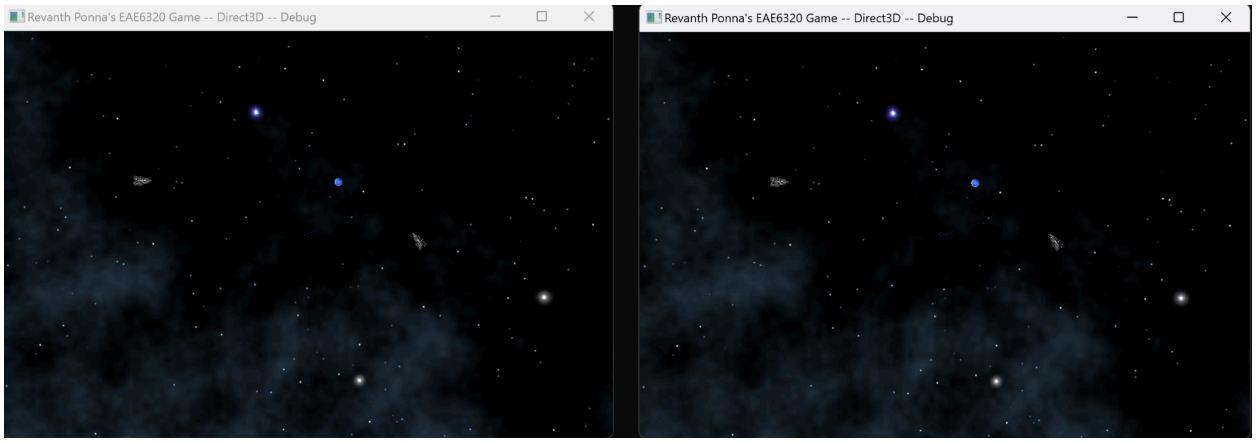
Final Project Write-Up

For my final game project, I created a top-down multiplayer space shooting game using my Networking system and my classmate Yunhao Ye's Physics Collider system. My primary goal for the final game project was to showcase the work I did for my Networking system and to visually represent information being passed across a client-server model. Additionally, I chose to use the Physics Collider system to detect collisions at key events in this interactive multiplayer space shooting game.

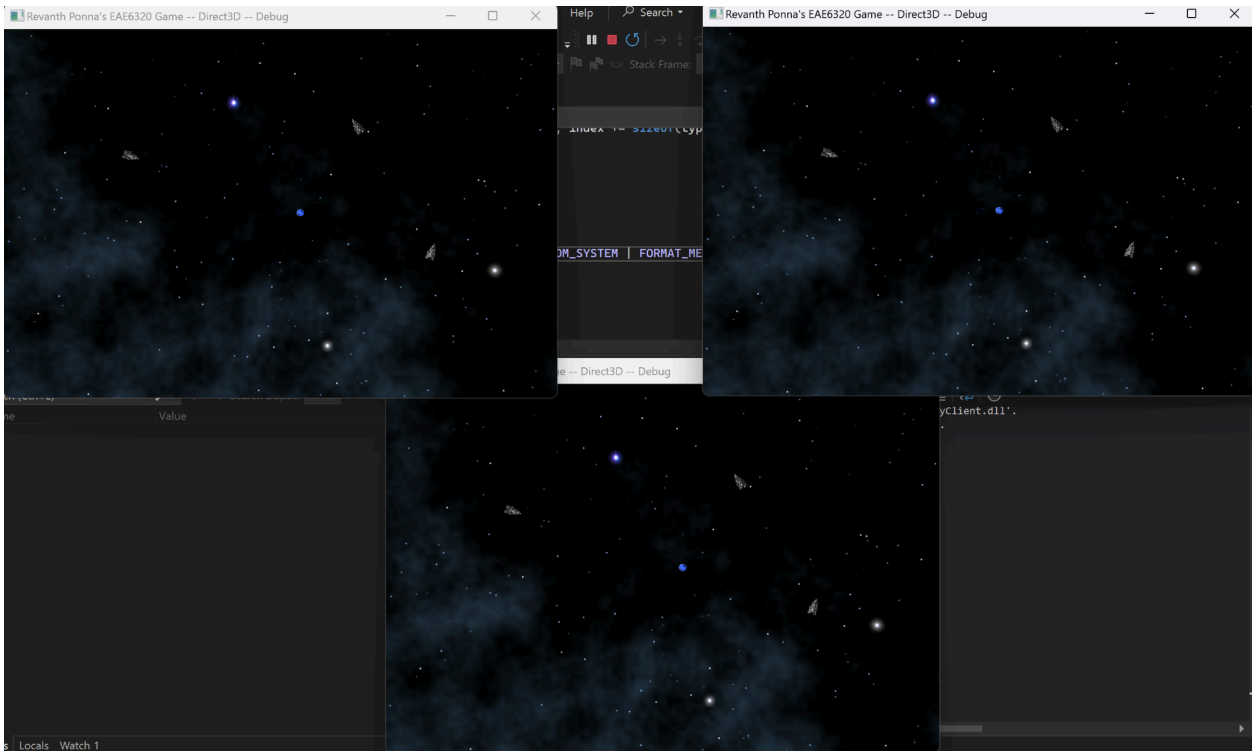


The original proposal for this final project was a little different from how it turned out. Initially, the plan was to showcase the networking system's ability to update player positions in real time, and I was hoping this would lead to funny, interesting movements and collisions in the game. However, after integrating the networking system into the game, I realized that it would be more fun and interactive if the players could do something else other than just move. So, I added the implementation of projectiles where players can shoot each other using the Space Bar and used the Physics Collider system to check for collisions between the spaceships and projectiles.

Using the Networking System

Some of the first things that I had to work on for this project was to integrate my networking system with the game itself, to visually display information being passed between the server and clients. I was honestly glad that I went with a minimalistic design for the networking system, where each client/game

instance only had to call three functions: Initialize, Update, and CleanUp. This approach made things easy and understandable for me to update information in the game.



When I originally finished implementing my networking system three weeks ago, I only put debug messages to show the flow of information between the server and clients. However, for the final game, I had to dig deeper and actually update the data that was being rendered by the Graphics system each frame.

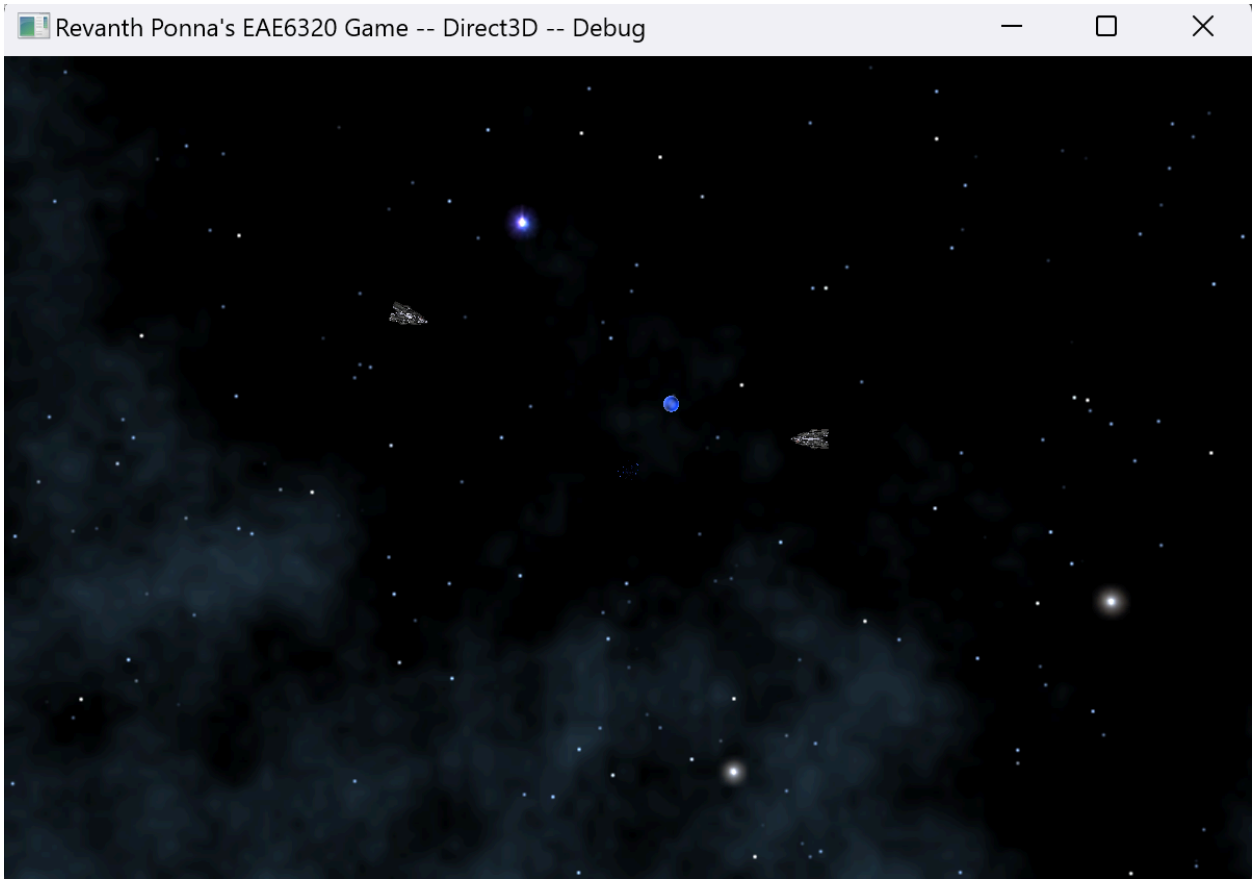
This is where I thought my engine system could have worked better. Since I didn't really think about mesh and effect data during my initial networking implementation, there was no seamless way to incorporate the Graphics system with the networking data when making the game. I believe this is something that could be worked on and improved to ensure all the engine systems work better together.

Using the Physics Collider System

For this final game project, I also used my classmate Yunhao Ye's Physics Collider System to detect collisions in the game and have specific events bind

to those collisions. For example, when a projectile collides with a spaceship, the spaceship is destroyed.

I had a reasonably easy and comfortable experience working with my classmate's Physics Collider System and code. Similar to my engine system, I felt like he also went with a minimal design, which made incorporating and understanding it much easier. There were only a few different types of colliders I could use, and the write-up made it clear how and where to call the interface functions.



One aspect of the Physics Collider system that I found really cool and neat was the use of inheritance in the engine system. First, there was a BaseCollider class that had all the essential functions needed for any type of collision. Then, child classes such as SphereCollider and BoxCollider were inherited from the BaseCollider to define any additional specific functions. This type of architecture ensured code reusability and a clean interface.

What I Have Learned

During the past four months, I believe my knowledge of game engineering and engine architecture has definitely improved a lot. In addition to making good design decisions, I have also learned plenty of new concepts and topics that are essential to a game engine.

The assignments provided great insight, and I gained experience working with an existing codebase. Adding features and developing the Graphics system was definitely not easy as I had never worked with graphics programming before. But, completing the assignments was a great learning curve.

One of the most important things that I took away from the assignments was the concept of making platform-independent interfaces. Especially in games developed for various platforms, I believe this is an essential skill to develop as a programmer. I learned that creating a platform-independent interface and having the platform-specific implementation hidden from the users is a clean way to design a game engine.

Another essential aspect I learned, especially in the latter assignments, was the importance, advantages, and disadvantages of having human-readable files and binary files. Initially, I thought that only binary files were essential and necessary due to faster processing. However, after completing the assignments, I realized the importance of human-readable files and why we would want to have both in a game engine.

In addition to general design principles, I also learned plenty of concepts and approaches specific to game engineering. For example, multi-threading or making a plug-in for Maya were things I was not expecting to learn. But they were fun exercises where I was introduced to new concepts I had no idea about. Also, our discussion in class about Debug vs Release configurations and optimizations was very valuable, as I've always wanted to learn those things.

Thoughts about Engineering, Architecture, and Design

My understanding of engine architecture and design was greatly influenced by the assignments, the engine system project, and the final game project that I

completed. I believe leaning toward a minimalistic design is better because requirements change all the time, especially in the games industry.

I remember one of our class discussions where the professor had to revamp and remove a lot of his previous work because the game's requirements changed in the middle of a development cycle. One of the essential design principles I learned from this class is that the more complex a system is, the slower it's going to be. So, if it doesn't need to be there, it shouldn't be. I would much rather start implementing and designing things as I go, only if there is a requirement.

In my opinion, a 'good' architecture and design is where each system has a clear goal, and each class controls a specific set of functions. Another important thing to keep in mind is that if two blocks of code require the same data/variables, they probably belong together. I learned the importance of refactoring where, as a developer, I try to ask myself why I am writing a piece of code twice? And how can I make it so that it is only written once?

On the other hand, the characteristics of a 'bad' design, in my opinion, is when a system tries to have lots of functionality and features, but doesn't do any of them well. I believe it's better to focus on just one thing and do it well. Then, we can always add small things to it. Trying to add too much early on can result in a messy architecture, which is not easily readable and usable by its users.