Player movement

For the player movement, I created a client authoritative movement component where the client updates the position of the player object and then shares it with all the other clients. For this, I used the NetworkTranform component, and set the authority mode to owner, and created a new script to read the player's input and update its character's position. The script also checks if the current machine is the owner, and disables itself if it is not. This is to prevent the player character from getting input from other clients. I also decided to use FixedUpdate instead of Update, so that the frame rate will not affect the movement speed.

Shooting

The shooting and the update of the bullet is mainly down on the server side. When the player shoots, the client sends a server RPC with the direction it wants to shoot. Server checks if the requesting player is still in cooldown, and if not, spawns a new bullet and sets its direction. Although the cooldown check and the spawning of the bullet is done on the server side, the direction of the bullet is done on the client.

The update of the bullet is done completely on the server side. This is to prevent clients from cheating, for instance setting the bullet position in front of other players, or make the bullet faster. The bullet movement script is a NetworkBehaviour, and is disabled if it is not on the host machine.

Collision

Collision is a server authoritative script. All collisions are done locally on the server, and then propagate the changes to all the clients. At first, I did the same check as all of the other server authoritative script does, if the script is not on the host machine, disable the script. But a while into development, I found out that despite it being disabled, the collision check is still being executed. After reading a little bit online, I found out that disabling the script only disables Start(), Update() FixedUpdate, etc. So to disable the collision, I had to check if the script is on host inside OnCollisionEnter2d.

Game Manager

The Game Manager is a server authoritative object. It is a singleton, since in any scenario, I only need one to exist, and many other objects need to access it. The game manager keeps track of the score for each player, and the state of the round, and the respawning of the player.

Game Manager stores a dictionary of client id to score, and a list of surviving players. When a player dies, it calls a function in GameManager to inform it that it has died. The GameManager then removes it from the surviving player list, and checks if only one player is alive, if so, increase that player's point by one. After that, the GameManager calls the action OnScoreChanged to inform all its subscribers to update. I used an action here to make it more robust. For instance if I were to add new mechanics to when the score changes, I don't

need to return to this script and add a function call to trigger each mechanic. To make sure that this action is triggered on both the server and all its clients, I create a RPC with the send to everyone attribute. When the score update's, the rpc is called, which then invokes the OnScoreChanged action on each client.

The GameManager is also responsible for respawning the player. When the GameManager updates the score, a timer starts to happen, after that all players are respawned. I thought of 3 methods of doing this, and tried to implement 2 of them. First I thought of despawning and respawning the player. But there's 2 problems with this. The first being that this will take more process power, since every time when it tries to respawn the player, it would need to instantiate a new player object. The second problem is that I'm not sure what is going on in the background. Since the player is created for us when we connect, there are a lot of configurations for the player object when it is created. But these configurations are not present if I spawned in an object for the client manually, which could cause some issues in the future. For this reason, I kept this as a backup and moved to a different method.

The second method involves deactivating the player and reactivating the player. When the player collides with the bullet, it sends a rpc to all its clients to deactivate the player object. And when the game restarts, the GameManager would cycle through the players and reactivate all the dead players. Here I wanted all the activation of the player object to be invoked by the GameManager, this way, it would be less likely for clients to cheat by reactivating their player object early.

But I ran into some issues when I tried to do so. Activating the player would require a reference to the GameObject, initially, I created a list of GameObject which stores a reference to all the player objects. Using the OnClientConnectedCallback I was able to add a listening function which finds the player object of the new client and adds it to the list. And with this list of player objects, I can cycle through them all, and reactivate them. This worked fine when I just had the host running, but when I added a second player to the game, the client never reactivated the player objects. The GameManager on the client side gets an error. The way I find the player objects is by searching in the dictionary ConnectedClients. This is a dictionary of ulong and NetworkClient stored in the NetworkManager. When trying to access the dictionary on the server side, everything is fine, the dictionary contains the information of all clients. But on the client side, this throws an error that ConnectedClient should only be accessed on the server.

From the problem above, I thought of the server first checking which objects need to be reactivated, store them into an array of gameobjects and send them through the rpc. Then on each client, the rpc call would cycle through the received list of gameobjects, and activate them. This causes another error, GameObject needs custom serialisation functions. I worked with RPC in Unreal Engine before, and there it is possible to send object references. So when I started creating the RPC call, I had the same assumption in mind, but this turned out to be false. Creating a custom serialisation function is not a method I can try, it would take too much time to research into it, and more time to try to implement it.

My final idea is to let the GameManager call a rpc on each player object to activate itself. Each player object contains a send to everyone rpc which activates itself locally. The server would cycle through the player object, get the component containing the rpc, and calls the

rpc for the player. This works, but proposes possible security issues. From what I've seen only send to everyone rpc have no calling restrictions, meaning everyone including clients can call it. And Since I need the GameManager to access the function, it is public as well. So in this case, it is much easier to cheat by respawning dead players with this function.