

Games Programming 2

## 

## James O’Callaghan | M3I625656-19-A

## 

*I confirm that the code contained in this file (other than that provided or authorised) is all my own work and has not been submitted elsewhere in fulfilment of this or any other award*.

*Signature*. James O’Callaghan 09/01/20.

[Associated Classes 0](#_Toc29557300)

[CreateMesh. 0](#_Toc29557301)

[ImportShader. 1](#_Toc29557302)

[MainGame. 2](#_Toc29557303)

[Mesh\_loader. 3](#_Toc29557304)

[SoundManager. 3](#_Toc29557305)

[Texture. 4](#_Toc29557306)

[Window. 4](#_Toc29557307)

# Associated Classes

To begin we will have glance over the header files, the header files are used for declaration. This is where we can declare our variables, functions/methods as well as structs, allowing us to then go into the class file and this is where we define our functions and variables.

## CreateMesh.

Within the CreateMesh class we have our constructor, which is used on creation to initialize the class. At this stage within the constructor is where we could give our declared variables their value. Then we also have destructor that does what you would expect, the opposite of the constructor and destroys anything that is contained within memory for that given created class.

We can then have look at LoadMesh which takes in string which is the filename/location of the given mesh that we are trying to load. We declare a local variable to hold the information to allow us to draw our meshes to our screen but before that, we must initialize the mesh within initMesh here we use the drawCount variable that is unsigned int, which we use to specify the number of elements which be rendered and allows openGL to go through our drawCount sequentially. Once we had already generated “VAO” vertex array object, we bind our VAO to allow us to apply any operation that works on VAO to our bound VAO. So, once we bound our data declared our buffers, passed our data, move the data to GPU and told openGL what sort data we are working with and stored it to the GPU we can unbind our VAO. And finally, for the mesh we also add simple meshSphere which is a simple collider struct which allows us to give each our loaded mesh to have collider should we wish to have collision detection on those objects in the future.

Now that we have all the data in order to finally draw our object, we can look at our Draw function. Really there is not a lot going on here it takes our VAO and passes it to openGL so we know how much of our object will be getting rendered never forgetting to bind our data.

### ImportShader.

ImportShader constructor contains a lot more than the create mesh constructor so let’s have an explanation to what is going on, well to begin we generate a shader program to be used by the code. After we have the shader executable, we can then load our files in at which point we can look at what is going to happen, with the given data we have just got from our file? We make a call ShaderCreate which is a function that creates a shader using the file that we give in. Using this it creates an array of chars that is converted to a list of c-strings. Sending the source code to openGL, so openGL can compile the shader code, check before returning the shader to see if it has any errors. Remembering that maintaining the order of which order we are loading the shaders or shader to match our shader enumerator.

We also have destructor in this class that must remove the shader from the shader program allow us to then delete them in the appropriate order from the stack. We can then look at the bind function which is rather simple but very important because this is where bind our shader program to be used in the next render buffer. We then have the Update function which is used to determine how the mesh are displayed to the camera depending on the positioning, the rotation and scale, passing our data into the “MVP” Model View Projection.

The last two function we have left are importShader and ShaderErrorCall. Firstly, importShader is where we stream the shader files in using ifstream which parsing from a file. If we cannot import the file, we will send an error to the console. Finally, we have the ShaderErrorCall which in layman terms should we get an error then print which error message is appropriate and print to the console.

##### MainGame.

In the main game constructor, we set up gameState equal GameCondition PLAY which is basically telling us that the game is running and will continue running. We create our gameWindow which is our game display where we will be rendering our objects to allow for visualization, we then create an array of meshes that will hold our three required mesh for our basic prototype and an extra to implement a simple skybox. We have a destructor also in MainGame but this time it does nothing because we close the application using our GameCondition.

After which we have the run function which make call to initGameSystems which is where we set up our level by initializing our window to display on our screen, it then loads our sounds using our SoundManager class. Next, we load our meshes from our array of meshes using our CreateMesh class after which we initialize our camera which is using the MainCamera.h and then we declare counter to be one which is used to move our boats to test collisions.

Now that we have initialized our systems, we ready to start our UpdateLoop is function using a while loop when the GameCondition is not EXIT. GameCondition is either PLAY or EXIT, meaning that while the game is playing, we will loop continually through the following functions Input which is we check for any keyboard or mouse input from the player.

We also make calls to the draw function which is where we had little experimenting with what can be done using c++ and OpenGL. So, to begin it by clearing our gameWindow, we then import our shader and load all our textures and then we move onto our level generation. We began by water generation which handle with a for loop going through rows and collums that was set up in the header file to both equal sixteen. Given us a total gird size of two hundred and fifty-six which all has a water block drawn. After the water we generate our land which again using for loop but not for the whole gridsize only for a portion of it to generate a landmass. Then thought land without tree didn’t look that great so this where we are using another set of for loops to similar as the water and land but again a smaller portion of the gridsize but each time we generate part of the level we are moving the transform to allow for height given a little sense of height to the world. After we generated all the land mass and water, I placed to random boat just out on the water that collide and detect that collision then play a sound it isn’t pretty but this only base for the future possibilities. Finally, we swap the buffers.

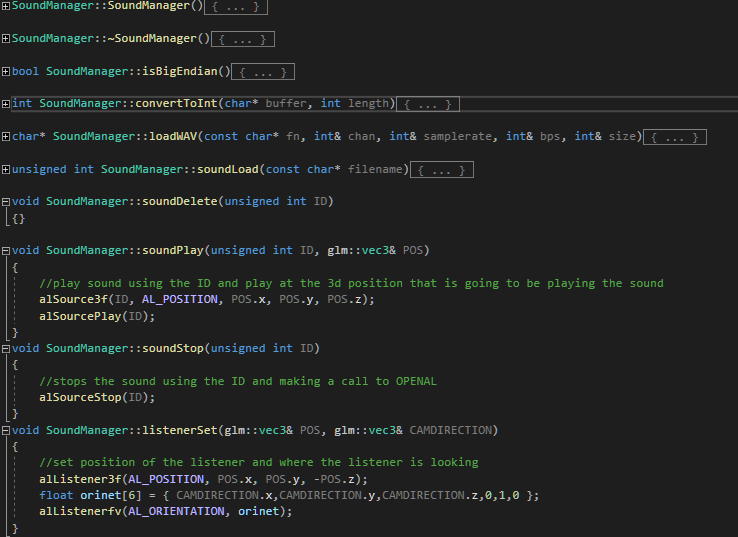
We then have collisDetect which is a bool that is used to detect when and if there is collision with object when checking for collisions. Which is done by using Pythagoras and check to see if and of our collider which just so happens be spheres so should there intersect with another the collision is set true else it is false. The last function within MainGame is playSnd which is used to play our background music or our collision sounds depending on which is making the call to the function.

###### Mesh\_loader.

The mesh\_loader is used in conjunction with our CreateMesh Class is this the brains of the whole parsing system being run in order to read our data from our obj file and allow us to make coherent data in order for us to pass it to OpenGL so we can allow us to use those meshes loaded in order to draw them to the screen. This is simplistic explanation for this class since this class could go well beyond page limit for each the classes so condensing it just felt smarter and more appropriate given what the class is being used for.

SoundManager.

With the SoundManager class we have same sort layout as all the other class constructor which sets everything up, destructor that clears everything and destroys it from the stack. If we get any errors, then send the errors. So rather than going through parsing sound file to convert to int so we can pass to the OpenAL there is attached image below showing our function considered needing an explanation.



Texture.

In our Texture class constructor, it starts by loading our image from file location, using the image code it then generates our data that can be bound, to be render the next frame. Wrap the texture using our outside width and height to set the parameters and the do our linear filtering by passing in float parameters for the texture. Confirm it is a 2D texture and clear the image.

Window.

Window constructor set up our sdlWindow to be a nullptr and then sets up our game resolution size using our declared floats. Then we have our destructor that removes our openGL contect from the program destroying the context and the deletes our sdl window and then finially making a call to the SDL\_Quit function. We have couple public function that allow us to get width and height of the sdlWindow. We also have returnError which is it says on the tin returns our error message and displays it to the consolel. Next, we have swapBuffer again this is rather simple swap the window buffers, then clearWindow clear the window using the parameters passed in to set the glClearColor and make a call to glClear using our color buffer and depth buffer. Lastly but not least we have initWindow which like all other init function it initialize the window to allow use in OpenGL.