Practical 2 – Sprites and UI

This practical is concerned with moving our animations about to create sprites and adding some user control via keyboard commands and mouse clicks.

Sprites
Now that we can successfully animate an image in one place, we are going to look at moving it around. This involves making an `Animation` object an attribute of a new `Sprite` class where the `Sprite` class handles the motion of the image and the `Animation` deals with displaying it. Start by creating a new project called Sprites in your IDE and then add the files from the 2DCode/Sprites folder that you unzipped in the previous practical. Make sure you also include the images sub-directory from 2DCode\Sprites.

This project should now contain 4 files, one of which you have already seen (Animation.java). The three new files are Sprite.java which manages the motion of a 'Sprite', SpriteTest.java which is the control class that contains the `main` method for the first part of the practical and MoveSprite.java which we will use in the second part of this practical.

Examine the contents of the file Sprite.java file and compare this with the material covered in the lectures. The `Sprite` class contains a number of utility methods that can be used to set and get the properties of a given sprite object.

The method that is of most interest to us is `Sprite.update`. This method is responsible for making changes to the position of the sprite and working out which animation frame should be use for the current frame. It performs the later part by passing the elapsed time information onto the `Animation` object that operates as before. Calculation of the new position of the sprite is achieved by multiplying its relevant change in velocity attributes by the amount of time that has elapsed since the last update.

The other part of the animation and movement process is contained in `SpriteTest.update`. Rather than put all the sprite and animation updates directly in `SpriteTest.animationLoop`, we have now introduced a new separate `SpriteTest.update` method that is called from `SpriteTest.animationLoop`. `SpriteTest.update` handles collision detection for the sprite (with the boundaries of the background) and also makes the call to the `Sprite.update`. You can see that if it wasn't for the collision detection process, this would be a very simple method.

We will build on this approach for later practicals and the assignment where your games `update` method will contain the majority of the control code for your game. The animation loop will eventually be contained in a separate `GameCore` library that you will not have to concern yourself with and it will make calls to your higher level update method to find out what to do.

1. Build this project and run it. Examine the behaviour of the sprite and ensure that you understand how its movement relates to the code in the `Sprite` and `SpriteTest` classes.

Practical 2 – Sprites and UI

2. Using the *sprite* object as a template, try adding one or two additional sprites with your own animations to this demonstration and get them all to move at the same time.

Practical 2 – Sprites and UI

User Input
In the previous section we implemented a sprite class that enabled us to have an animation that bounced around the screen. We would now like to control the motion of this sprite using the arrow keys on the keyboard so close the SpriteTest.java file and open MoveSprite.java instead.

The following changes have been made to the MoveSprite.java file:
1. The text `implements KeyListener` has been added to the declaration of the class and associated import for `java.awt.event.*` has been included.

2. The method call `addKeyListener(this)` has been added to the `go` method so that key events will be passed on to the MoveSprite object.

3. Since we state that we implement KeyListener, we have provided the required three key event listeners methods associated with KeyListener. These have been added at the bottom of MoveSprite class.

4. The animation loop is now controlled by a variable called 'stop'.

The main change that will be of interest to you is the addition of the `keyPressed` method. This method examines the values of keys that are pressed and takes an appropriate action. At the moment this method will either stop the program when the 'Escape' key is pressed or change the X velocity of the sprite if the right arrow is pressed.

1. Set the MoveSprite.java file to be the main file for this application, then compile and run the program. Observe that you can only make the sprite move right (or slow down if it is moving left). When you have finished adjusting the sprite speed with the right arrow button, try exiting the program using the 'Escape' key.

2. This functionality is a bit limited, so your next task is to add additional checks to see if the other arrow keys have been pressed and take some appropriate action.

3. MoveSprite.java only handles keyboard events at present. Normally in a game you would probably want to use mouse events to control actions as well so try adding the relevant code for a MouseListener to this class.

   This will require you to add a further interface to the one listed in the class declaration, inform the MoveSprite class that your class can handle MouseListener events and then implement the relevant MouseListener events. You should refer to your lecture notes for the relevant information on how to complete each of these stages and look closely at how keyboard events are handled in this code since the process is very similar.

   For an example of adding further functionality, try changing the current position of the sprite object to become the location at which the mouse is clicked. Note that you can set the sprite x and y position by using the `Sprite.setPosition` method.