Practical 1 – Graphics & Animations

This practical is concerned with looking at aspects of displaying graphics using Java. We will start by developing some simple applications that display and move images about the screen.

Before we start, download the *2DCode.zip* file from Java Graphics page on Canvas (the same page that you accessed this practical sheet) and unzip this to a suitable folder on your computer (it will be assumed that you have called this directory N6 in the following sheet). We will use the contents of the 2D directory as the starting point for the practicals that follow so please refer back to this directory in subsequent practicals.

Simple Java Graphics

1. We will start by looking at the simple 'Hello World' Java application shown in lectures. Create a project called SimpleFrame using your preferred IDE (ideally Eclipse or IntelliJ) and copy the file SimpleFrame.java to it.

   Within your IDE, look at the contents of the 'SimpleFrame.java' file and note how the format fits the layout of the console application structure that was discussed in the lectures. Near the top of the file we have the method `main` which is where we create an instance of a `SimpleFrame` object called `sf` and tell it to start running by calling the `go` method. The method `go` sets the size of the frame to 800 by 600 pixels and makes it visible. Since SimpleFrame is a form of JFrame the `paint` method will be called when the frame is made visible. If we had not provided our version of `paint`, then the default `paint` method from JFrame would have been called instead (you can confirm this by renaming the `paint` method to something like `paintx` and running this again).

   This whole process occurs using the Model/View/Controller architecture you should already have learnt in previous modules and we see more of this concept in operation later on when we use the same principle to build our own game engine. There will be default `update` and `draw` methods that you can override to perform your own custom operations.

2. Now that you have this application working, take some time to play with the code in the `paint` method. Try changing colours, refer to the lecture notes to draw and fill different shapes on screen (try all the different methods suggested in the lectures on 'Graphics & Java'). You could also copy the code regarding anti-aliasing and apply it to your text or graphics.

3. The next step is to add images to our frame since drawing objects using graphics primitives is quite hard work and it would be quite useful if we could do some of this work offline using a good graphics package. A separate set of Java classes has been provided for you to examine the process of drawing images.

   The relevant files are in the directory DisplayImages. Close the SimpleFrame project and create a new IDE project called DisplayImages then drag and drop the code and images from the DisplayImages directory into it.

   This project uses the same code as before but now adds the declaration and initialisation of two `Image` objects. The `Image` objects are declared at the top of the `DisplayImages` class file, just below the initial class declaration. The actual image contents are then loaded when we call the `go` method. We then have to wait until the `paint` method is called before they are displayed.

   In principle we could load and display the images every time the `paint` method was called but this would be very inefficient. The approach taken here is to declare them as attributes of the class DisplayImages, load them when we start up and then refer to them when we need them in `paint`.

   Try displaying more copies of the image around the screen. You could also try loading and

displaying other images from online sources. The images used in this application are all located within the 'images' sub-folder. If you have any other images you would like to use, put them in here and make sure you use the same references as the other image files, for example:

```
Image myImage = new ImageIcon("images/myImage.png").getImage();
```

Animation

Now that we have managed to draw images and graphics, we will look at animating our picture. The code for this section of the practical can be found in the directory Animation. As before, create a project in your IDE called Animations and copy the code from 2DCode/Animation into it. Once you have built this project, run AnimationTest1.java' and observe what happens.

This code relates to the animation example that was demonstrated in the lecture and consists of two classes. We have a new class called `Animation` that manages the separate animation frames and decides which animation frame should be shown at a given point. This class also contains a private class definition for the class `AnimFrame` which it uses to handle frame information. This is the code that was discussed in the previous lecture on Animations. The second class in this example is the `AnimationTest` class that is our controller and contains the code that starts the whole application going.

Open 'AnimationTest.java' and look at the initialisation code in the `main` method. Confirm that this fits the format for the console application code that you saw in the lectures. Above the declaration of the `main` method, you should see the declaration of a reference to an Image object that will store the background image and an Animation reference for the animation we will use. Note that both of these references will be null until we create the associated objects and load resources into them.

Below the main method, you should find the `loadImages` method which is responsible for loading in each of the images we are going to use. Once this method has loaded in the raw images, it inserts references to them into the `Animation` object `anim` with each image being allocated a specific time (in milliseconds) for a given frame to be shown. Note that we are only storing references to images in the animation object and not copies of the images themselves – this is a more efficient process than using copies and allows us to re-use the same image many times in the animation sequence with little overhead. The other key methods in this class are `go`, *`animationLoop`* and `draw`. Starting with the `go` method, examine how these methods work and inter-operate.

Once you think you understand what the code is doing, go to the method *`loadImages`* and try re-arranging the order in which the animation frames are added to the `Animation` object. See if you can observe your changes.

In the `AnimationTest.animationLoop` method, you should notice the use of an ImageBuffer object being used as a virtual graphics device to draw our animation frame to before finally draw it in one go to the actual graphics object. This is to prevent flickering and also significantly reduces real draw calls as the number of things you need to draw increases.

Practical 1 – Graphics & Animations

Animation Sheets

In order to avoid loading individual animation images and then building an animation from them, animations are often supplied as 'sheets' with all the animation frames in them. Each image is equally spaced out across the sheet so that it is exactly divisible by the size of a single animation frame (be careful of this if you edit or crop an animation sheet).

To save you time and effort, a couple of methods have been added to the Animation class to load a set of images from a sheet of animations. The method to call in the `loadImages` method of AnimationTest is *`loadAnimationFromSheet`*. If you have an animation object called *`anim`*, a sprite sheet called `spritesheet.png` (that you have put in your `images` directory and which has the individual images arranged in a grid with 5 columns and 4 rows) and a frame duration of 100 milliseconds, you would use it as follows:

```
anim.loadAnimationFromSheet("images/spritesheet.png",5,4,100)
```

There is a small sprite sheet called `landbird.png` that you should find in the `images` that you can use to test this method. The individual images are arranged in a grid of 4 columns and 1 row. Declare a new Animation reference called `bird` after the declaration of `anim` on line 9 of AnimationTest and initialise this animation in the `loadImages` method using the method calls:

```
bird = new Animation();
bird.loadAnimationFromSheet("images/landbird.png",4,1,100);
```

Add a call to update the bird animation in the `animationLoop` method and draw it in the draw method (look at the use of `anim` to see how to do these steps).

To finish off, see if you can locate potential animations for your assignment and try to get some of them to load and animate correctly either by adding single animation frames or using a sprite sheet.