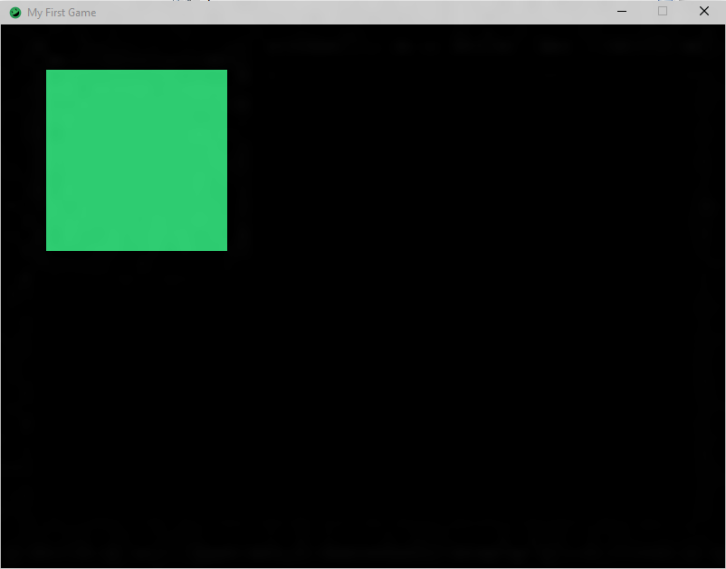
**Lab 2 - Geometry & Images**

**Part 1 – Understanding Positions**

So, we’ve already learned that we can put things anywhere within our window. However, we can also put things outside of our window. Any coordinates higher than the top of the window or to the left of the window are **negative**. Any coordinates lower than the bottom of the window or to the right of the window are **positive and greater than the window size**. Let’s take a look at the ranges:

Negative

Negative X 800+



Y

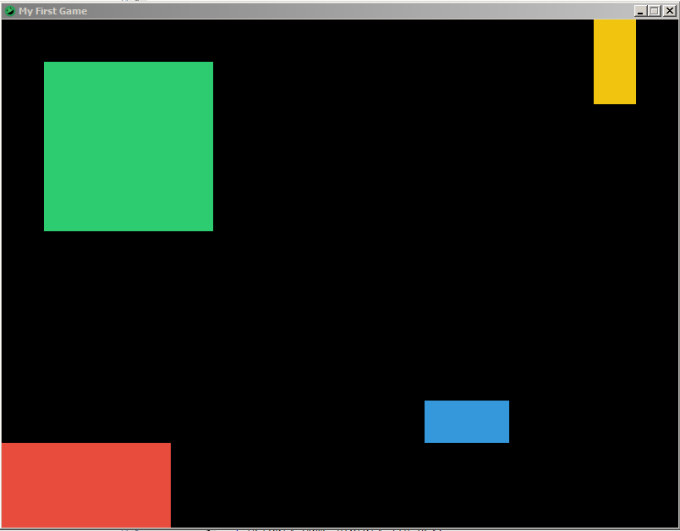
600+

So, if I have an X coordinate that is negative, the **upper-left** corner of the rectangle will be farther than the left side of the window and not visible. If the X coordinate is greater than 800, the **upper-left** corner of the rectangle will be farther than the right side of the window and not visible. The also applies to the Y coordinates.

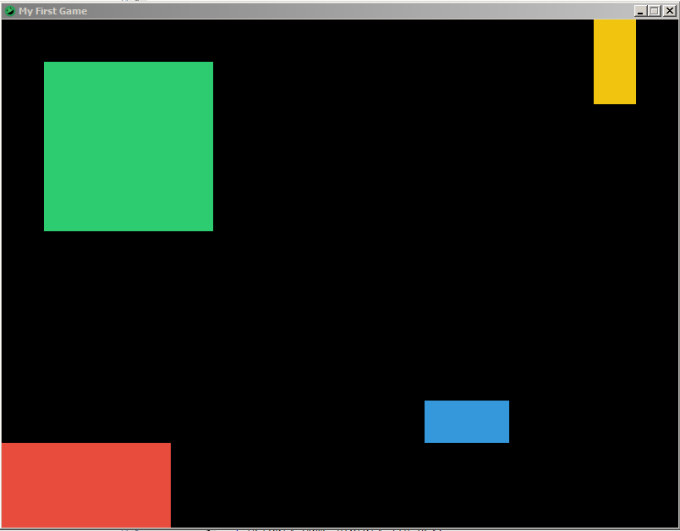
Rather than explain with words, let’s see an example. Let’s try making three more rectangles to show how coordinates outside our window work. Open up your **MyFirstGame** class and do the following:

* Create a **RED** rectangle that is at **( -50, 500 )** and is **250** **wide** by **250** **tall** and draw it
* Create a **YELLOW** rectangle that is at **( 700, -100 )** and is **50** **wide** by **200** **tall** and draw it
* Create a **PURPLE** rectangle that is at **( 900, 400 )** and is **100** **wide** by **100** **tall** and draw it

Once all three of these are done, compile and run your program. You should see the following:



So, why does our game look like this? Well, better than explain, let’s see what our rectangles ACTUALLY look like under the covers of the game:



Purple

Yellow

Red

Let’s look at each of this rectangles individually:

* The **RED** rectangle’s X coordinate is -50, which means the **upper-left** corner will be to the left of the window. Since the width is 250, the **upper-RIGHT** corner will be at 200 (-50 + 250 = 200). Since the height is 250, the bottom will be at 750 (500 + 250 = 750) and go off screen.
* The **YELLOW** rectangle’s Y coordinate is -100, which means the **upper-left** corner will be above the top of the window. Because the **YELLOW** rectangle’s height is 200, the bottom of the rectangle will be at 100 (-100 + 200 = 100).
* The **PURPLE** rectangle’s X coordinate is at 900, which means the **upper-left** corner will be to the right of the window. Since we draw rectangle’s left-to-right and the left side is off screen, the rest of the rectangle will be as well. This means we will not see the **PURPLE** rectangle.

**Part 2 – Images**

Although rectangles are nice, having actual images in our game would be nicer. Let’s create a new Java class called **MyFirstImage** and copy the following code in the class file:

import com.radirius.mercury.framework.\*;  
import com.radirius.mercury.graphics.\*;  
  
public class MyFirstImage extends Core {  
  
 public MyFirstImage(CoreSetup coreSetup) {  
 super(coreSetup);  
 }  
  
 public static void main(String[] args) {  
 CoreSetup coreSetup = new CoreSetup("My First Game");  
 coreSetup.width = 800;  
 coreSetup.height = 600;  
  
 new MyFirstImage(coreSetup).start();  
 }  
  
 public void init() {

}  
  
 public void update() {  
   
 }  
  
 public void render(Graphics g) {  
  
 }  
  
 public void cleanup() {  
 }  
}

To note, the code above is the basic code you will always need when creating a new game. First, let’s add some import statements to the top of our class:

import java.io.InputStream;

import com.radirius.mercury.resource.\*;

In order to show an image on the screen, we first have to have that image file on our computer. Go to Edmodo and look for the post titled “Game Development – Day 2”. Download the zip file and extra its contents into your “Game Tutorials” folder.

We’re going to be displaying the “mushroom.png” file, which is a 64 by 64 pixel image. In Mercury, image we show in our game are known as **Textures**. In your class, create a private **Texture** variable called **mushroom** like so:

private Texture mushroom;

Now we need to set an image to it. In order to create a **Texture** object, we need to read the data of the image file in, then load it into a **Texture**. Inside the **init()** method, add the following code:

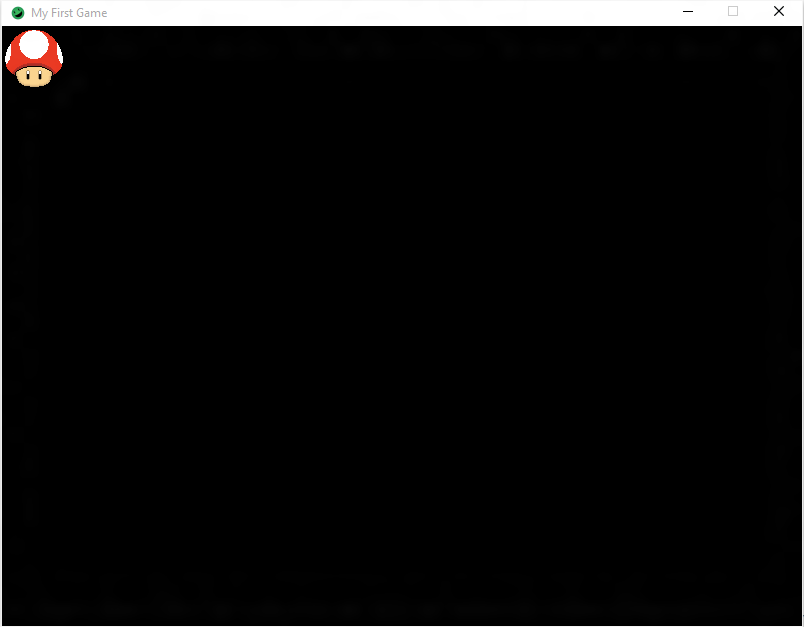
InputStream stream = Loader.getResourceAsStream("mushroom.png");  
this.mushroom = Texture.loadTexture(stream);

Let’s very briefly look at the code above. The first line uses the **Loader** class to read the data of the image file “**mushroom.png**” in to an object type called an **InputStream**, which stores the data of a file. We then create a **Texture** object by loading it from the **InputStream** we just created and setting our **mushroom** **Texture** variable equal to this new **Texture** object.

Our last bit of code is to draw the texture in to our window. Add the following code into your **render()** method:

g.drawTexture(mushroom, 0, 0);

This code will draw the **mushroom Texture** at position (0, 0). Compile and run you code and you should see the following in your game window:



**Part 3 – Basic Movement**

Our mushroom looks pretty good, but it would be even nicer if we could get our mushroom to move. Let’s try having our mushroom go from the left side of the screen to the right.

When we call **g.drawTexture( mushroom, 0, 0 )**, we draw the mushroom to position (0, 0). If we change the positions, we’ll draw the mushroom at that new location. Since we want our mushroom to go from left to right, we want to **increase** the **X coordinate**. Why? X positions increase as you go from left to right, so we simply need to make the X position we draw the mushroom at get larger over time.

To get this started, we need a variable to store the current X position we are drawing the mushroom at. Let’s create a private int variable called **xPos** that will store this value:

private int xPos;

We’ll want our mushroom to start a 0, so in our **init()** method, set **xPos** equal to 0:

this.xPos = 0;

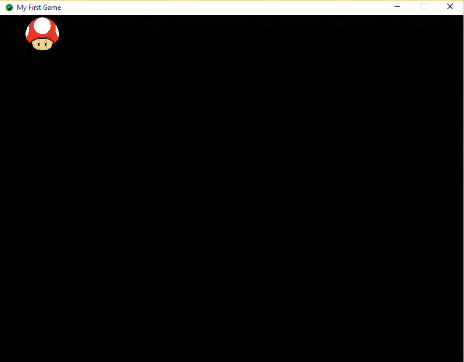
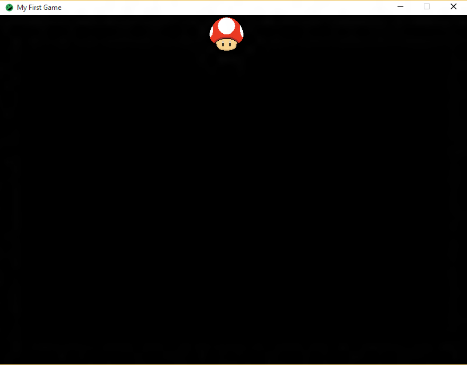
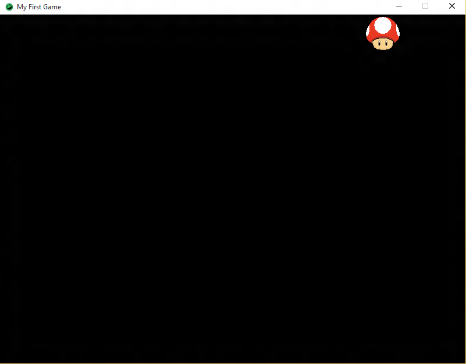
We now need a way to say “increase **xPos** every so often”. We can accomplish this inside of the **update()** method. We’ll discuss how this works in more detail later. For right now, all we need to know is that the **update()** method is called repeatedly throughout our game. Inside of the **update()** method, let’s add some code to increase the value of **xPos**:

this.xPos++;

Now we only need one more piece to finish this. Back in our **render()** method, we need to change our call to **drawTexture** to use **xPos** instead of 0 for the X position of the mushroom:

g.drawTexture(mushroom, **this.xPos**, 0);

Compile and run you code and you should see your mushroom slowly move from left to right and eventually move off screen:

**Part 4 – Final**: Try experimenting more with moving the mushroom around. Try doing it faster, slower, move on the Y coordinates, or move both coordinates.