Network Assignment Report

Overview of game design

I created a basic two-player-shooter game where you move your character with WASD and shoot with the left mouse button. The projectile is instantiated in the direction of the mouse on your screen. If you shoot your opponent, they lose health, and you eventually score a point. The health is displayed on the players and the scores are in the background of the map.

Implementation details of the network features

Basically, all inputs are checked locally, and then the server decides if something should happen. So, when I want to walk forward, I locally look for the input of W. Then I send that information to the server, and requests to have it moved. Depending on the server's logic, I will be granted to move or not. For example, when someone scores a point in my game, the server decides that movement is not allowed.

The chat system is implemented by locally looking for inputs 1-4 on the keyboard. If one of the buttons is pressed, a prewritten text will be sent to a server function. The reason the server should get the message before everyone else is because it will decide how long the message will be visible for everyone. The server will then distribute the message to everyone by calling yet another function that changes the text above the interacting player. To make sure that the message shows on top of the player that sent the message, and not any other, there is a check to see if the client id is identical to the one sending the message.

Both the chat- and health systems are implemented in very similar ways, with the main difference being that the triggering event of health change is on the server instead of locally. Both the health and death count (scores) are made network variables, to prevent cheating and so that both players can have individual stats and don't change the same values. In the server function, that resets health and adds a death count, the game manager's function is called to update the scores for everyone.

When the local player left clicks, it will locally find the mouse position on the screen. It's then calculated what the correct direction and rotation would be for a projectile if it was to instantiate, and then saves those values in local variables. This information is sent to the server via a function. In this server function, the local values are saved in network variables, which will then be

normalized. If the countdown isn't finished or a score just happened, it won't be able to shoot. If the countdown allows it, the projectiles will be instantiated and then receive velocity on the server. This makes sure that everything will look the same for all players. The projectiles will only be destroyed by the server, either when it collides with any object, or after three seconds have passed.

Challenges faced and solutions implemented

When creating the game, I first wanted it to host several players, but as time went on, I decided to only focus on two players because of the time limit. It would take some extra time to implement a new scoring system suited for more than two players, so I decided to continue with the one I already had in place. This is why some of the code is not coherent with the rest but can be built upon to easily add more players.

Some problems that I had weren't even network related, even though I thought so in the beginning. First, I wanted the score text to follow the camera, though it didn't. Second, I wanted the chat bubbles to follow the players, though it didn't. It turned out that the code was correct, but the use of canvas was incorrect. When a text is a child of a canvas, it's going to follow the camera. When it's a standalone GameObject, it's in world space. I had unfortunately used the canvas wrongly, and it took an unnecessary amount of time before I realized the problem.

One tricky thing to implement was the shooting direction. I first wanted to rotate the player and have the projectile instantiation in the direction of the player, but the camera kept spinning around. I think it was because in the hierarchy, it's locally placed on the player prefab. This would have it rotate together with the player. I ended up leaving the player's rotation at 0 and only control the projectile's rotation & direction.

When deciding the colors of the players, I now use OwnerClientId, which would have been useful for deciding the playerNumber as well. I had previously used NetworkObjectId and got seemingly random ulongs, so I decided to create my own way of tracking the players (playerNumber). Now I realized I wouldn't have needed that. At the moment, if more than two players join, they will still be in either team one or two, but with other colors.

A reflection on the learning experience

Though I had it in the back of my head throughout the making of this game, I still seemed to miss the importance of implementing inputs locally and logic on the server. It was harder to keep track of than expected, especially when doing the logic for projectile instantiation. I think the next network game I make is going to

be easier, as I probably already made all the mistakes I can. Though, I haven't learned how to optimize latency, I look forward to learning more about it in the future.

It wouldn't have hurt to have more lectures with in-depth examples. It feels like I learned one way to do everything, while I'm sure there are a hundred different ways.