Matthew Warren

CSC 481 Reflection

My game engine went through a ton of iterations as the semester progressed. Since we were warned constantly to be thinking about how our design would grow to support more features, it was constantly on my mind as I was thinking about how to build different features. I think I was successful in many ways to make an adaptable game engine, but also fell short in other areas. This reflection will go through my thought process and design throughout the semester, and look back at things I am proud of or would have done differently.

HW1 was all about building our game objects and making them change based on user input and collisions. I knew that collision was a critical part of any video game, so started by making a base “CollidableObject” class that extended a basic sfml shape class. This class had features to load textures, detect and resolve collisions, move, and more. This turned out to be one of my best decisions, as I used this base class throughout every other homework. Anytime I needed a new game object that cared about collisions, I just extended this class and added whatever other functionality it needed in its own class. It was so easy to add different game objects, and required very little changes as I went through the semester.

HW2 involved multithreading, time, and networking. After designing my timeline class, I did not need any major changes to it for the rest of the semester. The design closely followed what was provided in the lectures, and was robust enough to handle pausing, tic rate changes, etc. However, here I got lazy in my client-server design. We were only required to have 3 players, so I hard coded variables in my client to only work for up to 3 connections at once. This is something I had to go back and redesign in HW3, and regret not spending more time here to get it working better. Other than this, I think I created a solid foundation for thinking about how my client and server interact. I wanted the game to feel responsive and quick, so all movement was handled and displayed on the client, with the server just being sent updates. Communication string were as short as possible, for example: “5 100 200” meant player id 5 was currently in position (100, 200). This allowed for quick communication with the server while also feeling as responsive as possible. I used these ideas for the rest of the semester.

I already addressed a major change I had to make for HW3 in the previous section, which was fixed to allow an arbitrary number of players to play at once. I chose to continue to develop an object centric model as that is what I had started with. It was easy to adapt into the entire game, and had no problems building from it. One major thing I implemented here that I carried throughout my engine was a system for the server to send the entire world to the client whenever it connects. This would keep the world consistent across clients, even when one joins late.

The events for HW4 and scripting for HW5 was something I also followed the examples closely, and it made for a robust system that was very reusable. I did have to modify a lot of my client and server code to make changes happen with events vs hardcoded in the game loop, which was more work than I thought it would be. I don’t really see how I could have avoided this, though. I will expand on this more next, but I did not have to change very much after focusing my game loop around events.

When it was time to build the second game, I decided to start from scratch and copy the code I needed in. I originally started to try and modify my platformer game to make space invaders, but I had too many systems that deleting features I no longer needed (like platforms) would break a bunch of stuff all over the code, and I felt it would be more efficient to start from scratch. I was able to use my CollidableObject base class from before, as I knew I would be needing collisions with bullets to be detected. I had to create new classes for bullets and the enemy. Movement and event structure was also copied straight in - I used the same movement event I had in the platformer which worked well. I had to replace the jump input with shooting, which required a little modification. My client/server setup was the same, with the client handling movement of its player and the server sending some enemy instructions. Since I decided to make this game single player, I also kept enemy movement in the client, as it was a continuous thing that would be a better experience to have in the client. This also made it easy for tic rate adjustments to also apply to the enemies movement. The server sent instructions for which enemy should shoot and when, which was nice to give the client one less thing to figure out and handle the timing of. On startup, the server will send the player data and enemy setup. Scripting was also implemented the same way - with the only changes being to work with the new classes I wanted them to change. Outside of the movement event, a lot of my events needed to be changed. I made them very specific to the platformer game, and looking back would have been better to make them more general. The movement input event was the only one I could copy over, and had to make new ones for the rest of what happens in space invaders. My handlers also had to be slightly adjusted to take in the different objects I created in space invaders, but kept the same general format. I did have to make a small change to my event manager that I did not expect. I created a script that runs on an event, and creates a new event to add to the manager. However, the manager mutex is locked when processing that event (running the script) then locks it again when trying to add the new event to the queue, freezing the client. I had to add functionality to specify if the mutex needs to be locked when adding to the queue. This was not hard, but not something I expected to need to change.

Overall, I think I did a good job of making design decisions that were able to be adapted to making a different type of game with my engine. I did not have to add or overhaul any of my existing systems, and only had to make minor changes to implement the major features of the different game I was making. If I were to start over, I can’t think of much I would do differently. With the knowledge I have now, I would definitely start making more robust game objects from the start for features I now know are critical parts to game engines. I could definitely plan around what events I am using better and make sure my systems will work with them. I would have made sure to plan how my data is passed between threads better, as a lot of my debugging was figuring out why my program was freezing due to mutexes being locked. As for my design decisions, I think I would come up with a similar design if I were to start over. Part of this may be that it is now what I am familiar with, but it also worked well and made it easy to create 2 different types of games with, so I must have done something right.

Matthew Warren

Most of the features I implemented are relatively limited in scope to this class, and definitely need changing to enable more robust features that typical industry standard game engines have. All of my decisions were centered around what the requirements were for the class, and I did not consider how they would enable different things in the future. That being said, I am happy with how my code turned out, and definitely have more of an appreciation for how video games and their engines are made.

I did a diff of my two game folders, and exported it into a diff.txt in my hw5 parent folder. I did not see a way to get the percentage, but there are a lot of changes listed in that file.