Network Assignment Report

# Game design

The game is an old school twin stick shooter. Players on mouse/keyboard have separate 8-directional control of movement and aiming, while players on controller have fine-grain analog control. Projectiles and players are physics based to allow for built-in collisions.

The server has absolute authority as the client only has authority over its inputs (which are then sanitized before being used.

When the client player is spawned it will appear as blue, while the remote players appear as red. This is to better distinguish between characters.

Chatting is quite natural, using a chat box and sending messages on “return” being pressed. This is to allow the maximum amount of communication between the players. The only thing I would have wanted to add here is multiple channels (all, team).

The things that I would have added should I have made an actual game are:

* Dynamic camera
* Procedural obstacles
* Proximity based spawning
* Pickups
* Points system

# Network features

## Chat box

A computer screen shot of text

Description automatically generated

The chat box is quite simple. The client submits a message locally which is then formatted and sent off to the server. When the server receives the message, it has the option (not implemented) to sanitize/sensor as well as filter the message if the player is chat-banned (also not implemented). If all is well, the message is sent to everyone, and the chat log is updated.

I should have made sure the formatting happens on the server since it would be easier to implement the above stated potential features if the message wasn’t merged.

## Gun

The gun is responsible for spawning projectiles at a certain rate based on player input.

A screen shot of a computer program

Description automatically generated

When the client calls ToggleFire, the action is sent to the server which allows the spawning of projectiles. The projectiles are spawned with ownership to allow for score to be added to the correct player when the projectile damages another player. This is only needed since the projectile is a separate NetworkObject than the player and has a different ID.

A computer screen with white text

Description automatically generated

### Projectile

The projectile moves at a constant speed and removes itself if it doesn’t hit anything. Collision with players is handled server-side.

A computer screen with text on it

Description automatically generated

## Player

The client-side player is responsible for controlling the player character’s movement and look direction, as well as pressing and releasing the trigger of the gun. The client doesn’t have authority over anything else. Both \_moveInput and \_lookInput are NetworkVariables with Owner write permissions

A screen shot of a computer program

Description automatically generated

The server-side player is responsible for sanitation of input and transformation. We assume the player has modified the inputs to be out of range, so the normalization only happens here. Move and Look are run server-side during FixedUpdate.

A screen shot of a computer program

Description automatically generated

### Health

Health is a component attached to the Player gameobject, but it could be attached to anything.

A computer screen shot of a program code

Description automatically generated

It is vital that we only do this on the server since clients could inject code that does 10000 damage to all enemy players each frame. The variable \_currentAmount is a NetworkVariable to allow for things like client-side healthbars. Since the scoreboard was scrapped, the sourceClientId isn’t used here, but it would be used to add score to the correct player by sending the source id to the scoreboard.

# Challenges

The one major issue I did face was my attempt at implementing a scoreboard by the end of the time given. Because Unity doesn’t have a strong framework for storing player data (as unreal does) I tried to add a bunch of separate but connected scripts that would work as modules. Since there wasn’t much time and I had to write everything from scratch, I rushed it, and it turned into true spaghetti. After spending a lot of time trying to rewrite and fix it so I wouldn’t be the only one who understood how it worked, I realized that I would have to start over by extending the NetworkManager. At this point there was really no time left and the system wasn’t in a state I could call “good enough”, so I decided to scrap it. I will likely return to it quite soon and give myself a chance to add the previously mentioned scrapped features as well.

# Reflection

I will be completely honest and say that I didn’t learn anything from the lectures I couldn’t find better information on through the Unity NGO documentation. The reason I didn’t implement certain things (like client-side prediction or proximity-based spawn points) is the lack of information about common design patterns from the course material. Instead of spending a bunch of time researching these design patterns online, I chose to (for example) upgrade to Unity’s new input system and had to rewrite a lot of client-side code.

What I did learn was that Unity NGO is enough to build a simple game, and I will likely follow the Unreal approach and build all my side-projects using it going forward. Starting the server-client and allowing for new players to join was so simple and had such little overhead that there doesn’t seem to be any need to build your own solution. I didn’t end up playing with a lot of the features of the NetworkManager and I also didn’t extend it, but there seems to be enough to be able to avoid writing a lot of manager objects, e.g. SceneManager(s). I tend to avoid building god-objects, but with networking as a central concept it does make sense.