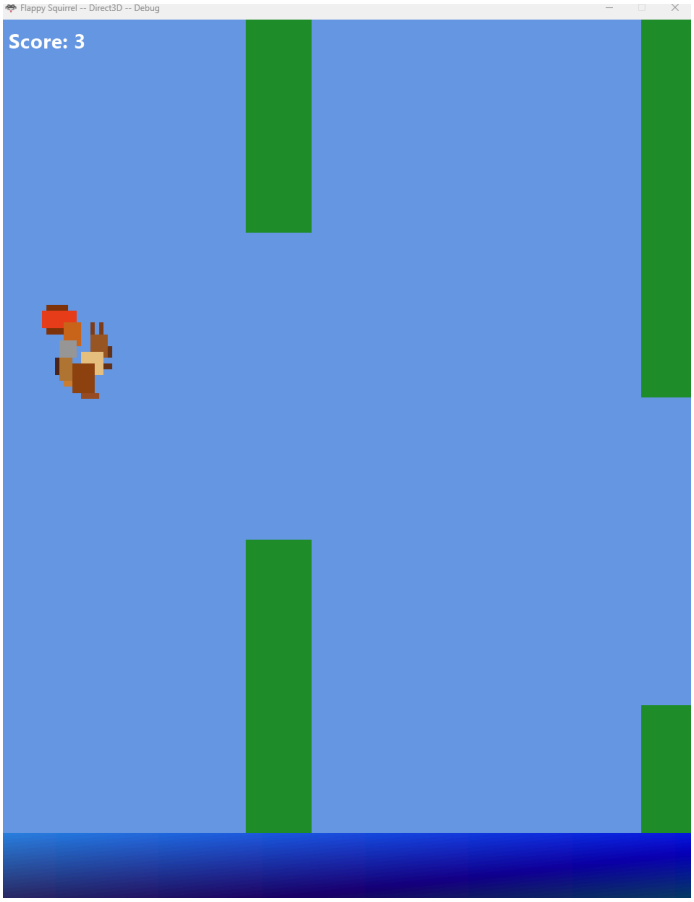
Final Project Write-Up

Game Screenshot:



My Final Game is Flappy Squirrel (Bird). I chose this game because it best went with my systems and the systems I wanted to use. Since I did advanced physics and collision, it was very straightforward to get the player controller movement working. The collision was also very straightforward, as it is AABB, so if the two meshes collide, then it ends the game. Flappy Bird is a simple game, so having an audio system made the experience more immersive, and the amount of sound was small as well. Having a score on screen also adds to the replayability and even the potential for the score to be saved as a high score in a JSON file that can load that value that is saved to file. All that was left to do was figure out how to procedurally generate the obstacles, which was a fun challenge and why I love gameplay programming. The reason I wanted to make Flappy Bird was to be my first project outside a mainstream engine. My initial plan was to use a library like Raylib, but doing it from scratch in our own engine felt more satisfying.

My time spent dissecting the code and files in the engine greatly benefited what I had available to me. Understanding the Math and Physics, as well as creating a game object project, made developing the game much easier. Being able to scale objects in code rather than editing the vertex and index data sped up asset usage and placement, and made reusing the same assets possible. Also, making the camera its own object allowed for the effect of moving the player forward. When implementing the UI, it seemed straightforward, but there was a hiccup with includes that I had to wait for help from another classmate to fix. Outside of that issue, reading their documentation was clear enough to get what I was looking for. A suggestion I have is to include these fixes in case the other developer's environment is different and has similar issues to mine.

This semester, I learned so much from dissecting unfamiliar code to building a game by refactoring the code. Each assignment builds off of the others to show how graphics can start with a simple triangle to have color and even 3D models that can be read from their own asset files. This process shows how important it is to build tools that allow you to read in data that other developers can supply, which aids in making meshes more efficient. How you protect that data is also important, so that if you work on a multiplayer game, there is some security from user data breaches, as well as hacker/modders. Platform Independence is also important, and building interfaces for it makes it so you don't have to code for multiple different platforms, and everything can be streamlined to save dev time.

Spending more time on the back end makes front-end development easier and faster to implement. When we were transferring our code to be platform independent, it meant that we didn’t need to change much after a certain point in the platform-dependent code; that way, when the code was built, there was no expectation of error for each version you built on. Building easier-to-read interfaces makes collaboration easier, especially when working on large teams. Having a coding standard helps everybody be on the same page and avoids many roadblocks in development, where time is everything. In my game, I never had to touch platform-dependent code due to the changes we made with the assignments, making it possible to just worry about building on one platform and not worry when building on the rest.
I had so many Ideas that I want to implement in my game, but I just didn’t have the systems in place; however, given how simple it is and with more time, I have too many ideas for systems I want to implement to continue to build another game in my own engine. This allows me to expand my knowledge of C++, engine and system design, and be satisfied in making a game from scratch.