Sprites - Speed and Direction

Contents

Time required: 45 minutes

All objects moving in the screen have their own characteristics such as position (x, y), speed and direction, etc. These characteristics can be isolated in an object that we are going to call a "Sprite".

Speed and Direction

In the last tutorial we got the ball (circle) to move. It moved downwards and to the right, one pixel every round of the Game Loop. When it got to the border of the screen the ball continued, vanishing from the canvas. We are going to make the ball bounce back once it touches the borders of the canvas, changing its direction.

```java
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class SimplePong extends JPanel {
   private static final long serialVersionUID = 1L;

   // Diameter of the ball
   final int BALL_DIAMETER = 30;
   final int MOVE = 3;

   // Starting coordinates of the circle
   // Upper left hand side of the JPanel
   int BallX = 0;
   int BallY = 0;

   // Variables to control the speed and direction
   // of the x & y movement of the ball
   int MoveX = MOVE;
   int MoveY = MOVE;

   private void moveBall() {
      // If the ball runs into the left border, reverse direction
      if (BallX + MoveX < 0) {
         MoveX = MOVE;
      }
      // If the ball runs into the right border, reverse direction
      else if (BallX + MoveX > getWidth() - BALL_DIAMETER) {
         MoveX = -MOVE;
      }
      // If the ball runs into the top border, reverse direction
      if (BallY + MoveY < 0) {
         MoveY = MOVE;
      }
      // If the ball runs into the bottom border, reverse direction
      else if (BallY + MoveY > getHeight() - BALL_DIAMETER) {
         MoveY = -MOVE;
      }
      // Set the movement direction based on the previous decisions
      BallX = BallX + MoveX;
      BallY = BallY + MoveY;
   }

```

```
53      @Override
54      public void paint(Graphics g) {
55          super.paint(g);
56
57          // Create a Graphics2D object to access methods
58          Graphics2D g2d = (Graphics2D) g;
59
60          // Turn on antialiasing and hinting to help rendering of graphics
61          g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
62                               RenderingHints.VALUE_ANTIALIAS_ON);
63
64          // Paint new position of the ball
65          g.fillOval(BallX, BallY, BALL_DIAMETER, BALL_DIAMETER);
66      }
67
68      public static void main(String[] args) throws InterruptedException {
69          JFrame frame = new JFrame("Simple Pong");
70          SimplePong simplePong = new SimplePong();
71          frame.add(simplePong);
72          frame.setSize(600, 400);
73          frame.setVisible(true);
74          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
75
76          // An infinite "Game Loop"
77          while (true) {
78
79              // Move the ball
80              simplePong.moveBall();
81
82              // Repaint the JPanel
83              simplePong.repaint();
84
85              // Sleep this thread for 17 ms (Approximately 60 FPS)
86              // Other threads can process, redrawing the screen
87              Thread.sleep(17);
88          }
89      }
90  }
```

In this code we can see that there are two more properties; **MoveX** and **MoveY**, which represents the speed in which the ball is moving. If **MoveX=1**, the ball moves to the right, one pixel every round of the Game Loop, if **MoveX=-1**, the ball moves to the left. In the same way **MoveY=1** moves the ball down and **MoveY=-1** moves the ball up. This is done with the lines, **"BallX = BallX + MoveX"** and **"BallY = BallY + MoveY"** of the **moveBall()** method.

Before running the previous instructions, we verify that the ball doesn't go out of the borders of the canvas. For example, when the ball gets to the right border, or when **BallX + MoveX > getWidth() – BALL_DIAMETER**, what we'll do is to change the direction of the movement on the "x" axis, we assign -1 to "MoveX"; **MoveX = -1**.

```
31    private void moveBall() {
32       // If the ball runs into the left border, reverse direction
33       if (BallX + MoveX < 0) {
34          MoveX = MOVE;
35       }
36       // If the ball runs into the right border, reverse direction
37       else if (BallX + MoveX > getWidth() - BALL_DIAMETER) {
38          MoveX = -MOVE;
39       }
40       // If the ball runs into the top border, reverse direction
41       if (BallY + MoveY < 0) {
42          MoveY = MOVE;
43       }
44       // If the ball runs into the bottom border, reverse direction
45       else if (BallY + MoveY > getHeight() - BALL_DIAMETER) {
46          MoveY = -MOVE;
47       }
48       // Set the movement direction based on the previous decisions
49       BallX = BallX + MoveX;
50       BallY = BallY + MoveY;
51    }
```

Each if statement limits a border of the canvas.

Creation of the "Ball" Sprite Class

A better way to code a game is to create a class called **Ball** which isolates everything that has to do with the ball. In the following code we see how we extract all the code from the class SimplePong2 which has to do with the ball and add it to our new class **Ball**.

```
 8 import java.awt.Graphics;
 9 import java.awt.Graphics2D;
10 import java.awt.RenderingHints;
11 import javax.swing.JFrame;
12 import javax.swing.JPanel;
13
14 public class SimplePong2 extends JPanel {
15
16    // Constants for the size of the JFrame
17    final static int GAME_WIDTH = 600;
18    final static int GAME_HEIGHT = 400;
19
20    // Approximately 60 FPS (Frames per Second)
21    static int gameSpeed = 17;
22
23    // Create a Ball object
24    Ball ball = new Ball(this);
25
26    // Call the Ball.move method
27    private void move() {
28       ball.move();
29    }
30
31    @Override // Override the default paint method
32    public void paint(Graphics g) {
33       super.paint(g);
34       // Create a Graphics2D object to access methods
35       Graphics2D g2d = (Graphics2D) g;
36       // Turn on antialiasing and hinting to help rendering of graphics
37       g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
38                            RenderingHints.VALUE_ANTIALIAS_ON);
39       ball.paint(g2d); // Paint the ball on the screen
40    }
```

```
42    public static void main(String[] args) throws InterruptedException {
43       JFrame frame = new JFrame("Simple Pong");
44       SimplePong2 simplePong = new SimplePong2();  // Create a game object
45       frame.add(simplePong);
46       frame.setSize(GAME_WIDTH, GAME_HEIGHT);
47       frame.setVisible(true);
48       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
49
50       // Game loop, loops forever
51       while (true) {
52          simplePong.move();       // Call the move methods
53          simplePong.repaint();    // Repaint the application screen
54          Thread.sleep(gameSpeed); // Pause for to let Frame redraw
55       }
56    }
57 }
```

The Ball sprite needs the reference to the SimplePong2 object to obtain the borders of the canvas and know when to change direction. In the **move()** method the Ball class calls the methods **game.getWidth()** and **game.getHeight().**

```java
import java.awt.Graphics2D;

public class Ball {
   private final int BALL_DIAMETER = 30;
   private final int TOP_LEFT_BORDER = 0;
   private final int MOVE = 3;
   private int BallX = 0;
   private int BallY = 0;
   private int MoveX = MOVE;
   private int MoveY = MOVE;
   private SimplePong2 simplePong; // Create a Game2 reference

   // Create a ball object with a reference to the game board
   public Ball(SimplePong2 simplePong) {
      this.simplePong = simplePong;
   }

   void move() {
      // If the ball runs into the left border, change direction
      if (BallX + MoveX < TOP_LEFT_BORDER)
         MoveX = MOVE;

      // If the ball runs into the right border, change direction
      if (BallX + MoveX > simplePong.getWidth() - BALL_DIAMETER)
         MoveX = -MOVE;

      // If the ball runs into the top border, change direction
      if (BallY + MoveY < TOP_LEFT_BORDER)
         MoveY = MOVE;

      // If the ball runs into the bottom border, change direction
      if (BallY + MoveY > simplePong.getHeight() - BALL_DIAMETER)
         MoveY = -MOVE;

      // Set the movement direction based on the previous decisions
      BallX = BallX + MoveX;
      BallY = BallY + MoveY;
   }

   // Create the ball/circle
   public void paint(Graphics2D g) {
      // Paint new position of the ball
      g.fillOval(BallX, BallY, BALL_DIAMETER, BALL_DIAMETER);
   }
}
```

If we execute SimplePong2, we will obtain the same result as if we executed the last version.

The convenience of putting the code of the Ball into a Sprite type class becomes clearer when we include the racquet as a new Sprite, in the next tutorial.

Assignment Submission

Attach the .java files to the assignment in Blackboard.