**Lab 6 - Sprites & Scrolling**

**Step 1 – Sprites**

A very common term in video game development is **Sprite**. A **Sprite** usually refers to a game object that that has physical representation (usually an image). For example, in Super Mario Bros game, Mario, Goombas, blocks, and mushrooms are all **Sprites**. A **Sprite** consists of two things; the **Texture** to draw to the screen and the **Rectangle** representing how to draw the **Texture** and for collision detection

Before we do anything, we need to download some of the files we’ll be using. Go to Edmodo, look for “Game Development - Day 6”, download the zip file, and extract its contents to the folder we’ve been using so far. We’re going to start on a new game that’s a side-scrolling platform game. Let’s get our game class setup first; create a new class called **PlatformerGame**:

import com.radirius.mercury.framework.\*;  
import com.radirius.mercury.graphics.\*;  
  
public class PlatformerGame extends Core {  
 public PlatformerGame(CoreSetup coreSetup) {  
 super(coreSetup);  
 }  
  
 public static void main(String[] args) {  
 CoreSetup coreSetup = new CoreSetup("Platformer Game");  
 coreSetup.width = 800;  
 coreSetup.height = 600;  
  
 new PlatformerGame(coreSetup).start();  
 }  
  
 public void init() {  
 }  
  
 public void update() {  
 }  
  
 public void render(Graphics g) {  
 }  
  
 public void cleanup() {  
   
 }  
}

Now that we have our game class, let’s start by creating a **Sprite** class. We’re going to be doing a few different things in this class, so add the following import statements to the top of it:

import java.io.InputStream;  
import com.radirius.mercury.framework.\*;  
import com.radirius.mercury.graphics.\*;  
import com.radirius.mercury.math.geometry.\*;  
import com.radirius.mercury.resource.\*;

The **Sprite** class will have the following things:

* 2 private variables, a **Texture** variable and a **Rectangle** variable
* A constructor that takes 3 parameters:
  + A String called **imagePath**
  + An int called **x**
  + An int called **y**
* **update()** method
* **render(Graphics g)** method

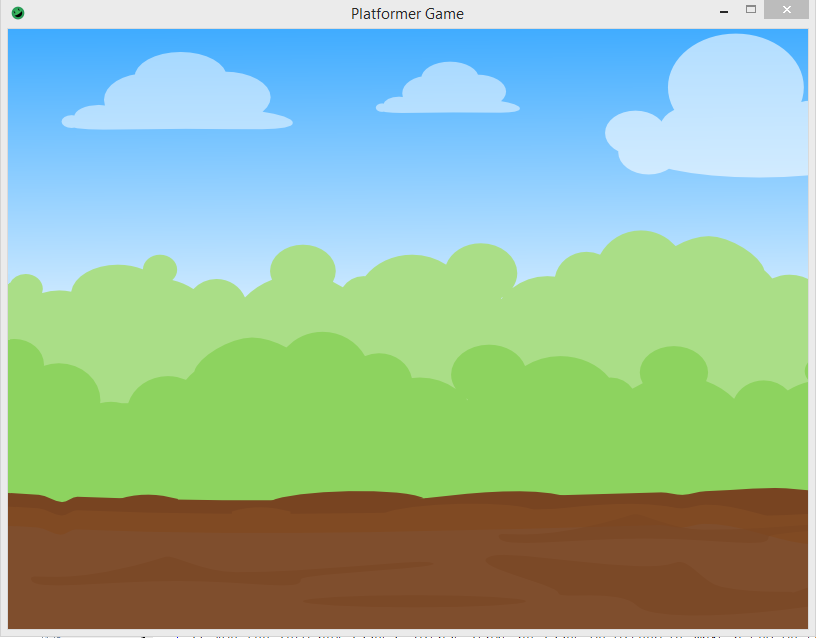
Set up the above in your class first, then we’ll go through each method step-by-step:

* **Constructor**
  + Load the image file in using the **imagePath** string, load a new **Texture** object, and set our private **Texture** variable equal to it.
    - Look back at the **MyFirstImage** class on how to do this
  + Create a new **Rectangle** object and set our private **Rectangle** variable equal to it.
    - The X and Y coordinates should be the **x** and **y** parameters
    - The width and height can be obtained by calling **getWidth()** and **getHeight()** on our private **Texture** variable
* **update()** - This method will be empty
* **render(Graphics g)**
  + Draw the texture to the screen
    - We’ll be using the **g.drawTexture()** method from the 2nd lab, but with a slight change. Instead of **g.drawTexture( Texture, int, int )**, we’ll be using **g.drawTexture( Texture, Rectangle )**
      * Look back at the **MyFirstImage** class on how to do this

There is a fair amount of work above, make sure you compile your **Sprite** class along the way to check for any compile errors. Once this is all done, let’s go back to our **PlatformerGame** class. Let’s test out our new **Sprite** class by showing a background. Do the following inside **PlatformerGame**:

* Add a private **Sprite** variable to our class called **background**
* **init() -** Create a new **Sprite** object and assign it to our private variable. Use “background.png” for our image path and the coordinates (0, 0).
* **render(Graphics g) -** Call the **render(Graphics g)** method on our **background**

Compile and run your code, you should now see a nice background image!:



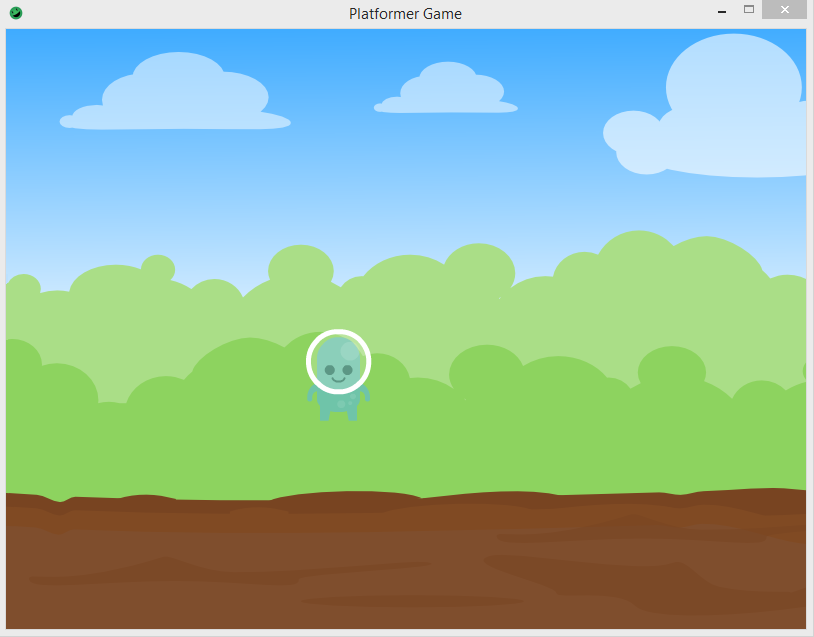
Our background is great, but it’s pretty lonely without any things with it. Let’s create a new class to represent our player called **Player** and have it inherit from our **Sprite** class. This class will be very small, mainly because we did most of the work already in our **Sprite** class!

* Create a constructor that takes 2 int parameters, x and y. Call our superclass’s constructor and pass the image path “player.png” and the x and y coordinates we get

Compile you code to make sure there are no issues then go back to the **PlatformerGame** class.

* Add a private **Player** variable to our class called **player**
* **init() -** Create a new **Player** object and assign it to our private variable; using (200, 300).
* **render(Graphics g) -** Call the **render(Graphics g)** method on our **Player**

Compile and run your code, you should now see our player!:



**Step 2 - Camera**

This is all great, by players don’t just sit; they run and jump! Let’s try adding movement for our player. Go back to the **Player** class and to its **update()** method. We’re going to control our player using the arrow keys; which are **Input.KEY\_LEFT** and **Input.KEY\_RIGHT**. Check to see if each key is being pressed and if it is, move our **Rectangle** by 5 pixels in the appropriate direction (EX: If LEFT is pressed, move 5 pixels to the left). Compile and run you code, you should now be able to control your player!

Something you’ll notice is that as the **Player** moves, they will start to move off the screen. What we want is for our **Player** to stay on screen as they move. In order to do this, we’ll need to use the game’s **Camera** to change what we see. Let’s quickly go through what the **Camera** is in our game and how it works. The **Camera** of a game is the viewing point under which the player sees the game. The **Camera** can move anywhere through the game without moving any other game objects, such as **Sprites**. Let’s take a look at a quick example:

|  |  |
| --- | --- |
|  | C:\Users\maolivo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\background.png |
| The **Camera** always starts at (0, 0) and is the width and height of the game window. The **Camera** is represented by the white box. | If I call **translate( 200, 200 )** on the **Camera**, it will move to (200, 200) and what the user sees will also change. |

Let’s try to control the camera. Go to our **Player** class and add a new method to move our **Camera** with our **Player.** The method will be called **updateCamera( Camera camera )**. For this method, we actually will copy the code from our **update()** method into this method, but we’re going to make one small change. Instead of doing **this.rectangle.translate(….)**, use the following:

camera.translate(...);

Compile and run your code, you should see your **Player** move but still stay with the **Camera**. This is because we are telling the **Camera** to move 5 pixels to the left/right with every frame with the **Player**.