Chester Merino
Professor Su
CSE 165
12 December 2021

INTRODUCTION

The idea I chose for this project was to create a two-dimensional top-down view mini golf style video game in OpenGL. The original plan was to create five different levels, but due to time constraints, I opted for three levels, each demonstrating increasing difficulty. The first level would be a "tutorial" kind of level, with no obstacles, just a clear path to the target hole, and an overlay of instructions on how to play. The second level would introduce obstacles, but static and immobile, to get the player used to "banking" the shot around corners to get it into the target hole. Lastly, the third level would include dynamic obstacles–a moving wall object in addition to the static obstacles to create a small window where the player would have to time their shot to get the ball through to the other side where the target goal would be. I also wanted the game to be controlled by mouse movements with only a couple keyboard keys to control level selection and start. Since I had no experience with OpenGL, I followed the tutorial on learnopengl.com to get started. The tutorial uses GLAD, which is a multi-language library function loader that has a custom file generator on their website based on whatever needs fits your project. This is the library that loads pointers to OpenGL functions at runtime, core as well as extensions and is required to access functions from OpenGL. The tutorial also uses GLFW (Graphics Library Framework), which is a cross-platform library that manages the OpenGL window and handling and processing of user inputs. Another library used was GLM (GL Mathematics) which uses vector calculations to determine object behavior. While the tutorial used STB (named after the creator's initials), as the library for loading textures, I was able to find an older tutorial that uses SOIL (Simple OpenGL Image Loader) in the exact same way as the two functions used took the same arguments. The reason was most likely due to being simpler, as SOIL uses STB as the base image loader, and also possibly due to SOIL being an outdated library (API has been considered deprecated on Mac OS for a while).

MOTIVATION

The reason I chose to do this project was that it was a way to apply the previous topics learned in class such as inheritance and polymorphism, as well as topics from other courses such as trigonometry and statics mechanics. It was also a great way to learn the details and the backbone of how video games work, specifically movement and physics, and collision detection and resolution. This was also a good head start to learn about computer graphics to prepare for the upcoming CSE 170 course in the Fall. Having this project would also look good to add to my portfolio or resume when searching for a career in game design.

DESIGN AND IMPLEMENTATION

When using the tutorial from learnopengl.com, it helped make the project clean and organized. It methodized everything by creating a resource manager class that took care of loading textures, sprites, and shaders, as well as having templates for classes and headers to keep it orderly when adding new objects. In addition, the tutorial used the game *Breakout* as an example, so it provided data such as the behavior of the ball object, including position and velocity, window border collision that flips the corresponding x or y screen coordinate, as well as providing the complex code for collision detection between a circular and rectangular object. Although the game *Breakout* demonstrated object movement, the ball object only moved at a constant speed once active, so I needed to figure out a way to slow the ball down realistically. I was able to find some code with a similar application which required me to add a mass variable member to the ball object class, declare a coefficient of friction constant variable, then using GLM's vector calculations, define a new vector–momentum–which is the product of velocity and mass. Then using GLM's normalize function, convert momentum to a unit vector, multiply this with the friction constant, and divide by mass, I could decrement the velocity of the object each frame to show a realistic deceleration of the ball. I also included this same calculation to all the collisions to shave a little speed every time the ball comes in contact with an obstacle or wall. With that done, my next objective was to set the initial 'shot speed', the instantaneous acceleration when a golf club hits the ball. To do this, I needed to grab two mouse point coordinates, one at the position when a mouse click was detected, and then when it was released. Then with the differences between the x and y coordinates, finding the velocity was a matter of using a simple physics calculation to get a magnitude of the vectors. After computing this vector, I had to scale the maximum speed by using a ternary operator on the individual vectors, so the ball wouldn't move too fast for collision detection. Then I decided to add features like a 'power gauge' that scaled with position and color from green to red. I then added an arrow sprite to compute the direction the ball would travel, and I did this by using trigonometry to get the inverse tangent of the two vectors to convert into an angle figure, as GLM had a way to rotate sprites, but only by the angle value. And since *Breakout* only had circle & rectangle collisions, I needed to create a boolean function that checks and returns true if the distance between the two centers are less than the sum of their radiuses.

OUTCOME

I was able to finish the main concept of the game which turned out well, although some issues I ran into during this project was the amount of time spent researching how to even set up OpenGL as there was no clear guide available that also implemented the use of SOIL. Another issue is that the distance calculated between the two mouse event points is at a 1:1 ratio with the value of the velocity. For example, a short distance between mouse events basically moved the ball very little, and larger distances had too much power. Ideally, if I had more time, I could find out a way to make the input of distance correspond logarithmically with the power of the ball shot. An even better solution would be to implement a physics library such as Bullet or Box2D,

although it would require more time to research how. This could also help with collision issues, as when the ball moves too fast, it would entirely skip a position coordinate between each frame. In addition to those libraries, I would have also liked to include ones for sound and text rendering.

CONCLUSION

Although this project was difficult due to lack of guidance for set up, and the fact that it was an entirely new concept to learn, I think it was important to acknowledge the practical uses of object oriented programming. It also let me demonstrate a little bit of knowledge gained from problem solving in previous engineering courses. With everything considered, once I started figuring out how things worked, it was actually a very fun project to do, and I may use these newfound skills to create future video game projects with past ideas.