Report for IMAT3904 – Game Architecture – P16195518

**Introduction**

In this assignment the objectives were to: create and implement a fully functioning game engine that includes a way to import assets, scenes and models, either via Maya or another modelling software. Include design patterns such as components and interfaces that could be used to aid in the creation of a game from this engine. Use JSON, XML or Text files to aid in the management of the game data, scenes and game flow. Allow for user input to be able to be created. Finally create a basic demonstration of what the game engine can do when used.

**Text Files & Level Creation.**

To create levels, the engine reads the JSON files line by line in order to display on the screen what is written in the file. Each object in the file is given a name, a reference to which model is going to be used and the positioning of the game object. The user would also be able to add scale and orientation to the JSON file code if they wish.

To switch levels the user should usecSceneStateComp.cpp which allows for a simple input command to be created to allow for the user to switch levels in the game as the press of a button, which ever button they state in InputHandler.cpp. This allows for as many levels as the user wanted to include with as many buttons to change the levels as needed as long as they have JSON files to let the engine know what needs to be drawn and rendered onto each level.

**Input Handling.**

Within InputHandler.cpp the user can add input commands for the player to be able to interact with the game. This could mean being able to move objects using key presses, mouse movements etc. These commands can be implemented separately to different objects within the game as to add more complexity to games. To be able to implement new input commands, new classes would have to be created for each command and referenced within the execute class to determine exactly what the command is going to do and to what object on the screen. To Implement the commands the user must first setup the mapping of the controls in InputHandler.cpp, this sets a specific key to the command class that they have created. InputCommand.cpp tells the user what each command will do. PlayerCharacter.cpp allows the user to specify how they would like the player to move depending on what buttons they are pressing.

**JSON Files.**

Implementing JSON files is similar to text files except the code would read the scale, position and orientation of the models, using obj files, within the JSON file and creates it as a game object and displays a model on the screen, with the help of the model and mesh cpp files, rather than simple shapes line by line. These game objects get created onto the screen and pushed back as either environmental objects or player objects. JSON files are a lot more appealing to the user for game creation as they can easily add new objects onto the scene with the textures of their choice in the positioning that they want. More than one JSON file can be created and specified within the code, in this game engine you are able to switch between two levels and two different JSON files using the “-“ and “=” buttons. To import a model, from Maya for example, the user would have to open their model, execute the Python code as a JSON file and add the JSON file into the correct section of the code to allow for the model to be added to the game. The user is able to set the specified texture they want for the model within their 3D model software (MAYA)

**Testing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Before** | **After** | **Works Correctly?** |
| **JSON files produce exactly what the file states it should.** |  |  | **YES** |
| **Player can move with the current input commands set up.** |  |  | **YES** |
| **Level can be changed** |  |  | **YES BUT USER HAS TO PRESS “=” TWICE TO GET TO SECOND LEVEL. THIS IS A BUG THAT SHOULD BE FIXED.** |