Games Programming 2

January 9, 2020

Michael Crainie

S1828588

2020

# Declaration

*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.*

*Michael Crainie*

# Display

## isClosed function

When the isClosed function is called it will return the value of the variable \_isClosed.

## clear function

A screen shot of a person

Description automatically generated

In the clear function it takes in 4 floats as arguments used to represent the red, green, blue and alpha values. The final line glClear is used to clear the colour buffer (GL\_COLOR\_BUFFER\_BIT) and fill it with the selected colour along with rendering textures that are in front of others (GL\_DEPTH\_BUFFER\_BIT).

## returnError function

A screen shot of a computer

Description automatically generated

returnError is used later on in this class with more defined error messages. It takes in a string as an argument. If returnError is called it will print the defined error message to the console followed by a prompt to press a key to quit. When said key is pressed it will run SDL\_Quit which shuts down all the subsystems of SDL.

## ~display function

The ~display function is the deconstructor for this class. When called it will simply delete the current game context, destroy the display window and then also run SDL\_Quit.

## update function

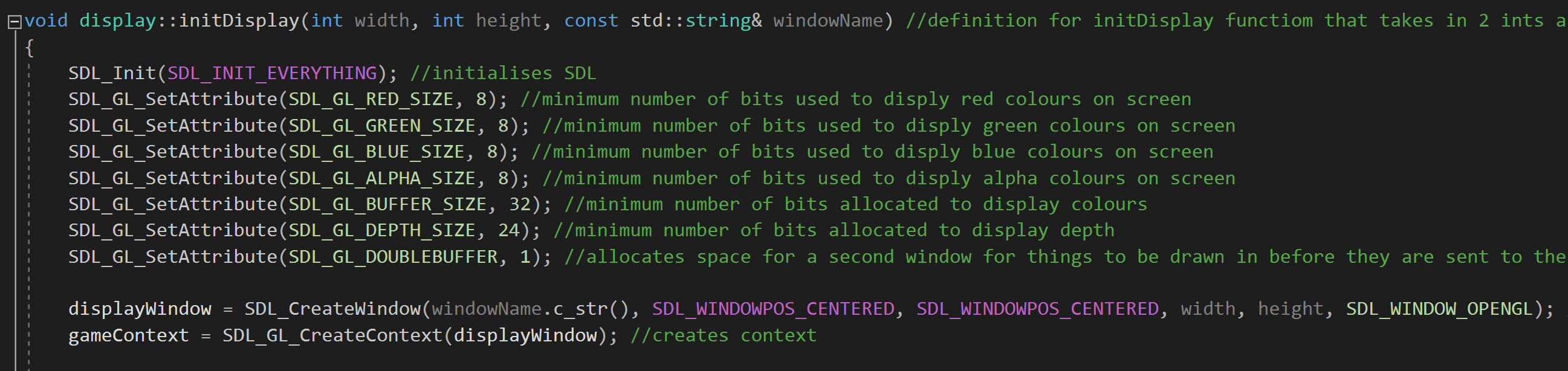
A screenshot of a cell phone

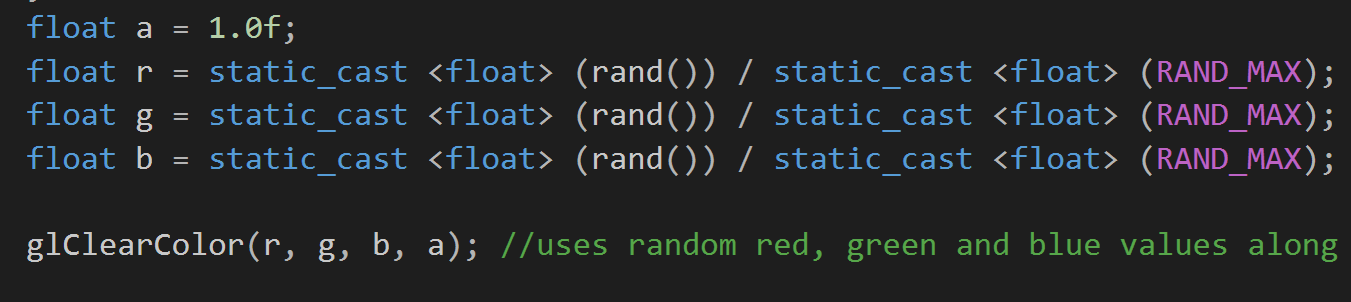
Description automatically generated

In the update function the buffer swap method occurs which takes what openGL has drawn in the buffer window and then swaps it to the display window when ready. An SDL\_Event named e is then created which is used in the while loop further to check if the event type SDL\_QUIT is called which then sets \_isClosed to true. The three lines glEnable(GL\_CULL\_FACE), glCullFace(GL\_BACK), glEnable(GL\_DEPTH\_TEST\_) stop the display from rendering things that are behind other objects which causes some weird visual bugs.

## initDisplay function

This function takes in two ints and a string reference as arguments. SDL is initialized in this function which is then followed by the minimum number of bits needed for each aspect of the window being allocated e.g. rgb colours, alpha, buffer size, double buffer, depth size.



After the bits have been allocated a window named display window is created along with a context named gameContext. Following this there are error checks that use the previously mentioned returnError function to check that a window and context are created and that GLEW initializes. The final part of this class calculates three random floats to be used in the glClearColour function along with a the with another float that is equal to 1.0f for the alpha. This results in the display window being a different colour every time upon running. 

# Shaders

## Shader function

The Shader function is the constructor and takes a const string reference as an argument. When this function is called a shader program is created along with two shaders (vertex and fragment). Vertex is used to specify position and fragment is used to specify colour. A for loop is then used to attach all of our shaders to the current project. Screen of a cell phone

Description automatically generated

glBindAttribLocation is then used to tell openGL which part of the data to read in as what variable in the shader program. Position is bound to attribute 0, texCoord bound to attribute 1 and normal bound to position 2 in the program.

A screenshot of a cell phone

Description automatically generated

The program is then linked and validated both of which are checked for errors using the checkShaderError function which will be covered further on. Access is then given to the program for the transform uniforms.

Screen of a cell phone

Description automatically generated

## ~Shader function

This is the deconstructor for the Shader class. A for loop is used to detach and delete every shader and then the program is deleted.

## Bind function

The program is binded in this function which means any reference to a program will affect the current program used in this function.

## Update function

This function takes in const Transform reference and a const Camera reference as arguments. Whenever this function is called the model view projection matrix is created, this is used to display where a model should be on the screen. The uniforms of the transform are then updated.

A screenshot of a cell phone

Description automatically generated

# Mesh

## Mesh function

A screenshot of a cell phone

Description automatically generated

This function is the constructor for this class and takes in a const string reference as an argument. Sets the model we wish to load into the game. The initialize mesh function is then called on this model and a Sphere named meshSphere is then created (collider).

## ~Mesh function

This is the deconstructer for this class and simply deletes the vertex array object

## InitializeMesh function

Takes in a const IndexedModel reference as an argument. Firstly, in this function the draw count is set to the number of indices the model (mesh) has. Space is then created for the vertexArrayObject. The vertexArrayObject is then bound which means any operation that would affect a vertex array object will affect this specific one.

A black sign with white text

Description automatically generated

glGenBuffers is then used to enable openGL to get data on the GPU and write to it. glBindBuffer is then used on vertextArrayBuffer[Position\_VB] so that any operation that would effect a buffer will effect this one. glBufferData is then used to get the size of the data going to the gpu for positions which is then told where to be stored by GL\_STATIC\_DRAW.

A close up of a screen

Description automatically generated

After this the first attribute in the array is enabled thanks to glEnableVertixAttribArray(0). This attribute is then accessed and it’s components are stored as floats by using glVertexAttribPointer(1, 2, GL\_FLOAT, GL\_FALSE, 0, 0).

A picture containing object, ball, hitting

Description automatically generated

This process is then repeated for TextureCoordinate\_VB, Normal\_VB and Index\_VB in order to enable different attributes until finally the vertex array object is unbound using glBindVertexArray(0). Note no attributes are accessed when doing this operation for Index\_VB

A close up of a screen

Description automatically generated

Find below the list of attributes from our basicShader.vert.

A screenshot of a cell phone

Description automatically generated

## Draw Function

glBindVertexArray object is used here again so that any operation that would affect a vertex array object will affect vertexArrayObject. glDrawElements is then used to specify what kind of shapes to use in the render and how many before we finally unbound vertex array object.

A screenshot of a cell phone

Description automatically generated

## updateMeshSphere

This function takes in a glm::vec3 and a float as arguments. This function is simply used to set the position and radius of the meshSphere.

# Texture

## Texture function

This is the constructor for this class and takes in const string reference as an argument. Initially three ints are made named height, width and numberOfComponents to pass into the stbi\_load. This is followed by another variable named image data being created which is an unsigned char pointer. This variable is then made to equal to stbi\_load which enables us to load texture data. Lastly an error check is performed to make sure the image data loaded in correctly and is not null.

A screenshot of a cell phone

Description automatically generated

Following this glGenTextures is used to generate space for the texture, this texture is then binded using glBindTexture which means any operation affecting a texture will affect this one. Next glTexParameteri is used twice to inform openGL what to do with the texture if a pixel is out with the texture bounds in terms of height and width. In this case we repeat the texture. This is followed by glTexParameterf being used twice in order to instruct openGL to do if a texture takes up more pixels than the resolution. In this case the texture scaled appropriately. The image data is then sent to the GPU using glTexImage2D and then finally the texture data is deleted when no longer in use thanks to stbi\_image\_free.

A picture containing wall, monitor, indoor, screen

Description automatically generated

## ~Texture function

This is the deconstructor for this class and is used to simply delete the texture using glDeleteTextures

## Bind function

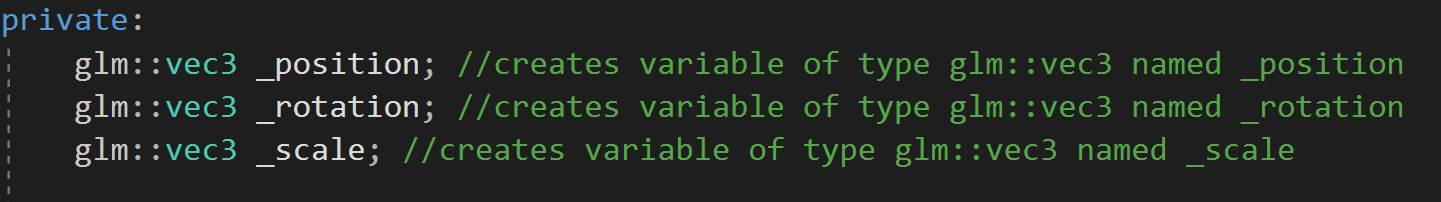
This function takes in and unsigned int as an argument. Firstly, in this function we use assert in order to keep the unsigned int within the desired scope (0 – 31) otherwise an error would occur. Following this glActiveTexture is used order to specify which texture openGL will work with. The texture is then bound.

# Transform

This class does not contain a cpp and is only a header file. The constructor for this class takes in three const vec3 references named position, rotation and scale. Scale is defaulted to (1.0f, 1.0f, 1.0f). The variables of \_position, \_rotation and \_scale are then initialized. Following this we create five different 4x4 matrixes. These are the position matrix, x rotation matrix, y rotation matrix, z rotation matrix and the scale matrix. Following this we create the rotation matrix by multiplying the x, y and z matrixes. The final rotation matrix is then returned. After this we create both getters and setters for our position, scale and rotation. In the private section we simply create the previously mentioned \_position, \_rotation and \_scale variables that are of type vec3.

A screenshot of a cell phone

Description automatically generated



# Camera

The Camera class also does not contain a cpp and is just a header file. The constructor takes in a const glm::vec3 reference which is the position and four floats which are the field of view, aspect ratio, near plane and far place. We then set \_perspective to glm::perspective and take the 4 previously mentioned floats in order to produce what the camera shows. This is followed by \_position being defined as whatever the position value passed into the function is. Next \_forwardVector is set to glm::vec3(0, 0, 1) in order to set which direction is forward. The two variables height and width are then set to 800.0f and 600.0f respectively. This is followed by the calculation of the view projection matrix. Finally in the private section the \_perspective, \_position, \_forwardVector, \_upVector, height and width variables are created.

A screenshot of a cell phone

Description automatically generated

# Game

## Game function

This is the constructor for the game class where we set \_gameState to GameState::PLAY and then create a display named mainDisplay().

## run function

In this run function we call initsystems() and also gameLoop()

## initSystems function

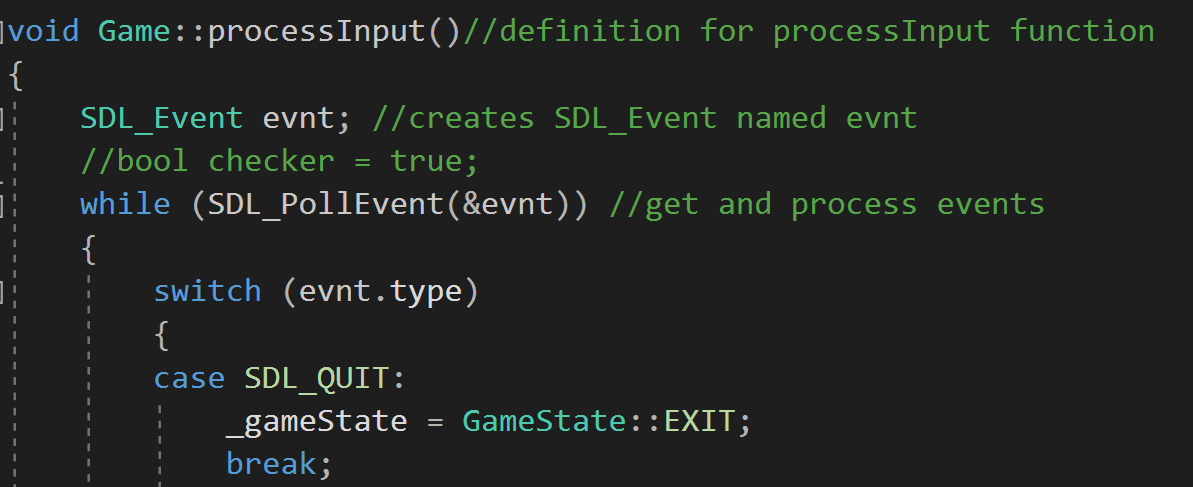
In this function instructions on how to play the game are sent to the console for the player on how to play the game. The initDisplay function from mainDisplay is called taking in 800, 600 and “window” as arguments. Three wav files are then loaded in using audioDevice.loadSound. These three files are for the background music, collision sound and firing sound.

## gameLoop function

In this function aslong as the variable \_gameState does not equal GameState::EXIT we will call the drawGame() and the processInput function.

## processInput function

An SDL\_Event named evnt is created to start off this function. SDL\_PollEvent is then used with a reference to evnt in order to get and process events. A switch statement is then used to check if the event SDL\_QUIT is called and if it is, \_gameState is set to GameState::EXIT.



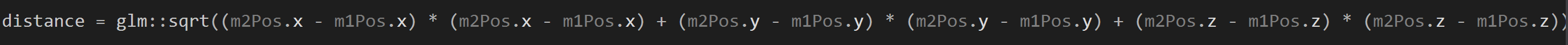
This is followed by the key down case which checks if the user inputs either the a, d or space key. A will cause the player to move to the left, d will cause the player to move to the right and space will fire a missile.

A picture containing indoor

Description automatically generated

## Collision function

The collision function takes in two floats and two glm::vec3’s which represent the position and radius of a mesh’s collision sphere. A float named distance is created in order to check the distance between two meshes which is calculated using Pythagoras.



If distance is less than the combined radius of the two meshes then information is output to the console, the collision audio plays using the playAudio function at m1Pos, the monkey spaceship loses one health, fire is set back to false enabling the player to fire again and the missile is returned to it’s starting position of -3.55f on the Y-axis

## playAudio function

This function takes in an unsigned int and a glm::vec3. It will play the specified Wav file aslong as it is not already playing.

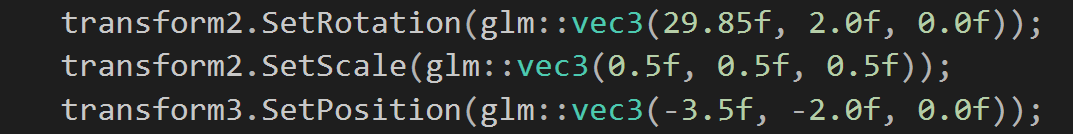
## drawGame function

The draw game function begins by creating our meshes, shader, textures, camera and transforms and then specifying which files to load in to correspond with these if they require a file type.

A screenshot of text

Description automatically generated

After this we create a while loop that runs aslong as the variable isClosed from the Display class is not equal to true. This while loop is where the transform variables for each mesh are set along with where we bind a set texture to a mesh. After the texture is bound the shader is then bound then the shader’s update method is called to get the model view projection matrix. After these steps the mesh is then drawn to our camera and it’s meshSphere (collider) is then updated.



A screenshot of a cell phone

Description automatically generated

This while loop also contains instructions for the monkey head mesh (mesh2) to move along the screen and drop down upon reaching an edge like in space invaders. Instructions for the monkey head mesh being destroyed by the player or reaching the bottom of the screen are also in this while loop. Contained in here is also instructions for the missile mesh (mesh 5) if it misses its target and goes off the screen. The collision function also checks if mesh5 and mesh2 are colliding while this loop is running. The processInput and mainDisplay.update functions are also called in this loop near the end.

A screen shot of a clock

Description automatically generated

# Main

The main is the starting point for our application and it very straightforward. Srand is used to enable the generation of random colour. Game game is used to create a Game object named game and then finally before returning 0 the run function from game is called.

A screen shot of a computer

Description automatically generated