Assignment 1 : A 2D Game Engine

Due: Fri Sep 1, 2017, 23:59:59

Late penalty: Penalties are off the maximum mark. 1.5 marks/day and 1 mark/day for the bonus game

Marks: 15 Marks plus up to 3 Bonus Marks

Note on bonus marks policy: bonus marks do not carry across to make up for final exam marks. They can only count towards the 40% allocated to assignment marks.

Intro

For the first assignment you will be building a simple 2D game engine.

This is an individual assignment

The aim of this assignment is to test:

1. Your understanding of the idea of a **scene graph**.
2. Your understanding of **2D affine transforms** (translation, rotation, scale)
3. Your ability to set the **model-view transform** in OpenGL
4. Your ability to use the **orthogonal projection**.
5. Your ability to **draw** simple shapes.
6. Your ability to **model** a simple scene using the scene graph

Task

You task is to complete the implementation of a scene-graph based game engine and use it to make a simple game (or animation). The engine is designed to allow a games programmer to implement simple 2D objects, connect them in a scene-graph (in this case, a tree) and animate them in response to user input.

Files

Download a set of base classes here. These classes implement the basic data-structures, but are incomplete. The files provided are:

1. **GameEngine** - the GLEventListener which handles init(), dispose(), update() and display() methods, mostly by passing them off to other components. You should read and understand this class but you should not need to modify it unless you do the bonus collision testing part.
2. **GameObject** - an object in the scene-graph. This class represents a node in the tree structure, with its own local coordinate frame. You will need to complete code to calculate the global position of this node, and to compute the model transform for its coordinate frame.
3. **PolygonalGameObject** - an extension of GameObject that represents a convex polygon. You will need to complete code to draw the polygon in the local coordinate frame.
4. **Camera** - an extension of GameObject that represents the camera. You will need to complete code to compute the view transform for the camera's position and orientation.
5. **MathUtil** - a small library of math functions that you may find useful, including matrix multiplication. You will need to complete methods to construct matrices for translation, rotation and scaling.

6. **Mouse** - a singleton class that keeps track of mouse button presses and mouse movement in world coordinates. You should not need to modify this class.

We will be using automarking to test some of these classes, so you need to make sure you **do not change class names or method signatures**. You may add additional classes and methods as you wish.

Detail

All the methods and classes you need to implement have been marked with the **TODO** tag. They are described in detail below

MathUtil

```
public static double[][] translationMatrix(double[] v)
public static double[][] rotationMatrix(double angle)
public static double[][] scaleMatrix(double scale)
```

These three methods should return 3x3 arrays of doubles representating a 2D translation, rotation and scale matrix respectively, to apply to a vector in homogeneous coordinations. The matrices should be specified in (row, column) order. So m[2][1] refers to the entry in row 2, col 1.

We have provided a class called MathUtilTest.java. This is a junit test class that you can use as a starting point to test your MathUtil class.

GameObject

A GameObject is an object that can move around in the game world. GameObjects are basically the 'nodes' that form a scene tree. The root of the tree is the special ROOT object. Each GameObject has a parent (which is null for the root node) and a list of children. Each GameObject is offset from its parent by a rotation, a translation and a scale factor. To simplify the maths involved in the assignment, we assume that transformations are always done in the order, Translate, Rotate, then Scale and that scaling is always uniform (ie we can't scale x and y axes by different amounts).

The methods that need to be implemented in this class include:

```
public void draw(GL2 gl)
```

This method needs to:

1. Update the model transform appropriately to establish the local coordinate frame for this object.

2. Call drawSelf() to draw the object.

3. Call draw() recursively on all children objects

```
public double[] getGlobalPosition()
public double getGlobalRotation()
public double getGlobalScale()
```

These methods should compute the global position, rotation and scale of this object in world coordinates.

Note: You are not provided a GL context in this case, so you cannot use GL methods to implement this. You should do this computation directly, using the matrices provided by MathUtil.

```
public void setParent(GameObject parent)
```

This method already handles moving an object to a different parent in the tree, but at the moment this causes the object to change location, orientation and scale. You should rewrite it so that it modifies the objects local transformation so that its global transformation is not changed when it moves.

Example: consider the scene-tree

```
    ROOT p = (0,0), r = 0, s =1
        Object 1 p = (1,1), r = 0, s = 1
            Object 2 p = (0, 1), r = 0, s = 3
        Object 3 p = (-1,-1), r = 90, s = 2
```

Each object is specified with its local position, rotation and scale.

Currently the **global** position of **Object 2** is (1,2). Its global rotation is 0 and its global scale is 3.

If we change its parent to **Object 3,** the current code creates the follow scene tree:

```
    ROOT p = (0,0), r = 0, s =1
        Object 1 p = (1,1), r = 0, s = 1
        Object 3 p = (-1,-1), r = 90, s = 2
            Object 2 p = (0, 1), r = 0, s = 3
```

Object 2 now has global position = (-3,-1), rotation = 90, scale = 6.

You need to change the local transform for Object 2 so that it maintains its **original** global transform. I.e:

```
    ROOT p = (0,0), r = 0, s =1
        Object 1 p = (1,1), r = 0, s = 1
        Object 3 p = (-1,-1), r = 90, s = 2
            Object 2 p = (1.5, -1), r = -90, s = 3/2
```

We have provided a class called GameObjectTest.java. This is a junit test class that you can use as a starting point to test your GameObject class.

PolygonalGameObject

PolygonalGameObject is a class that extends GameObject and can draw Polygonal shapes.

```
public void drawSelf(GL2 gl)
```

This method should draw the object's polygon using appropriate glVertex() calls. You can assume in this method that the model-view transformation has already been appropriately set by GameObject.draw() so you do not have to transform the points.

We will only be testing your class with convex polygons, so you do not have to consider the more difficult case of concave polygons. Of course feel free to try it for fun :)

CircularGameObject

CircularGameObject is a class that you must write. There is no stub provided. It should provide similar functionality to the PolygonalGameObject class but obviously be specialised to be able to draw circles. You should approximate your circles by drawing a 32 sided polygon.

The constructors you MUST implement are:

```
//Create a CircularGameObject with centre 0,0 and radius 1
public CircularGameObject(GameObject parent, double[] fillColour
                                             double[] lineColour);

//Create a CircularGameObject with centre 0,0 and a given radius
public CircularGameObject(GameObject parent, double radius,double[] fillColour
                                                            double[] lineColour);
```

It should also provide all similar functionality as the PolygonalGameObject class ie. You should be able to get points and draw the CircularGameObject etc.

Note: You can extend/use any classes or implement any interfaces you think are appropriate to complete this class. You can include additional methods.

LineGameObject

LineGameObject is a class that you must write. There is no stub provided. It should provide similar functionality to the PolygonalGameObject class but instead be able to draw a line.

The constructors you MUST implement are:

```
//Create a LineGameObject from (0,0) to (1,0)
public LineGameObject(GameObject parent, double[] lineColour);

//Create a LineGameObject from (x1,y1) to (x2,y2)
public LineGameObject(GameObject parent,  double x1, double y1,
                                          double x2, double y2,
                                          double[] lineColour);
```

It should also provide all the same functionality as the PolygonalGameObject class ie. You should be able to get and set points and draw the LineObject etc.

Note: You can extend any classes or implement any interfaces you think are appropriate to complete this class. You can include additional methods.

Camera

```
public void setView(GL2 gl)
```

This method must set the view transform to account for the camera's current position, orientation and scale.

It should also clear the viewport to the given background colour.

MyCoolGameObject

You need to write your own GameObject, that must extend GameObject (or another class that extends GameObject). The criteria for this class are as follows:

- Your object must have at least 2 descendants. In other words it must have at least 2 children, or have a child that also has a child. These children can be PolygonalGameObjects, CircularGameObjects, LineGameObjects or can be classes of your own design that extend GameObject.
- You must provide a default constructor for your object. We will be testing your object by using a default constructor. And will run it with the TestMyCoolGameObject.java class provided.
- Your object should not 'break' when we translate it, rotate it or scale it.
- You do not need to provide any animation for this object.

Bonus 1: Collision Detection

As an extension to the assignment, implement simple **collision testing**. Add a method on GameEngine:

```
public List<GameObject> collision(double[] p)
```

which takes a point in world coordinates and returns a list of any PolygonalGameObjects, CircularGameObjects or LineGameObjects in the scene-graph which contain that point.

Bonus 2: Write a game

Write a game with your engine.

Any serious attempt will get up to 0.5 bonus mark as decided by your tutor.

The best 2 games from each tutorial class (as judged by the tutor and or class vote) get put into running for the best game in the course. These games will all get 1 bonus mark.

The top game as voted by you will get 2 bonus marks, second and third top games will get 1.5 bonus marks.

Marking

This assignment is worth 15% of your final mark

Marks are assigned as follows:

Component	Mark
MathUtils methods	0.5
GameObject.draw()	2

GameObject.getGlobalTranslation()
GameObject.getGlobalRotation()
GameObject.getGlobalScale()
	3

GameObject.setParent()	2.5
PolygonalGameObject.draw()	1.5
CircularGameObject LineGameObject	2
Camera.setView()	1.5
MyCoolGameObject	2
Impressive Games	bonus 2
GameEngine.collision()	bonus 2

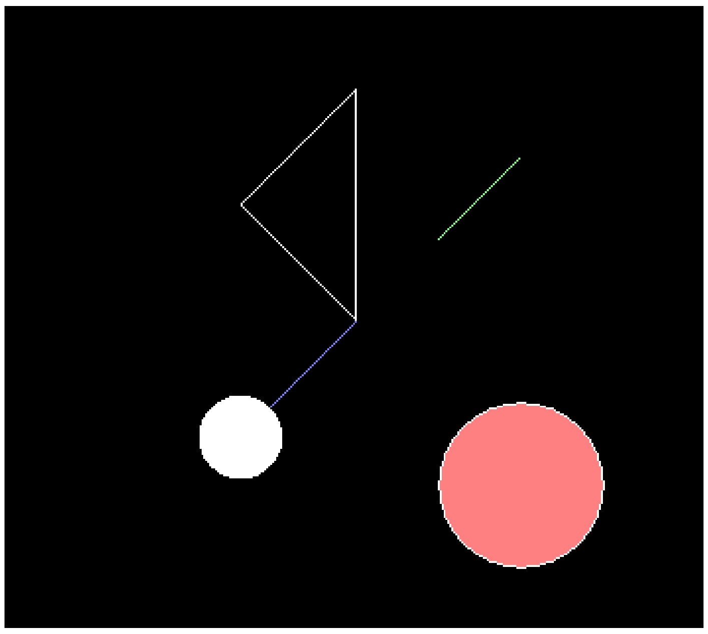
Note: The maximum mark for the assignment is 18.

Marks are based on **correctness** and **clarity** of code (My Cool Game Object and Impressive Games bonus marks are also marked on being cool or funny or beautiful etc).

Tests

We have provided some test files to help you test your assignment. These are not comprehensive and will not guarantee that your program is perfect, but are a good place to start. They include the following

- JUnit tests for the MathUtils and GameObject class. They use JUnit 4.
- We have provided a class called TestShapes.java. This is a simple class with a main method that creates a polygon, 2 circles and 2 lines. This file can be used as a starting point for testing your PolygonalGameObject, CircularGameObject and LineGameObject. Once you have implemented the relevant classes (including the Camera which is used in this test class), it should display the following:


Note: You still need to do your own testing. This only tests a few basics.

- We have provided a class called TestMyCoolGameObject.java. This is a simple class that uses your MyCoolGameObject class.

We have also provided the source of a sailing game as an example of a use-case for the engine you are developing. It is also another way to check if your code is working and to get inspiration for the bonus stage.

Submission

You can submit via webcms or via the command line using give. You must submit your bonus game (if you have one) separately from the rest of the assignment.