Network Assignment Report

## Preface

As a preface, I must mention that I could not be present for the entire first week of the course. I was away travelling with my family. As such, there are parts of the assignment that I believe I could have done better, polished more, or even features that I wish I had included. Additionally, I could not be present for the entire week, effectively giving me 3-4 days to both watch the lectures and to complete the assignment. Nevertheless, I believe I have completed all of the requirements.

## Design Overview

For this game, I wanted to create some kind of twin stick PvP shooter. However, I wanted some form of intriguing shooting mechanic. I decided on a “charge up” sort of attack, where the player is to hold the fire key down, and if it reaches a designer-defined threshold, a projectile fires. If the player charges for longer, the projectile will be faster, and deal more damage. Additionally, I added a recoil mechanic, both for the firing player, and the player hit by the projectile. My original idea was to add additional obstacles that would damage the player, such that the players would have to be very careful about their positioning. With such a mechanic, every offensive move might force you into defense if you are not careful. For example, if you are close to some spikes when you fire a projectile directly away from the spikes, you would be thrown into them, and lose health. In fact, my initial idea was for the projectiles to not deal any damage, and for the game to center around this mechanic a lot. But alas, I did not have time for that.

## Network Feature Implementation

Naturally, the network features center around the Network Manager. I did not inherit from the NGO Network Manager and instead used the base one, because I did not think that I needed to. On the same object I have a Game Manager, which is a MonoBehavior and not a NetworkBehavior. From this Game Manager, I displayed the placeholder GUI for starting and joining sessions. Additionally, the Game Manager kept a reference to the win screen, allowing the players to easily grab it and display the results given to them by the server.

#### Player

For the players, input was registered every frame. Then, the input was sent to the server in the physics thread, where it translated into the actual action that the player was trying to perform. The actions checked by the server were moving, looking, and firing. For firing, the process was the following: After charging is completed, and the client detects that the charge action has stopped, the client requests the fire action as a server-side Remote Procedure Call (RPC). This RPC simply executes the function Fire(), which spawns the bullet prefab, initializes the bullet with information about the charge on release, and applies a negative recoil to the player, sending them hurling backwards. The server also makes sure that the bullet’s NetworkObject component calls its Spawn() function.

#### Bullet, Damage, and Dying

The bullet essentially does two things. It travels in the direction defined by the player, and it tries to detect a collision with any player that isn’t the instigator. If it hits a player, it asks the player to damage itself, and then it sends a signal to the server that it should be destroyed.

When the player receives the signal that it should take damage, it applies the damage to the replicated variable Health. If the Value of this variable subceeds zero, the server is asked to handle the game’s end state. The dying client also sends its ClientId here in order to keep track of who actually died. The server then simply sends this signal to each client with the same ClientId passed along, and the clients are allowed to display the win screen according to whether the loser’s Id matches theirs.

#### Chat

When a player hits return, the input field is read, and the string is sent to the Server along with the Id of the player that sent it. When the server receives it, it simply sends both of these forward to each client. If I had a global profanity filter, I would apply it at the server step. If there was a user defined one, then I would apply it client side. When the client receives the message, it adds it to the screen, prefacing the message with either “You: “ if the Id is the same as yours, or “Player “ plus the Id if it is someone else.

## Challenges

Networking is undeniably difficult. Nevertheless, the path during this assignment was less troublesome than I had initially anticipated. Granted, I compromised on a lot of features like client movement prediction and so forth, but I was nonetheless positively surprised.

One of the bigger challenges during this project was simply wrapping my head around networking. I solved it by simply thinking a lot about the systems and the most logical way to organize them.

For a more concrete example of a challenge in this project, I will mention getting the movement to feel right for the client. To make it better, I would have to implement client-side prediction, but I did not have time for that. Nevertheless, I tried to moderate the effects of the latency by tweaking my movement parameters, and choosing carefully what to allow the client to handle themselves.

## Reflection

All in all, I would say that this was a very fun learning experience, if a little bit stressful at times. In the future, I would like to build a structurally sound backend for handling all of the networking business. As fun as this was, some of my solutions felt a bit slapped together. Had I had more time, I would have rebuilt many of my systems to make them more robust, but alas.