**Andrew MacFarlane**

**Student ID: S1511223**

**Computer Games (Software Development)**

**Games Programming 2**

**Coursework Documentation**

*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*.

*Andrew MacFarlane*

**Contents**

**1.0 – Mesh Class:Page 3**

**2.0 – Shader Class:Page 4**

**3.0 – Texture Class:Page 6**

**4.0 – Transform Class:Page 7**

**5.0 – Creating the Camera:Page 7**

**6.0 – Audio Class:Page 8**

**7.0 – GameDisplay Class :Page 11**

**8.0 – GameScript Class:Page 12**

**9.0 References:Page 17**

**Code Description**

The aim of this report is to give an in-depth explanation of the code that is used to generate the game.

**1.0 – Mesh Class**

Mesh in the game are handled by the createMesh class which utilizes the initialise, loadInModel, initialiseModel, checkSphereStatus, getSpherePos and getSphereRadius Methods. These are declared in the createMesh.h header file. Additonally with in createMesh.h there is a class to store vertices: “storeVectices” and a struct for the sphere colliders “sphereCollider”

1.1 Initialise

In the mesh class the initialise method is used to initialise the mesh into memory it does this by adding the vertices and indices into the lists which store the positions, texture co-ordinates, indices and the normals into the instance of the indexed model class from “obj\_loader”.

1.2 initialiseModel

The initialiseModel method is called in the loadInModel method and is used to load the mesh from file. It generates a vertex array, binds the vertex array object and generates the vertex array buffers. For each of the buffers such as the position buffer, texture coordinate buffer and normal buffer the method binds its array buffer and then moves the data and other relevant metadata such as size of the data, the data type, to the GPU and stores this data in the GPU. The method then assigns it to a vertex attribute array and gives it a vertex attribute pointer. Finally, it unbinds from the vertex array object. This code can be seen in Figure 1.

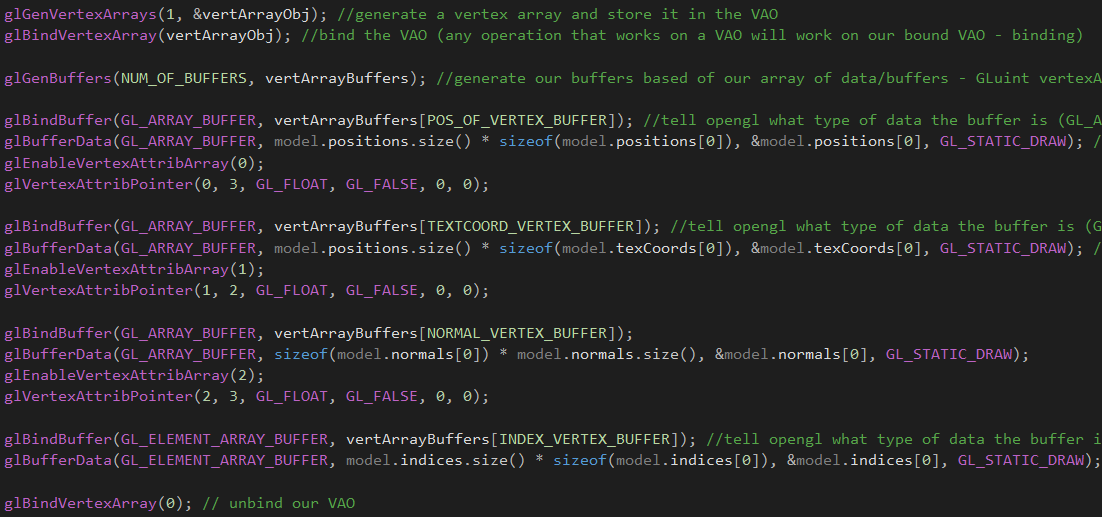


Figure1: Initialising the model

1.3 LoadInModel

The loadInModel method takes a file name and calls the initialiseModel discussed above in order to load to load the model into memory by assigning it to an instance of the Indexed model class. Additionally, the method also adds a collider to the model.

The destructor for this class “~createMesh” uses the glDeleteVertexArrays command to delete the mesh vertex arrays from memory this is called in the “GameScript.cpp” to delete the alien meshes when they are hit by a bullet.

When drawing a mesh in the game the DrawMesh method is called which binds the vertex array object and draws the triangles needed to create the mesh it then binds to the vertex array. This can be seen in Fiugre2.

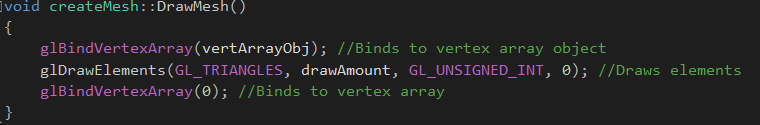


Figure2

1.4 checkSphereStatus

The checkSphereStatus assigns a position and radius to the colliders which are added to meshes in GameScript.cpp it takes in a vector3 position and float for the radius which are used in the broader Collison detection system.

**2.0 – Shader Class**

The games shaders are handed by the modelShader class. Its header file defines the BindShaders, Update, initialiseShader, LoadInShader,CheckForErrors and CreateShaders methods.

2.1-initialiseShader

This method is called when loading in the shader via a filename it creates a shader program and loads in both the vertex and fragment shaders and adds each of them to the shader program. The method then binds the shader program to an attribute variable, one for each of the buffers such as the position texture co-ordinate and meshNormal. It then uses the glLinkProgram command to creat and executable that will run on the GPU shaders. Additionally the method will check if the shader program is valid. An excert from this method can be seen in figure 3.

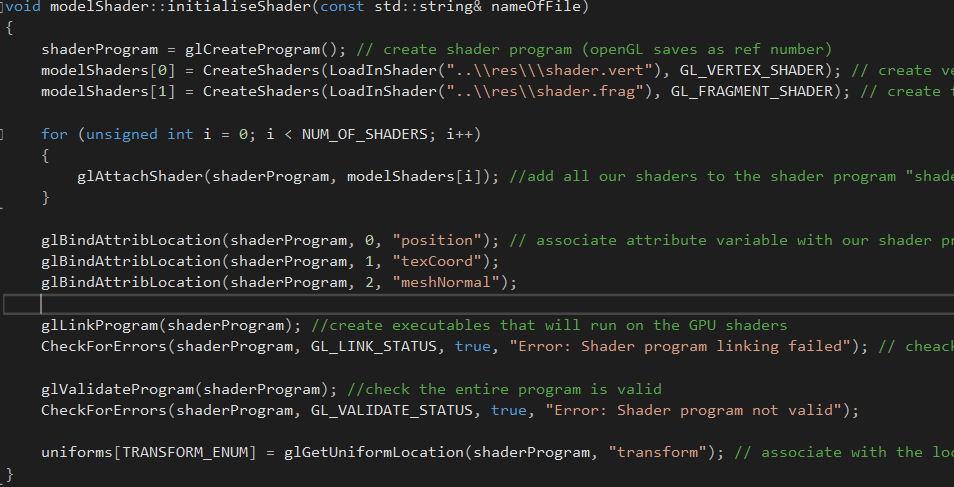


Figure3

2.2-Update

The update method takes in a transform and camera parameter and is used to update position of various game transforms relative to the game camera. It uses a matrix to store a representation of the view project from the camera and finds the passed in transform to find the model within that projection.

2.3-BindShaders

This method uses the glUseProgram command to run the shader program created in initialseShaders.

2.4- loadInShader

This method takes in a filename and loads the shader program from file. If the method is unable to read the file it outputs an error to the console

2.5 CreateShaders

This method is used to create the actual vertex and fragment shaders and is called in the initialiseShader method. It takes in a file path and a type which is then passed into the array of shaders used to create the shaderProgram. This method can be seen in Figure 4

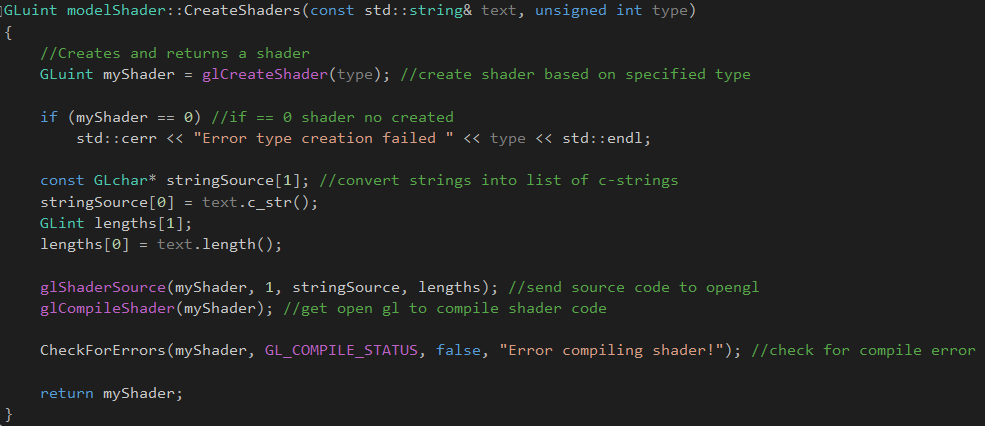


Figure 4

2.6 – Shader.vert and Shader.frag

These files are used to generate the vertex and fragment shaders and had to be changed significantly to in order to implement the lighting functionality. These can be seen in figures 5 and 6.

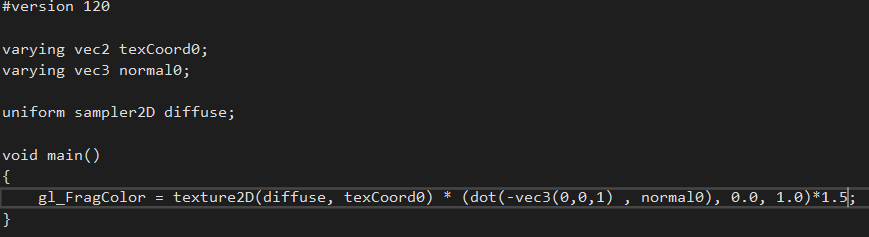


Figure 5: shader.frag

This shader was edited to multiply the fragment colour by the dot product of the simulated lighting direction and the give mesh’s normal. This is then again multiplied by 1.5 as to increase the effect.

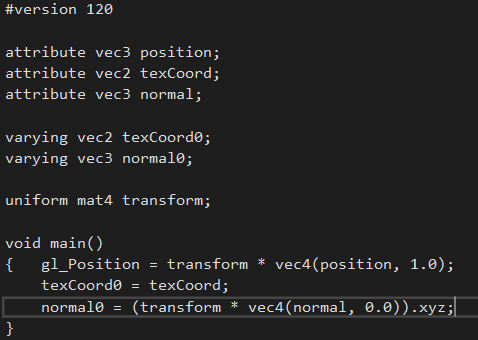


Figure 6: shader.vert

This shader was changed from the lab code to include a position for the normals which is cast to a vector3.

The combination of these changes to the lab shaders allowed the effect of increased lighting in the scene as well as a shadow effect being cast on to the alien meshes when the players ship moves to the edges of the screen.

**3.0 – Texture Class**

The texture class handles and stores the texture files which are applied to the games models.

3.1 initialiseTexture

This method is used to load the texture from file by taking in a file name and creates various pieces of metadata about the file such as its height and width. The method then bind the texture to array and wrap it to the relevant width and height and scales and maps the texture to the mesh. This method can be seen in figure 7.

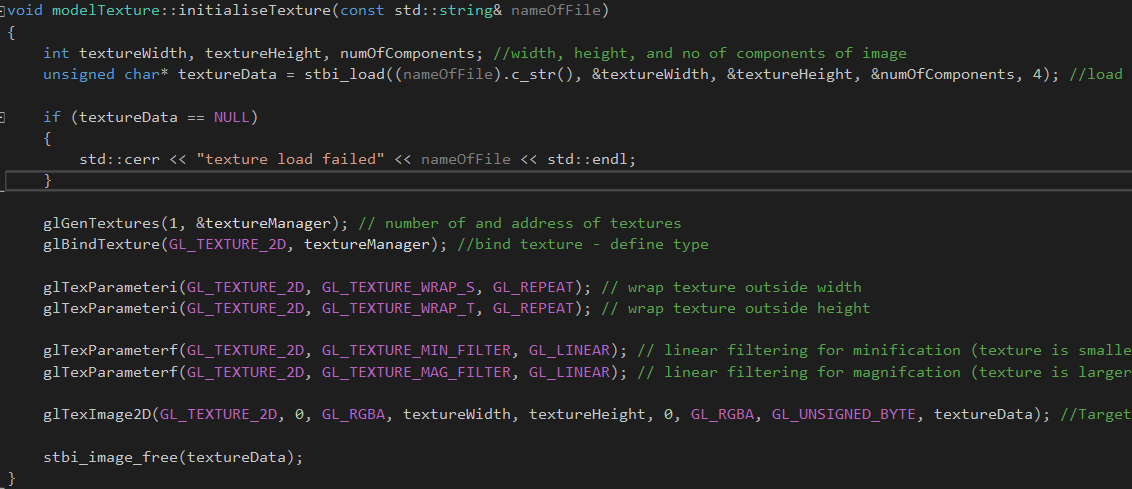


Figure 7

3.2 ~modelTexture

This classes destructor deletes the given texture using the glDeleteTextures command

3.3BindTexture

This method uses the glBindTexture command to bind the texture to the relevant location. Additionally it also sets the active texture unit with the glActiveTexture command.

**4.0 – Transform Class**

The transform class is used to store various information about an objects positioning and scale within the game world and is used for just about every object such as the player camera and skybox.

4.1 - Transform

This method is used to store the various information about an object such as its position, rotation and scale. This method can be seen in figure 8.

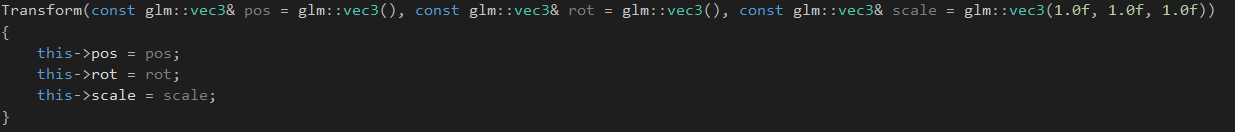


Figure 8

4.2 – Getters and Setters

These are used extensively in “GameScript.cpp” in order to access and manipulate a transform’s scale, position, and rotation. This is done by calling the various get and set methods such as “GetPos” or “SetPos”. An example of these getters and setters can be seen in Figure 9.

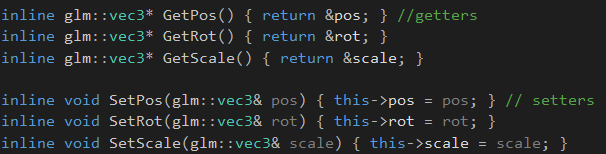


Figure 9

**5.0 - Creating the Camera**

The creation of the camera in the game is handled by the “createCamera” class. The camera is used to create a projection view of the scene and output it the game window.

5.1 – initialiseCamera

This method is used to assign various information about the camera such as its positon in the sceen the up and forward directions and its projection matrix. It also takes in values which define the near and far clipping planes, the camera’s field of view and its aspect ratio. This method is essentially used to create a camera in the scene. The code for this method can be found in figure 10.

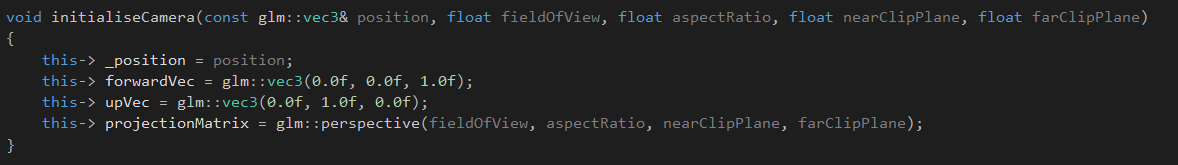


Figure 10

5.2 – GetViewProjection

This method is used to return the projection matrix from the direction in which the camera is looking, this direction is defined using the glm::lookAt command in which a position and directional vectors are given.

**6.0 – Audio Class**

This class “gameAudio” is used to handle all of the audio files by reading them from disk and allowing facility to play them at the appropriate time.

6.1 – Constructor: gameAudio

This method is used to open an AL device in the sound card and create and AL context. It checks to ensure an AL context has been created and then sets the current AL context to the context which it created. This method can be seen in Figure 11.

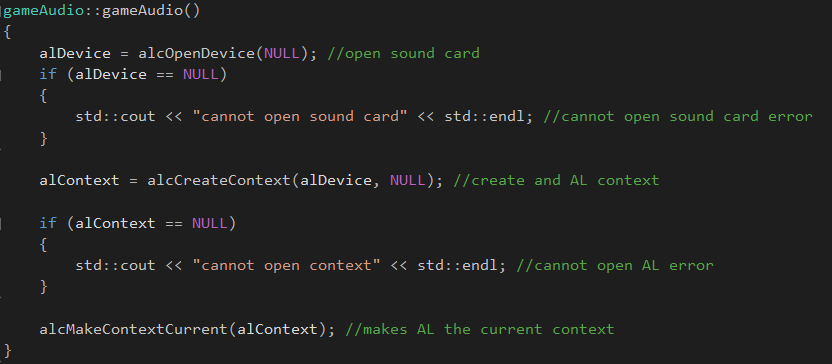


Figure11

6.2 – Destructor: ~gameAudio

This method deletes all the audio sources and buffers and destroys the AL context as well as closing the AL device. See figure 11.

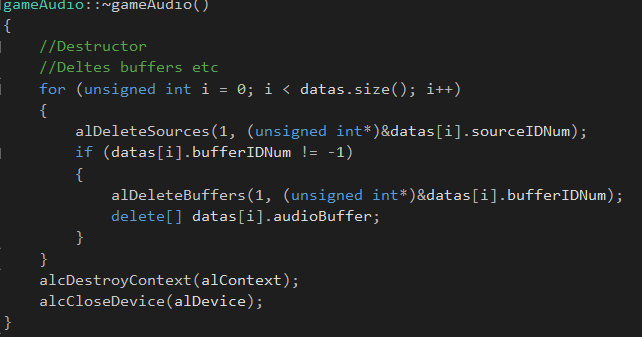


Figure 11

6.3 – loadWAVFile

This method reads in the actual .wav file from disk and is called in 6.4 to load in the sound effect. It loads from file using a variety of buffers to create soundData. See Figure 12

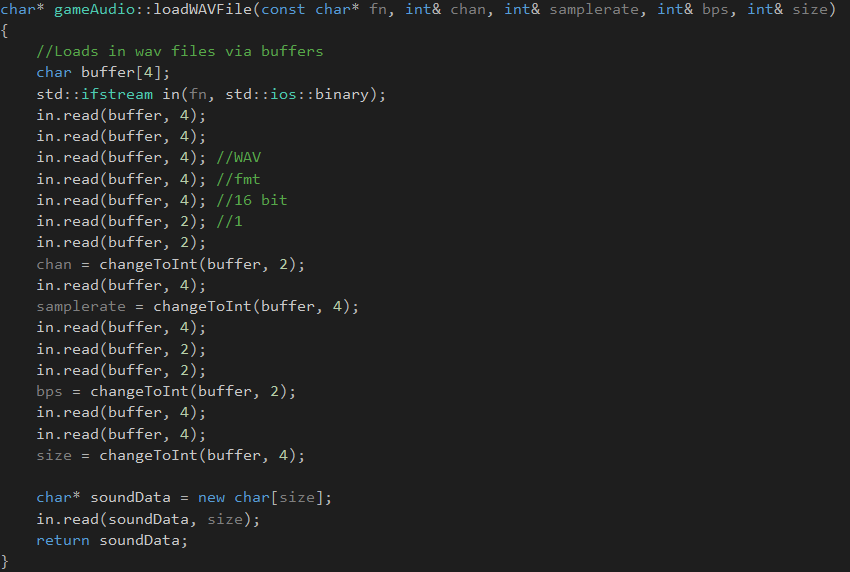


Figure 12

6.4 – loadInSoundFx

This method is called in the “GameScript.cpp” and uses 6.3 to load in the file, assign the soundData to an AL buffer and assign it a channel and format. This can be seen in figure 13.

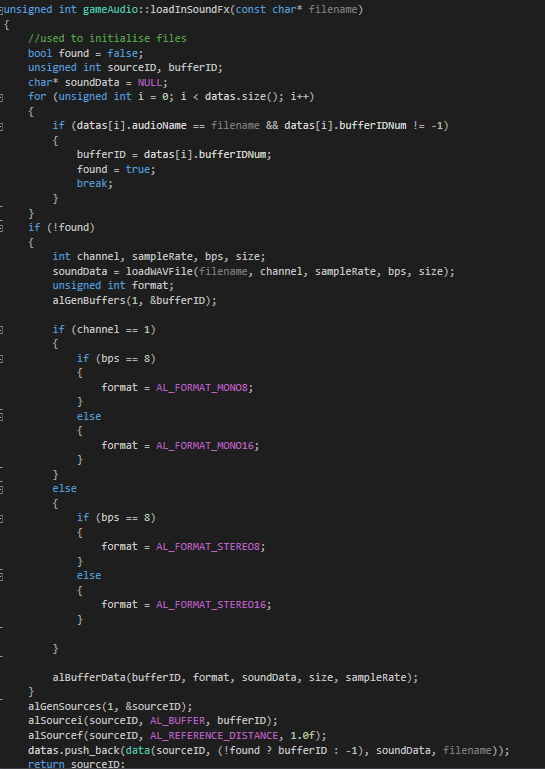


Figure 13

6.5-playSoundFx

This method is used to play the sound effect from a given vector 3 position within the game scene. Takes in an unsigned int to identify the audio clip and a vector 3 position for the sound to be played from.

6.6-setAudioListener

Used to set the position of the audio listener which will listen for the sounds made in the game scene and relay that information as output to the user.

**7.0 – DisplayGame Class**

7.1 – Constructor DisplayGame

This method is used to initialise the game window setting it width and height.

7.2 – Destructor ~DisplayGame

This method is used to delete the game window. It deletes the glContext the game window and exits the application.

7.3 – changeBuffers

This method is used to swap to the rendered buffer where the next frame is stored. This is done with the SDL\_GL\_SwapWindow command

7.4 – Start Display

This method is used to initialise the display. It sets attributes for red, blue and green colours in and 8 bit colour depth and sets up the double buffer. It then enables z buffering and sets up culling of mesh faces. This is done to the front of the meshes using the glCullFace command as this yielded the best result. It then clears the colour of the scene us glClearColor. This method can be seen in Figure 14

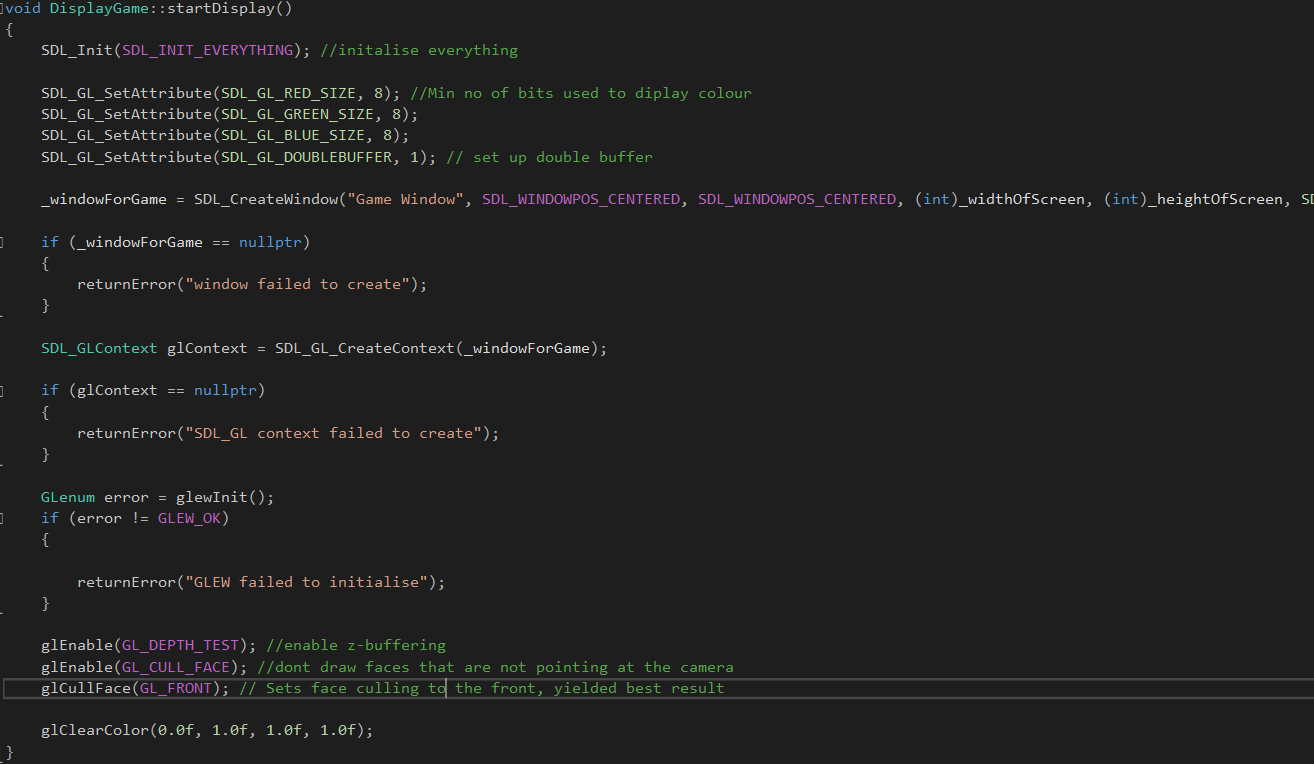


Figure 14

7.5-wipeDisplay

Used to clear the display in preparation for rendering the next frame it sets the glClearColor command to 0 and clears the colour and depth buffers.

**8.0 -GameScript class**

This script is where the bulk of the game aspects of the game are programmed including collision, the skybox user input and the various function calls used to load files from disk.

8.1 – runGame

Sets up the game by calling the startSystems class and createGameLoop class (See below).

8.2 – startSystems

Used to allow loading of resource files using various function calls for each type of file. This includes sounds, meshes, textures , shaders and camera. This is performed by using the appropriate loading method for each type. Also initialises camera. This method can be seen in Figure 15.

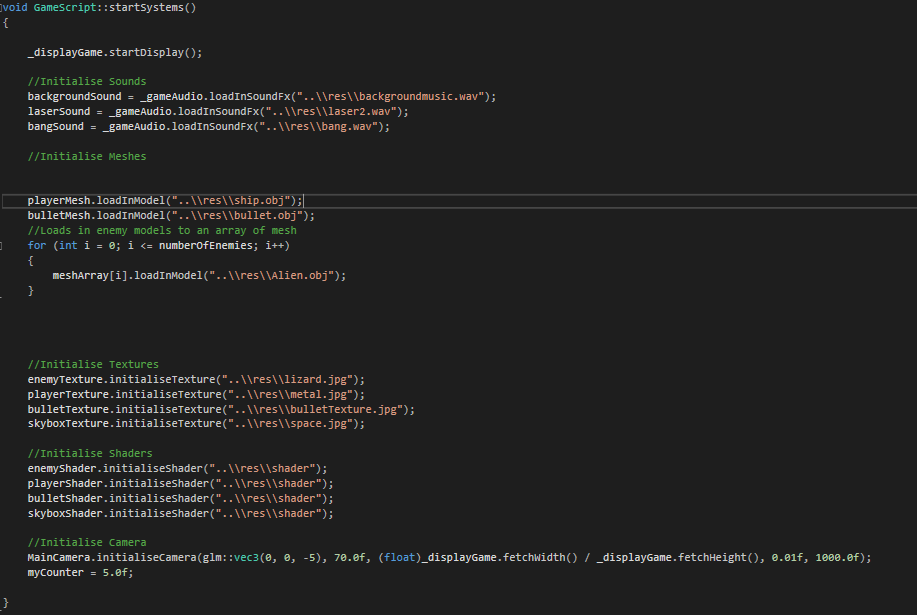


Figure 15

8.3 – createGameLoop

Runs every frame when the game state is playing it calls handleInput, CheckCollison, and renderGame.

8.4 – handleInput

Contains case statement which processes player input using (evnt.key.keysym.sym) system The various input in this system such as SDLK\_d for example allow the game to process which keys the player has pressed and respond accordingly. This method can be seen in Figure 16.

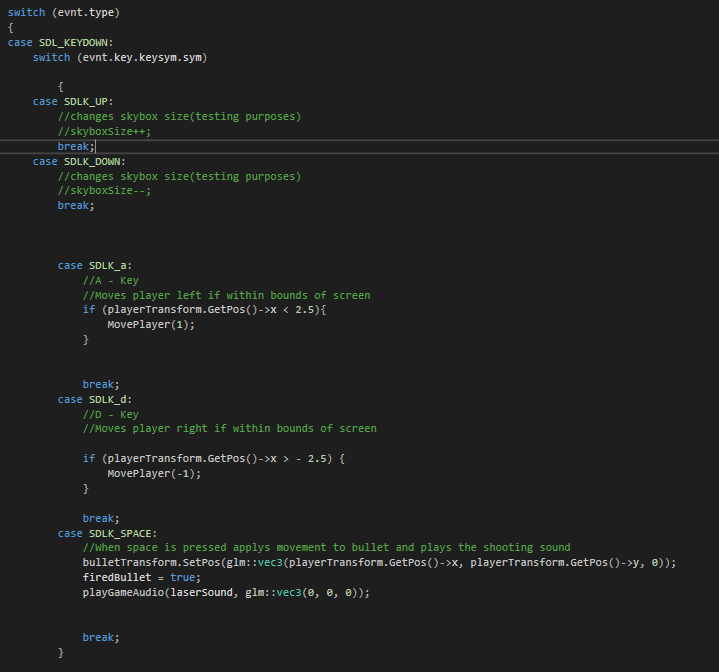


Figure 16

8.5 – sphereCollision

This method is used to determine if a Collison between the two objects which are passed in, has occurred. It calculates the distance between the two objects by using their transform positions and the radius of their respective colliders. Returns true/false depending upon wheteher or not a collison has occurred. This method can be seen in Figure 17.

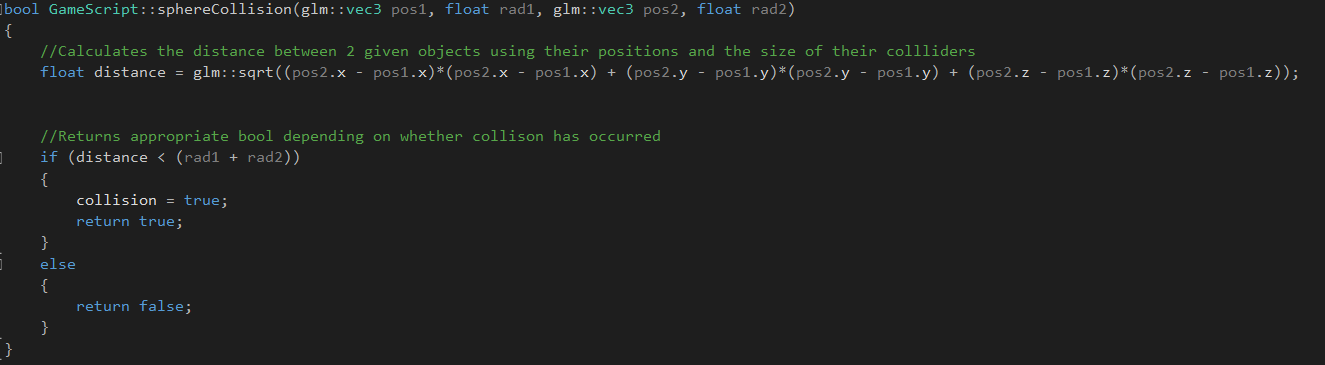


Figure 17

8.6 – playGameAudio

Called in order to play a sound effect when required by the game takes a sound clip and vector3 which defines the sounds point of origin.

8.7- renderGame

This method is called every frame and is used to draw the objects on the screen. It first wipes the last frame from the screen and then calls other relevant methods such as DrawEnemies which contains the code which draws the aliens to the screen. It then swaps the display buffers ready for the next frame. See Figure 18.

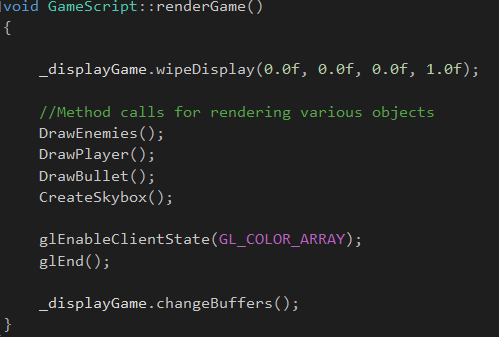


Figure 18

8.8 – MovePlayer

Takes in an integer which is multiplied by 0.25 and added to the x co-ordinate of the playerTransform in a set position command. This is called with the appropriate parameter when the user presses the a and d keys t move the player to the left or right of the screen.

8.9 – DrawBullet

Used to set the transforms and render the bullet if the bullet has been fired. This is monitored with the “firedBullet” Boolean which is set to true when the user presses the space bar. This is called in “renderGame” which allows for the rendering of the bullet. Also adds a collider to the bullet mesh See Figure 19.

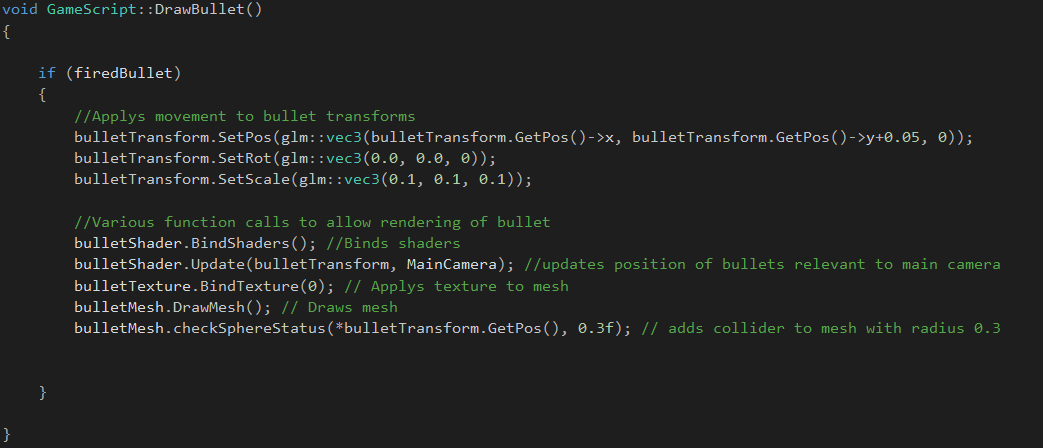


Figure 19

8.10 – DrawEnemies

Iterates through the array of enemy meshes to draw each alien at the top of the screen. Contains transform manipulations and render code for enemies which gives the aliens a spinning effect. Also adds a collider to each of the meshes. See Figure 20

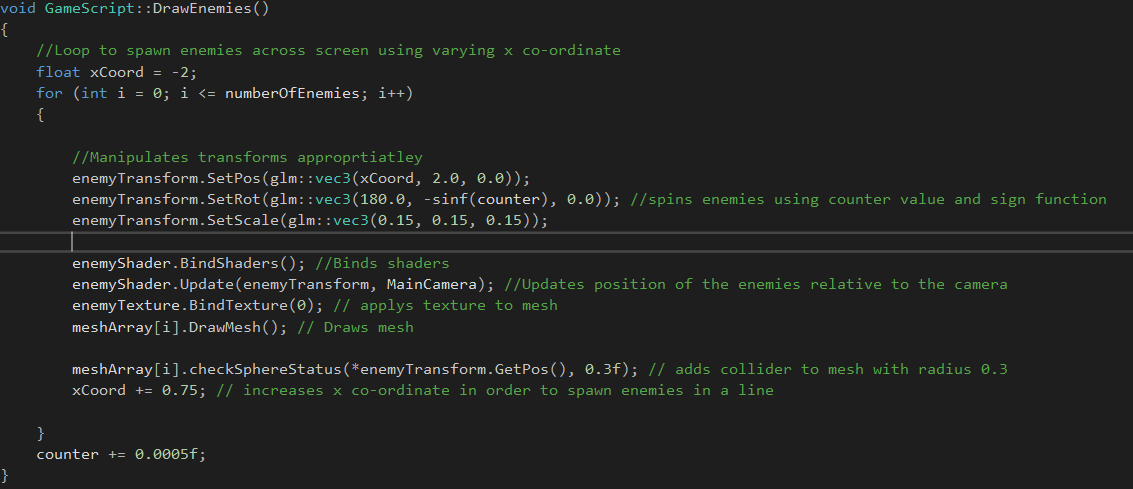


Figure 20

8.11 – DrawPlayer

Contains transform manipulations and rendering code for the player. Adds a collider to the players mesh. See Figure 21.

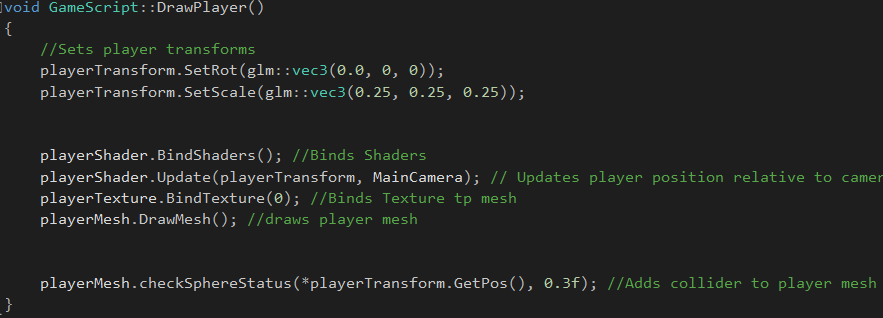


Figure 21

8.12 -CheckCollision

Checks for collision between the bullet and each of the enemies by iterating through the loop of enemy meshes. If there is a collision it will call that meshes destructor, deleting the mesh, and playing a bang sound. See figure 22.

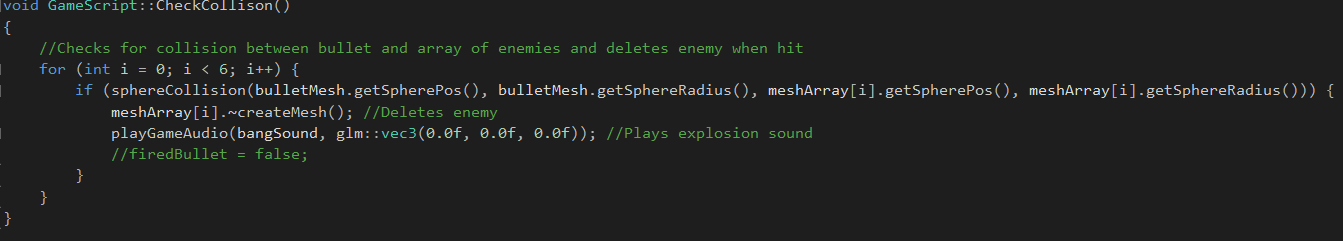


Figure 22

8.13 – CreateSkybox

Creates a skybox by creating vertices for each face and then rendering each face. Unfortunately, this does not work fully as the texture applied to each of the faces shows only as a solid colour. See Figure 23 for the code to create one of the faces.

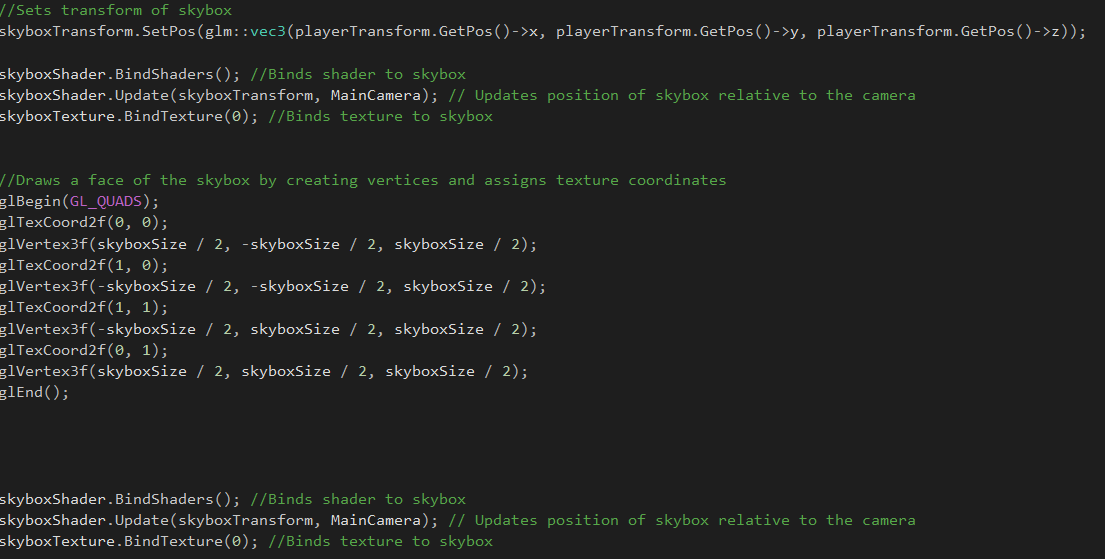


Figure 23

**9.0 - References**

<https://www.youtube.com/watch?v=imCNAWMC1Xs> : used to help create the skybox, adapted to fit architecture

<https://www.youtube.com/watch?v=NS980twY1ZE> : used to help create the lighting effect, shaders had to be changed to remove clamping from calculation as initial effect was very dark, changes to allow effect to fit into architecture