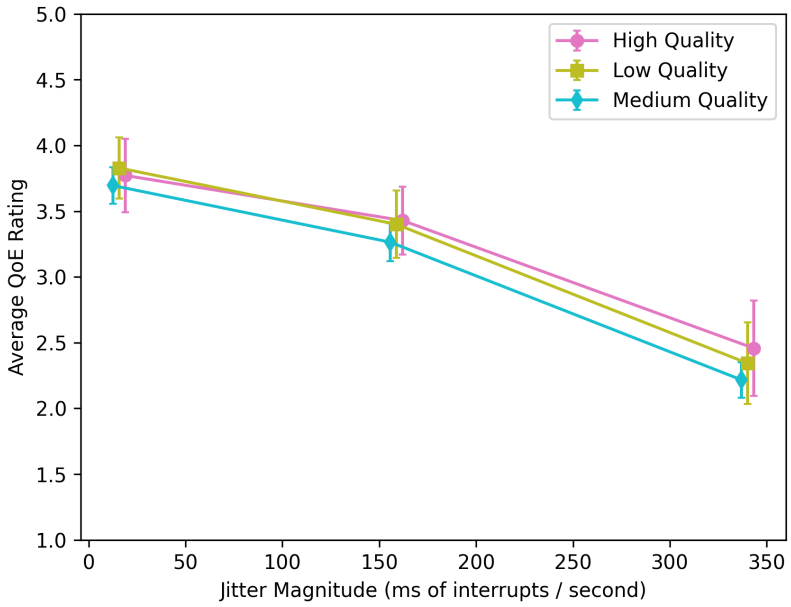


Figure 31: Average QoE Rating Versus Jitter Magnitude By Texture Quality

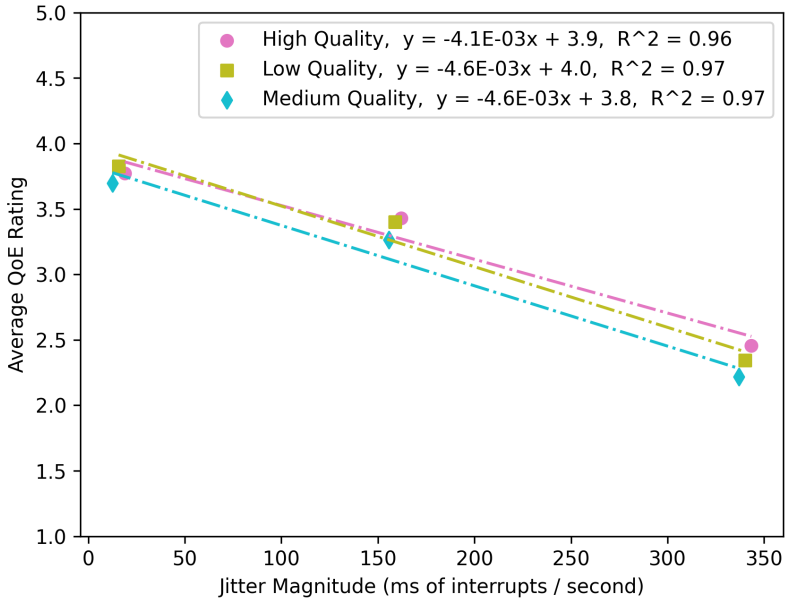


Figure 32: Average QoE Rating Versus Jitter Magnitude By Texture Quality with Trend Lines

As shown in Figure 31, medium texture quality scored the lowest ratings for all jitter magnitude levels. When no jitter was added, low texture quality had the highest rating, but later decreased below high texture quality as jitter magnitude increased. Three levels of texture quality had similar QoE ratings across all levels of jitter magnitude. Three QoE ratings all decreased as the jitter magnitude increased. It appears that texture quality did not have a significant impact on participants' QoE, as indicated by the overlapping confidence intervals as well as the similar slopes and high R^2 values.

Section 5.3. Participant Performance Analysis

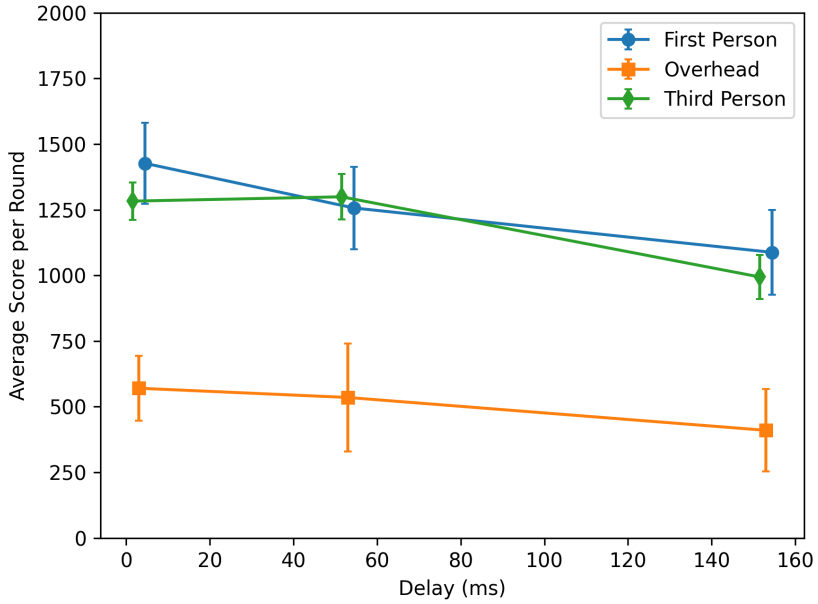


Figure 33: Average Score Per Round Versus Delay By Perspective

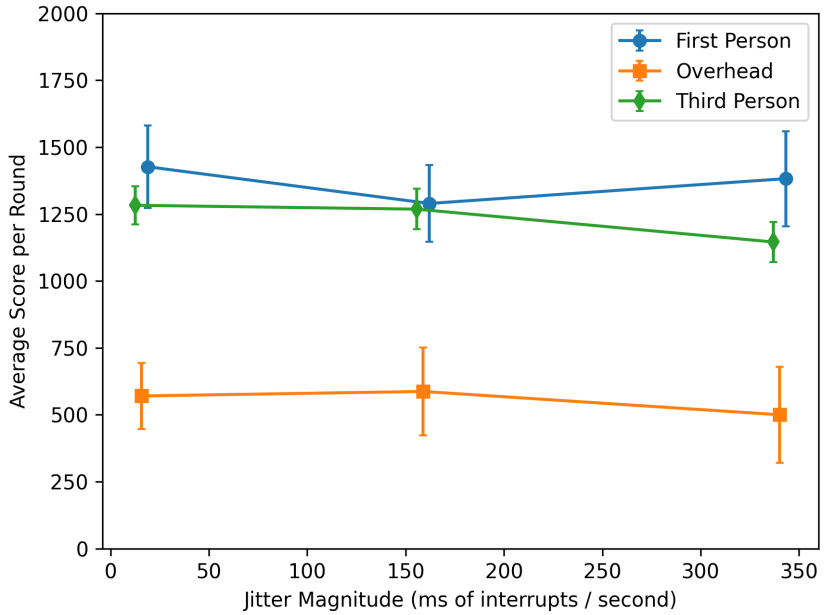


Figure 34: Average Score Per Round Versus Jitter Magnitude By Perspective

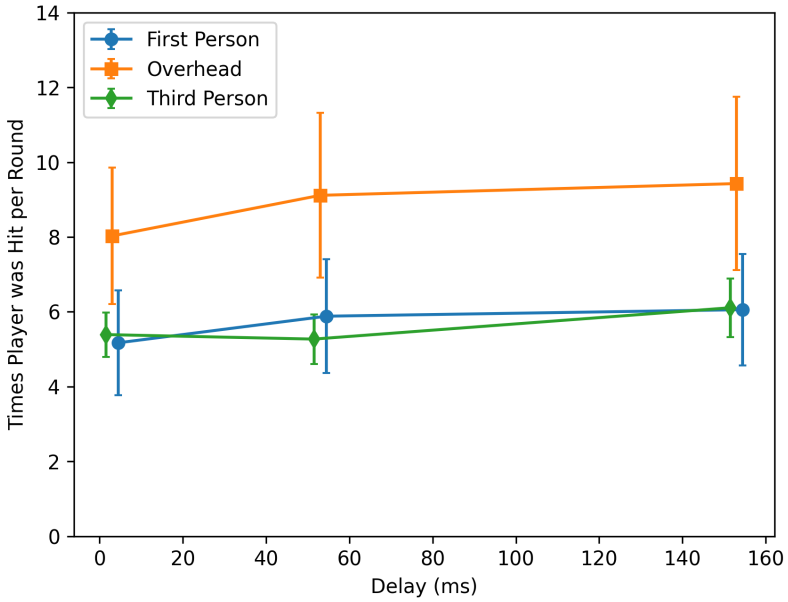


Figure 35: Damage Per Round Versus Delay By Perspective

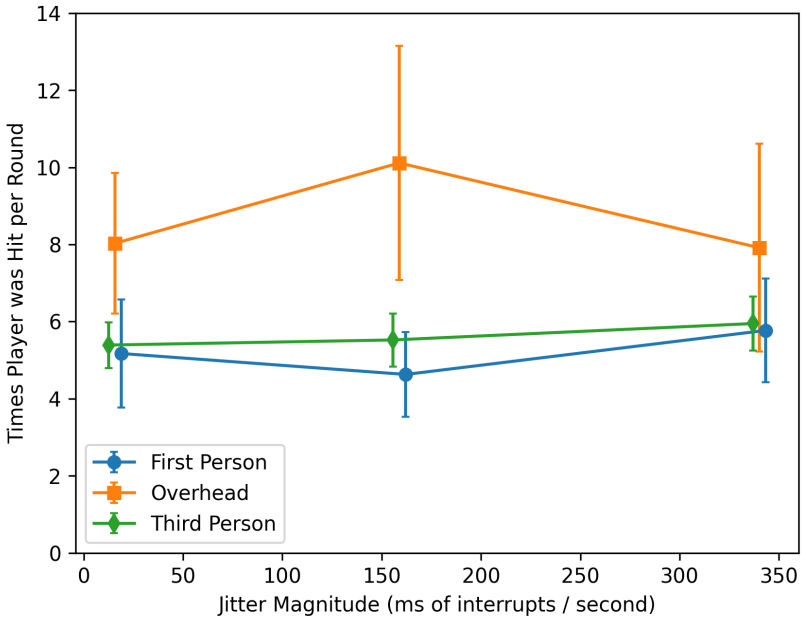


Figure 36: Damage Per Round Versus Jitter Magnitude By Perspective

The above figures (Figure 33 - 36) show various metrics of player performance vs. network conditions separated by perspective. Figure 33 and Figure 34 show that the overhead perspective resulted in statistically significantly lower score per round compared to the first and third person perspectives, while first and third person do not appear to have different average scores for the given network condition. Figure 35 and Figure 36 show damage taken versus delay and jitter respectively, also separated by perspective. These are less conclusive, but similarly indicate that participants took more damage (worse performance) in the overhead view compared to first and third person, while first and third person resulted in very similar amounts of damage taken per round. As seen in the figure named "Average Score Per Round Versus Delay By Perspective with Trend lines" from Appendix F, score is negatively affected the least in the overhead perspective while first and third person are similarly affected, with R^2 values of 0.98, 0.96, and 0.86 respectively. Additional charts for perspective that include the trend lines can be seen in Appendices F and G.

We performed the same analysis for score and damage taken separated by both texture quality and difficulty (as seen in Appendices H through K). As seen in the figure named "Damage Per Round Versus Jitter Magnitude By Difficulty with Trend Lines" in Appendix I, there seemed to be little correlation between jitter and damage taken for low and medium difficulty, but there was a positive correlation of 0.0095 hits per round per ms/s of jitter with an R^2 of 0.85 for high difficulty. Additionally, as seen in the figure named "Average Score Per Round Versus Delay By Texture Quality with Trend Lines" in Appendix J, average score per round is most negatively affected by delay for low textures, followed by medium textures, and least affected by high texture quality, with R^2 values of 0.97, 0.90, 0.98 respectively.

Section 5.4. Client-Side Buffer Enqueue Rate

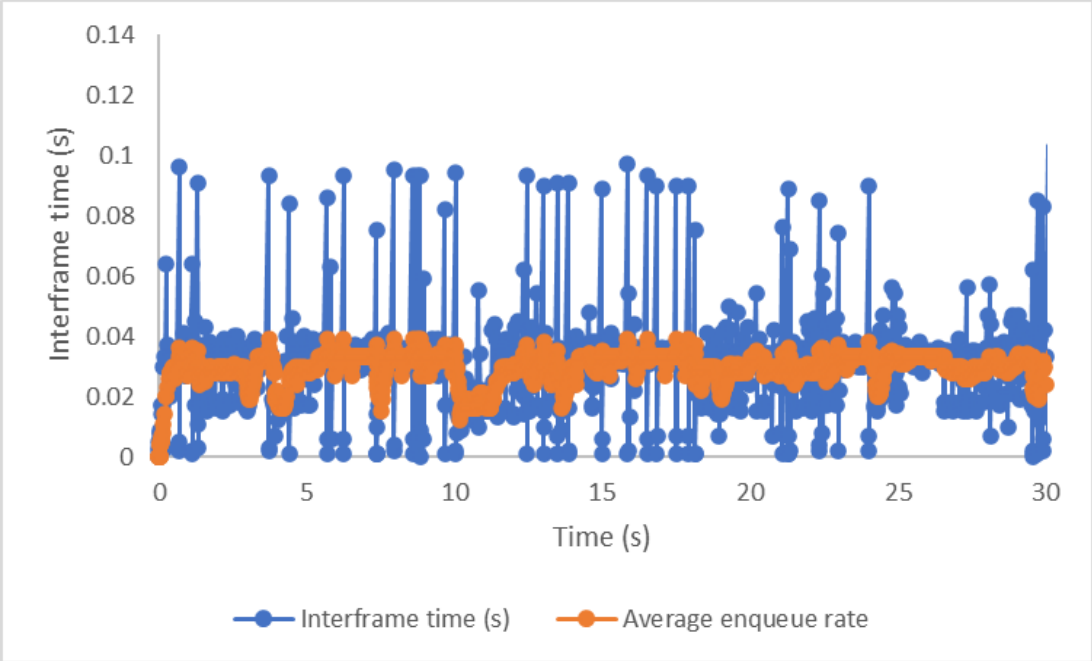


Figure 37: Enqueue Rate Versus Time for a 0 ms Buffer with 0 ms Jitter Magnitude

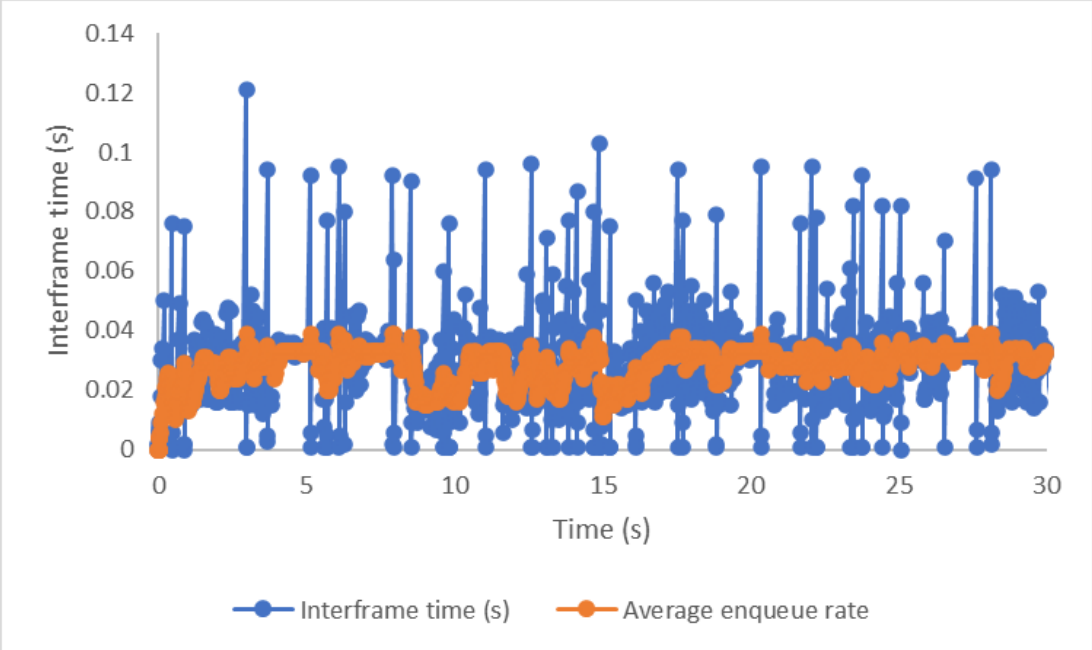


Figure 38: Enqueue Rate Versus Time for a 500 ms Buffer with 30 ms Jitter Magnitude

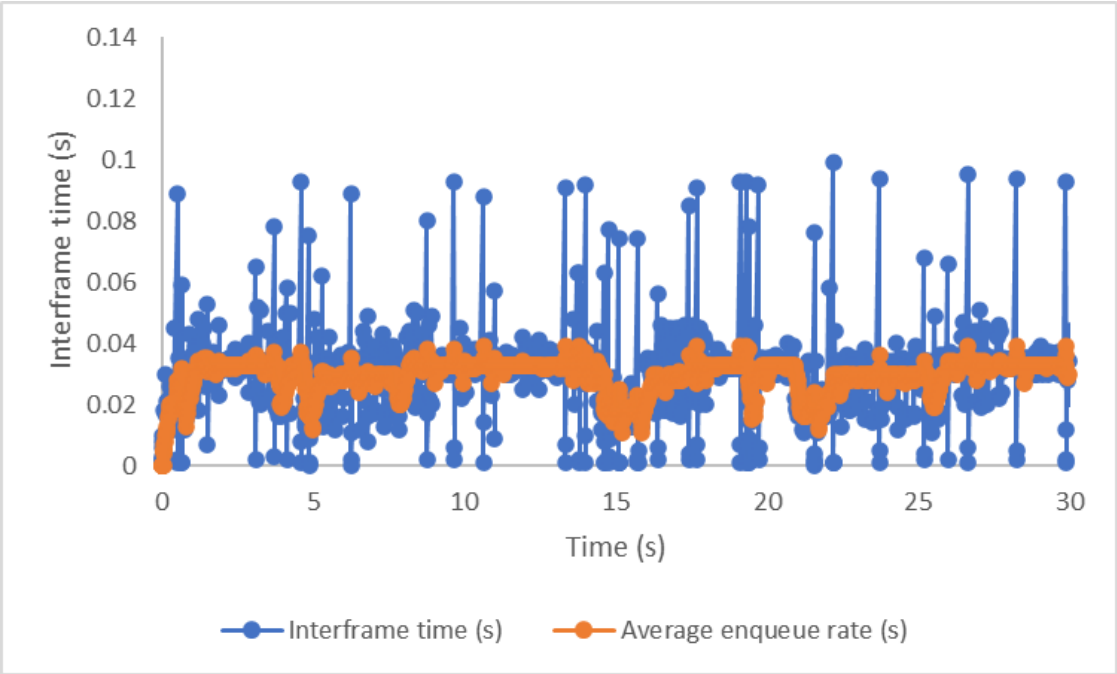


Figure 39: Enqueue Rate Versus Time for an 800 ms Buffer with 30 ms Jitter Magnitude

The interframe time (enqueue rate) was logged every time a frame was enqueued. The average enqueue rate was calculated by the interframe time of every ten frames in the enqueue thread. As depicted in Figures 37, 38, and 39, the average enqueue rate generally stayed around 0.033 seconds per frame for all three runs. Comparing the average enqueue rate lines for Figures 38 and 39, we could infer that with the same amount of jitter magnitude, an 800-millisecond buffer could provide a slightly stabler average enqueue rate. Based on Figures 38 and 39, we noticed the 500-millisecond buffer had more interframe time spikes that exceeded 0.05 than that of an 800-millisecond buffer.

Section 5.5. Client-Side Buffer Queue Occupancy

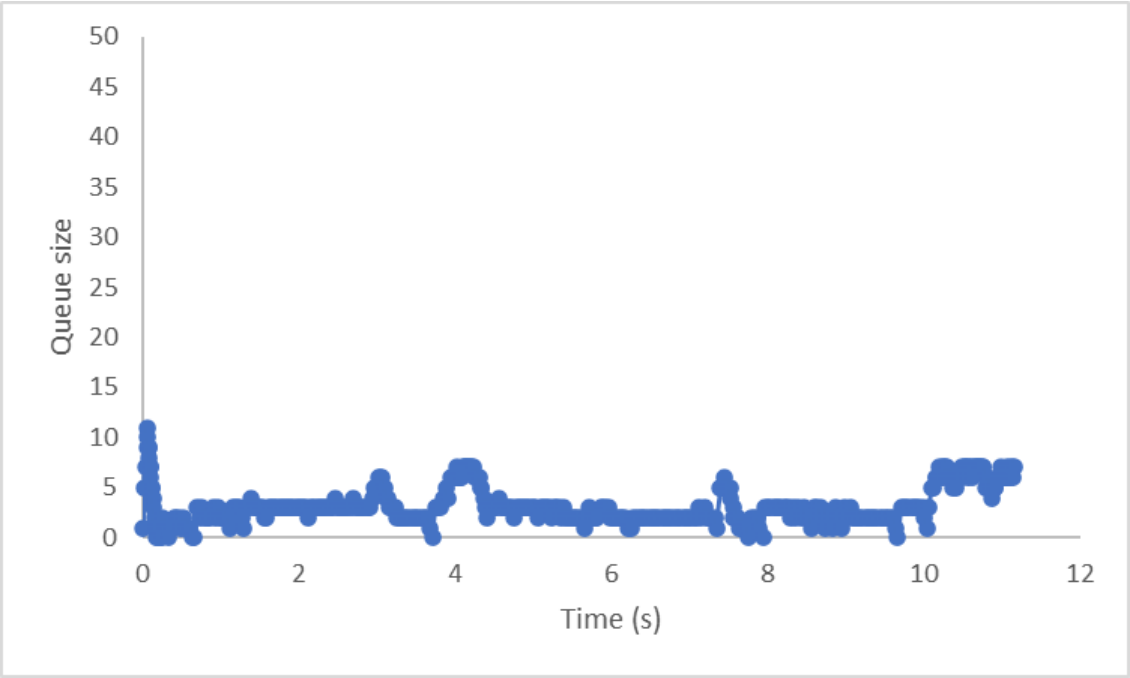


Figure 40: Queue Size Versus Time for a 0 ms Buffer with 0 ms Jitter Magnitude

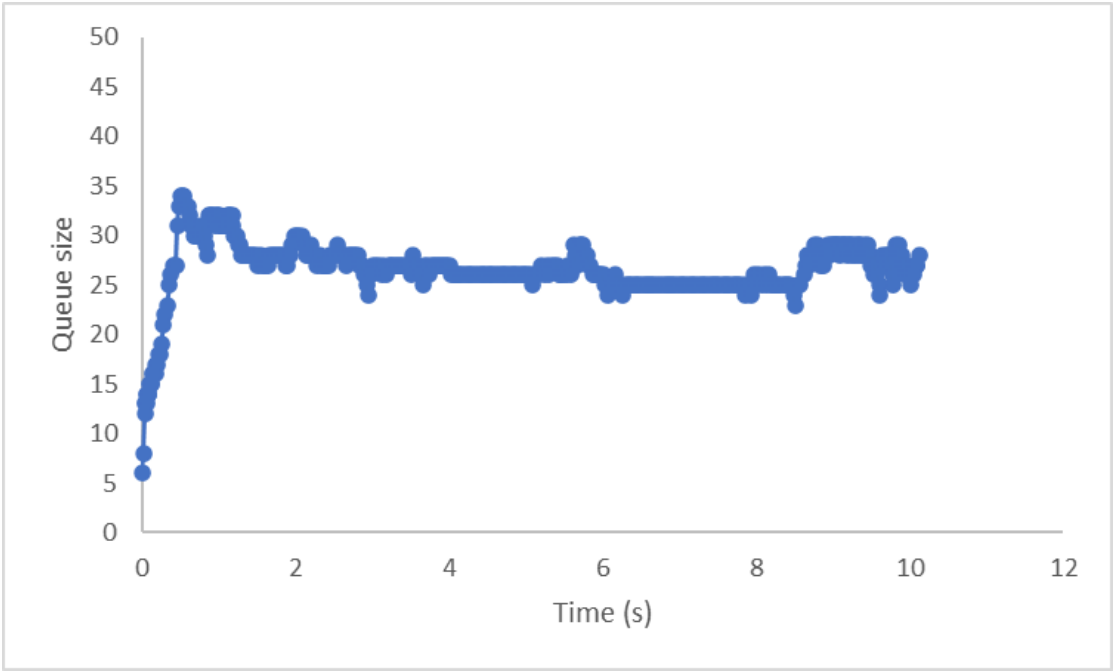


Figure 41: Queue Size Versus Time for a 500 ms Buffer with 30 ms Jitter Magnitude

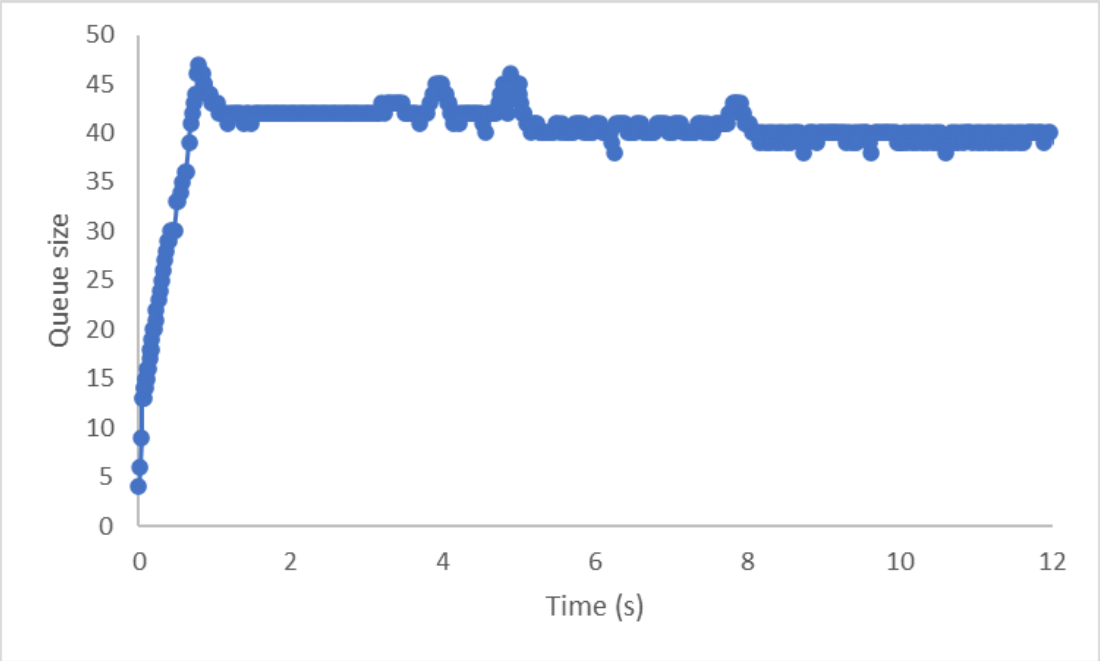


Figure 42: Queue Size Versus Time for an 800 ms Buffer with 30 ms Jitter Magnitude

In the dequeue thread, the queue occupancy was logged for each run. The queue size was logged every time a frame was enqueued or dequeued. The max queue size for a zero milliseconds buffer was zero. As mentioned in previous sections, although the max queue size was set after the initial buffer time, it was still possible for the buffer to exceed the max size for a small amount. Since it was impossible for the size to drop below zero, in Figure 40, the queue occupancy of a buffer with zero milliseconds of initial buffer time fluctuated around 0-10. In Figure 41, the queue occupancy for a 500-millisecond buffer had risen and fluctuated around 25-35 compared to the zero-millisecond buffer, which showed that increasing buffer time would lead to greater queue occupancy. In Figure 42, with a greater buffer time of 800 milliseconds, the queue occupancy raised and fluctuated around 38-46. Comparing Figures 41 and 42, with an 800-millisecond buffer, the queue occupancy was able to only fluctuate for a small amount around a constant size for a longer period of time compared to that of a 500-millisecond buffer.

According to these graphs, we were able to show that the buffer size can be controlled by the initial buffer time. Additionally, the buffer size and the initial buffer time were proportional.

Section 5.6. Client Side Buffer Dequeue Time

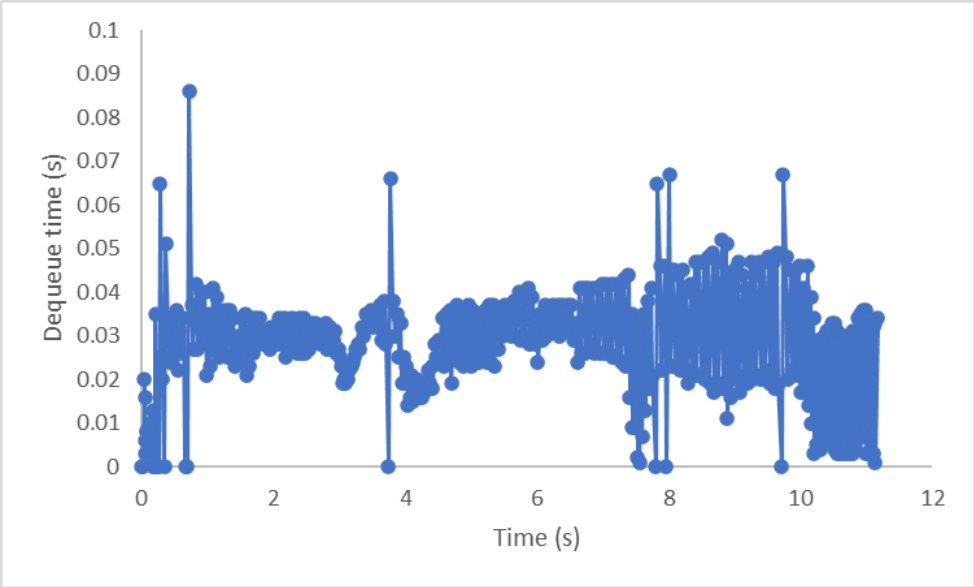


Figure 43: Dequeue Time Versus Time for a 0 ms Buffer with 0 ms Jitter Magnitude

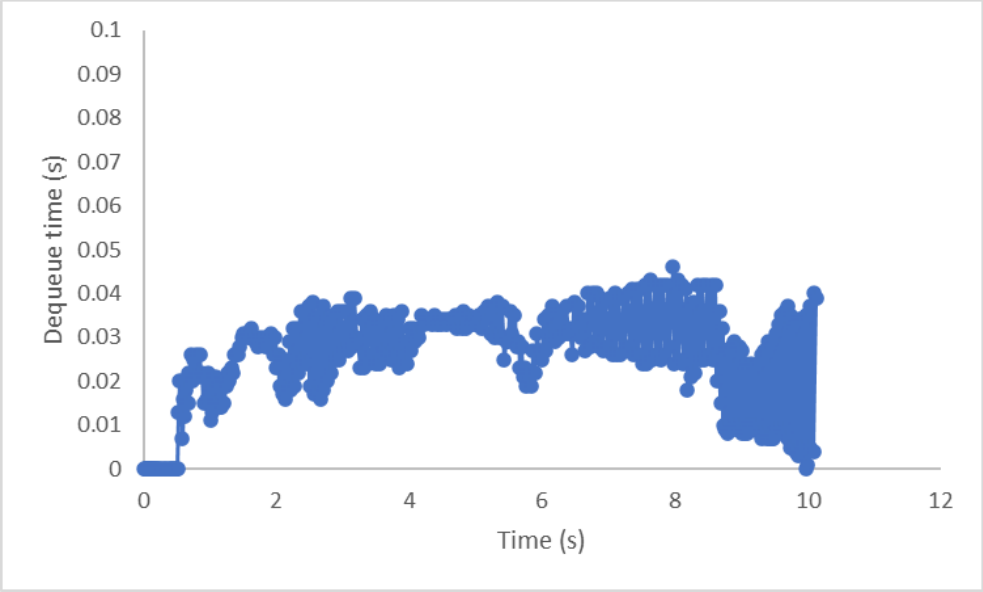


Figure 44: Dequeue Time Versus Time for a 500 ms Buffer with 30 ms Jitter Magnitude

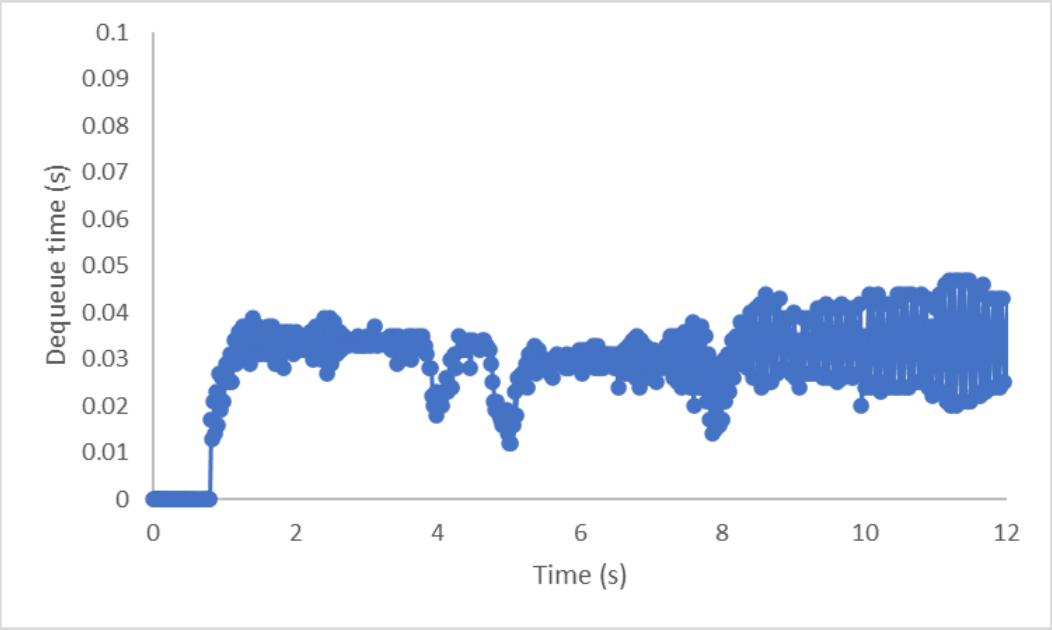


Figure 45: Dequeue Time Versus Time for an 800 ms Buffer with 30 ms Jitter Magnitude

The dequeue time was calculated in the dequeue thread, and the dequeue time logged the interframe time of every dequeued frame and drstatus. Comparing Figure 43 with Figures 44 and 45, we noticed that there were several significant spikes in the dequeue time. This could be caused by not having a buffer so that frames were being dequeued as soon as the frames arrived at the client, where frames were arriving at varying rates. Based on the data points in Figure 44, the dequeue time was able to stay around 0.010-0.046 seconds from the 0.5th to 8.7th second. We then noticed a decreasing trend, and the dequeue rate reached zero at the 10th second. As shown in Figure 45, the dequeue rate fluctuated between 0.012 and 0.043 seconds, which is a tighter range compared to that of Figure 44. In addition, the dequeue times in Figure 45 did not drop to zero, and the trend of the dequeue time stayed relatively flat after dequeue actions started compared to Figure 44.

Section 5.6 Analysis Summary

Based on the analysis in sections 5.1-5.2, all three categories were more negatively affected by delay than jitter based on the slope of the trend lines for QoE ratings. For perspectives, QoE for first-person and third-person appear to be similarly affected by delay and jitter, but first-person more so in both cases. The overhead view scored the lowest QoE ratings and had the worst performance overall among the three perspectives. This could indicate general dislike towards the perspective by participants in Robot Rampage, as verbal feedback from participants pointed to the perspective being less user-friendly. Despite this, it may be that the QoE for the overhead perspective is affected the least by the change in network conditions based on the trend lines. Meanwhile, the effects of texture quality and difficulty are far less pronounced and more inconsistent for QoE, score, and damage taken. A table comparing all the trend line equations gathered can be found in Appendix L.

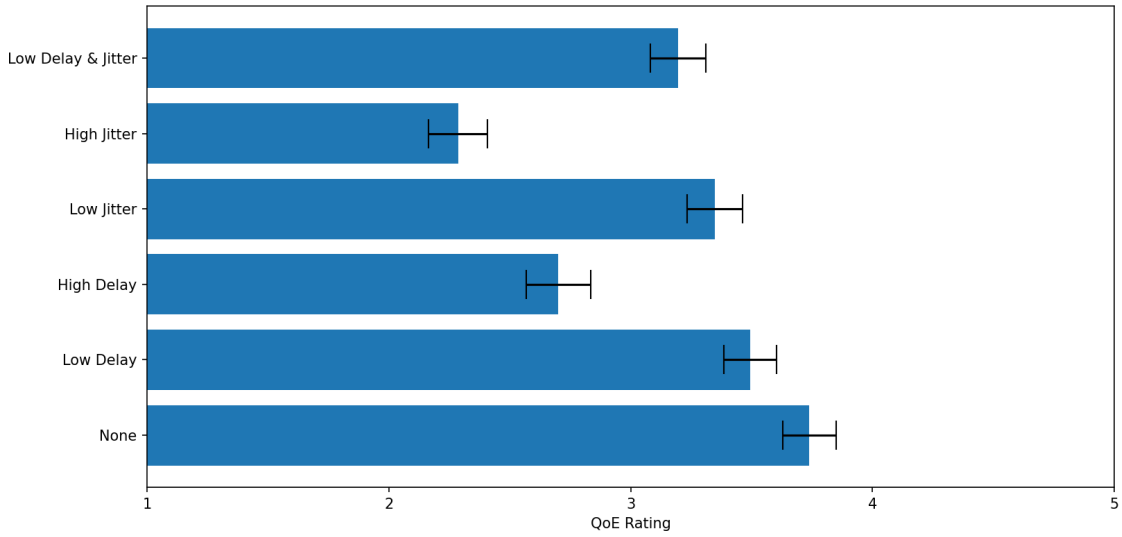


Figure 46: Average QoE Rating Separated by Network Conditions

Additionally, this project was interested in the effects on user's QoE when reducing jitter via buffering some number of frames in the client. This inherently adds delay, which theoretically is roughly equal (in ms) to the amount of jitter smoothed (in ms/s). Looking at low jitter (152 ms/s) compared to high delay (150 ms) can give an estimated indication of the effects of a client-side frame buffer on user's QoE. As shown in Figure 46, the average QoE for high delay (2.70) is significantly lower than the average QoE for low jitter (3.35). This would suggest that, at least for amounts of jitter around this level, having a large enough buffer to smoothly playout frames would result in a worse user experience.

From sections 5.3-5.5, we inferred a greater initial buffer time would produce a greater buffer size. Additionally, we showed the ability to control the buffer and produce relatively stable queue occupancy in addition to the ability to calculate enqueue rates, average enqueue rates, and dequeue rates.

Section 6. Conclusion

Cloud-based game streaming provides unique advantages over traditional methods of playing games while battling its own unique challenges in the process. Through the use of servers to stream games to user-side client machines, cloud gaming provides convenience to users who are unable to download and run the games on their local hardware such as phones or tablets. However, unfavorable network conditions such as latency and jitter affect the quality of experience a user could have with cloud gaming. To smooth out jitter, buffers can be implemented on the client side at the cost of added latency.

To address the problem of jitter and delay, the team ran a user study with a custom game to analyze how they affect the quality of experience within cloud-based game streaming. The custom game, Robot Rampage, was a 3D shooter created within the Unity game engine. It featured changeable game parameters, specifically three perspectives, three difficulties, and three texture quality settings. With these changeable game parameters and a set of different network settings with a range of delay and jitter values, the team was able to run a user study to measure how participants would rate their quality of experience based on combinations of specific game parameters and network conditions.

In addition, further research was done on implementing a buffer policy within an existing cloud game streaming service. The buffers stored frames in a container and streamed from the container at a specific rate instead of directly streaming frames from the server to the client to render. Thus when a jitter takes place, the buffer would output frames stored previously in the container instead of waiting for the next frame to arrive from the server. The buffer was not able to be implemented for the study due to the time and scope of the project.

The final results of the study indicate that perspective, difficulty, and texture quality QoE ratings were all more affected by delay than jitter. While perspective related QoE ratings displayed noticeable changes due to delay and jitter, QoE ratings showcased less pronounced changes across difficulties and texture qualities. The results also indicate potential design flaws in Robot Rampage, which can be improved in future versions. The results from the client side buffer analysis showed data that suggests it can control the number of frames buffered.

Section 7. Future Work

To expand the research on the project, we can select additional games from different genres to perform the user study. Gameplay mechanisms vary across game genres and thus can have different sensitivity in jitter and latency. By trying different game genres for the study, we can investigate the relation between game genres and jitter/latency sensitivity.

Only three levels (none, low, high) of jitter and latency were selected for the study. Future work can be done on streaming on a wider variety of jitter and latency values. With more values to test, we can produce a more specific and accurate analysis of the effect of jitter and latency on cloud streaming experience and the ability to smooth out jitter for different buffer sizes.

The client side buffer was not able to be deployed for the user study due to the time and scope of the project. From the log files and graphs generated, the buffer appeared to be bypassed by the original program and could not smooth out jitter spikes. This might be because the Moonlight client program had multiple functions that rendered frames. The sleep timer in the dequeue thread could not match the enqueue rate and caused the dequeue time to oscillate significantly. For future work, we need to examine the Moonlight client source code further, put all frame-rendering code under the dequeue thread, fix the sleep timer to properly match the enqueue rate, and implement a functioning buffer to the user study. Having a functioning buffer would help us to better understand the effect of different buffer sizes on users' QoE.

For future versions of this study, it is important to properly define the ratings used for QoE for participants. While QoE ratings are inherently subjective, making it clear to participants that the rating from 1 to 5 should be specifically in comparison to the practice rounds is important. Such a distinction could help normalize the data.

To further improve the custom game, the custom game needs to be playtested multiple times before the user study to ensure all game features and functionalities are best fitted for the purpose of the study. For example, users should not be able to fire their weapon without any penalty as it makes other aspects of the game such as accuracy less important. By playtesting early, design issues and dominant strategies can be found and dealt with ahead of time.

References

Allard, J., & Roskuski, A. (2015, March 6). *Measuring the Annoyance in Streaming Media Caused by Buffers and Interrupts* [Interactive Qualifying Project Paper]. Worcester Polytechnic Institute.

Aubuchon, B., & Langstaff, M. (2022, March 25). *Quality of Experience Evaluation for Buffer Sizing of Cloud-Based Game Streaming* [Major Qualifying Project Paper]. Worcester Polytechnic Institute.

bart. (2012, April 27). *Rockfloorbig* [Online Image]. OpenGameArt. https://opengameart.org/node/7391.

Claypool, M., & Claypool, K. (2009, April 26). *Perspectives, Frame Rates and Resolutions: It's all in the Game*. Proceedings of the 4th ACM International Conference on the Foundations of Digital Games, April 26-30, 2009, Orlando, FL, USA, ACM. http://www.cs.wpi.edu/~claypool/papers/perspective/

Connect to Work or Games from Anywhere | Parsec. (n.d.). Parsec. https://parsec.app

Fiber-optic Cable. (2023, April 12). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Fiber-optic_cable&oldid=1149460456

Franzese, T. (2021, October 13). *OK Boomer Shooter: How Indie Games Breathed New Life Into a Dying Genre*. Inverse. https://www.inverse.com/gaming/boomer-shooter-definition-origin

G-cluster. (2022, December 22). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=G-cluster&oldid=1128917683

GitHub - LizardByte/Sunshine: Self-Hosted Game Stream Host for Moonlight. (2019). GitHub. https://github.com/LizardByte/Sunshine

IR Team. (n.d.). *Network Jitter - Common Causes and Best Solutions | IR*. IR. https://www.ir.com/guides/what-is-network-jitter

Jarschel, M., Schlosser, D., Scheuring, S., & Hoßfeld, T. (2011, June 1). *An Evaluation of QoE in Cloud Gaming Based on Subjective Tests*. IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/5976180

Jope Anti-Studio. (2020). Robot JR-1 (Animated) + Mod1 & Mod2 (Version 1.0) [Digital File]. Available from https://assetstore.unity.com/packages/3d/characters/robots/robot-jr-1-animated-mod1-mod2-182628

Mangalindan, J. P. (2020, October 15). *Cloud Gaming's History of False Starts and Promising Reboots*. Polygon. https://www.polygon.com/features/2020/10/15/21499273/cloud-gaming-history-onlive-stadia-google

Moonlight Game Streaming: Play Your PC Games Remotely. (n.d.). Moonlight Open Source Nvidia Gamestream Client. https://moonlight-stream.org/

M-Stahl, KevinAsgari, aablackm, & v-chmccl. (2023, March 16). *Game Streaming Latency Measurement - Microsoft Game Development Kit*. Microsoft Learn. https://learn.microsoft.com/en-us/gaming/gdk/_content/gc/system/overviews/game-streaming/game-streaming-latency-measurement

Open-Stream – The First Completely Integrated Open-Source Game Streaming Platform. (n.d.). OPEN-STREAM. https://open-stream.net/

Omabuarts Studio. (2022). Goat - Quirky Series (Version 1.3) [Digital File]. Available from https://assetstore.unity.com/packages/3d/characters/animals/mammals/goat-quirky-series-178683

Out of Ram Studios. (2020). PolyGun Sample Pack (Version 3.0) [Digital File]. Available from https://assetstore.unity.com/packages/3d/props/guns/polygun-sample-pack-146654

Roach, J., & Parrish, K. (2021, March 29). *What is cloud gaming?* Digital Trends. https://www.digitaltrends.com/gaming/what-is-cloud-gaming-explained/

Sabet, S. S., Schmidt, S. K., Zadtootaghaj, S., Griwodz, C., & Möller, S. (2020, June 8). *Delay Sensitivity Classification of Cloud Gaming Content.* MMVE '20: Proceedings of the 12th ACM International Workshop on Immersive Mixed and Virtual Environment Systems, June 8, 2020, New York, NY, USA, ACM. https://doi.org/10.1145/3386293.3397116

qubodup. (2012, April 27). *Paving 6* [Online Image]. OpenGameArt. https://opengameart.org/node/8020.

Appendices

Appendix A: Informed Consent Form

Informed Consent Agreement for Participation in a Research Study

Investigator: Ryan Darcey, Botao Han, Sean O'Connor, Wenjie Zhang

Contact Information: stadiamqp22@gmail.com, gr-StadiaMQP22-23@wpi.edu

Title of Research Study: Latency and Jitter in Cloud Game Streaming

Sponsor: None

Introduction

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

Purpose of the study: The purpose of this study is to test the effects of varying network conditions on games played through cloud game streaming. Specifically, the effects of jitter and latency are to be tested through the use of a game with varying parameters and a customized cloud game streaming client. Data from the user study will be used to find effective solutions for latency and jitter based on the varying game parameters and network conditions.

Procedures to be followed: The participant will first be asked to fill out a demographic survey before playtesting. Then the participant will have a trial round to get familiar with the game. Once the trial play has been completed, the participant will play the game for several rounds.

After each round, the participant will complete a survey regarding their experience in the preceding round. Users will spend about 30 minutes in total playing the game, with the total study taking about an hour to complete.

Risks to study participants: There will not be any physical risks caused by our study. Our study gathers information about in-game actions, as well as email addresses for those who want to receive IMGD playtesting credits (which is entirely optional). This study is voluntary, you can withdraw from the study anytime you wish to.

Benefits to research participants and others: There are no benefits to research participants.

Record keeping and confidentiality: This is an anonymous interview/survey, and individual responses will not be published. Information collected will be stored in a Google Drive secured by WPI online security and will be destroyed after the research is completed. Responses will be collectively analyzed. Records of your participation in this study will be held confidential so far as permitted by law. However, the study investigators or its designee and, under certain circumstances, the Worcester Polytechnic Institute Institutional Review Board (WPI IRB) will be able to inspect and have access to confidential data that identify you by name. Any publication or presentation of the data will not identify you.

Compensation or treatment in the event of injury: This research does not involve more than minimal risk or harm. If you would like to have compensation or treatment of any kind, you are welcome to contact us with the information given below. You do not give up any of your legal rights by signing this statement.

Cost/Payment: Participants will be given a $10 gift card for participating in the research study.

For more information about this research or about the rights of research participants, or in case of research-related injury, contact:

Ryan Darcey, Botao Han, Sean O'Connor, and Wenjie Zhang, Email: gr-StadiaMQP22-23@wpi.edu

Mark Claypool, Tel. 508-831-5409, Email: claypool@wpi.edu

IRB Manager, Ruth McKeogh, Tel. 508-831- 6699, Email: irb@wpi.edu

Human Protection Administrator, Gabriel Johnson, Tel. 508-831-4989, Email: gjohnson@wpi.edu

Your participation in this research is voluntary. Your refusal to participate will not result in any penalty to you or any loss of benefits to which you may otherwise be entitled. You may decide to stop participating in the research at any time without penalty or loss of other benefits. The project investigators retain the right to cancel or postpone the experimental procedures at any time they see fit.

By signing below, you acknowledge that you have been informed about and consent to be a participant in the study described above. Make sure that your questions are answered to your satisfaction before signing. You are entitled to retain a copy of this consent agreement.

_____ Date: _____ Study Participant Signature

_____

Study Participant Name (Please print)

_____ Date: _____ Signature of Person who explained this study

Appendix B: User Demographics Survey Questions

1. Participation Number:
2. Age:
3. Gender:
    a. Male
    b. Female
    c. Other:
4. Do you play video games?
    a. Yes
    b. No
5. How often do you play video games?
    a. 1 to 3 times a week
    b. 4 to 5 times a week
    c. Almost everyday
    d. I do not play video games
6. Have you used any cloud game streaming services?
    a. Yes
    b. No
7. Select all cloud game streaming services that you have used:
    a. Amazon Luna
    b. Google Stadia
    c. NVIDIA GeForce Now
    d. PlayStation Now

e. Xbox Cloud Gaming

f. Other

g. None

8. If you would like playtesting credit, please include your WPI email:

Appendix C: In-between Round Survey Questions

1. Rate the quality of the previous game round (Please enter a number from 1.0 to 5.0):
2. Is the experience acceptable?
    a. Yes
    b. No

Appendix D: User Study Recruitment Email

Hello,

We are a MQP team studying the effects of jitter and latency on games played with cloud game streaming services. We are looking for participants to play a few rounds of our custom shooter game Robot Rampage and fill out an in-between round survey about your experience of playing the game.

Robot Rampage is a 3D shooter in Unity with three perspectives: first-person, third-person, and overhead. The game has players control Gompei as he traverses a variety of rooms and takes down robot enemies. Each playtesting session will last around 45-60 minutes. Participation is voluntary and participants can quit at any time. Participants are paid $10 in gift cards for each session and receive IMGD playtesting credit if needed.

If interested, please select a time slot here.

For any further questions, please email gr-stadiamqp22@wpi.edu.

Thank you,

Jitter and Latency in Cloud Game Streaming Team

Appendix E: Game Audio References

Music:

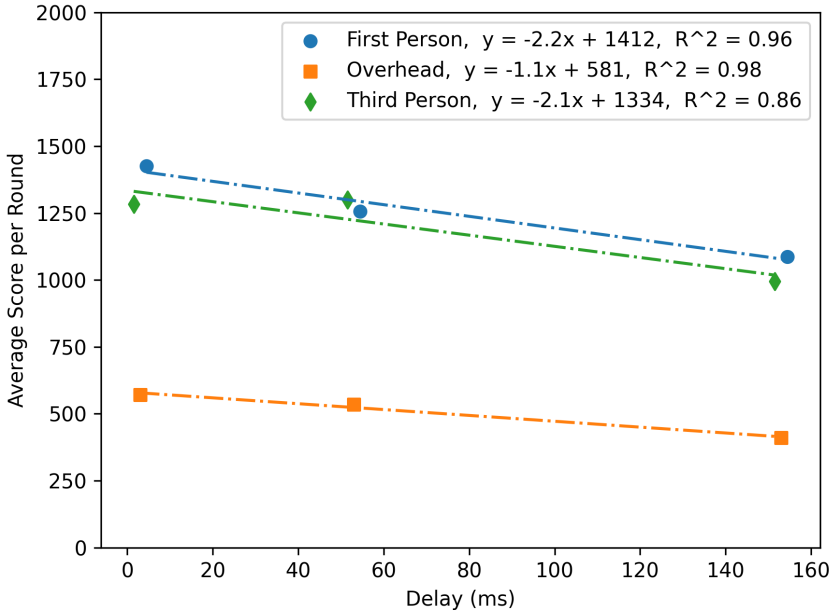
Name	Artist	Game Description	Source
More	Eric Matyas	Level Theme 1	https://soundimage.org/wp-content/uploads/2019/01/More-8-Bit-Drama_Looping.mp3
Craffeine Crazed Coin-Op Kids	Eric Matyas	Level Theme 2	https://soundimage.org/wp-content/uploads/2020/07/Caffeine-Crazed-Coin-Op-Kids.mp3
8-Bit Espionage	Eric Matyas	Level Theme 3	https://soundimage.org/wp-content/uploads/2017/10/8-Bit-Espionage_Looping.mp3
Arcade Drama	Eric Matyas	Level Theme 4	http://soundimage.org/wp-content/uploads/2017/12/Arcade-Drama.mp3
Cyberpunk Arcade 2	Eric Matyas	Level Theme 5	https://soundimage.org/wp-content/uploads/2021/09/Cyberpunk-Arcade-2.mp3
8-Bit Drama	Eric Matyas	Menu Theme	https://soundimage.org/wp-content/uploads/2017/03/8-Bit-Drama.mp3

Sound Effects:

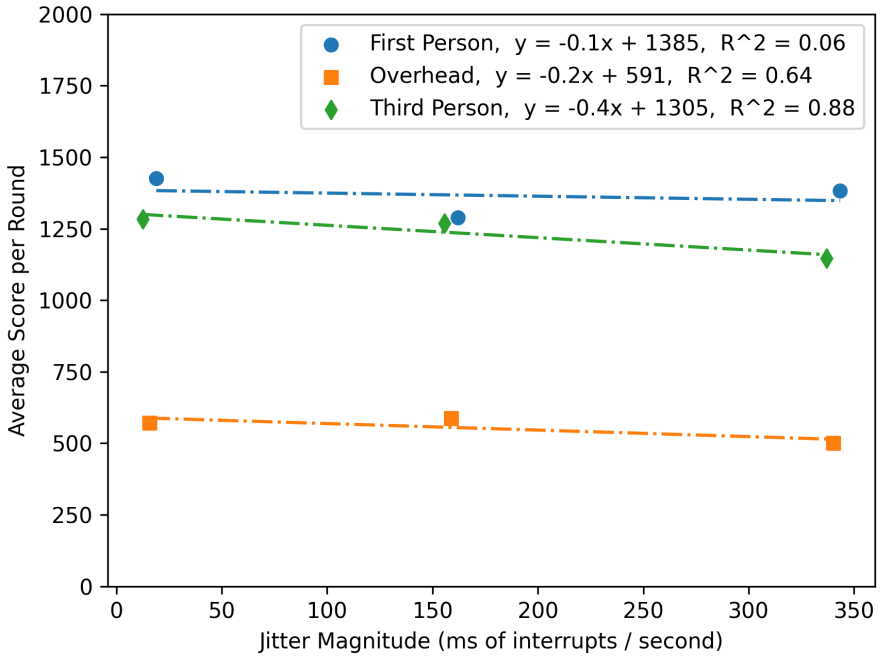
Name	Artist	Game Description	Source
Laser	kafokafo	Enemy Attack	https://freesound.org/people/kafokafo/sounds/128229/
8bit_hit_11	Soundholder	Enemy Damage	https://freesound.org/people/Soundholder/sounds/425348/
ROBOT WHAT IS HAPPENING TO ME	metrostock99	Enemy Death 1 - 3	https://freesound.org/people/metrostock99/sounds/514696/
Jumping On a Bed	deleted_user_7146007	Enemy Run	https://freesound.org/people/deleted_user_7146007/sounds/383753/

Retro Gun Shot	Jofae	Player Attack	https://freesound.org/people/Jofae/sounds/363698/
8-bit damage sound	EVRetro	Player Damage	https://freesound.org/people/EVRetro/sounds/501104/
Cartoon jump	Bastianhallo	Player Jump	https://freesound.org/people/Bastianhallo/sounds/462958/
Retro Bonus Pickup SFX	suntemple	Player Pickup	https://freesound.org/people/suntemple/sounds/253172/
8-bit footsteps	EVRetro	Player Walk	https://freesound.org/people/EVRetro/sounds/501102/

Appendix F: Average Score Per Round By Perspective Trendlines

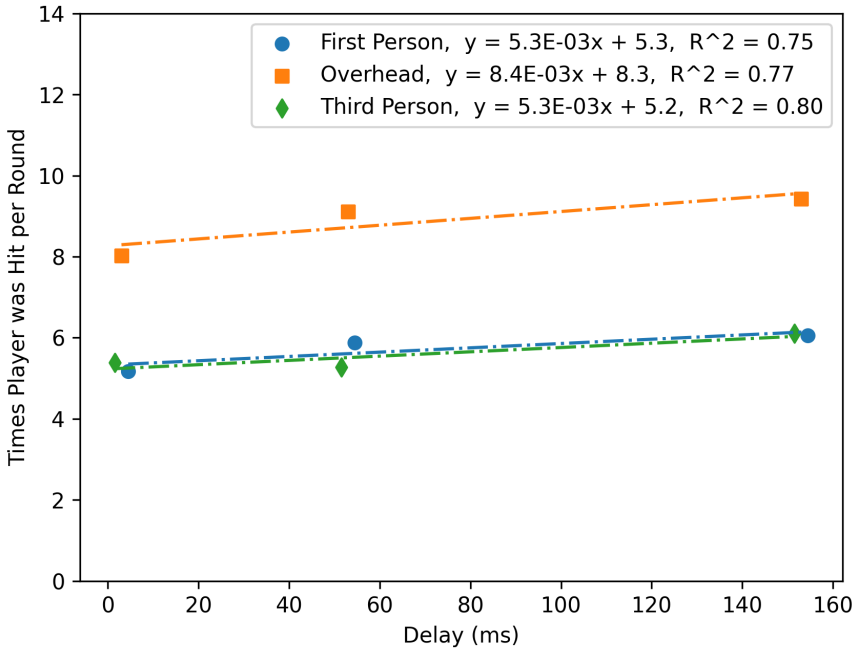


Average Score Per Round Versus Delay By Perspective with Trend lines

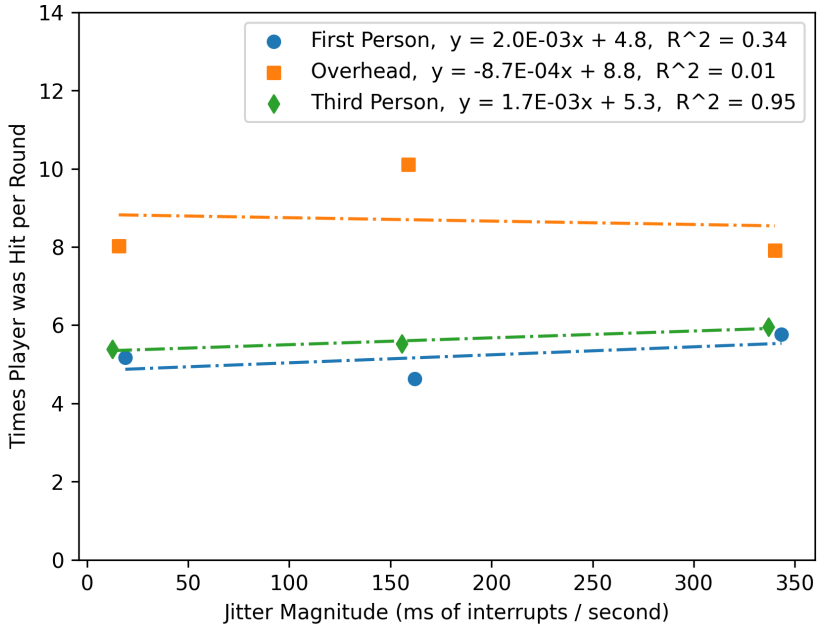


Average Score Per Round Versus Jitter Magnitude By Perspective with Trend lines

Appendix G: Damage Per Round By Perspective Trendlines

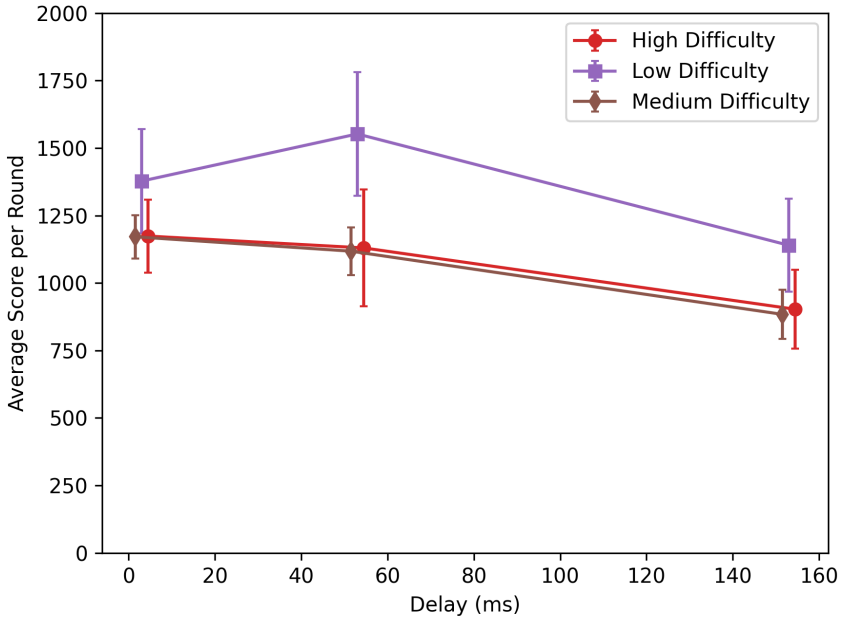


Damage Per Round Versus Delay By Perspective with Trend Lines

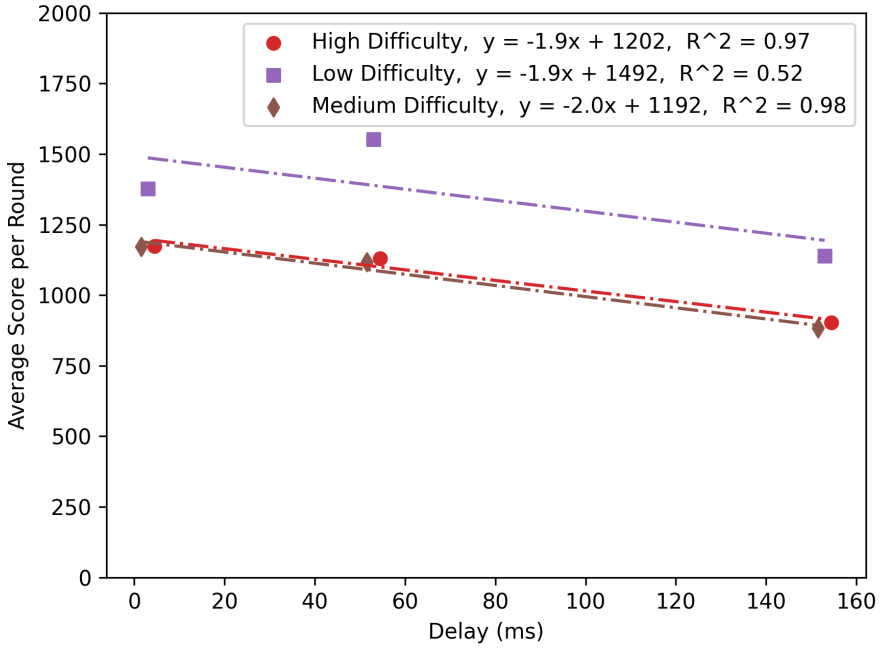


Damage Per Round Versus Jitter Magnitude By Perspective with Trend Lines

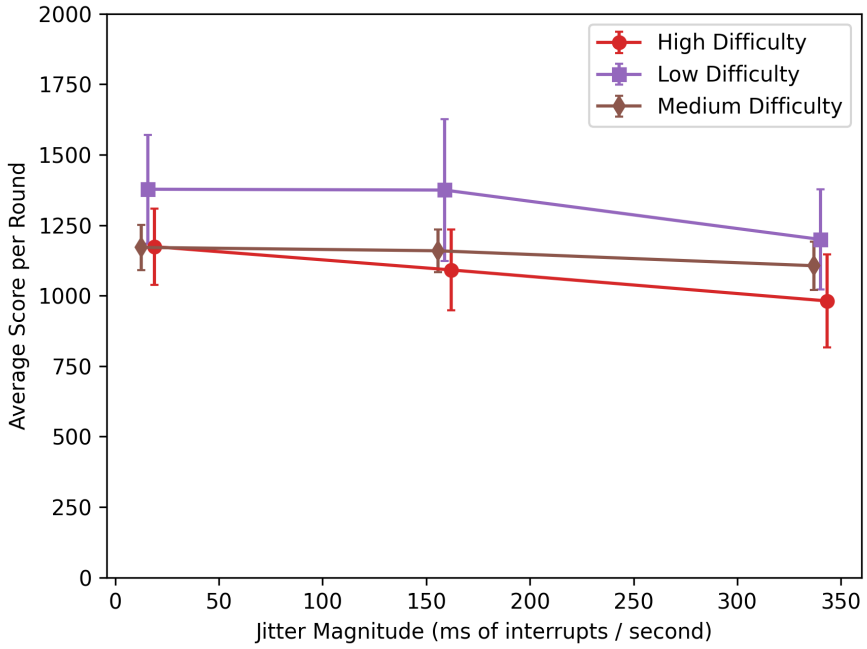
Appendix H: Average Score Per Round By Difficulty



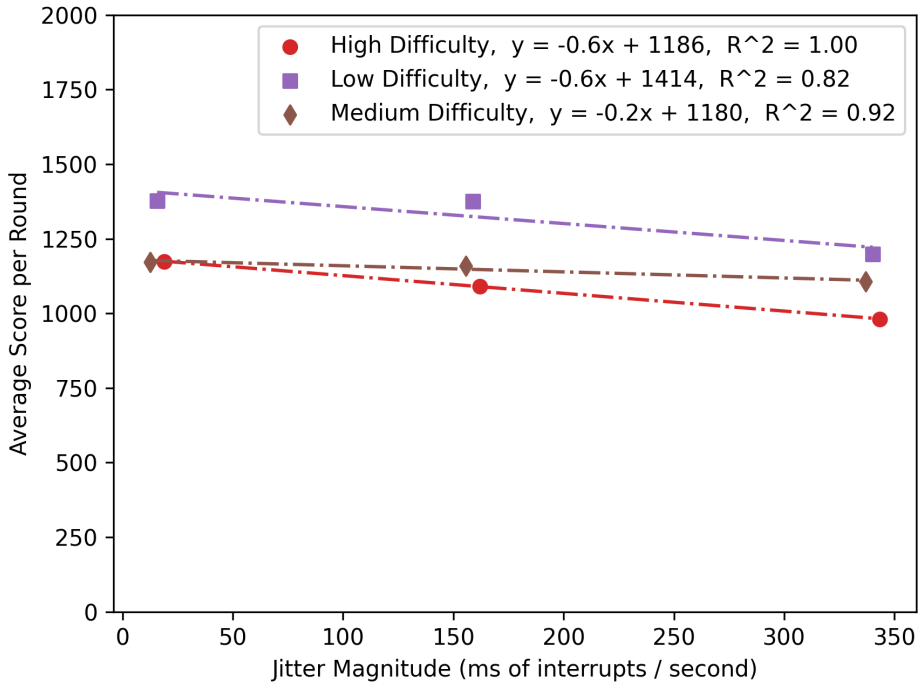
Average Score Per Round Versus Delay By Difficulty



Average Score Per Round Versus Delay By Difficulty with Trend lines

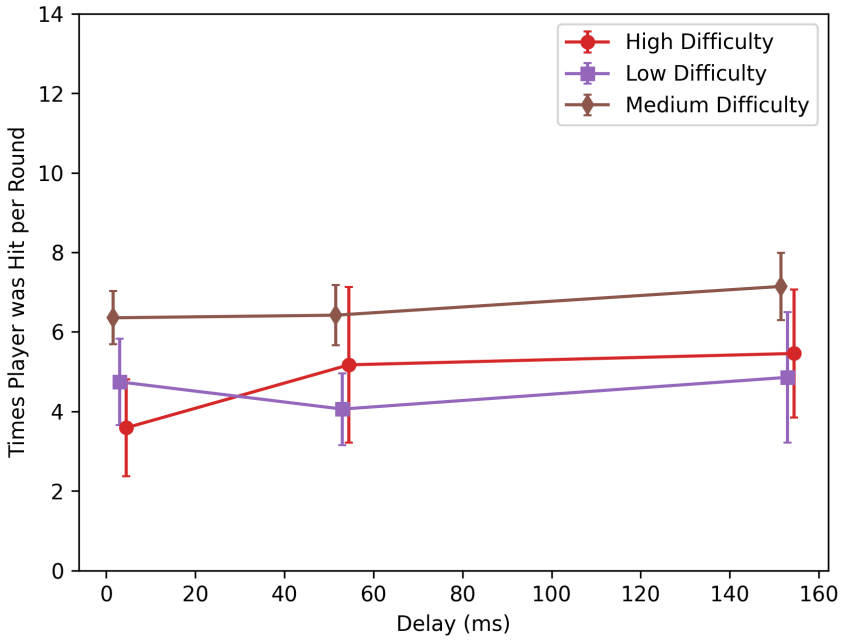


Average Score Per Round Versus Jitter Magnitude By Difficulty

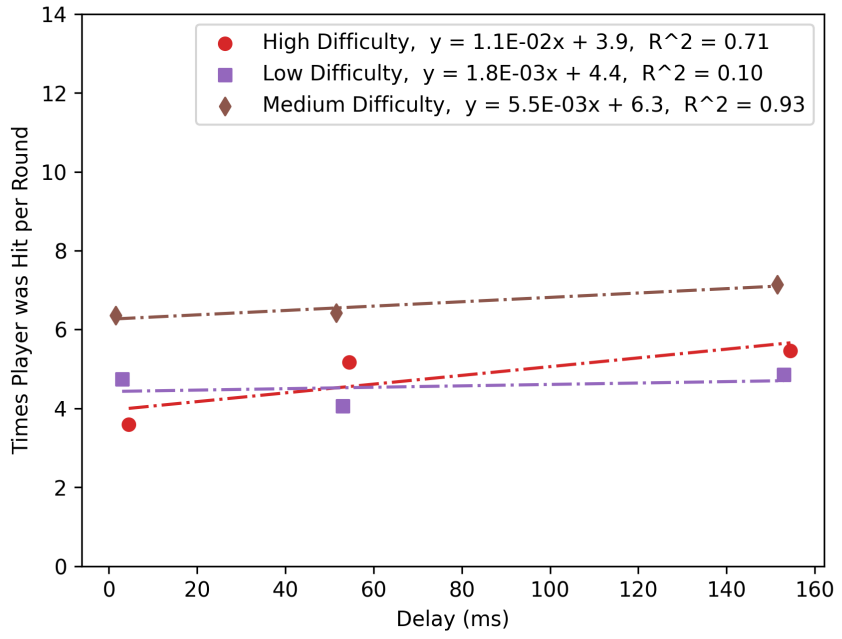


Average Score Per Round Versus Jitter Magnitude By Difficulty with Trend lines

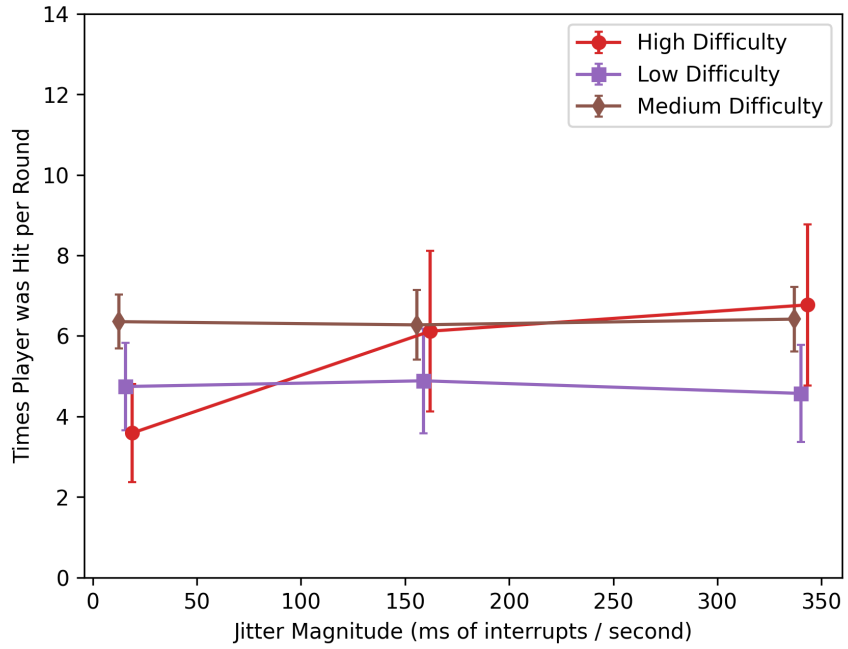
Appendix I: Damage Per Round By Difficulty



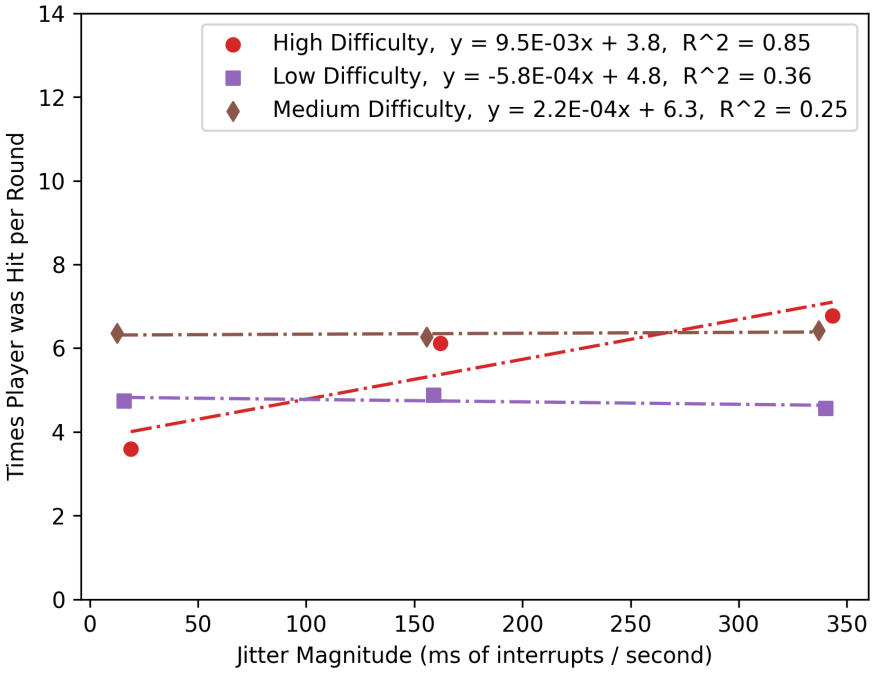
Damage Per Round Versus Delay By Difficulty



Damage Per Round Versus Delay By Difficulty with Trend Lines

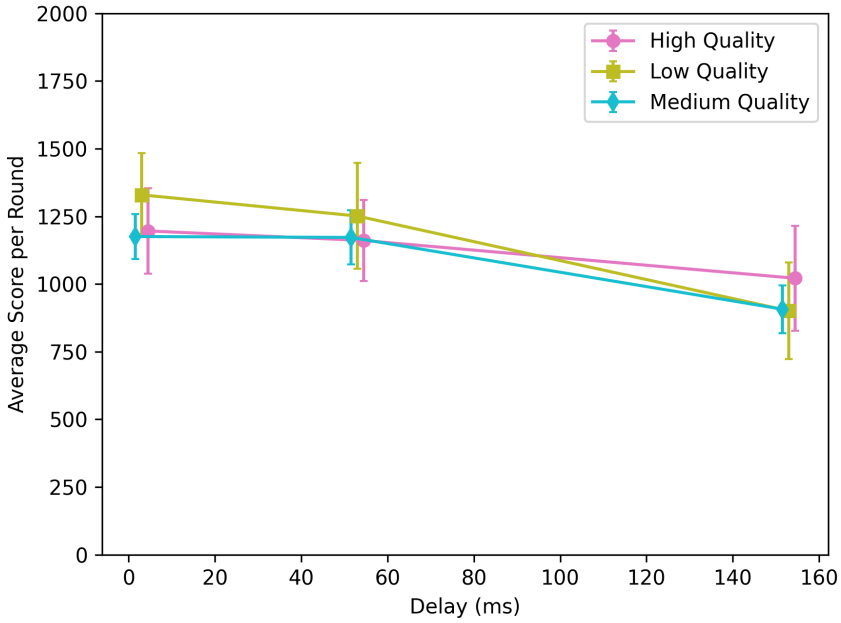


Damage Per Round Versus Jitter Magnitude By Difficulty

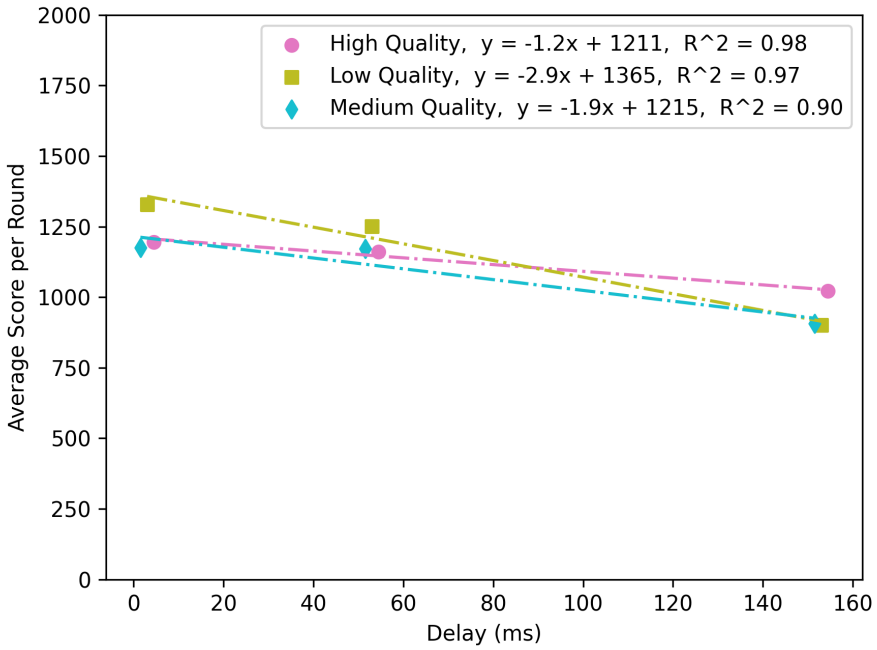


Damage Per Round Versus Jitter Magnitude By Difficulty with Trend Lines

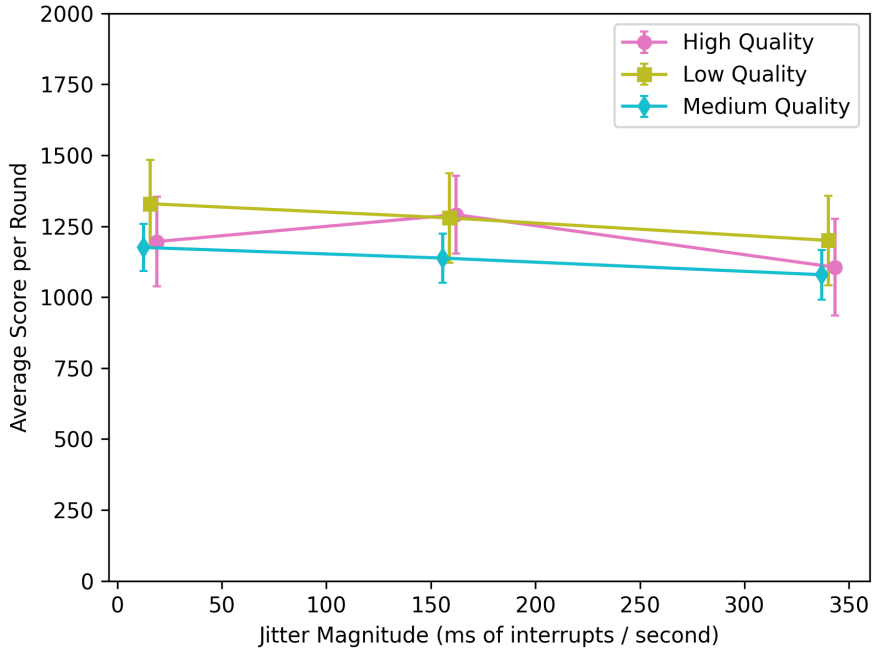
Appendix J: Average Score Per Round By Texture Quality



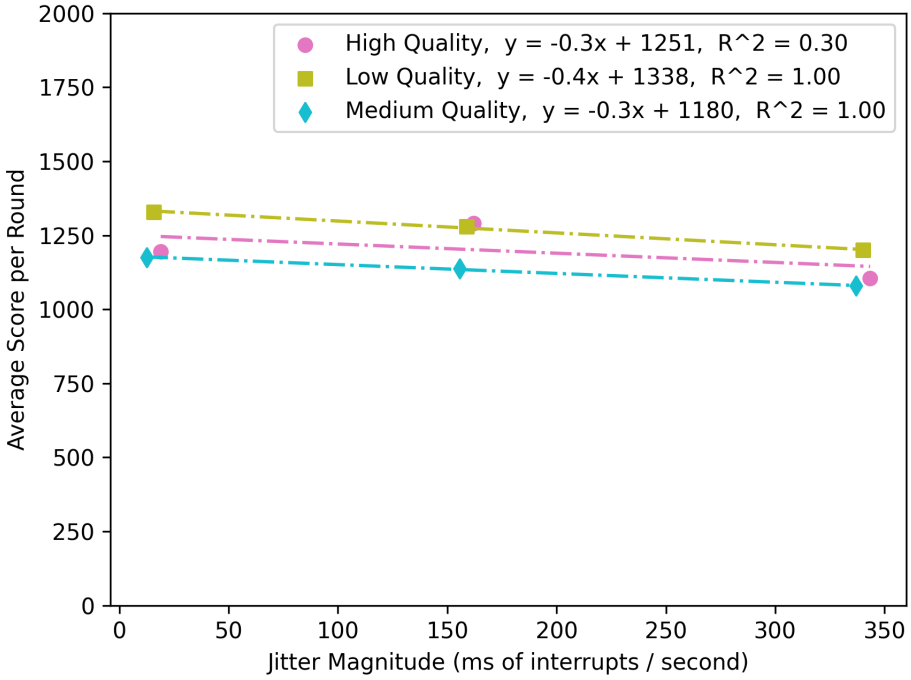
Average Score Per Round Versus Delay By Texture Quality



Average Score Per Round Versus Delay By Texture Quality with Trend Lines

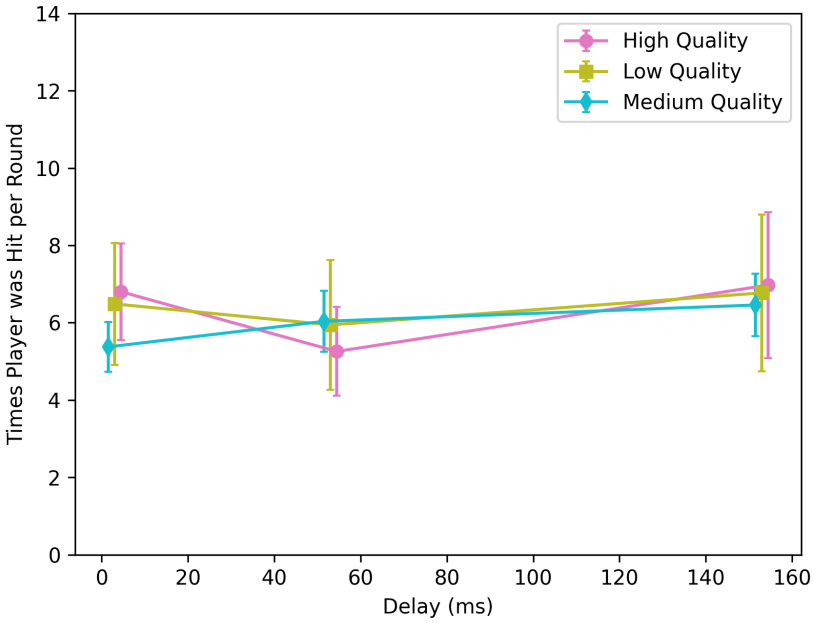


Average Score Per Round Versus Jitter Magnitude By Texture Quality

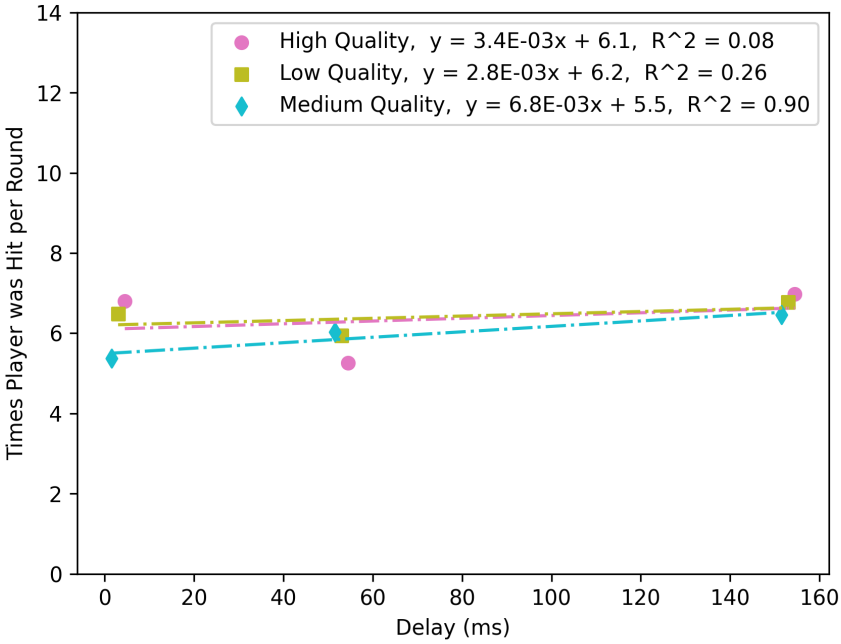


Average Score Per Round Versus Jitter Magnitude By Texture Quality with Trend Lines

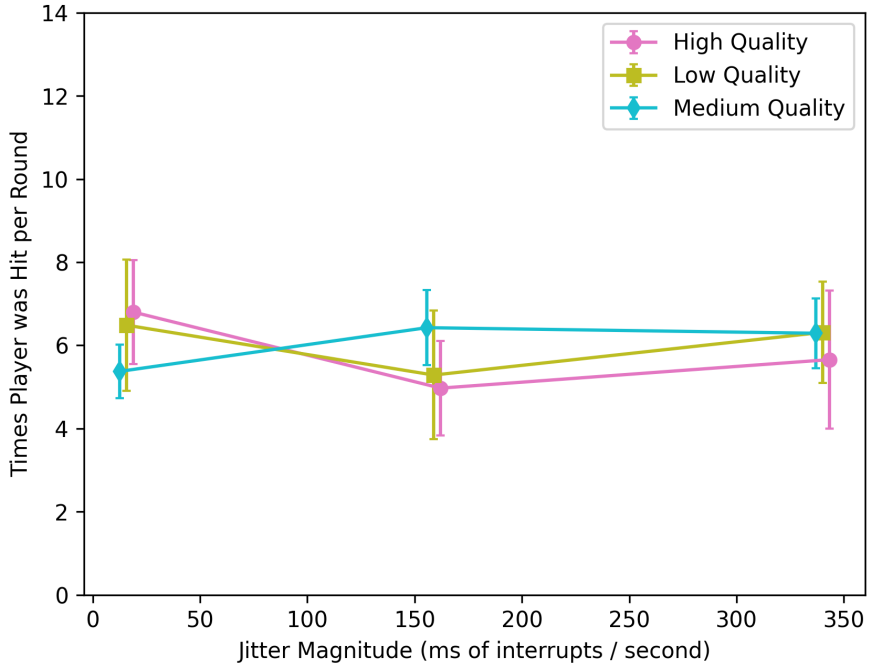
Appendix K: Damage Per Round By Texture Quality



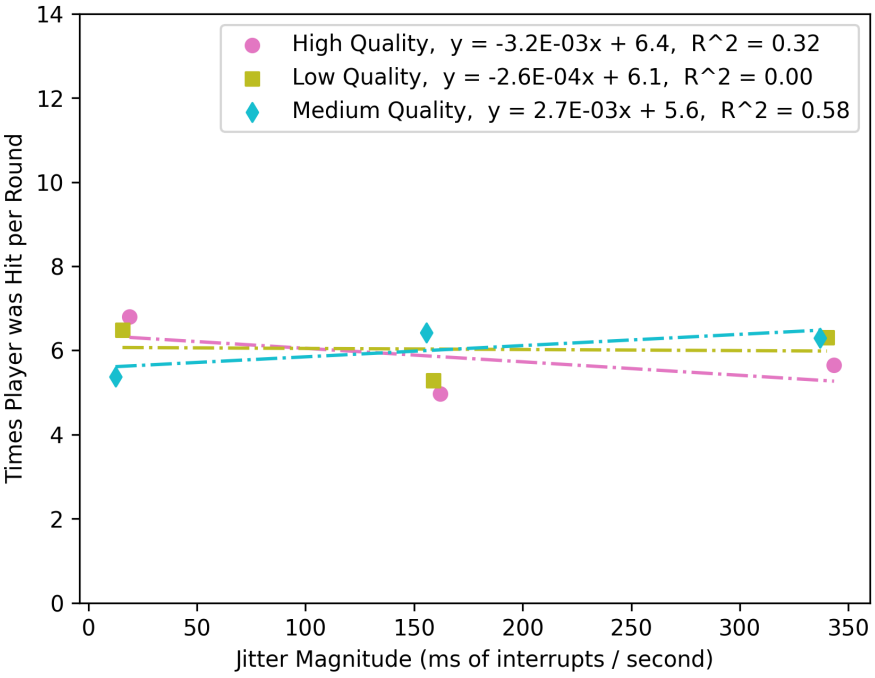
Damage Per Round Versus Delay By Texture Quality



Damage Per Round Versus Delay By Texture Quality with Trend Lines



Damage Per Round Versus Jitter Magnitude By Texture Quality



Damage Per Round Versus Jitter Magnitude By Texture Quality with Trend Lines

Appendix L: All Trend Line Equations

Metric	Network Condition	Game Parameter	Trendline Equation	R^2
QoE	Delay	Perspective	y = -0.009495x + 4.06	0.992
QoE	Delay	Perspective	y = -0.004449x + 2.92	0.98
QoE	Delay	Perspective	y = -0.007314x + 3.91	0.973
QoE	Delay	Texture Quality	y = -0.007673x + 3.88	0.973
QoE	Delay	Texture Quality	y = -0.007551x + 3.95	0.956
QoE	Delay	Texture Quality	y = -0.007050x + 3.74	0.991
QoE	Delay	Difficulty	y = -0.007777x + 3.97	0.986
QoE	Delay	Difficulty	y = -0.006922x + 3.93	0.998
QoE	Delay	Difficulty	y = -0.007163x + 3.73	0.981
QoE	Jitter	Perspective	y = -0.005301x + 4.10	0.998
QoE	Jitter	Perspective	y = -0.002573x + 3.01	0.988
QoE	Jitter	Perspective	y = -0.004786x + 4.00	0.958
QoE	Jitter	Texture Quality	y = -0.004108x + 3.94	0.96
QoE	Jitter	Texture Quality	y = -0.004634x + 3.99	0.971
QoE	Jitter	Texture Quality	y = -0.004609x + 3.83	0.972
QoE	Jitter	Difficulty	y = -0.005286x + 4.14	0.927
QoE	Jitter	Difficulty	y = -0.005265x + 4.07	0.964
QoE	Jitter	Difficulty	y = -0.004251x + 3.78	0.979
Round Score	Delay	Perspective	y = -2.179460x + 1412.22	0.964
Round Score	Delay	Perspective	y = -1.094694x + 581.41	0.985
Round Score	Delay	Perspective	y = -2.086434x + 1334.37	0.861
Round Score	Delay	Texture Quality	y = -1.198281x + 1211.30	0.98
Round Score	Delay	Texture Quality	y = -2.949307x + 1365.98	0.974
Round Score	Delay	Texture Quality	y = -1.912434x + 1215.23	0.899
Round Score	Delay	Difficulty	y = -1.874459x + 1202.29	0.967
Round Score	Delay	Difficulty	y = -1.949223x + 1492.36	0.517

Round Score	Delay	Difficulty	y = -1.976779x + 1192.48	0.975
Round Score	Jitter	Perspective	y = -0.108232x + 1385.30	0.063
Round Score	Jitter	Perspective	y = -0.228129x + 591.60	0.644
Round Score	Jitter	Perspective	y = -0.433759x + 1305.38	0.876
Round Score	Jitter	Texture Quality	y = -0.311567x + 1251.90	0.298
Round Score	Jitter	Texture Quality	y = -0.400632x + 1338.35	0.996
Round Score	Jitter	Texture Quality	y = -0.297856x + 1180.91	0.997
Round Score	Jitter	Difficulty	y = -0.594688x + 1186.06	1
Round Score	Jitter	Difficulty	y = -0.567610x + 1414.64	0.817
Round Score	Jitter	Difficulty	y = -0.204810x + 1180.06	0.923
Damage Taken	Delay	Perspective	y = 0.005311x + 5.33	0.748
Damage Taken	Delay	Perspective	y = 0.008449x + 8.27	0.772
Damage Taken	Delay	Perspective	y = 0.005299x + 5.23	0.797
Damage Taken	Delay	Texture Quality	y = 0.003429x + 6.10	0.077
Damage Taken	Delay	Texture Quality	y = 0.002816x + 6.20	0.261
Damage Taken	Delay	Texture Quality	y = 0.006795x + 5.49	0.903
Damage Taken	Delay	Difficulty	y = 0.011088x + 3.95	0.708
Damage Taken	Delay	Difficulty	y = 0.001794x + 4.43	0.101
Damage Taken	Delay	Difficulty	y = 0.005539x + 6.26	0.934
Damage Taken	Jitter	Perspective	y = 0.002043x + 4.83	0.338
Damage Taken	Jitter	Perspective	y = -0.000866x + 8.83	0.013
Damage Taken	Jitter	Perspective	y = 0.001747x + 5.33	0.949
Damage Taken	Jitter	Texture Quality	y = -0.003204x + 6.37	0.318
Damage Taken	Jitter	Texture Quality	y = -0.000258x + 6.07	0.004
Damage Taken	Jitter	Texture Quality	y = 0.002674x + 5.58	0.577
Damage Taken	Jitter	Difficulty	y = 0.009541x + 3.82	0.851
Damage Taken	Jitter	Difficulty	y = -0.000581x + 4.83	0.36
Damage Taken	Jitter	Difficulty	y = 0.000220x + 6.31	0.249