**ICT397 Group Self-Assessment**

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

**Group Name:** Group Carré

**Introduction:**

This document will outline the testing details we have used for our program, our suggestions for improvement on our current project, and reuse of our software.

**Testing details:**

|  |  |  |
| --- | --- | --- |
| **Test** | **How it was done** | **Results** |
| Gravity | Command prompt (std::cout)  Visual (object falling) | Command prompt printed out correct location of game object rigid body as it fell. Visually, the game object fell due to gravity. |
| Object Collision | Command prompt (std::cout)  Visual (objects colliding) | The command prompt printed out the correct locations of game object rigid bodies. Visually, game objects collided as they should. |
| Camera Collision | Command prompt (std::cout)  Visual (move into other objects) | Command prompt printed out correct camera rigid body location. Visually, camera was used to push other game objects. |
| Shaders | Visual (draw floor and objects) | A floor was visible, as well as some cubes. |
| Loading .obj files | Debugging | Debugging mode was used to ensure that .obj files were loaded and stored correctly. |
| Displaying .obj files | Visual (drew a vehicle) | A vehicle was visible, but with no texture. |
| Scripting | Command prompt (std::cout) | A script was loaded and printed out to the command prompt correctly. |

V2.0

These are the testing details we conducted for version 2.0.

|  |  |  |
| --- | --- | --- |
| **Test** | **How it was done** | **Results** |
| Camera init via script | Command prompt (std::cout)  Debug mode | Camera initialization values were read from script, then printed to console to compare to values in script. Debug mode showed correct data being read in. |
| Models init via script | Command prompt (std::cout)  Debug mode | Models initialization values were read from script, then printed to console to compare to values in script. Debug mode showed correct data being read in. |
| Terrains init via script | Command prompt (std::cout)  Debug mode | Terrains initialization values were read from script, then printed to console to compare to values in script. Debug mode showed correct data being read in. |
| Textures init via script | Command prompt (std::cout)  Debug mode | Textures initialization values were read from script, then printed to console to compare to values in script. Debug mode showed correct data being read in. |
| Windows init via script | Command prompt (std::cout)  Debug mode  Visual | Windows initialization values were read from script, then printed to console to compare to values in script. Debug mode showed correct data being read in. Screen size changed when changing values in script. |
| Load texture into Texture manager | Command prompt (std::cout)  Debug mode | Message would be sent to command prompt if texture was loaded. Debug mode showed map of textures increment by 1. |
| Load same texture | Command prompt (std::cout)  Debug mode | Message would be sent to command prompt if texture was already loaded. Debug mode showed map of textures not increase in size. |
| Load terrain | Visual | Terrain loaded and was visible. |
| Give terrain texture | Visual  Debug | Terrain loaded and a texture was visible on it. Debug showed correct texture being loaded on terrain. |
| Camera follow terrain | Visual  Debug | Camera changed height when moving across terrain. Debug showed camera changing y-value. |
| Load model | Visual | Model loaded and was visible. |
| Give model texture | Visual  Debug | Model loaded and texture was visible and mapped correctly. Debug showed correct texture being applied. |
| Model follow terrain | Visual  Debug | Model changed height when moving across terrain. Debug showed model changing y-value. |
| Give model AI | Visual | Model was moving between set waypoints correctly. |
| Give multiple models AI | Visual | Models were moving to same waypoint, instead of random waypoint each. |
| AI swap states | Command prompt (std::cout) | Command prompt printed when changing states, and what state it was currently in. |

**Suggestions for improvement:**

**Communication:**

The first suggestion is to communicate more as a team. Despite being fairly communicative, there were often times where we were slow to update the other members on our current progress, issues we had run into, or plans that would interfere with working on the project. This assisted in the group falling behind time on tasks, and not being able to complete their sections as much as they would have liked. With better communication, the group would have a better idea on the overall current progress of each member, the project as a whole, and would be able to devise solutions to problems sooner.

**Task delegation:**

Another suggestion would be to delegate the tasks more evenly amongst the group members. The task delegation we initially went with ended with some members having more work to do than others, and some group members waiting on others to finish their sections. With better talk delegation, we could have it so the workload is more even amongst all members, and that each member always has something to work on in parallel to the other members.

**Time management:**

One of the bigger suggestions is better time management. Taking in the size of the project, plus each group member’s responsibilities to other units and jobs, better time management is a necessity. Poor time management assisted in the group falling behind on this project, and is one of the bigger contributors.

**Do things as you go:**

This suggestion is based on doing related tasks in tandem with each other. One example of this is putting in doxygen comments while writing code. Another example is filling in sections of documentation as you complete related tasks (test cases, task breakdown list, etc). The group didn’t do this for Assignment 2, and it resulted in spending extra time going back to complete tasks that would have otherwise been done easily while in the progress.

**Rewrite classes with better design:**

This suggestion is based on the scenario that some classes were written in a way that got them to work quickly, not in a way that had a good design and worked efficiently. This is partly attributed to there not being enough time left to create a well designed and efficient class, partly due to the extra workload for each member due to another member dropping out, and partly because we were already behind from assignment 1. The combination of already being behind, the sudden extra workload, and the few weeks remaining made it difficult to implement all classes with a good design. By having another member, more time, and/or not falling behind is a few ways that this could have been prevented.

**Software reuse:**

Reflecting on the architecture and design of the Carre Game Engine, we feel that the design of each component represents the use of good software design patterns that are directly lead to how well the overall engine can be reused. In saying this, there is always ways to make it better and something to improve on in the future would be how modular each component is and its ease of use. An example of this would be, being able to removed the current physics engine oor input manager class and apply that to another engine. This is not to say it can't be done, just requires more under the hood work to be reused.