Engine System Update #1

Project - GAMES6320

The first week of creating the networking system for my project involved plenty of research and study about game networking, where I learned a lot about the fundamentals of such a system, and things to keep in mind when making decisions going forward. This knowledge helped me better understand how to structure my engine system and what features to develop in accordance with my final game.

Additionally, I also got started on setting up the networking project in the Visual Studio solution and wrote the code for the initial representation of a game server, including both platform-dependent code and a platform-independent interface.

UDP vs TCP Protocols for Game Networking

One of the first things I learned from my initial research was the difference between the two types of transfer protocols: User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). Following are the advantages and disadvantages of both:

UDP:

Advantages:

- Generally faster and more straightforward than TCP
- Good for low latency and can tolerate some packet loss

Disadvantages:

- Much less reliable than TCP
- Cannot sequence or arrange data

TCP:

Advantages:

- More reliable and ordered, guarantee that data will be delivered
- Can retransmit data if packets fail to arrive

Disadvantages:

- Slower than UDP
- Requires an established connection before sending data

Looking at the above points and summarizing the two transfer protocols, I decided to implement a UDP approach for my game networking system. This is because UDP is faster and much simpler to design than TCP. I also believe this choice largely depends on the type of game you're making. For example, UDP is a much better approach for fighting or shooting games, where we want fast transmission of data with low latency. On the other hand, something like TCGs or turn-based games would be better suited for a TCP approach, where the order of the data matters more than speed. So, I really had to think forward about what kind of game I wanted to make for my final project, and also what would be most useful for my fellow classmates if they were to use my system. In the end, I decided to go forward with a UDP implementation, since I believe most people (including me), would want to create games that are fast-paced.

Game Server Implementation

During this first week, I also got started with an initial implementation of a game server, separating out the platform-dependent Windows code with a platform-independent interface. Right now, the goal is to make sure that player data is being transmitted correctly between server and client, and between clients. For this, I made a simple Player struct that contains playerId/clientId, a position vector, and a bool isAlive variable.

The GameServer class currently has interface functions for initializing and cleaning up the server, reading data, and sending the data to all clients. These functions are then implemented in GameServer.win.cpp by creating and binding Windows sockets to connect the players/clients to specific ports. At first, I was a little confused about how to structure my code and functions. But, I looked back on the existing projects that are in the engine already (like Concurrency), and this gave me a good idea of how to refactor my networking code.

My plans going into the next week are to finish the client-side implementation of the UDP model, and make sure that the data in the player struct is being transmitted correctly across multiple clients.