CSD2341/CSD5102 Computer Graphics Programming

Assignment 2: Antipodion

Related Objectives from the Unit outline:

- understand the programming techniques and algorithms used for developing 2D graphics software;
- design and develop an interactive graphics package using one of the standard programming languages.

Learning Outcomes:

In this assignment, you will learn about using openGL to create 2D graphics.

Due date:

Midnight, Wednesday 28th May, 2014 (the day before the final class)

Note: You are required to demonstrate your solution in the workshop session of the last week of semester.

In this assignment, you will be using openGL to create the graphics for a simple 2D game. This will be achieved by writing a Java class that uses JOGL.

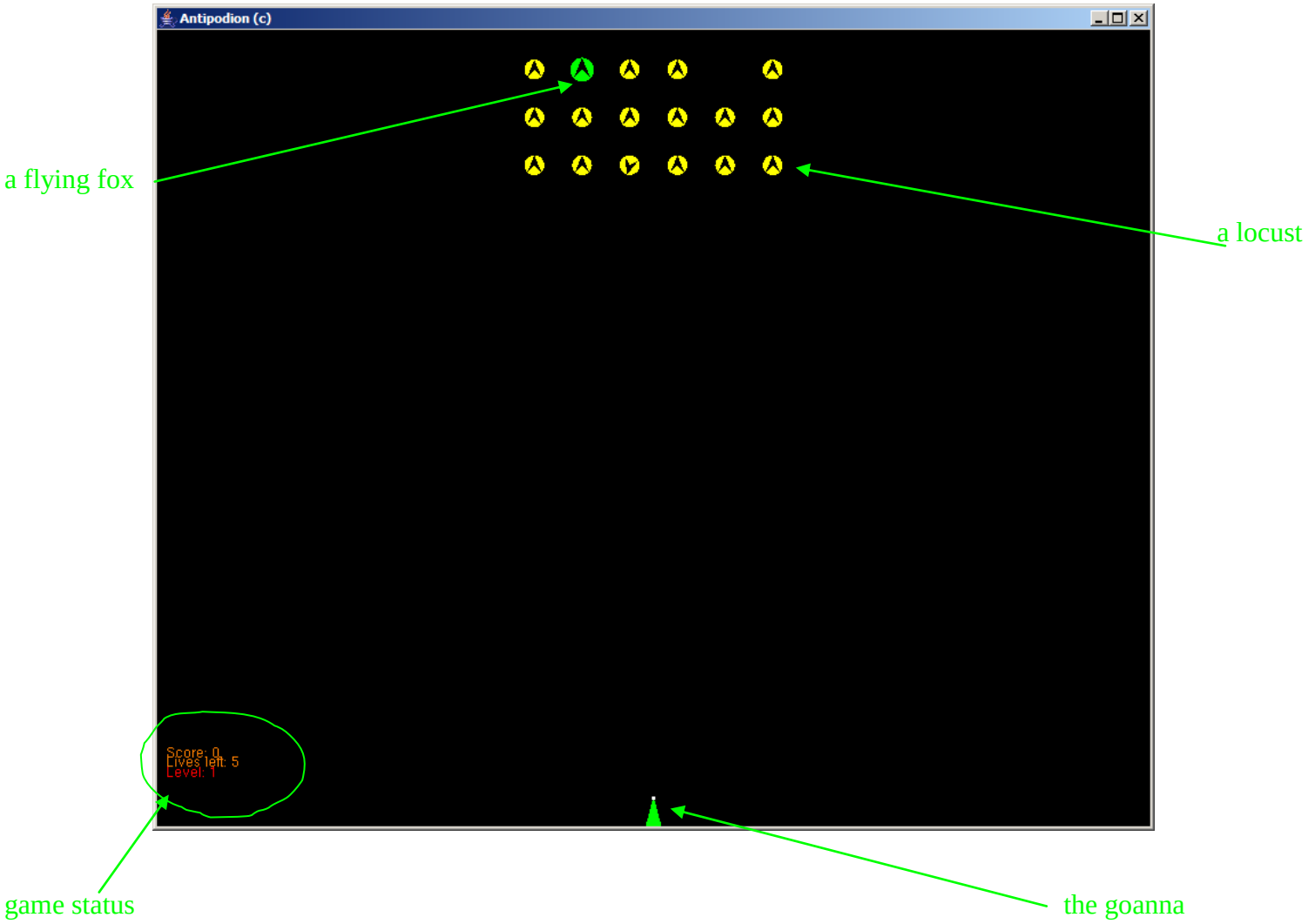
You will be provided with source for a game called Antipodion ©. Download the source from BlackBoard, load it into your Java IDE, and compile it up (don't forget to import the JAR files and set VM options, as for the workshops). The main class is called `Main.class`. As well as the source files, there is a zip file containing sound files in `.wav` format. Unzip these into the project folder. (These files should not be distributed further.)

This game is set at a point in the distant past in Australia, when there was a terrible war between the goannas and the flying things (locusts, flying foxes, wedge tailed eagles and teradactyls). The player takes the part of the last five surviving goannas, who, in the final battle, must defeat three large swarms of flying things.

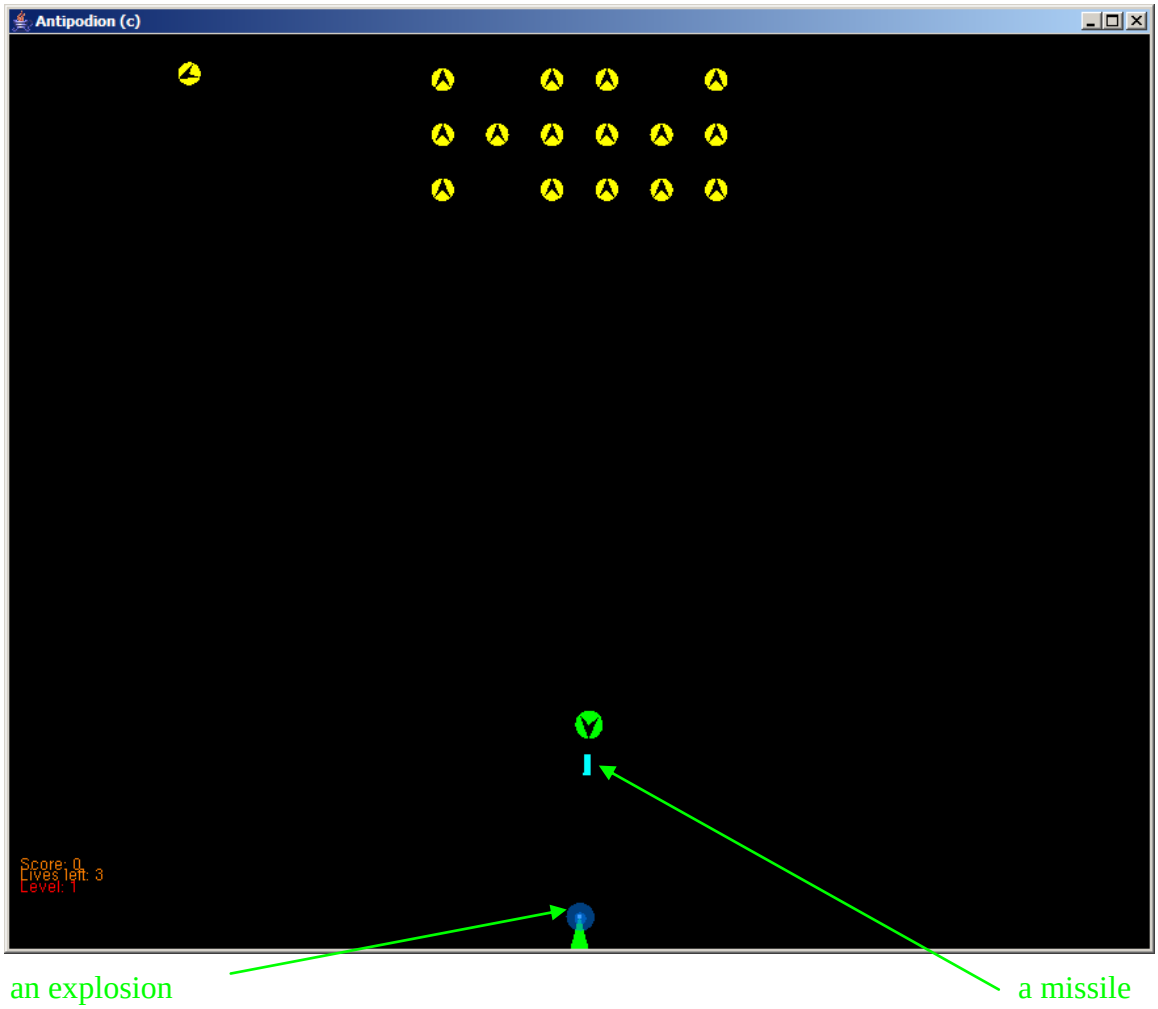
These swarms dive bomb the goanna (the player), unleashing "missiles". If the goanna is hit, it is blown away and replaced by the next goanna. The goanna can also shoot missiles at the flying things. Some flying things must be hit several times before they are knocked out. When all the flying things in a swarm have been defeated, the next swarm attacks.

To win the game, all three swarms must be cleared before the five goannas are lost.

Start the game by running `Main`. The game starts up immediately, with a screen like this:



A little later, it might look like the next screenshot. Here a flying fox is charging and has just hit the goanna with a missile. You can see an explosion happening.



As you can see, the graphical elements are primitive. **Your task is to provide interesting, more professional-looking graphics!!**

Game play

To play the game, use the following keys from the keyboard:

1. “w” to move upwards;
2. “a” to turn left;
3. “d” to turn right;
4. “s” to move downwards;
5. the enter key to fire a missile.

You can pause the game by pressing the “escape” key. A dialog will appear, and you can choose to restart the game, quit the game, or continue.

The Drawer interface

To complete the assignment, you must write a Java class that implements the `Drawer` interface, defined in the source file `Drawer.java`. An example implementation, `DefaultDrawer.java`, has been provided. The methods you will need to write are described below:

```
public void init(GL gl);
```

This method is called once, when the GL drawing context is first created. You can use it to do any one-time initialisation that you need to do, such as setting GL options, enabling GL features, initialising any of your own data structures. Otherwise, just write a method with an empty body.

```
public void drawBackground(GL gl, double width, double
height, int level, int frame);
```

This method is called first when a frame is drawn. All other drawing is done over the top of this. The DefaultDrawer do not draw a background. The meaning of the parameters is:

- `gl` – this is the GL context for your graphics commands,
- `width` – this is the width of the playfield,
- `height` – this is the height of the playfield (so the bottom left of the playfield is at (0, 0), and the top right corner is at (width, height)),
- `level` – this integer is the level number (you could use this in order to have a different background for each level),
- `frame` – this can be used to time an animation feature, such as having some moving objects in the background. Each time the method is called, `frame` progresses through a cycle between 0 and `Constants.MAX_FRAME`. (Note that because the game is multi-threaded, some frame numbers may be skipped over.) Many of the other methods also use this parameter for the same purpose.

```
public void drawGoanna(GL gl, Point2D.Double location,
double size, boolean haveMissiles, boolean moving, int
frame);
```

This method is used to draw the goanna. The `DefaultDrawer` simply draws a green triangle, with a small white square at the top to show when the goanna is ready to fire a missile). The parameters are:

- `gl` – this is the GL context for your graphics commands,
- `location` – this is the position of the centre of the goanna (a `Point2D.Double` has two fields called `x` and `y`),
- `size` – this is the radius of a circle that the goanna should be drawn within,

- `haveMissiles` – true if the goanna has a missile ready to fire (this should be used to visually let the player know whether they can fire or not),
- `moving` – true if the goanna is currently moving (the `DefaultDrawer` does not use this, but you should give use some animation so that the goanna does not look like it is sliding from side to side),
- `frame` – this can be used for animation, as before.

```
public void drawLocust(GL gl, Point2D.Double location,
double size, double rotation, int frame);
```

```
public void drawFlyingFox(GL gl, Point2D.Double location,
double size, double rotation, int frame);
```

```
public void drawWedgeTailedEagle(GL gl, Point2D.Double
location, double size, double rotation, int frame);
```

```
public void drawTeradactyl(GL gl, Point2D.Double location,
double size, double rotation, int frame);
```

These methods are used to draw flying things. `DefaultDrawer` just draws a circle with an arrow shape inside. Your version should make them look like the animals they are supposed to be. The parameters are:

- `gl` – this is the GL context for your graphics commands,
- `location` – this is the position of the centre of the flying thing,
- `size` – this is the radius of a circle that the flying thing should be drawn within,
- `rotation` – this is the number of degrees by which the flying thing is rotated (anticlockwise) – rotation of 0 means facing straight up the screen,
- `frame` – this can be used for animation, as before.

```
public void drawMissile(GL gl, Point2D.Double location,
double rotation, double width, double height, int frame);
```

This method is used to draw a missile in flight. The parameters are:

- `gl` – this is the GL context for your graphics commands,
- `location` – this is the position of the centre of the missile,
- `width, height` – these are the width and height of a rectangle that the missile should be drawn within,
- `rotation` – this is the number of degrees by which the rectangle is rotated (anticlockwise) – rotation of 0 means facing straight up the screen,
- `frame` – this can be used for animation, as before.

```
public void drawExplosionSequence(GL gl, Point2D.Double
location, int frame);
```

This method is used to draw an explosion. When an explosion occurs, this method will be called when drawing the following few frames. The parameters are:

- `gl` – this is the GL context for your graphics commands,
- `location` – this is the position of the centre of the explosion,
- `frame` – this will run from 0 to `Constants.ANIM_FRAMES`.

```
public void drawLocustDeathSequence(GL gl, Point2D.Double
location, double size, double rotation, int frame);
```

```
public void drawFlyingFoxDeathSequence(GL gl,
Point2D.Double location, double size, double rotation, int
frame);
```

```
public void drawWedgeTailedEagleDeathSequence(GL gl,
Point2D.Double location, double size, double rotation, int
frame);
```

```
public void drawTeradactylDeathSequence(GL gl,
Point2D.Double location, double size, double rotation, int
frame);
```

These methods are used to draw an animated sequence when a flying thing dies (note that some flying things have to be hit more than once before they die). The parameters are the same as for the `draw<FlyingThing>()` methods above, except that frame runs from 0 to `Constants.ANIM_FRAMES`.

```
public void drawForeground(GL gl, double width, double
height, int level, int score, int goannasLeft, int frame);
```

This method is like `drawBackground`, except that this is called last when drawing a frame, so whatever is drawn will be on top of the rest of the image. The extra parameters are:

- `score` – this is the total score that the player has achieved. Points are earned by destroying flying things (flying foxes are worth more than locusts, wedge tailed eagles are worth more than flying foxes, and teradactyls are worth more than wedge tailed eagles – and each thing is worth twice as much if destroyed when it is diving),
- `goannasLeft` – this is how many goannas have not been killed yet.

Your task:

Write a Java class that implements the `Drawer` interface for the Antipodion game. Your class will use JOGL library functions, similar to those in `DefaultDrawer`. You can use `DefaultDrawer` as a starting point, but you are expected to significantly improve each method. Change the name of the class to something unique. **You may not use any image files or similar – all the graphics must be generated in your code.**

Once you have created your class, you can test it by changing the constructor in the `GUI` class so that it constructs an instance of your drawer class instead of `DefaultDrawer`. **You may not alter any of the other game source files.**

Some ideas for improvements (but this list is neither prescriptive nor exhaustive):

- use geometric transformations
- use animation
- use procedural textures
- use particle systems
- use fractals
- use transparency

Marking scheme:

A grade will be awarded for each of the drawing methods: one of these grades

- NOT IMPLEMENTED (no change from `DefaultDrawer`) – 0
- POOR (no significant change from `DefaultDrawer`, or buggy) – 1
- ACCEPTABLE (different from `DefaultDrawer`) – 2

  To be ACCEPTABLE, some methods have additional requirements:
  - Objects must be correctly located and sized;
  - Rotated objects must be correctly rotated (missiles, flying things);
  - They must not flip upside-down;
  - The goanna moving must feature some animation;
  - The Sequence methods must feature animation;
  - Objects must look like what their names suggest – e.g. The goanna must look like a goanna, not a circle or a rectangle or a submarine.

- ATTRACTIVE (graphically interesting effects and no bugs) – 3

The results for all the methods will then be added together. A grading of ATTRACTIVE for all methods will get you 40 out of 40% for this assignment. In addition, bonus marks may be awarded for EXCEPTIONAL features (the lecturer is the only judge on this).

You will find an Excel worksheet that will be used for scoring.

Submit:

…your Java source file. It is acceptable to use additional classes. In this case, put them in their own package and submit a zip file containing the source files for this package. **Do not submit the other source files for the game**.