Problem Solving Task A

IGB283 – Assessment 01

Due: Friday, 15th September 2017, 11:50pm

Weight: 30%

**Group Memebers:**

Ethan Farley

Jared Carey

Contents

[2 Instructions 3](#_Toc493271490)

[Speed Changing 3](#_Toc493271491)

[Vertical Movement 3](#_Toc493271492)

[To change the vertical height of the moving gameobjects one needs to click and drag the bright pink boxes that are in line with the gameobject with the left mouse button. To stop changing the vertical height of the gameobject simply stop pressing the left mouse button. 3](#_Toc493271493)

[3 Scripts 4](#_Toc493271494)

[GameController 4](#_Toc493271495)

[GraphicalObject 4](#_Toc493271496)

[IGB283Transform 4](#_Toc493271497)

[Matrix3x3 4](#_Toc493271498)

[ColourLerp 4](#_Toc493271499)

[Moveable 4](#_Toc493271500)

[4 Task 1 5](#_Toc493271501)

[4.1 Summary 5](#_Toc493271502)

[4.2 Creating the Vertices 5](#_Toc493271503)

[4.3 Constructing Triangles with the Vertices 5](#_Toc493271504)

[4.4 Centring the Mesh 6](#_Toc493271505)

[4.5 Transformation Class 7](#_Toc493271506)

# Instructions

### Speed Changing

To increase the speed of the moving objects, press either the **W Key** or **Up Arrow Key** on the keyboard. To decrease the speed of the moving objects, press either the **S Key** or **Down Arrow Key** on the keyboard.

The **speed of the moving objects can be increased / decreased five times** from the beginning speed. In each gameobject has about **eleven different speeds**.

### Vertical Movement

### To change the vertical height of the moving gameobjects one needs to click and drag the bright pink boxes that are in line with the gameobject with the left mouse button. To stop changing the vertical height of the gameobject simply stop pressing the left mouse button.

# Scripts

GameController

Instantiates gameobjects pivotal to the game and assigns random values to the gameobject’s component’s variables. Also controls functionality for speed changing.

GraphicalObject

Creates the mesh for the gameobject based on two public variables and makes sure the mesh is centred round the gameobject’s pivot when the game starts.

IGB283Transform

Contains the transformation functions used in the game and translates, scales and rotates gameobjects vert by vert using the Matrix3x3 class.

Matrix3x3

The class where all the matrix functions are located.

ColourLerp

Simple script that lerps between two colours, using the gameobject’s mesh’s x position as the step value.

Moveable

Provides the functionality relevant to moving the gameobjects and their parent’s horizontally.

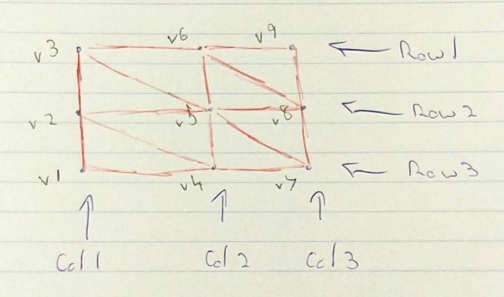
# Task 1

## Summary

To create the mesh of my gameobject, I constructed Vector3 locations based on two variables: xSize and ySize. Using these Vector3 locations I dynamically construct the vertices of my mesh with for loops. Once completed, I use more for loops to assign the colour of each vertice and construct the triangles of my the gameobject’s mesh.

**Important**: the values of both xSize and ySize must equal at least 5 for the mesh to meet the criteria of the assignment.

## Creating the Vertices

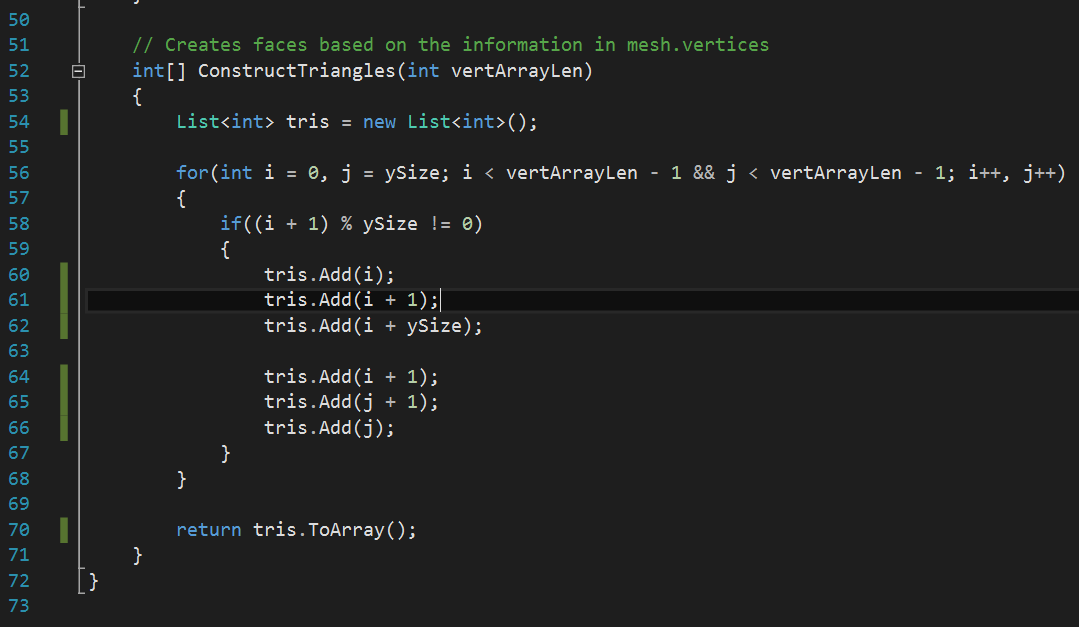


When creating the **Vector3** locations of the **GameObject’s** **Mesh** one can see that I construct the columns of the mesh first. Furthermore, one can see that it takes an **xSize** and **ySize** of 3 to create a square that contains 8 triangles – or four quads; **xSize** and **ySize** represents the number of verts in a column/row not the amount triangles in the mesh. If one were to have either **xSize** or **ySize** equal 1 then there would be no mesh. Due to this design, one can also construct rectangles with the scripts I created.

All of these **Vector3** locations are stored in an array so they can be easily accessed and used when constructing the **GameObject’s** **Mesh**.

## Constructing Triangles with the Vertices

Below one can see a picture of the code used to construct the triangles of the **Mesh**.



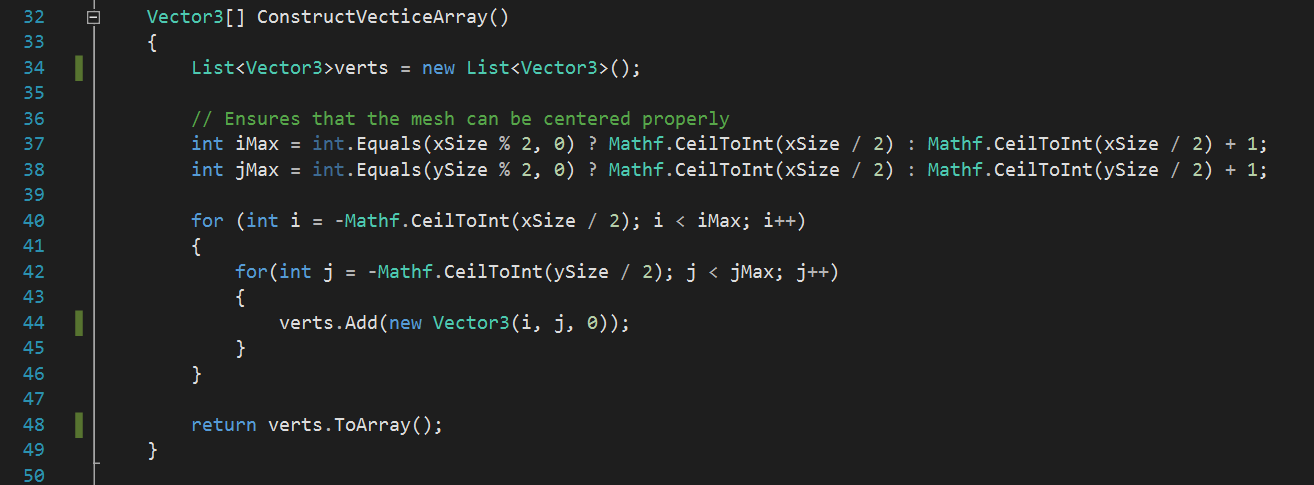
Using the peculiar for loop displayed above I construct the mesh quad by quad, adding each vertice’s **Vector3** location in threes to create triangles. For example, the first triangle would be constructed from vertice’s v1, v2 and v4 if we go by the image in Section 4.2.

As the vertices are constructed column by column we need to add the ySize / column size to i to get the next vertice in the row. Finally, j receives its own variant of this process to create the other triangle of the first quad.

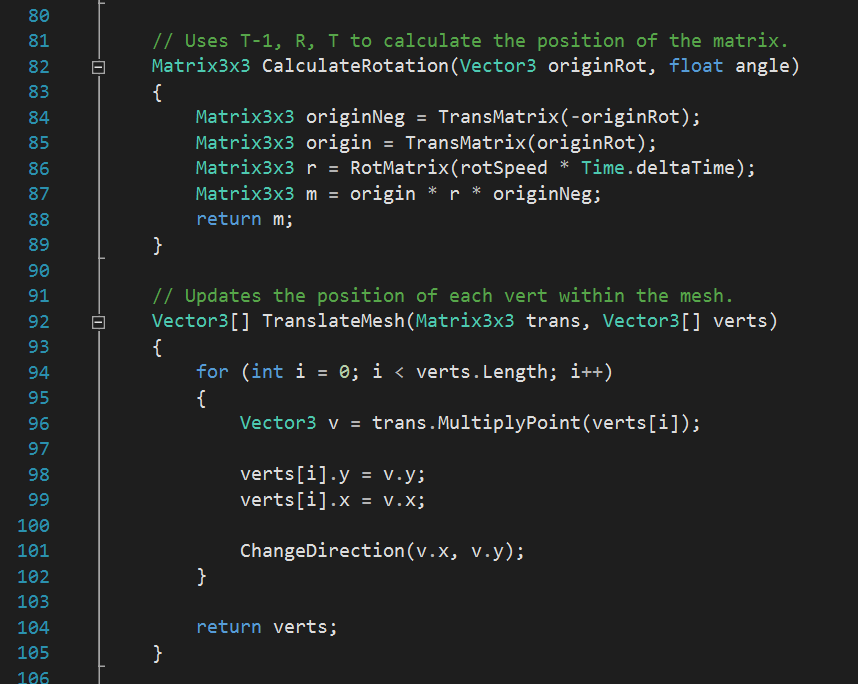
To avoid an issue where a triangle that would span across two entire columns was created an if statement was constructed. This if statement checks if i + 1 is equal to the column size, if it is then it’ll that iteration of the loop as the problematic triangle would be spawned otherwise.

## Centring the Mesh

To initially centre the **GameObject’s Mesh** around the **GameObject’s Mesh** alterations were needed to be made to the construction of the vertice locations. The solution was simply: divide the row and column size in half and use these new values to offset the initial **Mesh**. A picture of the aforementioned solution can be seen below.

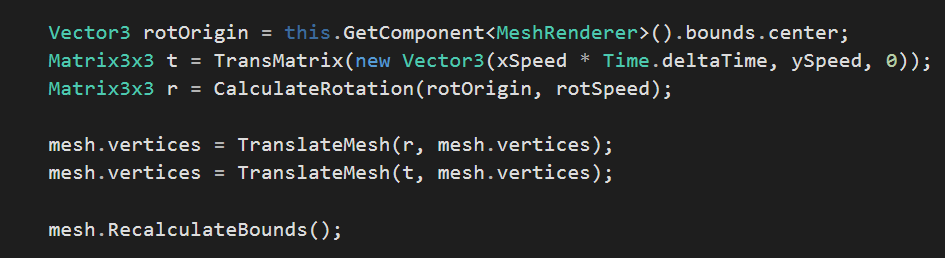


## Transformation Class



Above one can see the functions I created to translate, rotate and scale the **Mesh** created using the vertice and triangle construction logic that was shown priorly.

Using a combination of my own matrix construction functions and the **Matrix3x3** functionality I can create my own matrices and then manipulate them with these two functions. By performing these computations in update and multiplying it with **deltaTime** the **Mesh** can move around the level in a smooth fashion.



One can see the code for my matrix construction functions in the image below.