Victor Haskins

Game Programming II

Lab 05 Report – UNet Overview

Introduction

We took the time on this lab to watch and follow the instructions of Youtuber *Gamer to Game Developer* over a two part video where he explains the basics behind the new networking suite UNet. He explains the steps to quickly connect sessions over a localhost. He then removes some components so he can show us a smoother, easier, and more robust version of what was already allowed.

Methods and Work

He starts off by having us use the standard assets package from the unity store and the base engine to create a few core components to manipulate: a plane for traversal, two cubes of differing size, and a first person controller named player with an added capsule for visual. He added a quick change to the plane by adding a simple gray material and then set all but the player in an empty game object to remove clutter from the Hierarchy. He also added a modified cube to act as a pointer and fake gun for the player.

Now we get to the meat of the issue. He created a blank object called NetworkManager and searched for a Network Manager component from the Add component tab. He then used the same technique to add a Network Manager HUD.

We make a prefab of the player and then add that prefab to the “Player Prefab” option of the Network Manager component so it is added when the person playing is added to the scene. He then adds two new components to the player: the Network Identity and Network Transform scripts. He activates the Local Player Authority node of the Network Transform to have the active player of the pc use it and sets the Transform Sync Mode to “Sync Transform” on the Network Transform instead of the default so the players transform is communicated.

At this point, we can test to see if the host works. Opening the project in Unity brings up the project with a player prefab to move around and if we build the project, placing it in its own folder for better organization, and run the two simultaneously, we notice that the client information is jittery and no matter which window is used, both respond to the controls. In response to this, he disables controls, cameras, and audio listeners,(because that also threw out an error,) on the player prefab to be activated by individual players. This is easier than having to deactivate other objects as they appear.

After we make our first build—adding both the offline menu scene and online main scene—we drag the menu and main scenes into the offline and online scenes respectively. This switches between the two scenes depending on the online status.

Now we create a script called Player\_NetworkSetup where we can activate the player's controls, camera and listener, while deactivating the main camera of the scene. We need to add the library “UnityEngine.Networking” so we can change this script from Monobehaviour to NetworkBehaviour. This allows the script to affect the server and either this or other clients' interfaces.

Using a network boolean “isLocalPlayer”, we disable the main camera of the scene, using the line “GameObject.Find(“SceneCamera”).SetActive(false);”. Next we call GetComponent functions to enable the Character Controller and First Person Controller. He switches tactics when activating the player's camera and audio listener to create serialize fields and enabling them. He then attaches the script, drags the player into the new fields made. Running two instances now shows that the players are individually controlled, but the client still has jerky motions on the host game.

To combat this, we remove the Network Transform from the player prefab and create a new script called Player\_SyncPosition, setting this file up for networking like the last one. We hold a Vector 3 to hold the sync Position, and two serialize fields to hold the player's transform and lerp rate (as a float). We then create a new function LerpPos() where if the object is not the local player, we Lerp the player from the current location to the sync position it should be at. We then create a Command function called CmdProvidePositionToServer that will set the sync Position to the player's true location. For this to work, the server needs to be fed this sync Position variable, so it is preceded by the segment “[SyncVar]”. Next, we offer a ClientCallback function called TransmitPosition() to have the prefab regularly update the sync Position by calling the command created earlier. Finally, we create a function called FixedUpdate() that will replace the normal Update() function. It will simply call LerpPosition and the new Transmit Position.

Starting is video 2, we modify the player prefab a little to demonstrate the when the camera moves, move the network manager from the Main scene to the menu scene, which should be placed first in the build. Only one of those is needed for the whole game. We then move onto the final script for the game called Sync\_PlayerRotation, where we, not surprisingly, update the server to the player's rotation. This works almost identically to the previous script and is indeed set up that way. We just have an extra Serialize Field because we want to rotate both the player and the player's camera and we must do both. The big difference now is that we are working with Quaternions instead of Vectors.

Conclusion

While I was told that this is not nearly as effective as learning networking from the older material, I found it immensely helpful. I have never worked with networking outside of linux command lines and actually laying line between computers. It is nice to see the process laid out for me and will be useful in my Agile class in just a few weeks.

Report videos can be found at the following:

<https://www.youtube.com/watch?v=NLnzlwCRjgc>

<https://www.youtube.com/watch?v=PVUT6SR70wg>