# How was the program designed?

## Worlds

The Worlds act as different scenes within our program, each having its own OpenGL callbacks for:

* Special Function – Movement Keys
* Special Up Function – Release Key
* Keyboard Up Function – Release Keys
* Keyboard Function – Keyboard
* Display Function – Display
* Idle Function – Idle
* Mouse Function – Mouse
* Motion Function – Mouse Move
* Reshape Function – Reshape
* Passive Motion Function – Mouse Move

The *main.cpp* file sets up these callbacks and allows for changing between worlds. The main function has a member variable of type World\* which keeps track of the currently activated world. They can be switched between using the variable *currWorld* variable available in the World abstract class which records which World to change to.

# What issues were encountered and how were they resolved?

# What special features are in your program?

# What issues, if any, arose when you were doing the maintenance part of the assignment?

# Testing Details

# Suggestions for Improvement

The biggest improvement that could be made is not related to the actual construction of the program, but is instead to do with time management. The major working periods on this program were upon initially receiving the assignment and during the few weeks leading up to due date, this was due to underestimating the time required to do tasks and the number of tasks involved. Doing it differently we would ensure progress is made each week thus enabling us to better judge what exactly is required to be added and the sort of time required to complete those tasks.

There were of course many program related areas in which we could greatly improve also, the largest of these would be memory management, as we were low on time much our code is functional rather than well written and set out, therefore many objects are not deleted after their use is fulfilled so more memory is continually taken up as the program runs. The biggest memory sink was related to reading in models using GLM, every time a model was required it would be read in by GLM then displayed, even if the same model had already been used before. This was a huge waste of resources as nearly every in game model is used very often. What we would have liked to have had was a class that managed the models so that one model was only read in once and then could be used anywhere.

Multiple classes were also poorly written, with some instances of public variables being used simply to get things working. Had we the time we would very much like to overhaul most of the classes to get them running in compliance with low coupling and high cohesion.

The Finite State Machine AI also needs improvement as currently each AI uses only one behaviour, this behaviour handles everything that AI does instead of transitioning between multiple behaviours. Splitting these single behaviours would have made the code much more readable and understandable. This also currently defeats the purpose of having the finite state machine class.

Given more time we would also increase the current scope of our game, by adding more AI and more abilities so that the game play could be mixed up more to keep it fresh, currently the only change is difficulty as the AI come out at a faster and faster rate, this means the game gets stale very quickly.

# How is code reuse taken care of?

Code Reuse was something done reasonably well in this program, the major way was through inheritance, the majority of the class in the game inherit from the game object class, that contains variables and functions that get used in all of the children, the same also occurs with the projectile class, where projectiles of different types inherit from a parent projectile. It can also been seen in the AI where multiple AI can use the same state, in the case of our game the seek behaviour is used by two different AI’s who simply set their own speed to differ from each other.

# Can the assets/objects/items be reused easily in other programs?

Some of the parent classes could be very useful in other programs, the gameobject class is a basic class that contains variables and functions of use to any object placed in a game, this would be quite useful in another game program however details to do with displaying may need to be changed. The finite state machine and associated state class are probably the most portable pieces of code, zero changes would be required to use these classes in another program and it would not be restricted to game. The same also holds true for the sound class.

Vector3