ANGRY Post-Mortem

For this assignment, I had to make an angry birds clone using the Box2D physics library and OpenGL for the graphics. For my version of the game, I decided to strip away the idea of Angry Birds into its core essence, so the obvious choice was to take away the birds and just leave the angry. The game will be presented with a more abstract (but still easily readable) art style, consisting of angry shapes and sprites, with a fitting name of "ANGRY".

The first problem I came across during the development, was the initial integration of the Box2D dependencies with my own game engine (DragonDrop 0.1.0). Firstly, the .lib files were not included with the version of Box2D that was downloaded, so I had to figure out a way to build the .lib files myself using visual studio. This required the downloading of a program called premake5, which was needed to create the visual studio files for building Box2D. Then after building it in the debug mode, the .lib now works with my project. However I soon came upon another problem when I tried to build my project in release build, which ended up being the fact that Box2D requires a .lib file built specifically in release mode to be able to build a project in release. After that was done the rest of the implementation of Box2D into my engine went smoothly enough.

For the physics to work with my engine, all I needed to do was to take the outputted transform values from a Box2D body, and have those be used for my GameObject's transform values, that way whenever a Box2D body moves or rotates, its corresponding GameObject will also do the same and render its attached sprite in the same manner. Once this was the done for both the "birds" (which I refer to as "balls" instead) and the boxes, making a simple Angry Birds style level was quite straightforward.

The first level was first done with some test objects, in which I just made three piles of boxes and had a ball hit them with a predefined force. This proved that the implementation of the Box2D library was working. Next was to figure out how to detect which objects are colliding, so that I could have destructible objects "destroy" themselves when they are hit whilst above a certain velocity. This way I can have destructible enemies (which I refer to as "Evils") and also destructible boxes be destroyed in a satisfying manner. This required me to create a class that communicated with the Box2D physics world, and "listened" to the collisions that happened per frame of the physics update. This way I could find out which Box2D body was colliding with which other Box2D body and any given point in time, meaning that I can now check exactly when two objects collide with a specific velocity. Once this was done, the destruction was pretty much done.

I decided that instead of needing to have separate objects for the Box and the Plank, it made more sense for them to be the same object, but is able to be dynamically sized and scaled to fit whichever needs the level requires. This was easy enough to do, but did take some time to implement, nevertheless it definitely helped a lot for when it came to making the levels.

In order to have joints in the game as well, I created a "machine" class that could take boxes and balls and join them together using joints to create various things. The first machine called the "Windmill" was created with two box bodies connected by a motorised Revolute joint, this was

dynamically size-able due to using the boxes to make up the machine. This windmill machine would be used as a further obstacle to block the player's projectiles so as to create more strategic gameplay. The second machine was a cart that used a box body as the cart and two ball bodies as wheels, which in turn are joined together using Wheel joints to the cart, the Wheel joint is a different joint to the revolute joint, in that it also creates a spring-like property to give the wheels the effect of having suspension. The third machine was the pulley, which used two boxes as platforms that were joined together using a Pulley joint, the pulley joint itself was fine as far as implementation goes, but I had to create a new Line class in order to render the pulley's ropes and stuff properly.

After the joints were implemented, I started on creating the three different kinds of balls, one was to be a ball that can dash, one was a ball that shot pellets like a shotgun, and the third was a ball that could explode like a bomb. The dash ball simply multiplied its current velocity vector by 100 whenever the ability was activated. The shotgun and bomb balls however required a bit more work. I made it so when they shot or exploded, they would send out a number of smaller ball objects (that also had physics) to act as the shrapnel. This was done and tested with various different sized balls and explosion forces until I had ones that I was satisfied with.

Once that was done, I was able to start working on the cosmetic polishes and particle effects. The first was figuring out how to do a blood splatter effect for when the Evils get destroyed, so since I already had a system in place for exploding balls, I decided to just use a similar method for the blood as well. The main difference with this was that this was set to a different collision filter than the rest of the physics objects, so that it will not collide with anything else but other blood objects. This worked pretty well to give a gratifying blood splatter effect.
The next effect was to create debris when the destructible boxes are destroyed, this used a similar system to the blood splatter, but instead of balls, it spawns thin boxes (that do collide with everything else) of varying sizes to simulate the look of debris.

Once all of these elements were implemented, I used them to create 4 different levels that each showcase either a different ball type or a different machine type with different joints. These were playtested multiple times and iterated upon with different layouts and different balls. Once these were done, the game was in a state that I was satisfied with.