**LOG**

Second Commit

*Description*

Added functionality to menu buttons, fixed bugs (program shuts down properly), added transition between game states (main menu / gameplay) and started working on terrain.

*Challenges*

1. Creating main menu and gameplay classes with as little connection as possible.

MMO games don’t always have a menu as part of the executable file, sometimes the game is run from a website. That is why I wanted to make the main menu and gameplay as independent from each other as possible. I have achieved this by implementing a state manager class which handles the transition between main menu and gameplay.

1. Passing input control and Ogre3D data between main menu and gameplay classes.

Because both of the classes need to manage input in their own way as well as require access to ogre I had to implement a way of passing this information between them. Passing ogre information was easily achieved by instantiating the ogre manager class inside the state manager and passing it as a parameter. However, that wasn’t possible with the input manager because it either had to be the superclass of the one in control or it had to pass the control to the class in charge. First I’ve tried to solve this by using inheritance and type casting during initialisation of the gameplay however that was a very bad idea and did not work. After that I’ve tried to stay with the inheritance but instead of passing the input manager to the next class I’ve only changed the class to which the listeners pointed. However, this created a strange behaviour during runtime which made the mouse jump during switching between main menu and gameplay. This was probably due to the movement being captured twice during the transition. Finally I have decided to go with the option I didn’t want to use due to it creating a connection between the menu and gameplay classes. I made the input manager responsible for passing control over input. In order to do this I had to create an interface class from which both menu and gameplay would inherit so that I could pass them as a parameter to the input manager.

Third Commit

*Description*

Added player character, movement and 1 terrain slot with skybox.

*Challenges*

1. Object inheritance hierarchy.

Since many different types of objects have similar or same functions or attributes it is a good idea to reuse the code for them. In order to do this I had to come up with a hierarchy that allowed different objects inherit shared functionality and attributes. Another reason for doing this was to create a system where any game character can be controlled using the same input code. In order to achieve this I have created a “MoveableObject” class that implements simple movement mechanics which can be then overridden by a subclass such as “PhysicsObject” for example. This kind of design will allow me in the future to pass control over characters between AI and players as well as create entities with different movement mechanics.

1. Unique IDs and Names.

Because I will be using the same code for creating many game objects, there was a problem of giving them unique names and IDs so that they don’t override each other or cause any errors. In order to do this I have created a base class “Object” which is responsible only for assigning unique IDs by storing the next available one in a static variable and incrementing it every time the constructor is called. I have then used this ID to create names for entities and nodes contained by the subclasses by converting it to a string followed by name of the object.

*Fourth commit*

*Description*

Started working on map editor, a place for my procedural code. At the moment it is able to create any size terrain and divide it into equally sized bmp files.

*Challenges*

1. Creating a height map

In order to create a decent height map for the terrain I had to learn the mid-point displacement technique. Over the course of writing the code for it I had trouble with making it reusable for any size of the terrain. At first the produced image didn’t have smooth transition from lower values to higher because I forgot to scale it into the 0-255 zone and when I converted the height values from floats to chars the values lower than 0 were changed into higher than 200. After I have solved that problem I started working on creating bigger terrain than 512 x 512, at first the terrain didn’t turn out how I expected as it looked like a grid of dots and spaces between them. As I’ve played with the values for creating the terrain the height maps changed however they still didn’t look how they should until I’ve noticed that treating the terrain as bigger than 512 by starting from the middle point of height/2 x height/2 created this weird behaviour. I’ve solved the problem by treating every size of terrain (512+) as 512 and starting the calculations by misplacing the 512 / 2 x 512 / 2 points first. In order to debug the code for creating the height map I saved all values of individual segments of the terrain into text files as looping through an array of size 5120 \* 5120 didn’t seem “fun”.

1. Saving the terrain to files

Because the terrain values were saved in a single dimensional float array I first had to convert them into RGB values and then write a code that saved them into bmp files as an (x,y) grid. I also had to learn how to write to said files by creating headers with information about the file.