Lab Report 08
Low-Level API Networking
Darrick Hilburn

Introduction

The last few labs have focused on using Unity's built-in UNET for networking and creating a persistent Network Manager object. This methodology focuses on higher-level API, which sits more at the top layers of the TCP/IP Networking Model. Not all networking follows the standards of Unity's built-in methods, however, so we must also know how to program at the lower levels of the Networking model. This lab focuses on using sockets, which are generally used more within the second and third levels of the TCP/IP model.

Methods

Two scripts are built for this lab: Server Connection and Client Connection. These scripts are attached to two Camera objects in one scene. There is a second scene with only a cube in it that the player can transition to if they press the space bar.

The Server script begins by setting up Global Configurations for the Network, namely setting up a Reactor Model and a Thread Timeout. The Reactor Model is set to Fixed Rate, so that it handles packets periodically, and the Thread Timeout is set to 10ms, meaning the packets are handled every 10ms. With the global configurations set up, the channels are added to the server script through Connection Configurations. We set up both reliable and unreliable channels for TCP and UDP respectively. Next is the host topology, which combines the channels with the maximum connections allowed on the server. With all this preliminary information set up, we can finally initialize the Network through Network Initialize and by setting up a server socket ID through this initialization by adding a host.

With the server network up and running, we can check every few frames if an event has occurred and what to do with each event. The events we are interested in are if nothing happens, if a client connect or disconnects, and if a client sends data through the server; we are especially curious about the last option. Should a data message be sent, we want to read this data, then respond to this data. To do this, a memory stream is created based on a buffer initialized at the start of this update, a binary formatter is created, and the message in the buffer is deserialized from the memory stream via the formatter. The deserialized message is sent through a function call to respond to the message to determine what is done.

The client side is near identical to the server side, except the client has a function call to connect to the server in its startup, and its update only checks if the player has pressed either the space bar or 'R' button. This information is used to send messages to the server. The client also uses checks for if they're connected to the server when updating to determine if they received a server message, and will update if they're connected or not should they perform either action.

Conclusion

I have learned how to open sockets for networking and using them to send binary information over a network. This is extremely useful because it is a universal networking method unlike Unity's Network Manager methods, meaning it can be utilized in other places outside of Unity and can be expanded upon greatly.