Network Programming Assignment – Documentation/Report

Pre-Production

I started the assignment by thinking of what type of game I would make to utilize the network functionality required. I wanted to make something that wasn’t too hard but at the same time I wanted to practice making player movement. I considered making it a simple FPS, TPS or a platformer. I originally decided to go with FPS as I wanted to experiment with having the players be represented differently based on client. However, due to illness and reduced time to work on the assignment I changed the type of game to a simple top-down shooter in 2d instead.

The game itself is very simple. Two players on one screen, they can shoot at each other. If a player is hit by the other player’s projectile they will lose 1 health. When one player has lost all its health the other player wins.

Reflections/Challenges

I found Network for gameobject to be prety comfortable to work with, it took some time intuitively understand what had to be a network variable and what had to be handled by the server. However, most times when trying to play the game these issues would often present themselves pretty handily, such as one player’s screen looking different from the other’s. I think most of my trouble honestly came from returning to Unity after both not having worked with it for many months but also spending that time with Unreal Engine.

I did find it difficult to both mentally and programmatically differentiate between the server and the client on the host. I often felt like I couldn’t separate the two and had some difficulty making it make sense for myself. To make it easier to understand for myself, I put all of the server related things that used deltaTime inside of FixedUpdate with the conditon IsServer.

I didn’t really have an idea for how to implement the chat functions in way it is used traditionally with a chatbox and a list of previous chat messages. So I opted to make it akin to quick chat with preset messages that appears on screen for a certain amount of time.

Devlog

I started off by creating and importing all the required packages. I have not used the new input system in Unity yet so I’m taking this chance to familiarise myself with it. I created the player sprite and added an input mapping and input actions for movement and shooting. I wanted the player to rotate towards the mouse to aim where to shoot but decided to hold off on it for the time being.

Next, I added a NetworkManager to the scene and added the network components from the unity package to it. I set up the player prefab to spawn when the client started and tried starting two versions of the game, one build and one in editor. I got the two players to spawn in, but they were still sharing input. I fixed this by having the input vector modify a network variable that only had writing permissions from the owner. I later realised that this was an unreliable method of communicating input as it let the client modify the value directly. This was fixed by making the input vector an argument in a method that used RPC to communicate the input value to the server. The value was then sanitized by the server by normalizing it, meaning that even if a massive vector was sent it would not cause the player to move faster.

When I was going to implement the spawning of the projectile, I considered whether to add aiming before or after I had managed to get the projectile to spawn on the server. I decided to once again to hold off on implementing aiming, my goal at that moment was to get something to spawn for both players.

I managed to get the projectile to spawn on the server and could see it from both client's perspective regardless of if the host or the joining player spawned it. I did it by instantiating with Instantiate and then grabbing the object and spawning it with Spawn in an RPC to the server.

This felt like the correct point to implement aiming at. I did it by getting the mouse position and setting a vector from the player to that location in WorldSpace. I moved the projectile using the same method as the player, by modifying the position with the velocity vector on the server.

I wanted the players to have a fire rate but at first only the host was bound by it, the joining player could fire as quickly as they could press the button. I ended up first removing the fire rate limit after realising that it was more fun to fire as quickly as you could press the button. However, I eventually realised that this meant the fire rate was not handled on the server. Meaning that if a players input was 1 million clicks a second the server would accept and spawn that many objects. I went back and changed this by adding fire rate network variables that were modified by the server to stop the player from spawning too many projectiles.

Originally, I used normal box colliders on the projectiles, but it caused the player that was hit to move slightly from the impact, so I changed the projectile to be a trigger instead. When I continued to test the trigger event, I noticed that the joining player’s projectile no longer functioned. Since it was always the joining player that had this issue, I correctly guess that it had to do with something not getting sent to the server. The issue was the mouse position used for the velocity of the projectile was not the network variable I had created, thus causing the joining players projectile velocity to always be 0.

I split the collision logic between the player and the game manager. The player that recieves a trigger event checks if the tag on the trigger is correct and if the clientId that spawned the projectile is different from its own. If both are true, the player fires a UnityEvent with its OwnerClientId passed along. The GameManager is subscribed to this event and uses the OwnerClientId to determine which player was hit. It then subtracts one health from the player health network variable. This subtraction is subscribed to with OnValueChanged to update the UI for both players.

The chat system I wanted to implement was a system that used pre-set messages like you would see in online games like Rocket League. I made a list of preset messages and linked them to the 1, 2 and 3 on the number row. I set up network variables for each of the players relating to what message to show on what UI element and how long it should be displayed for. These inputs were sent to the server in the same way the movement was and then the server communicated to the UI elements on both clients with what to change them to.

For the winner screen I made a network variable in the game manager that would keep track of if one player’s health had reached 0. If that happened, all input from both players gets disabled and both players get a message on screen with the winning player, this was implemented in the same way as the quick chat messages.