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 – Ruari McGhee

Coursework

Games Programming 2 - M3I625656-19-A

Ruari McGhee – S1432402

Table of Contents

[Main Game 1](#_Toc29552586)

[Class Description 1](#_Toc29552587)

[Class Behaviour 1](#_Toc29552588)

[Setting Up Game 1](#_Toc29552589)

[Running Game 1](#_Toc29552590)

[Display 2](#_Toc29552591)

[Class Description 2](#_Toc29552592)

[Class Behaviour 2](#_Toc29552593)

[Setting Up Display Devices 2](#_Toc29552594)

[Double Buffer Rendering 2](#_Toc29552595)

[Game Object 3](#_Toc29552596)

[Class Description 3](#_Toc29552597)

[Class Behaviour 3](#_Toc29552598)

[Constructor 3](#_Toc29552599)

[Component Handling 3](#_Toc29552600)

[GameObject Implementations 4](#_Toc29552601)

[Transform 5](#_Toc29552602)

[Class Description 5](#_Toc29552603)

[Class Behaviour 5](#_Toc29552604)

[Position 5](#_Toc29552605)

[Rotation 6](#_Toc29552606)

[Parenting 6](#_Toc29552607)

[Game Scene 7](#_Toc29552608)

[Class Description 7](#_Toc29552609)

[Class Behaviour 7](#_Toc29552610)

[Scene GameObjects 7](#_Toc29552611)

[Handling Scene Information 7](#_Toc29552612)

[Scene Manager 8](#_Toc29552613)

[Class Description 8](#_Toc29552614)

[Class Behaviour 8](#_Toc29552615)

[Changing Scene 8](#_Toc29552616)

[Audio Manager, Font Manager, Mesh Manager, Shader Manager, Texture Manager 9](#_Toc29552617)

[Class Description 9](#_Toc29552618)

[Class Behaviour 9](#_Toc29552619)

[Storing Loaded Items 9](#_Toc29552620)

[Mesh 10](#_Toc29552621)

[Class Description 10](#_Toc29552622)

[Class Behaviour 10](#_Toc29552623)

[Loading A Model 10](#_Toc29552624)

[Changes to Obj\_Loader 10](#_Toc29552625)

[Shader 11](#_Toc29552626)

[Class Description 11](#_Toc29552627)

[Class Behaviour 11](#_Toc29552628)

[Loading a Shader From Resources 11](#_Toc29552629)

[Shader Lighting 11](#_Toc29552630)

[Time 12](#_Toc29552631)

[Class Description 12](#_Toc29552632)

[Class Behaviour 12](#_Toc29552633)

[Calculating Delta Time 12](#_Toc29552634)

[Information Accessibility 12](#_Toc29552635)

[Input Manager 13](#_Toc29552636)

[Class Description 13](#_Toc29552637)

[Class Behaviour 13](#_Toc29552638)

[Input Detection 13](#_Toc29552639)

[Information Accessibility 14](#_Toc29552640)

[Component 15](#_Toc29552641)

[Class Description 15](#_Toc29552642)

[Class Behaviour 15](#_Toc29552643)

[Constructor 15](#_Toc29552644)

[Update 15](#_Toc29552645)

[Ball Controller 16](#_Toc29552646)

[Class Description 16](#_Toc29552647)

[Class Behaviour 16](#_Toc29552648)

[Respawn 16](#_Toc29552649)

[Scene Changing 16](#_Toc29552650)

[Camera 17](#_Toc29552651)

[Class Description 17](#_Toc29552652)

[Class Behaviour 17](#_Toc29552653)

[Constructor 17](#_Toc29552654)

[Update 17](#_Toc29552655)

[Camera Mount 18](#_Toc29552656)

[Class Description 18](#_Toc29552657)

[Class Behaviour 18](#_Toc29552658)

[Following another Gameobject 18](#_Toc29552659)

[Environment Controller 19](#_Toc29552660)

[Class Description 19](#_Toc29552661)

[Class Behaviour 19](#_Toc29552662)

[Environment Movement 19](#_Toc29552663)

[Physics Handler 20](#_Toc29552664)

[Class Description 20](#_Toc29552665)

[Class Behaviour 20](#_Toc29552666)

[Velocity 20](#_Toc29552667)

[Torque 20](#_Toc29552668)

[Collision Detection 20](#_Toc29552669)

[Collision Resolution 21](#_Toc29552670)

[Mesh Renderer 22](#_Toc29552671)

[Class Description 22](#_Toc29552672)

[Class Behaviour 22](#_Toc29552673)

[Renderer Setup 22](#_Toc29552674)

[Rendering Object 22](#_Toc29552675)

[Font Renderer 23](#_Toc29552676)

[Class Description 23](#_Toc29552677)

[Class Behaviour 23](#_Toc29552678)

[Renderer Setup 23](#_Toc29552679)

[Rendering Text To Screen Space 23](#_Toc29552680)

# Main Game

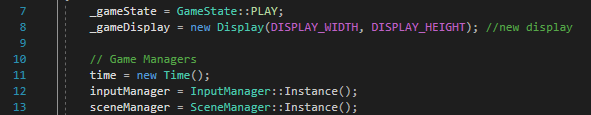
## Class Description

Handles Initialising the game display and managers, also handles running the game loop

## Class Behaviour

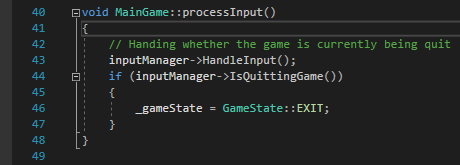
### Setting Up Game

To properly set up the various elements that the game needs the main game class handles it so that they are handled at the beginning of the game’s lifetime. The class handles instantiating the managers and creating a display for the game to render to.



### Running Game

The main game class uses an instance of a game state enum to compare whether the game is playing or closed. While the enum is set to playing then the class uses the previously instanced scene manager to find what scene the game is currently in and runs it. If the game is quitting class breaks out of its run loop and reaches the end of the int main and the program ends.

To change the game state enum the class uses the previously instanced input manager and checks if the current event is to quit the game. This was changed from the previous implementation the input manager needs to be the object that consumes currently pending user inputs. 

The other aspect of the game that the loop handles is to properly update the time class so that the time between frames can be calculated by it.

# Display

## Class Description

Sets up SDL and Creates a viewport for the game to render to

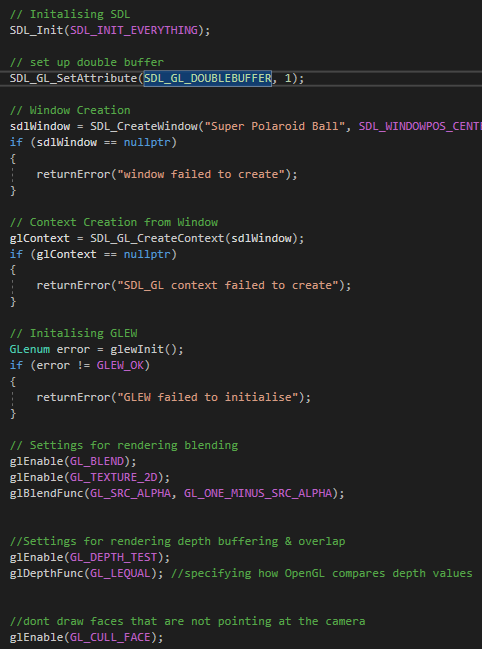
## Class Behaviour

### Setting Up Display Devices

The display’s width & height are defined so that other classes don’t need a reference to the display class to access this information



When creating a window, the class also sets up different settings for handling rendering later down the line.



### Double Buffer Rendering

Allows the program to prevent flickering on the screen by rendering on the back buffer an only moving the back buffer to the front when its fully renderer

# Game Object

## Class Description

A base from which a game entity is built by adding different components

## Class Behaviour

### Constructor

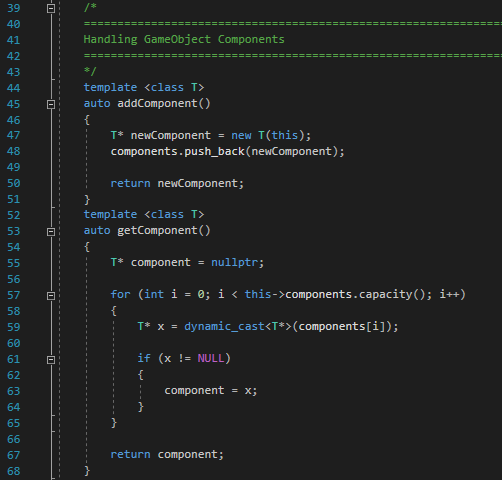
Gameobjects require a reference to the scene that they are in so that if a component of the gameobject needs other information from that scene then it has a path to do so. To avoid circular dependencies the class uses forward declaration to declare the GameScene class again.

### Component Handling

Each gameobject uses a unity inspired component pattern to handle adding additional behaviours to the gameobject

#### Adding & Getting Components

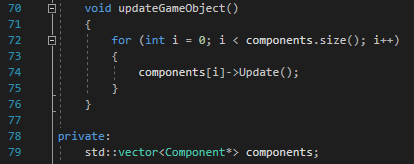
The methods for adding and getting components use generic types so that the user just needs to declare the requested type to the gameobject and the gameobject itself can handle the instantiation of the new component.



When adding a new component, the method also returns the newly added component through the use of the auto keyword so that the user can have a reference to it to change variables.

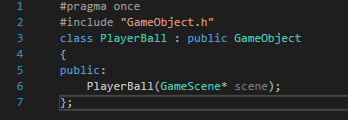
#### Updating Components

Each GameObject stores it’s added components in a vector. This makes it easy for the gameobject to iterate through it’s added components and call the update method on each component

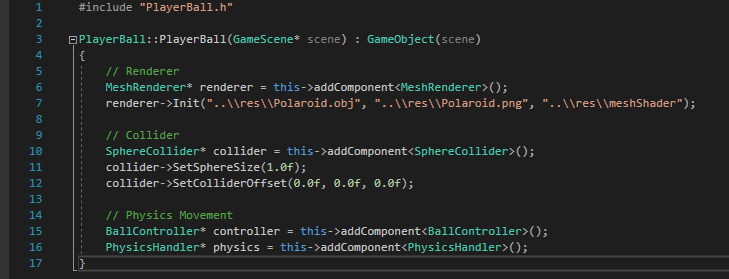


### GameObject Implementations

Subclassing from the GameObject class allows the user to create premade GameObjects for GameScenes to implement



The subclassed gameobject uses its constructor to set up any components that the gameobject may need.



# Transform

## Class Description

Stores the object’s position and rotation values, also allows for the gameobject to have a parent which allows the gameobject to inherit its parent’s transform values

## Class Behaviour

### Position

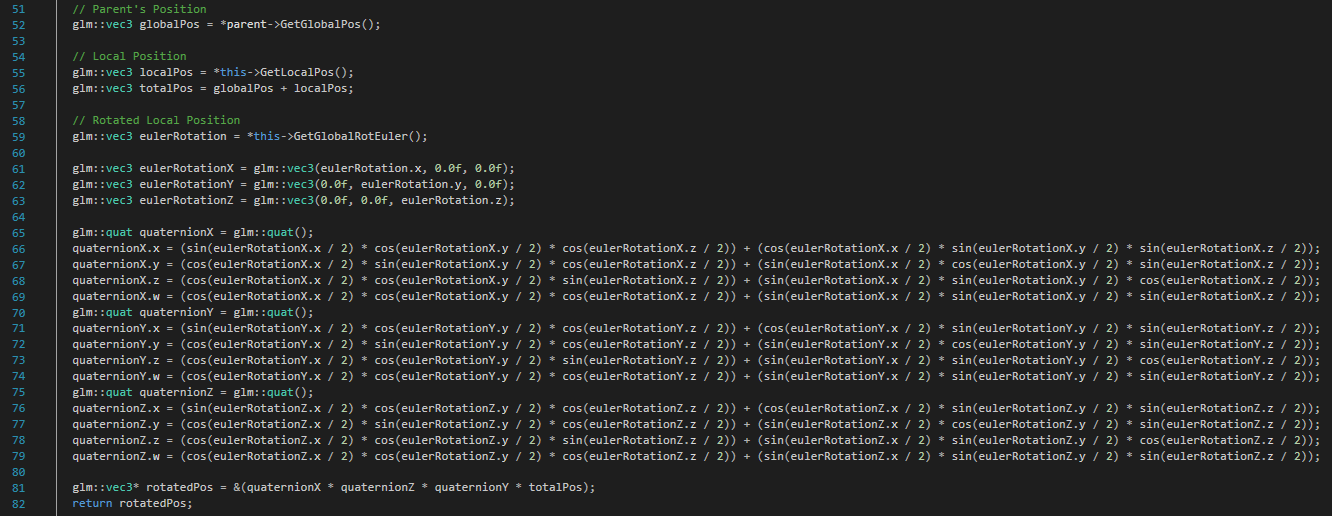
#### Local Position

Represented by a vector 3 to show the gameobject’s position relative to its parent. Most gameobjects use it to represent its position however UI based gameobjects can use it to represent their position in screen space (based on the screen’s width & height)

#### Global Position

If the gameobject has a parent, then the object’s new local position is calculated by getting the parent’s global position. If the gameobject has no parent, then the object’s local position will be the same as its global position (the gameobject’s parent could be considered as being the gamescene’s origin)

The position also needs to be rotated so that it take’s the parent’s rotation into account. Instead of getting this object’s rotation as a quaternion directly the rotation is instead gathered from it’s Euler form and a quaternion is created for each the Euler’s X, Y & Z values. This is done manually so that the order of multiplication can be controlled (Y then Z then X)



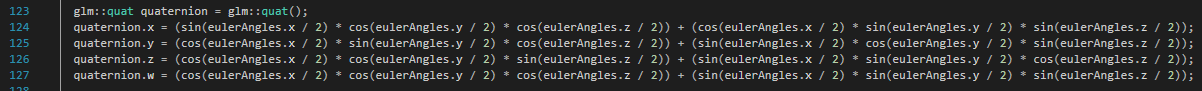
### Rotation

#### Euler Rotation

Represents the rotation of the object in radians, when getting the global Euler rotation, the transform class is simply able to add the parent’s global Euler rotation and this object’s local Euler rotation together.

#### Quaternion Rotation

Represents the object’s rotation as a quaternion, Both the global and local getters are manually calculated from the relevant Euler rotation value using the following equation.



### Parenting

When an object’s parent is changed the class recalculates the object’s position and rotation values so that they are relative to its new parent

# Game Scene

## Class Description

A container for holding the multiple gameobjects required to build & run a level for the game.

## Class Behaviour

### Scene GameObjects

Each scene contains a vector of GameObjects

When creating a new gameobject in the scene it must be added to this vector so that the scene can correctly update it throughout the scene’s lifetime

### Handling Scene Information

Some GameObjects need to access information from other gameobject’s in the scene

An example of this is when the physics handler class needs to know all the box collider’s in the scene, the scene itself handles getting this information and then can pass It to the physics handler

# Scene Manager

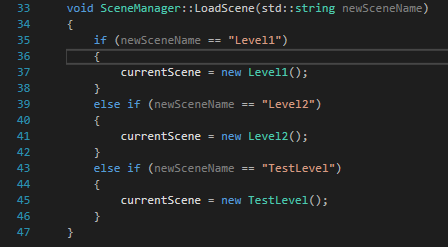
## Class Description

Implements a singleton pattern to ensure only one instance on the class exists in the project. For this reason, the constructor is also kept private so that the only entrance point into the class is through the Instance method.

## Class Behaviour

### Changing Scene

Any class that has a reference to the game’s scene manager can change the game’s current scene. To keep those classes from needing to know about the possible GameScene implementations this is handled by the LoadScene method only needing a string to determine what scene to move to.



# Audio Manager, Font Manager, Mesh Manager, Shader Manager, Texture Manager

## Class Description

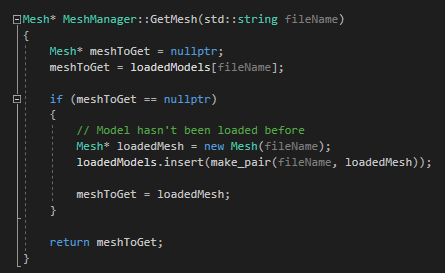
Each manager handles the storage of a type of loaded items, they also ensure that items are only loaded once to optimize memory usage.

Like the SceneManager these manager’s implement a singleton pattern to ensure that only one instance of each class exists in the project.

## Class Behaviour

### Storing Loaded Items

Each manager has a map to store loaded items, when an object requests to load a new item from resources the managers first checks to see if the item has already been loaded.

If not, then the manager loads the item and then stores it in the map by using its file path as the key. 

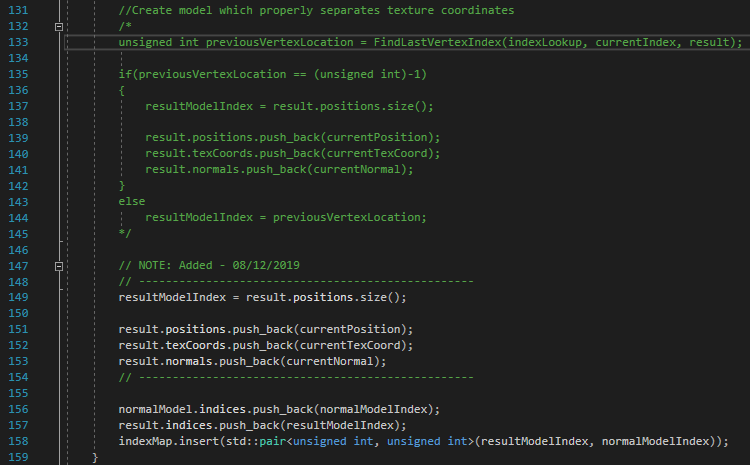
# Mesh

## Class Description

Handles loading a model file

## Class Behaviour

### Changes to Obj\_Loader

Having the texture coordinates separated out often caused problems when loading new meshes if that mesh had a vertex in the same position as another one. The indices would end up referencing the previous one in the mesh and due to the usage of findlastvertexindex the model would resume from the last index. After this point all the indices in the model would be 1 off so rendering faces would be messed up.

By ignoring the FindLastVertexIndex check the model would have indices explicitly made for each vertex, this made it so that the obj loader was more consistent when loading in meshes from the resources folder

# Shader

## Class Description

The Shader class handles loading shaders from the resources folder to create a shader program

## Class Behaviour

### Loading a Shader From Resources

Since a shader is made from a fragment shader and a vertex shader the parameter used for loading the shader file ignores the file type.



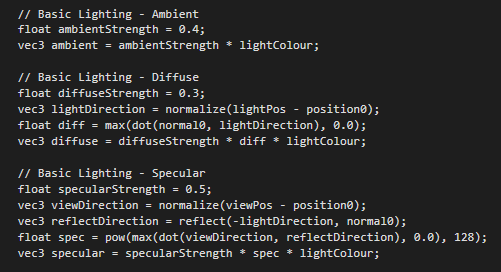
### Shader Lighting

The fragment shader handles getting the color of the object and the lighting affecting it. Lighting is split into 3 parts.

Ambient light – a universal color applied to the object

Diffuse Light – Lighting that is created from a source and affected by the object’s normal in world. For the purposes of this project the details are created in the shader for simplicity

Specular Light – light that hit the object and is reflected to the camera.



# Time

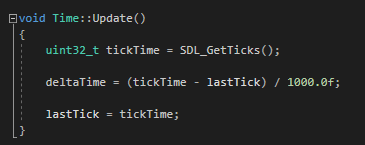
## Class Description

Handles calculating the difference in time between game frames

## Class Behaviour

### Calculating Delta Time

SDL contains a method to get the current time of the of the program since SDL was initialised by using SDL\_GetTicks(). this means that the class can store the time of the last frame, get the current frame’s time and then subtract the previous time from the current time to get the difference in time between the frames / delta time.



### Information Accessibility

Delta time is kept as a static variable however is kept private in the time class however to prevent other classes from changing its value it is kept private within the class. For other classes to use its value it has a static getter method.

# Input Manager

## Class Description

Handles detection and consuming user inputs

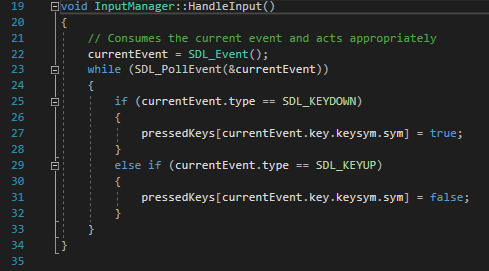
## Class Behaviour

### Input Detection

SDL only has the ability to detect when keys are being pressed down or when keys are being unpressed, to expand on this and let the program know about keys that are being held down the manager has a map of key states.

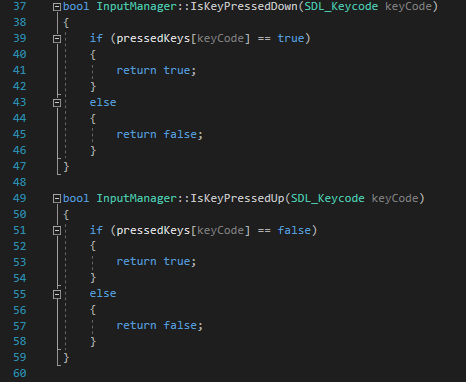


This map uses SDL’s keycodes as keys and Booleans as values to represent if a key is currently being held down. Every frame the manager consumes the currently pending events and updates its map of inputs depending on the event. If the user has pressed the key down the relevant value on the map is set to true (if there is not a value for the requested key the map automatically adds one). If the user has pressed up, then the value is set to false.



### Information Accessibility

Other classes can check if the user has pressed a specific key down using the input manager’s isKeyPressed methods, if the user request a check for a keycode not in the map then the map again automatically adds an element for that keycode



# Component

## Class Description

Components are classes designed to be added to gameobjects to store information or add behaviour

## Class Behaviour

### Constructor

Let’s the component know what gameobject that it is attached to, this is key so that each component can perform basic changes to the gameobject & access other information through the gameobject.

Since each gameobject needs to know its components and each component needs to know its parent gameobject this would usually cause circular dependencies. To avoid this the component class uses forward declaration to declare the gameobject class again in it’s header file and then lets the cpp files of the subclassed implementations include the gameobject source file.

### Update

Since the Update method is intended to contain additional behaviour for subclassed components to define the base Update method is left defined as a pure virtual method. This ensures that even if the subclassed component doesn’t need to define additional behaviour to call every frame it must at least define its own version of the Update method.

# Ball Controller

Inherits from - Component

## Class Description

Contains various sets of behaviours to simplify showing off elements in the coursework project. Intended to be added to a gameobject in combination with the physics handler component

## Class Behaviour

### Respawning

The main function of the ball controller is to detect if the ball has fallen out of bounds of the level, if the ball’s global position falls below a certain amount on its y axis then the ball is automatically returned to its start position on the level.

### Scene Changing

The Ball controller can also manually switch between the 3 available scenes in the game by combining the available behaviours from the input manager and the scene manager. The user can quickly change the current scene by pressing down

# Camera Mount

Inherits from - Component

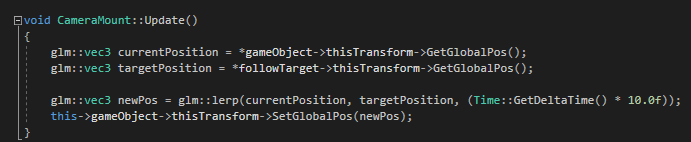
## Class Description

When added to a gameobject the camera mount component can allow the gameobject to follow another target gameobject

## Class Behaviour

### Following another Gameobject

Rather than just setting the parent gameobject’s positions to the target gameobject’s position the class lerps between the two position to create a smoother transition. It uses the time class’s delta time calculation to ensure that the lerp speed is independent of the program’s frame rate.



# Environment Controller

Inherits from - Component

## Class Description

When added to a gameobject it allows the user to control the object’s rotation through the use of keyboard inputs, it is used on the scene environments by having one element in the environment contain the environment controller component (the anchor) and then the rest of the environment gameobjects in the scene attach themselves to the anchor through the use of the transform’s parenting function.

## Class Behaviour

### Environment Movement

The component uses the input manager to check if the user has pressed and of the WASD keys and rotates the gameobject appropriately. Pressing the W or S keys affects the gameobject rotation on the x axis and pressing the A or D keys affects the gameobjects rotation on the z axis.

To prevent the environment from rotating too far in any direction there is a limit imposed by the component.

# Physics Handler

Inherits from - Component

## Class Description

A component that when added to a gameobject allows the gameobject to move with forces and react to other colliders in its scene, the physics handler has been created in a way so that it must be attached to an object with a sphere collider and the only other colliders that it can do collision checks against is other gameobjects with box colliders

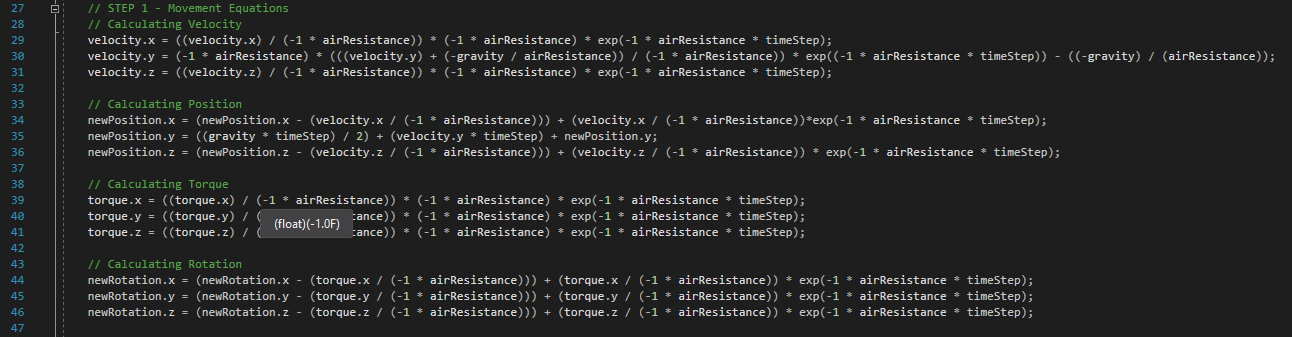
## Class Behaviour

### Velocity

A measurement of the translational forces currently affecting the object, it is required to calculate the objects new velocity each frame due to the object constantly having air resistance affect its velocity on the X, Y & Z axis’ and also gravity on it Y axis. From this new velocity the objects new position can be calculated, these calculations are derived from the initial velocity calculations, so they also have air resistance and gravity affecting the same axis’.

### Torque

A measurement of the rotational forces currently affecting the object, uses similar calculations to handling velocity but the y axis only has air resistance affecting it. Calculating the new rotation is then handled like before (again excluding gravity)



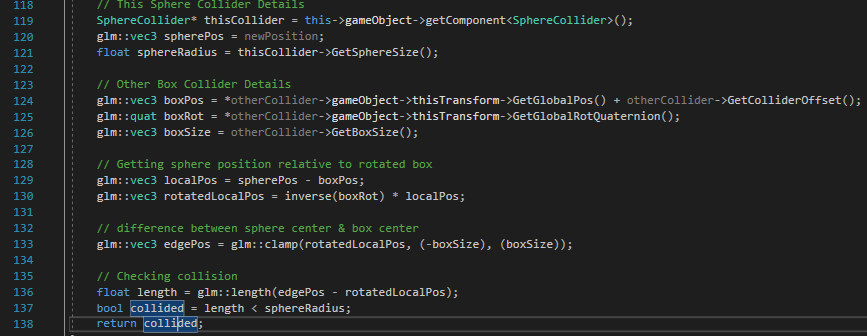
### Collision Detection

Before the object’s new position & rotation is applied the class checks if the object is would be colliding with anything in its new position

#### Box Vs Sphere Collision

When colliding, the object first needs to find out where it is colliding on the other object. First the class rotates the two objects so that the box vs sphere collision can be calculated as a regular axis aligned bounding box vs sphere collision.

From this the following equation is used to get the sphere’s nearest position on the box’s edge, if the distance between the sphere’s position and edge position is less than the sphere’s radius then the sphere is currently colliding with the box.

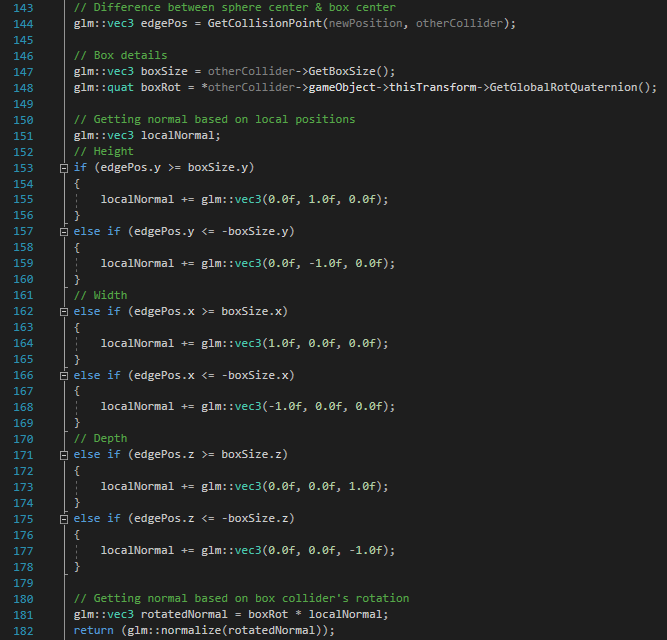


If the object is colliding, then the class must resolve the collision and if it’s not then the class can just apply the previously calculated new position and rotation

### Collision Resolution

#### Reflecting Velocity

To reflect velocity of the ball away from the collider the class needs to know the normal of the surface that it hit, since it’s already calculated the point of impact in the box’s local space the class can easily use this to determine the normal that was hit. This normal needs to be rotated by the box collider’s rotation to put it back into world space though.



# Mesh Renderer

Inherits from - Component

## Class Description

Component for handling rendering a gameobject, requires a mesh, texture and shader to function properly

## Class Behaviour

### Renderer Setup

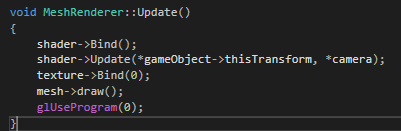
After being added to a component the render must be setup through the use of the component’s Init function. This is a mandatory step for the mesh renderer to render an object on the screen

The component uses the relevant managers to check if the assets have been loaded and uses pointer to get their locations in memory.

### Rendering Object

Since the renderer already contains references to a mesh shader it just needs to bind and unbind the relevant parts.

The component ensures to unbind any programs after rendering to ensure that the next object that needs to render starts from an empty slate.



# Font Renderer

Inherits from - Component

## Class Description

Component for handling rendering a text to the UI, requires a font to render.

## Class Behaviour

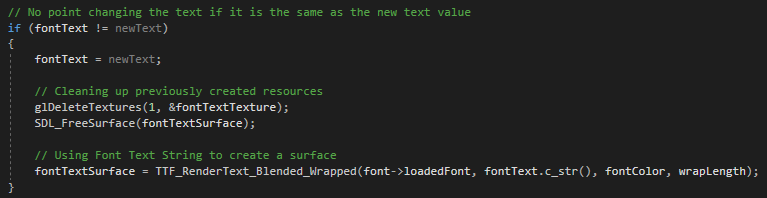
### Renderer Setup

The Font Renderer behaves the same as the Mesh Renderer when needing to be set up for an object, after adding it to a gameobject the component’s Init method must be called with the fonts file path passed through

### Rendering Text To Screen Space

#### Creating a String

From the loaded font the user can create a



#### Rendering to The Screen

The class renders to the screen by creating the text texture as a standard 4-sided shape that matches the texture’s width & height.

