Breakout: Project 2, Milestone 4

Due Monday, November 16
Total Points: 40 + extra credit
Requirements:
1. Breakout with block requirements of milestone 3.
2. Game over when ball hits bottom edge.
3. A user (mouse) controlled paddle with 2D movement across panel width and bottom 200 pixels.
4. Collisions must look realistic, but can have a few hard to notice glitches.
5. Paddle must reflect ball and impart some amount vertical and horizontal velocity to the ball that is a function of the paddles velocity at the time of impact.
6. Javadoc
Read Mouse Pointer Location in screen coordinates

```java
public class HelloWorld extends JFrame implements ActionListener {
    public HelloWorld() {
        setBounds(10, 10, 100, 100);
        setVisible(true);
        javax.swing.Timer myTimer = new javax.swing.Timer(500, this);
        myTimer.start();
    }

    public void actionPerformed(ActionEvent arg0) {
        java.awt.PointerInfo mouse = java.awt.MouseInfo.getPointerInfo();
        java.awt.Point p = mouse.getLocation();
        System.out.println("Mouse: (" + p.x + ", " + p.y + ")");
    }

    public static void main(String[] args) {
        new HelloWorld();
    }
}
```

Read Mouse Pointer Location by MouseMotionListener in container coordinates

```java
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
// Other imports and class variables

int mouseX, mouseY; // Global so all methods can access.

public class BreakoutDraw extends JPanel implements MouseMotionListener {
    public BreakoutDraw(BreakoutFrame frame) {
        // Other statements
        this.addMouseMotionListener(this);
    }

    public void mouseDragged(MouseEvent arg0) {}
    public void mouseMoved(MouseEvent evt) {
        mouseX = evt.getX();
        mouseY = evt.getY();
        System.out.println("Mouse: [" + mouseX + ", " + mouseY + "]");
    }
}
```
Throttle Paddle Movement: Idea

- The user requests to move the paddle from \((x_1, y_1)\) to \((x_2, y_2)\).
- If the distance between \((x_1, y_1)\) to \((x_2, y_2)\) is less than or equal to \(PADDLE\_MAX\_SPEED\), then on the next timer event, move the paddle from \((x_1, y_1)\) to \((x_2, y_2)\) or to a collision.
- If the distance from \((x_1, y_1)\) to \((x_2, y_2)\) is greater than \(PADDLE\_MAX\_SPEED\), then on the next timer event, try moving the paddle from \((x_1, y_1)\) to \((x_3, y_3)\) where \((x_3, y_3)\) is on the line between \((x_1, y_1)\), \((x_2, y_2)\) and a distance of \(PADDLE\_MAX\_SPEED\) from \((x_1, y_1)\).

![Diagram showing movement of paddle from \((x_1, y_1)\) to \((x_2, y_2)\) with \((x_3, y_3)\) on the line between \((x_1, y_1)\) and \((x_2, y_2)\).]

Throttle Paddle Movement: Code

```
// Use Pythagorean theorem to find the distance between two points in two-dimensional space:
// \[ d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \]

public static int getPixelDistance(int x1, int y1, int x2, int y2) {
    return (int)Math.sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
}

...  
int x3=x2; int y3=y2;  
double dist = getPixelDistance(x1, y1, x2, y2);  
if (dist > PADDLE_MAX_SPEED) {
    double scale = PADDLE_MAX_SPEED/dist;  
    x3 = x1 + (int)((x2-x1)*scale);  
    y3 = y1 + (int)((y2-y1)*scale);  
}
```
class Collision: Using enum

class Collision
{
    public enum SurfaceType
    {
        UNKNOWN,
        VERTICAL_EDGE, // Hit Vertical surface
        HORIZONTAL_EDGE, // Hit Horizontal surface
        CORNER // Hits both Vertical and Horizontal surfaces
    };
    public SurfaceType surface = SurfaceType.UNKNOWN;
    public boolean hit = false; // True when any type of collision occurs
    public boolean hitBlock = false; // True when collision with a block.
    public boolean hitPaddle = false; // True when collision with paddle.
    public int col = -1; // If hitBlock is true, column index of block.
    public int row = -1; // If hitBlock is true, row index of block.

    public void clear()
    {
        hit = false;
        hitBlock = false;
        hitPaddle = false;
        surface = SurfaceType.UNKNOWN;
    }
}

Breakout v4 Suggestions

1. In mouseMoved(), read the mouse location, but do not move paddle.
2. Move both paddle and ball in nextTurn() (called each timer event).
3. Erase ball and paddle from off-screen buffer at start of nextTurn().
4. Let fastestComponent be the maximum speed (in pixels) of either the x or y speed of the ball or paddle.
5. During nextTurn(), take at most fastestComponent pixel steps. Stop taking pixel steps if a collision occurs. Do not render graphics during these pixel steps.
6. During each pixel step, first move the paddle, and check to see if it hits the ball. If the paddle does not hit the ball, then move the ball and check for collisions.
7. At the end of nextTurn(), render both ball and paddle either the full distance they tried to move or one pixel step BEFORE a detected collision.
Ball and Paddle Collision Cases

1. Ball reverses vertical direction.
2. Some of paddle speed is added to ball speed.

1. Ball reverses vertical direction.
2. Some of paddle speed is subtracted from ball speed.

1. Ball does not change direction!
2. Ball is nudged to an increased speed.

Tricky Ball / Paddle Case: Double Hit

Turn $n$

1. Ball is moving slowly down.
2. Paddle is moving quickly up.
3. Ball and Paddle collide. Then ball is moving up with medium speed, but paddle is still moving up quickly.

Turn $n+1$

1. Ball and Paddle collide again, both moving upward, but paddle moving faster. The ball should not turn around again.
Tricky Ball / Paddle Case: Infinite Loop

Turn \( n \)

1. In each micro-step, Paddle is moved, then Ball is moved.
2. In a micro-step, \textit{Paddle hits Ball}.
3. Paddle was moving up, Ball was moving down and Ball’s vertical speed is reversed.
4. Turn ends on this micro-step with Paddle adjacent to Ball.

Turn \( n+1 \)

1. After the collision, neither Paddle nor ball moved. Now, in the first micro-step, since Paddle moves first, and the ball is still adjacent to it, the Paddle will hit the ball again.
2. This will be the first micro-step the turn ends with \textit{no change in positions}: infinite loop.

Code: What Case is this?

```java
if ( ( Math.abs(paddleSpeedY) > Math.abs(ballSpeedY) )
    &&
    ( paddleSpeedY * ballSpeedY > 0) )
```