**Post Mortem – Chris Stone**

I began this project on Wednesday, 3rd of April. I completed it on Sunday, 8th of April. I have altogether put about 24 hours of work into it.

**Goals:** One of my main goals for this project was to make the shape and the animation very much independent. This is so that any animation could be run on any shape, without having to know information about which shape it was. I did this by passing the Shape interface to the animation’s update function, instead of a concrete class. This created a flexible system that is easily modifiable.

Another goal was to make sure to handle errors properly. I have several functions to check OpenGL errors which I make use of when initializing OpenGL properties. Because the program is reliant so heavily on user input, I had to make sure to check that all inputs were valid, especially on dimensions of shapes. I make use of throw, try, and catch statements to handle them as cleanly as possible. I also made sure I was free of memory leaks with the CRT library.

Extensibility was another goal that I had. I set up a lot of aspects that support extensibility but are not fleshed out because they were not in the scope of the brief. One example is the Entity class, which manages shapes and animations, but could be extended with other functionality (for an enemy with a gun, for example). The fact that Entity supports composite shapes and multiple animations also contributes to this extensibility. Another example is the Canvas class supporting multiple controllers. With this, a second player can be added, who could perhaps control a different aspect of the game.

**Decisions:** For my graphics library, I chose OpenGL, as I had recently been revising it, and had previous projects that used it that I could reference. I used PugiXML to read definitions of shapes, because I had worked with it before, and I believed it to be faster and easier to implement than using streams and text documents. The XML file also makes it very clear and easy to identify which values to edit.

There were several different ways I could have approached the actual program. Either as a general graphics framework, as a drawing program, or as a game. It made more sense to me to approach it as if it were a game, since the animating shapes reminded me of AI patrolling (although originally, I had a different approach, hence why my game manager is called canvas).

I chose in this program to use the prototype pattern to handle spawning shapes. Approaching as a game, I decided that I would want to spawn many shapes from one definition (akin to spawning many enemies of a certain type). If I could construct an entity from the definition that I loaded in, I could then spawn shapes in the game by copying that entity. This seemed like an elegant way to tackle the problem.

**Improvements:** One aspect that I am not so happy with is the way I load in the definitions, and the layout of definitions themselves. I would like to separate the definitions for shapes, animations, and entities, and find a cleaner way to assemble the entity. This is no doubt the hardest part of the program to modify if I were to need to change it, which suggests that a more elegant solution could be found.

Also a few of the naming conventions that I have used throughout the program have been rather inconsistent and unclear (Canvas just being one example). Tidying this up would go a long way to improving readability.