**ICT397 Issues Encountered Sheet**

**Assignment/Project Name:** ICT397 Assignment 1 - OOber Taxi

**Group Name:** Group Carré

Jack Matters:

Overall, I didn’t encounter too many issues. When I was first trying to implement Bullet Physics API into our project, I was having troubles with using CMake just to find out I was selecting the wrong version of Visual Studios. In the case of creating simple rigid bodies (sphere shape, box shape, etc), I didn’t have any issues. Having previous experience with making a Physics Engine, and following the examples that accompanied the Bullet Physics API made it rather simple. The only issue I had was when I was trying to visually test that collision was working. It took longer than expected to draw shapes to the screen that matched the rigid body shapes I had created, so I could visually see that they were colliding as they should be. For more difficult shapes (heightfield), I ran into several issues. The first being actually reading in the data to create the heightfield terrain shape, but this was solved by implementing code from a previous lab for reading in data from a heightfield data file. The next issue was all the data that was being passed into the function to create the terrain shape. I am unsure if I passed in the correct data, or what some pieces of data actually did. The last issue was that without having a visible terrain, I was unable to confirm whether what I had implemented was working as it should or not. I was able to create some form of rigid body, as the camera was colliding with something, but without being able to visually see the terrain, it was hard to continue with that part of the Physics Engine. I ran into similar issues with the code I wrote that would take a data structure containing all game object data, and create a rigid body for all the different game objects. I wrote the code to implement this, but without the data structure being implemented, I was unable to continue working on this section of code. The general layout is coded, but has been commented out until I can continue to work on it.

Overall, the biggest issue I ran into was being unable to continue with my part of the game engine until sections that other members were working on were completed. I believe this is partially my fault as I didn’t offer to assist in those areas, and I didn’t take the initiative to start on other sections of the game engine and/or documentation as I was unsure what to work on, or the sections I could work on weren’t due for this assignment (audio engine, animation engine). If I was to assist more, we probably would have had a better game engine to submit.

Cordell Smith:

### Structure

So initially I had a fairly good idea on how we wanted the architecture of our game engine to be structured. Although the transition from planning to execution was not a smooth ride as I encountered various issues along the way. The biggest fault was not creating an initial UML diagram to help layout the components containing basic functions and member variables of each class and their dependencies to one another. The UML was created later than it should have been however helped a lot in the final understanding of the design structure.

An issue that was caused from this was how the Game Asset Factory would interact with the GameWorld and its objects I couldn't find an efficient way to have the information stored in the data structures be accessed by individual GameWorld objects such as if an object had a specific model to be updated or texture to be applied. This issue set me back longer than it should have but was resolved after proper planning with pen-paper and creating a UML diagram with starUML.

### Assimp Linking

I decided to use Assimp Import Library to handle the reading of what could be many different file types to their data structures. Linking assimp was just a small issue in that I had never used CMake before to create a workable program from the source provided on the Assimp website to link to. This setback took me longer than I had liked and was resolved after further research.

### 3D Rendering Engine (facading)

Creating a self-contained 3D engine was quite difficult as I was learning at the same time as trying to implement it. I managed to put together a 3D render function within a model class that is essentially an object however this was not efficient in abstracting the OpenGL code using a facade class.

### OpenGL Shaders

We planned to use simple shaders in our game engine as it was the most efficient way to load and render model data by sharing the workload from the CPU to the GPU which is also much more efficient and powerful when rendering complex objects. I had very limited knowledge of the OpenGL rendering pipeline and ended up doing extensive research about shaders to try and implement our simple models. I managed to get a simple model loader working using a basic shader class however, when it came to the structure of which class will contain what OpenGL calls I lost track.

#### 

#### Models and Meshes

Using the Assimp Import Library to read in whatever filetype we needed, in this case we were testing with .OBJ files and the format of this file reads in many meshes that make up a single model. When I was creating the classes, I had a vector list of meshes that each contained mesh data such as vertex information and indices information. When it came time to set up my vertex array object per mesh using OpenGL calls, it didn’t respond at all. OpenGL is not user friendly or kind when it comes to making mistakes as I was continually debugging and making changes to get a result but to no avail. I feel like I have an idea of what I was doing wrong however have not had time to implement and test it.

#### OpenGL Shading Language (GLSL)

From various tutorials online, I learnt the basics of the GLSL and how data is read in from the VBO to your vertex shader, then through your fragment shader to produce an output. I was able to understand the exactly how a vertex buffer object works to implement a shader that took in 4 attributes however did not know enough about the OpenGL shading language to process my desired tasks in this C language such as texture sampling which restricted us from applying textures to our models.

### Textures

I spent too much time on the facade class to render objects that I wasn’t proficient in reading in texture data and storing them to be used by the models in the rendering pipeline. The texture uv coordinates and normals were stored in their respected data structures however there was no way to link them to the actual texture that needed to be loaded in.

### Workload

I took it upon myself to get the rendering engine completed however I think that during task delegation, I underestimated the workload of handling both the model rendering and data storing as well as the graphics facade class using OpenGL shaders. I understood at the time that these would need to be worked on closely as they both tie in together and it is hard to try and explain how certain things work to another group member if they have no prior knowledge however I guess that is where abstraction and low-coupling comes in handy. I will learn from this experience in the future and properly assess the requirements of each component.

Michael Bell:

I ran into quite a few issues implementing very basic functions of the game engine such as lua readers and the game world class which all stems from my lack of input during initial setup of the game engine. I lost track of how the game engine was functioning as a whole which made very simple tasks take a lot longer to implement. I need to ensure I keep my involvement up to prevent this from happening in the future.