# Gooey Cannon Report

Hour 1

During the first hour, I decided to implement the complex collision code (found in “Implementations of Collisions 2” (Francik, Implementation of Collisions 2, p. 13)) that allowed for object rotation, since I had previously done the simpler code with the bouncy ball game. This was straightforward, as all I had to do was use the code provided in the collisions slides and use the objects provided by the game instead. In one instance where I was confused about why it wasn’t working properly, I remembered that my bouncy ball code used a different function instead of GetTime(), so I had a look at that code and replaced it with GetDeltaTime() \* 1000f, which allowed it to work properly.

Hour 2

Admittedly, I did not realise that the complex collision code only worked for one side of any rectangle at this point and spent the hour wondering if the values for “alpha” should have been changed, although nothing I did helped.

Hour 3

I had a look at my bouncy ball code and realised that I likely needed to use the code three more times for the other three sides. I used Dot(v,n) > 0 instead of Dot(v,n) < 0 and used d = u – (Y + R) \* n instead of d = u + (Y + R) \* n which successfully implemented collisions for the other horizontal edge of the rectangles. However, I struggled to find a way to collide the vertical slides. I tried using cross product in stead of dot product, and I tried Dot(n,v), both to failed results.

Hour 4

I went back and had a look at the slides and found the “Implementation Of Collisions” (Francik, Implementations of Collisions, p. 39) slides, which on slide 39, helpfully reminded me of how to find the normals for each side of a rectangle at any angle. With this, I went back to my code and made a new attribute, n2, which held:

CVector(-cos(alpha), sin(alpha))

With this new value, I was able to implement vertical collision once I also swapped the X and Y attributes in the rest of the function. Dot(v,n2) < 0 was used for left vertical collision and Dot(v,n2) > 0 was used for right vertical collision, again with very minor changes to the function for both of them.

Conclusion

As a reflection, I find it convenient that a simple change such as greater than or less than zero allows one to not have to make four separate attributes. Although in hindsight, perhaps it would be more showing of my knowledge of physics if I had indeed made four attributes for each normal. I also feel that there must be a way to optimise this function to make it even neater to implement all four sides.

As I did not feel confident with rounded object collision, I decided not to make a new level, as I felt that it would not be demonstrating any new skills that I hadn’t already.

As a final conclusion, this exercise helped me understand that collision code was a lot simpler than I was initially worried about, as the hours I spent scrutinising the code and combining it with my understanding of vector physics helped me get a better feel of how vectors worked in code, and that the solution to my problem was just that I wasn’t aware it had to be implemented four times.

# Works Cited

Francik, J. (n.d.). *Implementation of Collisions 2.* Retrieved from https://kingston.app.box.com/s/mc0av99htvydkzik464j1n4ndklhe6v4

Francik, J. (n.d.). *Implementations of Collisions.* Retrieved from https://kingston.app.box.com/s/75abfvja2yxv10adns30rccu50hi9g85