Evaluation Report

This document is an evaluation of the development and result of the Games Architecture ACW, discussing specific outcomes of the program as well as evaluating the development process of the coursework. The assignment was to build a component-based engine system and using the built component system, create a maze game.

My result for the assignment is a rather disappointing maze game that only barely achieves the tasks listed within the specification. The game is built using the component-based system, independent of the engine and making use of abstract classes and interface systems. The game itself is a basic rendition of the maze game with the inclusion of 3 keys to open the portal. There is an enemy 'drone' on the map that is able to cause a game over if you run out of lives, but the drone does not have any form of artificial intelligence to track the player; instead the drone patrols the centre of the map. There is additionally no implementation of any of the choice implementations specified within the ACW. This is due to slow progression resulting in a lack of time to include any of the choice implementations.

Development during the ACW was very problematic and by the end of it, the result still has plenty of major bugs and issues in the code. This section of the document will go through the various bugs and issues that were, and in some cases still present, with the source code and development.

- **Line to Sphere Collision (Ongoing Issue):** The biggest issue with the program right now is the functionality of the program is the line to sphere collision used for detecting collisions with walls. Currently the system is very inconsistent in action, allowing the player to pass through walls majority of the time. There are times where the collision detection is functional but due to its high inconsistency, there is no apparent way to troubleshoot the issue. In addition to this, occasional attempts to stop the player going through the wall causes them to launch off the map. In terms of the cause of the issue, my current opinion is that the 2nd player position point is causing the results of the calculations to be incorrect – this would also explain why it is inconsistent rather than not working outright.
- **Texturing (Ongoing Issue):** The game makes use of a common moon shape object for keys and portal; these were to have different coloured textures to differentiate them from one another. These changes in colour however don't appear when running the game. This not only makes the distinguishing the keys from one another impossible, but it is concerning that there is no visible reasoning as to why the changes in colour are not apparent.
- **Audio System (Ongoing Issue):** The audio system has had a few minor issues during development and while the system is in place, it is currently not functioning. Initially there was an issue with reading specific types of .wav files, most files I attempted to use were not compatible due to their sampling rate. This was sorted with changes to the audio files, this however brought up another issue that was apparent in the code in that the audio doesn't run at all. This is most confusing as a previous code that was built during lab assignments uses similar code and functions.
- **Missing Geometry & Mass Deleted Geometry (Solved Issue):** As I was using the same moon objects for keys and portal, I was using the same object file for each entity. However, it was made quickly apparent to me during development that OpenGL can't have OBJ loaders using the same objects simultaneously. Finding this out, I solved this with duplicates that were separate files or replacement models. After this another bug came where attempted removal of an entity's geometry removed the geometry from all entities currently rendered. This was due to my initial attempt at geometry removal removing the entire list of objects instead of specific entities. The solution to this was to access the components through the game scene in a similar fashion to the systems. This means I can manipulate entities live

during play. This method was forwarded to use in key counting and management as well as collision.

- **Rapid Counter Decrement (Solved Issue):** The current system for analysing successful collisions is to send messages from the collision manager back to the game scene and have the actions take place in the scene. When initially implemented, messages from the collision manager were sent ad infinitum. This caused situations such as negative lives and infinite keys. What I didn't do beforehand was clear the list after every collision, this means that the collision test wasn't sending out repeating messages to the game scene.

Overall, I am not happy with the outcome of this assignment development execution and the final product of the coursework. There were multiple hiccups during development and it was made apparent that there is a hole in my programming knowledge that needs to be worked on.