**Hangar 13 Engineering Test – July 2017**

Izzy Benavente

Time Spent: 8 hours

VS Version: Visual Studio Community 2015

Added files: Projectile.cpp, Projectile.h, Clock.cpp, Clock.h, Vector2.h

**Explanations of Implementation and Design Choices:**

1. Positioning the cannon

I positioned the cannon using relative positioning based on screen width and height, so that the cannon would still be in the bottom middle even if the screen size were to change.

1. Rotating the cannon

I made variables for the cannon’s rotation angle and rotation speed so they could be easily tuned. The angle is always kept between -90 and 90. I made a helper function to draw the individual lines of the cannon. I use the bottom middle of the cannon as the anchor point and rotate each point of the line around that anchor.

1. Shooting a projectile

In my approach to creating the projectiles and clocks, I decided that I should create a quick Vector2 implementation to make movement and other calculations more manageable. Then I made a Projectile class, which keeps of the projectile’s head, tail, and direction. I use vector math to keep the projectile moving in its direction. I flag the projectile as off screen and ready for deletion once the tail vector goes past the screen boundaries. C\_Application stores all projectiles in a vector, deleting any that go offscreen.

1. Shoot multiple projectiles while holding SPACE

After implementing part 3, this seemed to be already accomplished, but with an extremely high firing rate. I added a very short cooldown to slow down the firing rate to match the example result.

1. Creating the clock object

I made a new class for the clock and kept track of its position and radius. Like I did with projectiles, I store each clock in a vector within C\_Application so that it can iterate through each object and call update. I made draw functions for the clock’s square frame and for its hands. Using the included GetTime function, I take each hour, minute, and second value then convert it to an angle from 0 to 360. With that angle, I draw a line from the clock’s center point outwards using a length and color specific to each time hand.

1. Show two clocks on the screen at random positions

I added a SpawnClocks function to C\_Application which spawns exactly two clocks each time it’s called. In the Clock class, I added a function called FindNewSpawnPosition which sets the clock position to a random point on the screen where the clock can fit, factoring in its size. I also added a function in Clock to check collision (using AABB) with other clocks. SpawnClocks will check if new clocks collide with any existing clocks. If they do, FindNewSpawnPosition is called until a satisfactory position is found. Then the new clock is added to the vector of clocks.

1. Set the clocks in motion, able to rebound off walls and other clocks

I added variables for speed and direction within Clock, which are set to a random value in the constructor. The clock will move using those values each time Update is called. For collision with other clocks, I use the AABB collision check made in step 6. If two clocks collide, I reverse the direction of both clocks. For wall collision, I had to make a new function that checks which wall was collided with, then calculates the reflection vector of the clock’s direction to bounce off the wall.

1. Test projectiles and clock collision, clocks should divide into two smaller clocks

To start, I added a life count variable to clocks. The radius of the clock is set in the constructor as a function of the life count and the original radius. In Clock, I added a collision function that does a line-box intersection test. In C\_Application, the collision check between projectile and clock is done during the iteration through all the projectiles. If a collision occurs, the projectile and clock are destroyed, but two new clocks are spawned with one less life than the previous clock. These new clocks are spawned inside of the area that the destroyed clock occupied.

1. Two big clocks spawn when the last clocks are shot down

When the vector of clocks in C\_Application has a size of zero, two clocks are spawned with the maximum amount of lives, which gives them the original, large size.

\*A note about the time taken and a mistake on my part:

I started this test shortly after finishing another programming test. In my haste, I didn’t catch the sentence that said you could change the test result file into an exe. For most of the assignment, I was working purely off of the test instructions without knowing there was an example executable. When I finally saw it, I had to go back and change behaviors that didn’t exactly match the example, so that added to the time spent on the test. I apologize for asking questions that could have been answered by reading the document thouroughly the first time and finding the test example. I’m grateful for this opportunity to take this test and I thought it was pretty fun.