**Lab 4 - Movement & Collision Detection**

**Part 1 – Movement:**

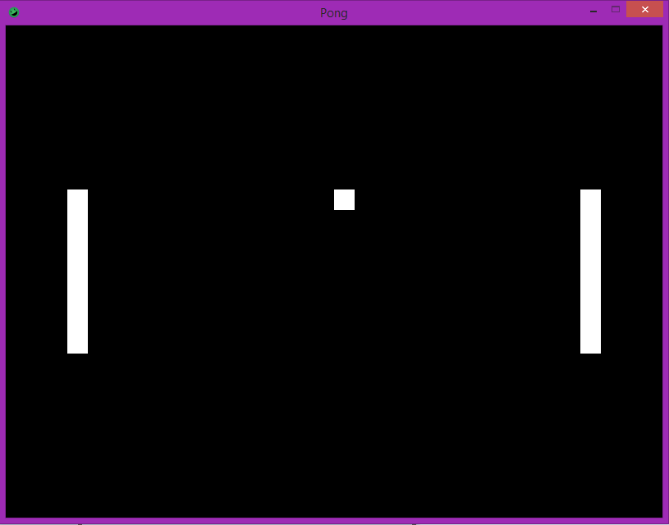
Since we got the Pong paddles done in the last checkpoint, it’s time to add the ball into our mix. Let’s start by creating a **PongBall** class. To make life easier, copy the code you have in your **PongPaddle** and make the following changes:

* Change the class name from **PongPaddle** to **PongBall**
* Make the size of the **Rectangle** be **25 wide** by **25 tall**
* Remove all the code from inside the **update()** method, but keep the method in the class.

Go to your **PongGame** class and do the following:

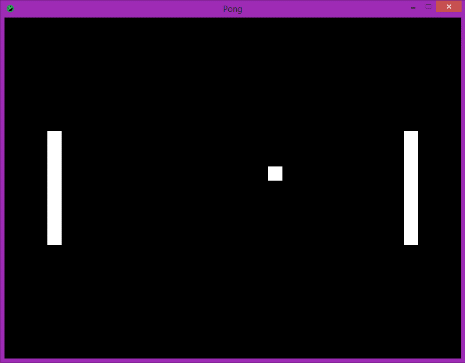
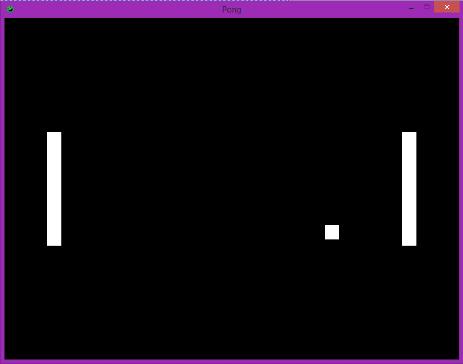
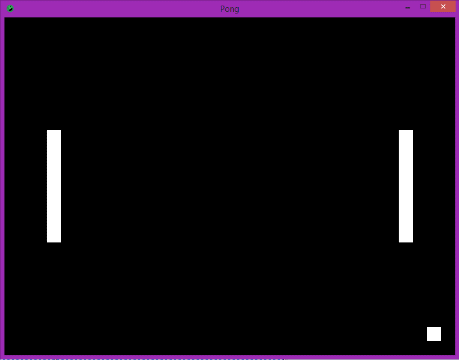
* Create a private **PongBall** variable
* **init()**
  + Create a new **PongBall** object and set our ball variable to it
  + Pass in the coordinates (400, 200) for its position
* **update()**
  + Call the **update()** method for our ball variable
* **render(Graphics g)**
  + Call the **render(Graphics g)** method for our ball variable

Compile and run your code, you should now see a small white ball in the center of your window:



This ball is pretty bland right now, let’s make it more interesting with some movement. To do this, we need to have some way to keep track of the **velocity** of our ball. The **velocity** of an object is how far it will move with every frame (every **update()** call). For example, if the **X velocity** of an object **10**, then that object will move **10** pixels to the right every frame. If the **Y velocity** is -5, then the object will move **5** pixels up every frame.

To keep track of the velocity of our ball, let’s create two private int variables in our **PongBall** class; one for the **X velocity** and one for the **Y velocity**. Let’s set their initial values both to 2. Now that we have our velocities, let’s have our ball actually move. Go to our **update()** method for the **PongBall** class and call the **translate( int x, int y )** method on our **rectangle**, but use our **X velocity** and **Y velocity** variables we just made. Compile and run your code and you should now see the ball moving!:

…right off the edge of the screen. We need to fix our ball so that it does the following:

* If it hits a paddle, it bounces the other way (left 🡪 right, right 🡪 left)
* If it hits the top or bottom of the window, it bounces the other way (down 🡪 up, up 🡪 down)

To do this, we’ll need to learn about collision detection.

**Part 2 - Collision Detection**

To define it simply, **collision detection** is determining whether or not two objects intersect within the same area. Let’s look at 3 different examples of collision detection and whether they intersect or not:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| No Collision | Collision | Collision |

Normally, doing collision detection between objects can be very difficult. However, Mercury allows us to easily tell us if two **Rectangles** intersect. The method **intersects( Rectangle other)** on the **Rectangle** class return true if **this** **Rectangle** intersects with the **other Rectangle**. We will use this to determine if our Pong ball collides with our Pong paddles.

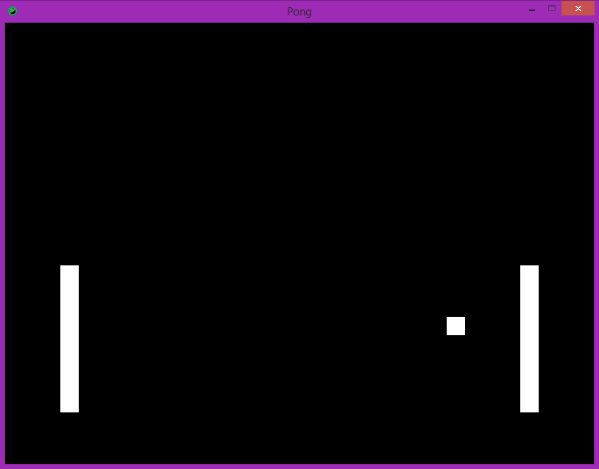
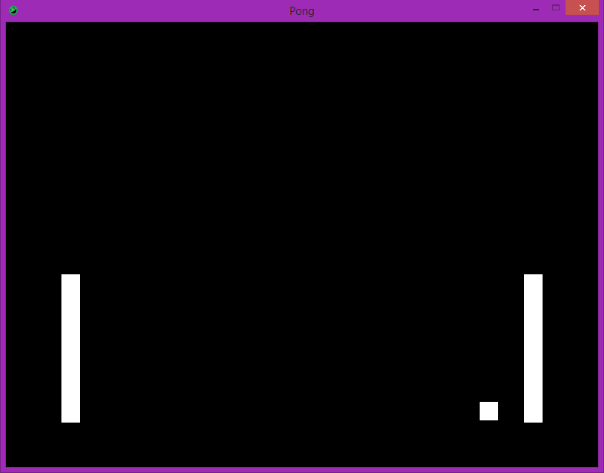
Before we can get started, we need to go to our **PongPaddle** class. The **intersects( Rectangle other )** method works on a **Rectangle** and expects a **Rectangle** as its parameter. However, the **PongPaddle** class’s **Rectangle** variable is private. So we need to add a method to the **PongPaddle** class to give us access to it. Create a method on the **PongPaddle** class called **getRectangle()** that returns the **Rectangle** object inside of the class.

Let’s go to our **PongBall** class now. We’ll be creating a method that will check to see if the ball intersects with a paddle. Create a method called **checkPaddleCollision( PongPaddle paddle)** in the **PongBall** class. This method should check if the **PongBall’s** rectangle **intersects** with the **PongPaddle’s** rectangle. If this is true, we need to flip our left to right direction.

Now, how do we flip our left to right direction? Let’s think about it for a second. Our X velocity tells us how much we’re moving left or right. We know that a positive X moves us right and a negative X moves us left. So, what do we need to do to our **x velocity** to flip our left to right direction?

Once you’ve figured out the above, let’s go back to our **PongGame** class. In our **update()** method and after we have called the **update()** method on our two paddles and ball, we need to call our new **checkPaddleCollision( PongPaddle paddle )** method on the **PongBall** class. We need to call this method twice, one for each of our paddles.

Once the above is done, compile and run your code. Try to move your right paddle to collide with the ball. You should see the ball hit your right paddle and then start moving left:

However, once the ball hits the paddle, it’ll move off the bottom of the screen. Now we need to add collision detection in for the top and bottom of our window.

Let’s go back to our **PongBall** class to add a new method to check for this collision. Add the new method **checkWindowCollision( int height )** to our class. This method will check to see if the ball has reached the top or bottom of the screen.

Unfortunately, there is no way to check if the ball intersects with the edges of the window. However, we can check the position of the ball in relation to the window. We know that the top of the window is at **Y position** 0 and the bottom of the window is at **Y position** **HEIGHT** (which is the height of the window). If the **top** of the ball hits the **top** of the window or the **bottom** of the ball hits the **bottom** of the window, we want to flip our **Y velocity**.

The **Rectangle** class has some methods to get the different position parts of itself. The **getY()** method returns the **Y position** of the **top** of the **Rectangle**. The **getY2()** method returns the **Y position** of the **bottom** of the **Rectangle**. Using all the information above, have the **checkWindowCollision()** method flip the **Y velocity** when the ball hits the top or bottom of the window.

Once this is done, go back to our **PongGame** class and the **update()** method. Add to the bottom a call to the **checkWindowCollision()** method we just made and pass in 800 and 600 for our window **width** and **height**. Compile and run your code and watch the ball bounce off your paddles and the top and bottom of the window!:

