Péter Gulyás Network programming assignment

Game design

The game is a 2D top down shooter where the players are playing as tanks and the goal is to eliminate each other. You can move with WASD, break with space and shoot with the left mouse button.

Implementation details

I started with the base that you shared with us on the second lecture.

Player script

On **Update**, I handle the movement by first setting some variables locally based on the player's input. These variables are then sent to the server using an RPC. Once received on the server, we call the **UpdateServer** function, which applies forces to the Rigidbody based on the input we fed in through the RPC.

The braking works in a similar way. If the player presses the spacebar, it changes the state of a bool variable, which is then sent to the server with an RPC. Then the server applies a force to the Rigidbody, causing the tank to slow down.

After the movement and braking logic, in **LateUpdate**, we handle the camera movement for each player locally.

The last thing in this class is the shooting. When the left mouse button is pressed, we retrieve an object from the ObjectPool, spawn it on the server, and feed necessary data such as direction, velocity, and owner.

Object Pooling

It is basically a design pattern that allows you to have a collection of reusable objects. “By pre-instantiating and reusing the instances of those objects, object pooling removes the need to create or destroy objects at runtime”[1].

We have a list of **PoolConfigObjects**, which consists of a prefab and a count indicating how many instances of this object we should spawn. We also have a dictionary where the prefabs act as keys, and the values are queues filled with the instantiated objects. When a player joins a game, we populate the dictionary with the object we assign in the editor.

[1] https://docs-multiplayer.unity3d.com/netcode/current/advanced-topics/object-pooling/

To use the available objects, we call the **GetNetworkObject()** function. This function calls an internal method that checks if the queue has any available objects. If it does, after setting some parameters on the GameObject, it returns it. If it doesn't, we create a new one, set the parameters, and return the newly created object.

With this pattern, we can improve performance and decrease the load on the garbage collector. In the context of a networked game we can even decrease potential latency.

Bullet

This is a fairly simple script. It primarily handles setting the bullet's velocity, checking for collisions, and managing the visuals. If the bullet collides with something—and since the collision checks happen on the server—we send an RPC to inform the clients that they should play the explosion effect. Additionally, the script manages the bullet's lifetime with a timer, resets the bullet's state after it is destroyed.

Attribute Component

This is another straightforward class. It features a **NetworkVariable** for health. During initialization, we subscribe to the **OnValueChanged** event of this variable, which will trigger an update to the health bar whenever the value changes. Since only the server has the authority to modify this variable, the update occurs on the server side. So the health bar update is performed via an RPC that is sent to both clients and the host.

Chat

Lastly, I have implemented the chat system based on the example shown during our second lecture, with a slight modification. The implementation now includes an **InputField** for players to enter their messages and a **ScrollView** to display the message history.

Reflection

I did not face any specific challenges during development but rather struggled with understanding the overall concept of networking. But, after going through some sample projects, I managed to get a better grasp of the topic. I learned about object pooling, which seems like a strong concept even for non-networked games. Overall, I feel more confident in my networking skills now and believe I have managed to build a strong foundation to further develop in the future.