ICT397 Issues Encountered Sheet

# **Assignment/Project Name:** ICT397 Assignment 2 - OOber Taxi

# **Group Name:** Group Carré

## Jack Matters:

### Script Manager Class

A lot of time was spent trying to design the script manager class, and the accompanying scripts. Trying to create a well designed class that could handle any form of script turned out to take far longer than I would have liked. Similarly with the scripts themselves, deciding on they would be laid out and how the data would be read in took more time than I would have liked. Eventually, I opted to just have a separate function for each type of script just so we had something working and I could move on. I would like to go back and change the way I implemented scripts and the Script Manager, as I am not overly happy with how it turned out.

### Giving AI An Collision Body

The way I initially set up the physics collision bodies was they would only move when acted upon by an external force. Having an AI that moves according to its current state, and not by an external force, made it difficult to give an AI model a collision body in the current physics implementation. I have an idea of how to fix it, by doing something similar to what I have done for the player controlled object, but the time that would take to implement is something I didn’t have. So for the final version, none of the AI have a collision body.

### More Than One ‘Smart’ AI

I was able to implement more than one AI in our finished game engine, but they weren’t ‘smart’. By this, I mean they didn’t act independently of each other, instead all moving towards the same waypoint. When one reached a way point, they all started moving onto the next. I would have liked to fix this to have better AI and FSM implementations, but time constraints would not allow me the chance to fix their behaviour. So for the final version, all AI follow the same behaviour.

### Workload

Having our 3rd member drop out fairly late in the semester was quite unfortunate, as this meant my current workload would increase to include what they did not get completed prior to them dropping out, and the work they would have completed if they continued the unit. Playing catch up from several weeks ago, while also continuing to work on my current tasks, as well as ensuring enough time was left to complete the rest of the work proved to be quite the issue. Due to size of the workload, and the time remaining, it was impossible to meet every assessed requirement, or to perform them to a satisfactory level.

## Cordell Smith:

### Workload

After the unfortunate incident of one of our group members dropping out of the unit, we were tasked to split the extra workload up evenly between myself and my partner. Because we were down a person, this restricted our capabilities to complete all the functionalities of the game as well as made us rushed for time, having to compensate for the extra work.

### Design Structure

With each component that I was tasked to complete, I had to plan out exactly how I would implement it using good software design patterns and keeping to the principle of low coupling and high cohesion. The factory method pattern was used most frequently and in doing so, I had to make sure I correctly categorised the same types together. These could be assets, objects, renderer, player, entities, each to be grouped together so that I could create common functionality in the base interface class they inherited from.

### Assimp Loader Flag

There was a minor issue with how the UV data would be read in from the aiScene. When the models were loaded and texture applied, they were not entirely correct. Setting the flag ‘aiProcess\_FlipUVs’ fixed the issue, as the coordinate system is read differently by openGL with coordinates (0, 0) starting in the top left hand corner. This fixed the textures being slightly off.

### Heightmap .raw File Creation

When creating a heightmap image to be procedurally generated into a terrain mesh, the easiest part seemed to be making the image file to be read it on a 3rd party software such as Gimp or Photoshop. This was actually not the case. The .raw files I initially generated were not being read in correctly and produced some unusual terrain meshes. I had to google the process in creating a heightmap and found that you had to make sure the file was Greyscale and something that wasn't found online, that I found through testing was that when exporting from Gimp (image editing software) when you saved it as a .data file (Gimps .raw file) you have to specifically select the ‘Planar (RRR GGG BBB)’ option. This fixed the data being read in and the terrains were correctly generated using the brute force method.

### Modern OpenGL

For assignment 2 I felt more comfortable using modern openGL shaders however there were still a few issues that arose.

#### VAO, VBO Data

The data that was being stored in each models VBO was incorrect. I realised that I had defined the size of each segment that was too large and as a result, when the vertex shader was passing in the UV texture coordinates, it was in fact another attributes data (unsure which one) and this was the cause of the textures not applying correctly to their respected models.

#### Binding Textures

When abstracting the facade openGL render class, I had to also design a way to correctly bind the appropriate textures to the current shader being used and to also be accessed by the uniform sampler2D() in the fragment shader which is what maps the texture data to the correct triangle drawn. This was done by setting the default name of a uniform to ‘diffuse\_1’ and then binding the texture to it.

### GLM Maths and Matrices

Using the glm library to handle all of our maths was immensely helpful however it still required us to understand how the library works. This required reading the documentation of specific functions and variables of the glm library and what can be done. I had issues with some of the syntax such as defining a 4x4 matrix like so glm::mat4(1.0). I didn’t know exactly where the 1.0 would be placed and did some testing to see that it adds it to the x, y, z, a component of the ordered columns, first column being the x, second column being the y, third column being the z and fourth the a. Knowing these helped me understand the syntax and use it within the game engine.

Another small issue was that I was using incorrect values in the glm::rotate() function. When defining the degrees to rotate on a particular axis, I was using radians instead of degrees which only let, say the camera rotate a very small amount. To fix this I identified where it needed to be in degrees and used glm::degrees() to convert it.

### Game Not Compiling on All Systems

This was the biggest issue I encountered that also affected the entire group. If the program does not compile on all systems then the marker cannot assess what we’ve done. The issue came in three parts. The first was that the cmake compiled 3rd party libraries used such as our physics engine needed to have the same code generation configuration as the game engine solution. The second issue was that the major and minor version of openGL was too high. It was change from max 4, min 4, to max 3, min 3. And the third part was to include 3 specific .dll files with the executable that are ‘msvcp140d.dll’, ‘ucrtbased.dll’, ‘vcruntime140d.dll’.