# Assignment 1 Report

## Youtube Links:

**Main game:** https://youtu.be/tBmkqZIBL0c

**Updated game:** https://youtu.be/aMzHOANtHeo

## Introduction:

In this assignment I developed a variant of “DX Ball” game where player controls a mobile paddle and aiming to hit and break all the bricks with ball without letting the ball touch to the bottom of the canvas. The game consists of a dynamically moving ball, a thin rod which is for player to throw the ball at what angle he/she wants , a paddle that the player controls using arrow keys, and several bricks that the ball can hit and break upon collision. Game have features like pause/start and status like angle, score and finishing messages to let the user know.

I used StdDraw library which is useful for basic graphical applications. I used it for drawing shapes (filled circle, filled rectangle), writing texts (angle, score, game status, victory/game over) onto canvas, to detect key events (with isKeyPressed() method) for ball and paddle movement and pause and start the game and also used double buffering and show() for faster animations.

### Game Mechanisms:

Line Movement Mechanism: Firstly i drew a line at proper location and length. I calculated the length of the line by Pythogoras Theorem ;

$$\text{LineLength} = \sqrt{(x2-x1)2 + (y2-y1)2} \qquad (1)$$

where x1 and y1 are starting points of the line and x2 and y2 are end points of the line respectively. Later that, when left key or right key is pressed angle variable is increasing or decreasing respectively. Then i decide the new end points of the line by equations:

$$x2 = x1 + r * \cos(angle)\ ,\ y2 = y1 + r * \sin(angle). \qquad (2)$$

*Lastly displaying the new line on the canvas.*

*Paddle Control Mechanism: Firstly i drew a rectangle with respect to given height and width corresponding to the paddle. Then in a while loop i checked if left key or right key is pressed , if so i incremented or decremented the x component of paddle position at a magnitüde of given paddle_speed variable respectively. Then i cleared and drew the paddle again with new coordinates after each movement.*

*(ex.: paddle_pos[0] -= paddle_speed;)* (3)

*Ball Movement Mechanism: Firstly i drew the ball with given features (radius, color) then after the very first space bar press ball starts moving with respect to the angle we calculated using a thin rod. To do so i used the ball_velocity and the angle the rod directed in equations;*

$$x\_Velocity = ball\_velocity * \cos(movingAngle)$$

$$y\_Velocity = ball\_velocity * \sin(movingAngle). \qquad (4)$$

*Then summed the ball's x and y coordinate with x_Velocity and y_Velocity in a while loop respectively. So ball will move until it collides a brick or wall. I determined if the ball collided with anything and changed the velocity of it properly. To do so, firstly, i determined if the ball collided any way then if this collision occured with a corner or not. For the first decision i used statements like;*

$$xBall + 8 >= xPaddle - paddle\_halfwidth \ \&\&$$

$$xBall - 8 <= xPaddle + paddle\_halfwidth \ \&\& \ \ldots \ etc. \qquad (5)$$

*For the second one i used the equation;*

$$cornerHit = (|xBall - xBrick| > brick\_halfwidth) \ \&\&$$

$$(|yBall - yBrick| > brick\_halfheight) \qquad (6)$$

*(for brick corner collision) and if it is true it means collision occured with a corner. Lastly i determined the x and y component of ball's velocity after collisions. If the collision occured with a horizontal edge i changed y component's sign, if it is with a vertical edge i changed x component's sign, if it is with a corner i changed both x and y components by drawing a standard normal vector where normal vector*

$$N = (xBall - xBrick, yBall – yBrick) \qquad (7)$$

*and by diving N to its magnitude i found the standard normal vector. Then i used*

$$Vnew = Vold - 2(Vold . N) N \qquad (8)$$

*formula to calculate new velocity vector. Then with the new components*

*Bricks: Firstly i drew the brick as filled rectangles with given features ( height, width, color, coordinates). Then after then every collision i remove the collided brick and its color from respective arrays and incremented the score 10 points. Then after those i drew the whole bricks rest again.*

*Implementation Details:*

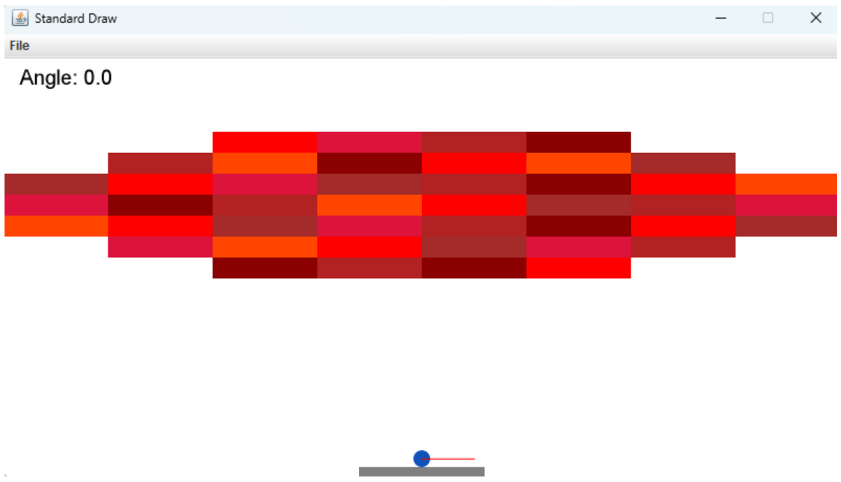


*Figure 1: Initial Game*

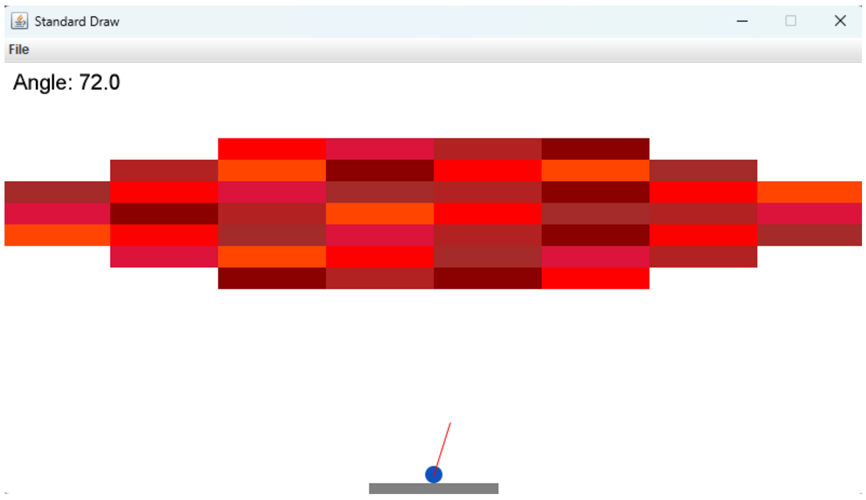


*Figure 2: Angle determining*

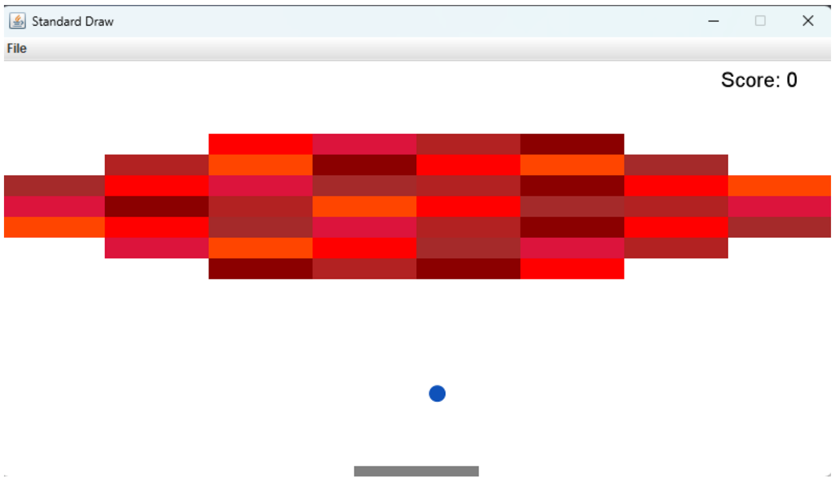


*Figure 3: First shooting*

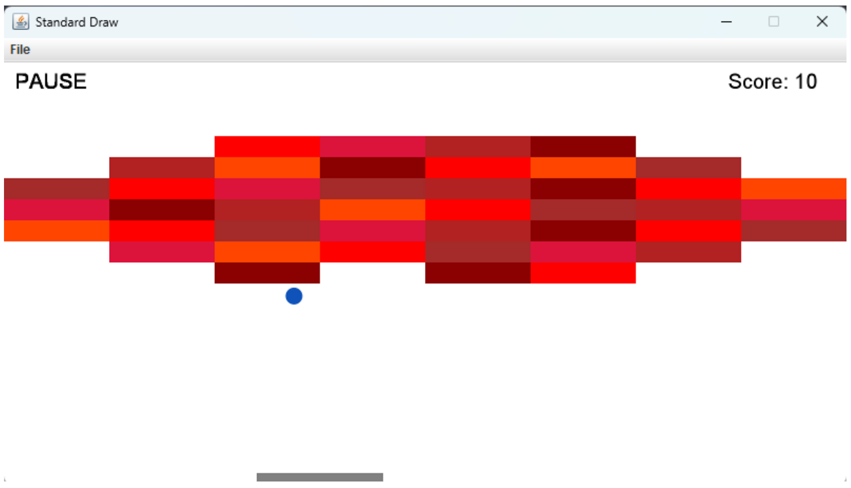


Figure 4: Pause screen

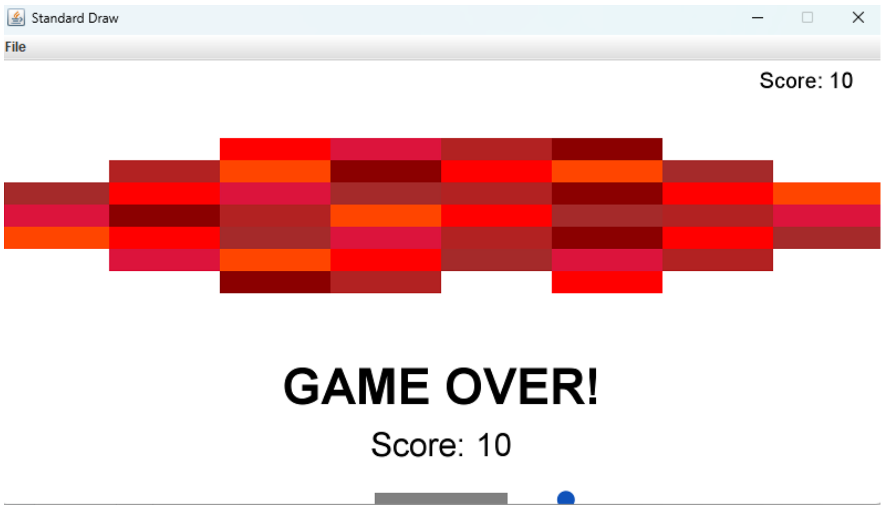


*Figure 5: Game over screen*

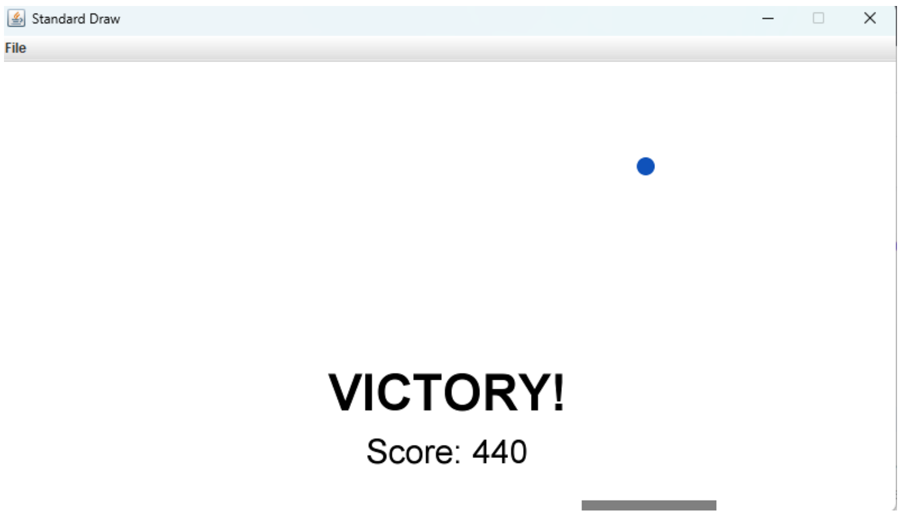


Figure 6: Victory  screen

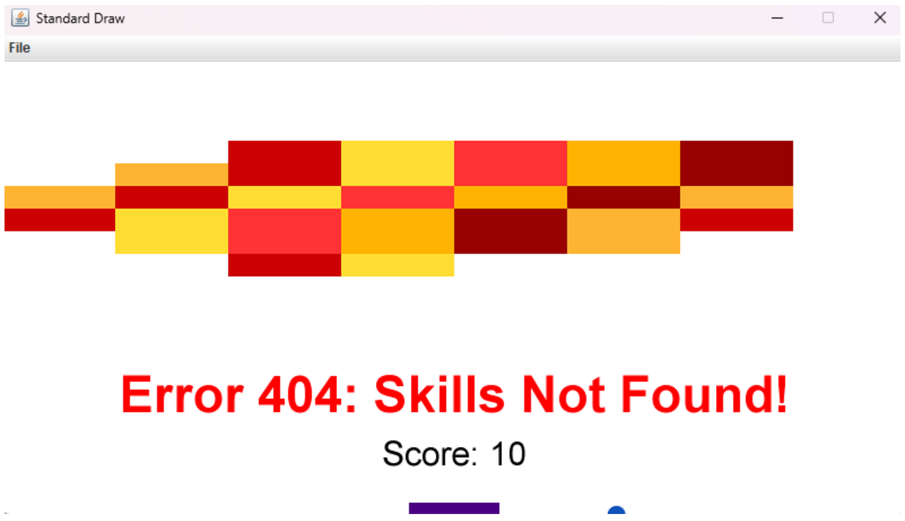
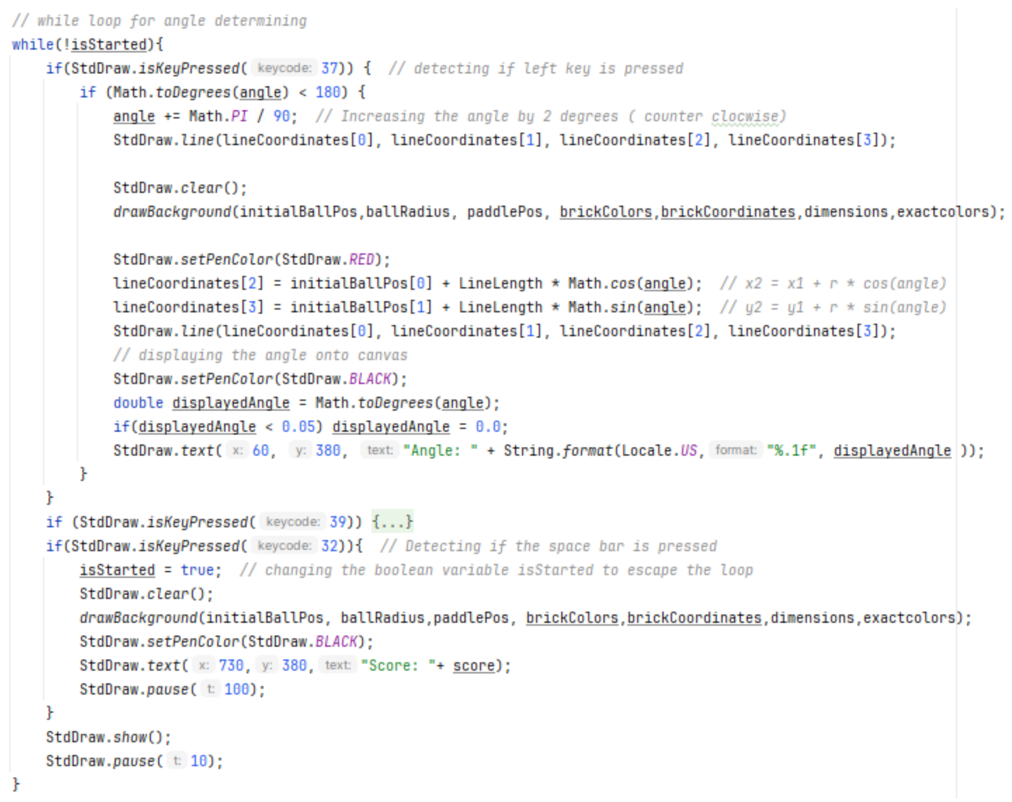


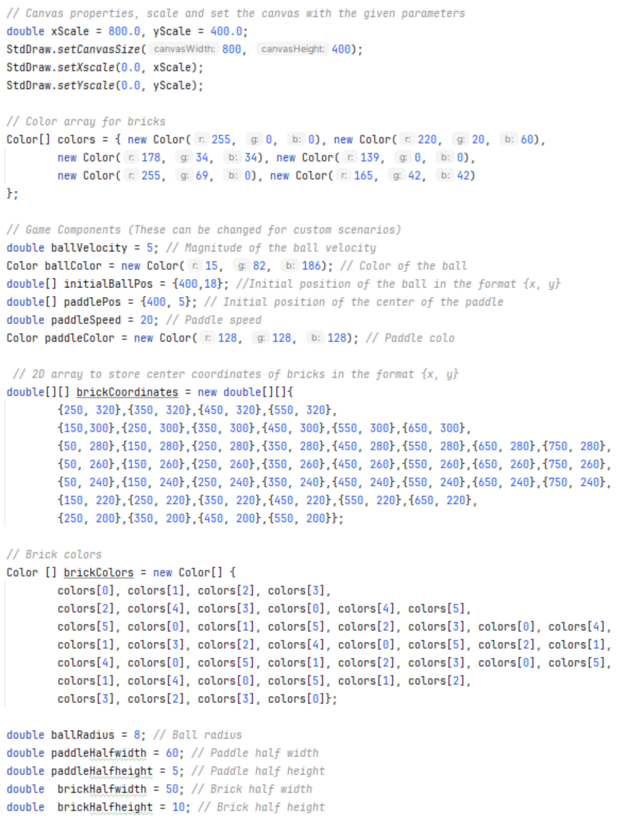
Figure 7: Updated game Game Over scree

## Code 1: Angle determining rod movement



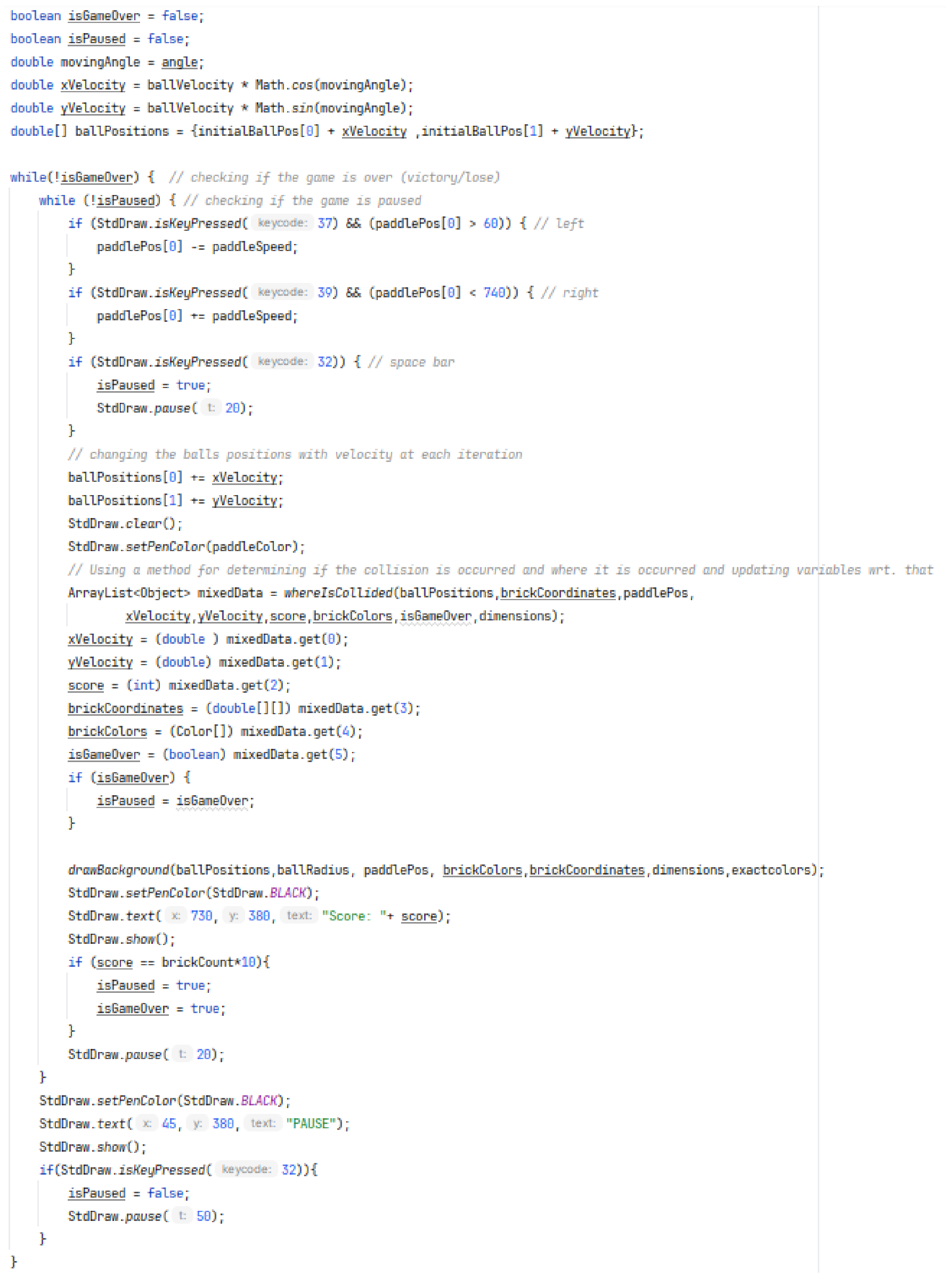
```java
// while loop for angle determining
while(!isStarted){
    if(StdDraw.isKeyPressed( keycode: 37)) {  // detecting if left key is pressed
        if (Math.toDegrees(angle) < 180) {
            angle += Math.PI / 90;  // Increasing the angle by 2 degrees ( counter clocwise)
            StdDraw.line(lineCoordinates[0], lineCoordinates[1], lineCoordinates[2], lineCoordinates[3]);

            StdDraw.clear();
            drawBackground(initialBallPos,ballRadius, paddlePos, brickColors,brickCoordinates,dimensions,exactcolors);

            StdDraw.setPenColor(StdDraw.RED);
            lineCoordinates[2] = initialBallPos[0] + LineLength * Math.cos(angle);  // x2 = x1 + r * cos(angle)
            lineCoordinates[3] = initialBallPos[1] + LineLength * Math.sin(angle);  // y2 = y1 + r * sin(angle)
            StdDraw.line(lineCoordinates[0], lineCoordinates[1], lineCoordinates[2], lineCoordinates[3]);
            // displaying the angle onto canvas
            StdDraw.setPenColor(StdDraw.BLACK);
            double displayedAngle = Math.toDegrees(angle);
            if(displayedAngle < 0.05) displayedAngle = 0.0;
            StdDraw.text( x: 60,  y: 380,  text: "Angle: " + String.format(Locale.US,  format: "%.1f", displayedAngle ));
        }
    }
    if (StdDraw.isKeyPressed( keycode: 39)) {...}
    if(StdDraw.isKeyPressed( keycode: 32)){  // Detecting if the space bar is pressed
        isStarted = true;  // changing the boolean variable isStarted to escape the loop
        StdDraw.clear();
        drawBackground(initialBallPos, ballRadius,paddlePos, brickColors,brickCoordinates,dimensions,exactcolors);
        StdDraw.setPenColor(StdDraw.BLACK);
        StdDraw.text( x: 730,  y: 380,  text: "Score: "+ score);
        StdDraw.pause( t: 100);
    }
    StdDraw.show();
    StdDraw.pause( t: 10);
}
```

## Code 2: Game parameters and variables (same as in description)



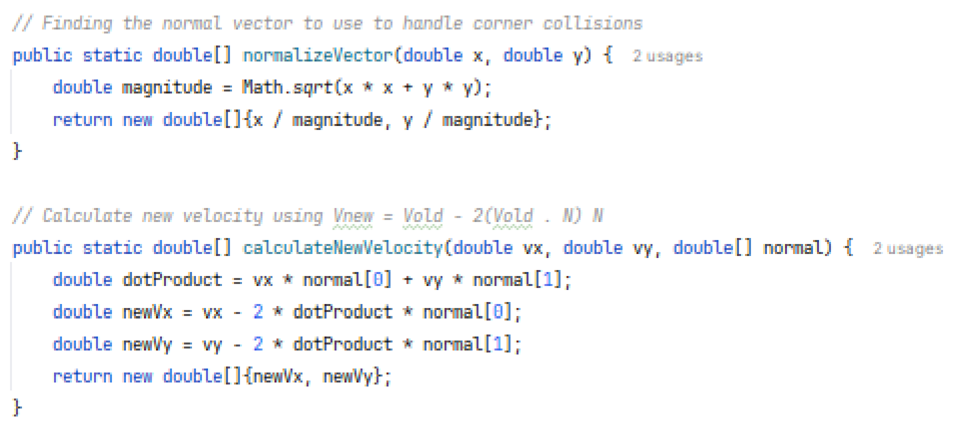
```java
// Canvas properties, scale and set the canvas with the given parameters
double xScale = 800.0, yScale = 400.0;
StdDraw.setCanvasSize( canvasWidth: 800,  canvasHeight: 400);
StdDraw.setXscale(0.0, xScale);
StdDraw.setYscale(0.0, yScale);

// Color array for bricks
Color[] colors = { new Color( r: 255,  g: 0,  b: 0), new Color( r: 220,  g: 20,  b: 60),
        new Color( r: 178,  g: 34,  b: 34), new Color( r: 139,  g: 0,  b: 0),
        new Color( r: 255,  g: 69,  b: 0), new Color( r: 165,  g: 42,  b: 42)
};

// Game Components (These can be changed for custom scenarios)
double ballVelocity = 5; // Magnitude of the ball velocity
Color ballColor = new Color( r: 15,  g: 82,  b: 186); // Color of the ball
double[] initialBallPos = {400,18}; //Initial position of the ball in the format {x, y}
double[] paddlePos = {400, 5}; // Initial position of the center of the paddle
double paddleSpeed = 20; // Paddle speed
Color paddleColor = new Color( r: 128,  g: 128,  b: 128); // Paddle colo

 // 2D array to store center coordinates of bricks in the format {x, y}
double[][] brickCoordinates = new double[][]{
        {250, 320},{350, 320},{450, 320},{550, 320},
        {150,300},{250, 300},{350, 300},{450, 300},{550, 300},{650, 300},
        {50, 280},{150, 280},{250, 280},{350, 280},{450, 280},{550, 280},{650, 280},{750, 280},
        {50, 260},{150, 260},{250, 260},{350, 260},{450, 260},{550, 260},{650, 260},{750, 260},
        {50, 240},{150, 240},{250, 240},{350, 240},{450, 240},{550, 240},{650, 240},{750, 240},
        {150, 220},{250, 220},{350, 220},{450, 220},{550, 220},{650, 220},
        {250, 200},{350, 200},{450, 200},{550, 200}};

// Brick colors
Color [] brickColors = new Color[] {
        colors[0], colors[1], colors[2], colors[3],
        colors[2], colors[4], colors[3], colors[0], colors[4], colors[5],
        colors[5], colors[0], colors[1], colors[5], colors[2], colors[3], colors[0], colors[4],
        colors[1], colors[3], colors[2], colors[4], colors[0], colors[5], colors[2], colors[1],
        colors[4], colors[0], colors[5], colors[1], colors[2], colors[3], colors[0], colors[5],
        colors[1], colors[4], colors[0], colors[5], colors[1], colors[2],
        colors[3], colors[2], colors[3], colors[0]};

double ballRadius = 8; // Ball radius
double paddleHalfwidth = 60; // Paddle half width
double paddleHalfheight = 5; // Paddle half height
double  brickHalfwidth = 50; // Brick half width
double  brickHalfheight = 10; // Brick half height
```

## Code 3: Ball and paddle movement, collision and game status arrangement



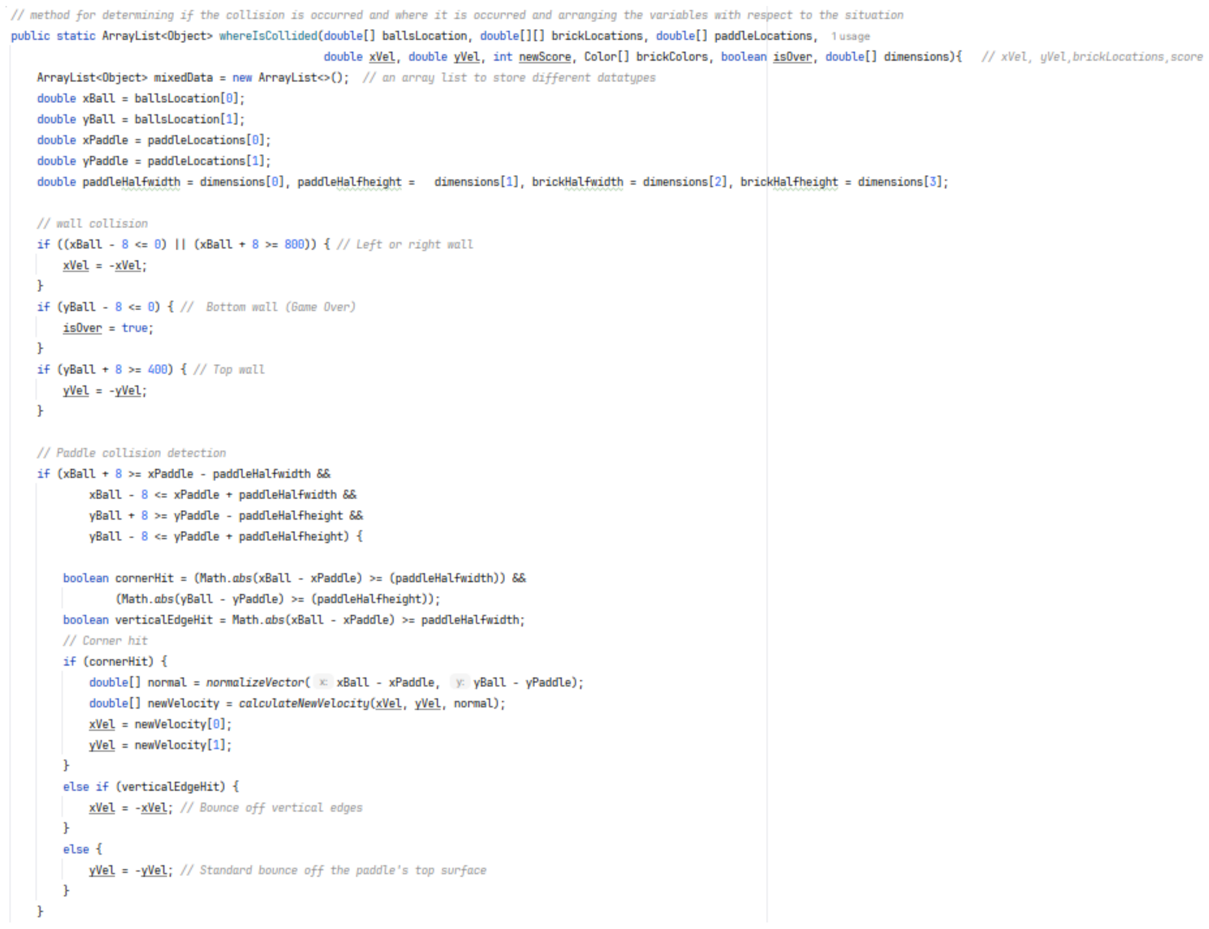
```java
boolean isGameOver = false;
boolean isPaused = false;
double movingAngle = angle;
double xVelocity = ballVelocity * Math.cos(movingAngle);
double yVelocity = ballVelocity * Math.sin(movingAngle);
double[] ballPositions = {initialBallPos[0] + xVelocity ,initialBallPos[1] + yVelocity};

while(!isGameOver) {  // checking if the game is over (victory/lose)
    while (!isPaused) { // checking if the game is paused
        if (StdDraw.isKeyPressed( keycode: 37) && (paddlePos[0] > 60)) { // left
            paddlePos[0] -= paddleSpeed;
        }
        if (StdDraw.isKeyPressed( keycode: 39) && (paddlePos[0] < 740)) { // right
            paddlePos[0] += paddleSpeed;
        }
        if (StdDraw.isKeyPressed( keycode: 32)) { // space bar
            isPaused = true;
            StdDraw.pause( t: 20);
        }
        // changing the balls positions with velocity at each iteration
        ballPositions[0] += xVelocity;
        ballPositions[1] += yVelocity;
        StdDraw.clear();
        StdDraw.setPenColor(paddleColor);
        // Using a method for determining if the collision is occurred and where it is occurred and updating variables wrt. that
        ArrayList<Object> mixedData = whereIsCollided(ballPositions,brickCoordinates,paddlePos,
                xVelocity,yVelocity,score,brickColors,isGameOver,dimensions);
        xVelocity = (double ) mixedData.get(0);
        yVelocity = (double) mixedData.get(1);
        score = (int) mixedData.get(2);
        brickCoordinates = (double[][]) mixedData.get(3);
        brickColors = (Color[]) mixedData.get(4);
        isGameOver = (boolean) mixedData.get(5);
        if (isGameOver) {
            isPaused = isGameOver;
        }

        drawBackground(ballPositions,ballRadius, paddlePos, brickColors,brickCoordinates,dimensions,exactcolors);
        StdDraw.setPenColor(StdDraw.BLACK);
        StdDraw.text( x: 730, y: 380, text: "Score: "+ score);
        StdDraw.show();
        if (score == brickCount*10){
            isPaused = true;
            isGameOver = true;
        }
        StdDraw.pause( t: 20);
    }
    StdDraw.setPenColor(StdDraw.BLACK);
    StdDraw.text( x: 45, y: 380, text: "PAUSE");
    StdDraw.show();
    if(StdDraw.isKeyPressed( keycode: 32)){
        isPaused = false;
        StdDraw.pause( t: 50);
    }
}
```

## Code 4: Methods for corner collisions



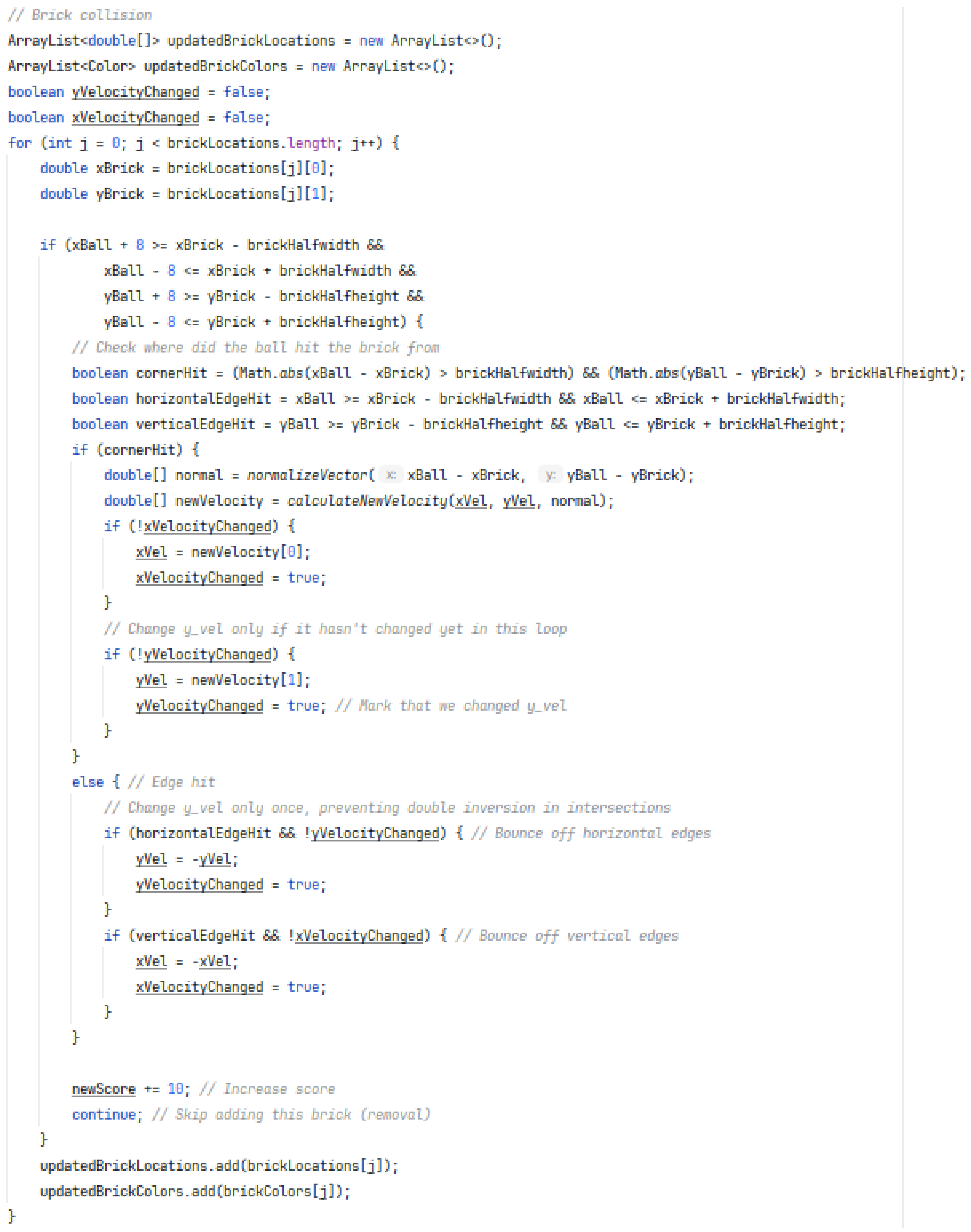
```java
// Finding the normal vector to use to handle corner collisions
public static double[] normalizeVector(double x, double y) {  2 usages
    double magnitude = Math.sqrt(x * x + y * y);
    return new double[]{x / magnitude, y / magnitude};
}

// Calculate new velocity using Vnew = Vold - 2(Vold . N) N
public static double[] calculateNewVelocity(double vx, double vy, double[] normal) {  2 usages
    double dotProduct = vx * normal[0] + vy * normal[1];
    double newVx = vx - 2 * dotProduct * normal[0];
    double newVy = vy - 2 * dotProduct * normal[1];
    return new double[]{newVx, newVy};
}
```

## Code 5: Wall and paddle collisions handling



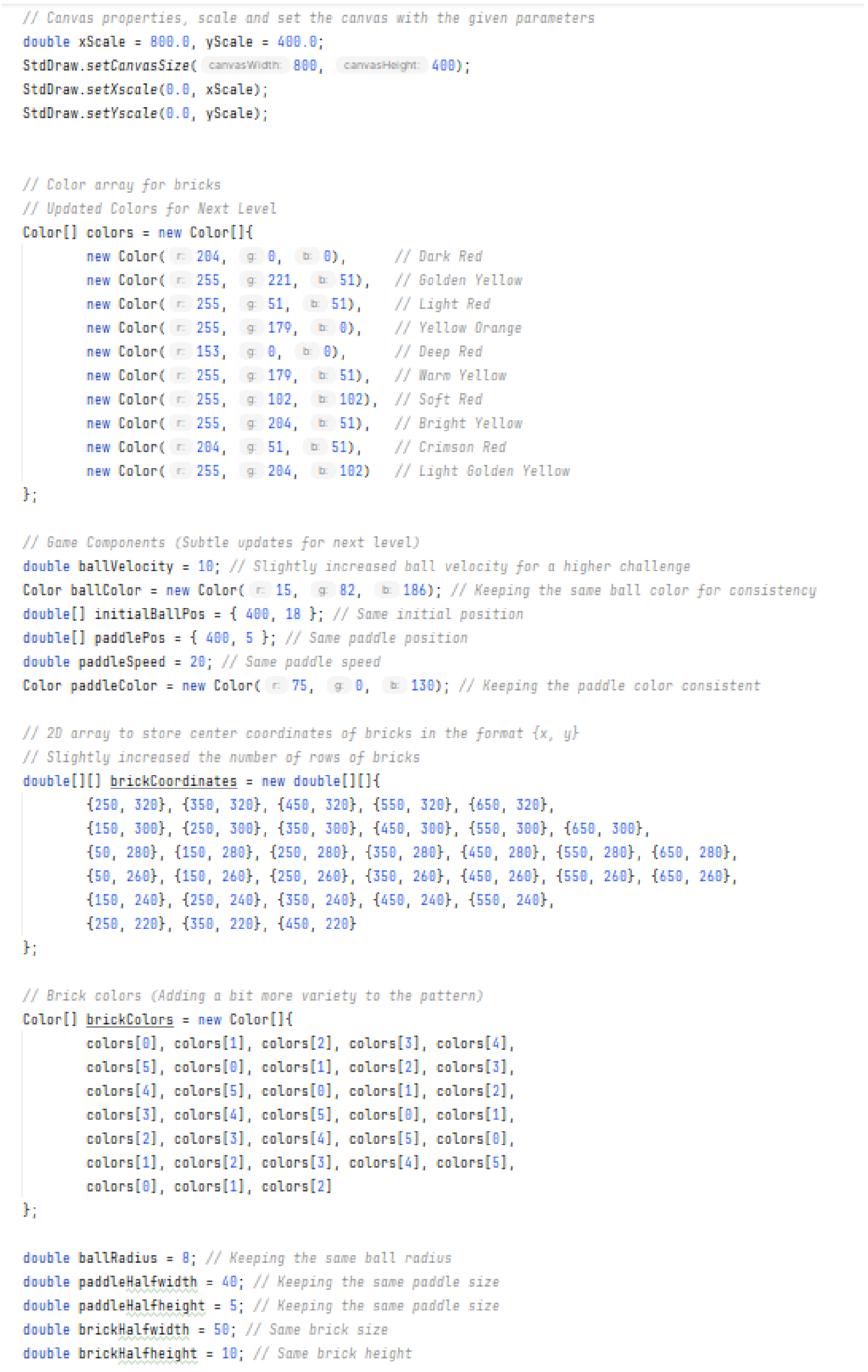
```java
// method for determining if the collision is occurred and where it is occurred and arranging the variables with respect to the situation
public static ArrayList<Object> whereIsCollided(double[] ballsLocation, double[][] brickLocations, double[] paddleLocations,  1 usage
                                                double xVel, double yVel, int newScore, Color[] brickColors, boolean isOver, double[] dimensions){   // xVel, yVel,brickLocations,score
    ArrayList<Object> mixedData = new ArrayList<>();  // an array list to store different datatypes
    double xBall = ballsLocation[0];
    double yBall = ballsLocation[1];
    double xPaddle = paddleLocations[0];
    double yPaddle = paddleLocations[1];
    double paddleHalfwidth = dimensions[0], paddleHalfheight =   dimensions[1], brickHalfwidth = dimensions[2], brickHalfheight = dimensions[3];

    // wall collision
    if ((xBall - 8 <= 0) || (xBall + 8 >= 800)) { // Left or right wall
        xVel = -xVel;
    }
    if (yBall - 8 <= 0) { //  Bottom wall (Game Over)
        isOver = true;
    }
    if (yBall + 8 >= 400) { // Top wall
        yVel = -yVel;
    }

    // Paddle collision detection
    if (xBall + 8 >= xPaddle - paddleHalfwidth &&
            xBall - 8 <= xPaddle + paddleHalfwidth &&
            yBall + 8 >= yPaddle - paddleHalfheight &&
            yBall - 8 <= yPaddle + paddleHalfheight) {

        boolean cornerHit = (Math.abs(xBall - xPaddle) >= (paddleHalfwidth)) &&
                (Math.abs(yBall - yPaddle) >= (paddleHalfheight));
        boolean verticalEdgeHit = Math.abs(xBall - xPaddle) >= paddleHalfwidth;
        // Corner hit
        if (cornerHit) {
            double[] normal = normalizeVector( x: xBall - xPaddle,  y: yBall - yPaddle);
            double[] newVelocity = calculateNewVelocity(xVel, yVel, normal);
            xVel = newVelocity[0];
            yVel = newVelocity[1];
        }
        else if (verticalEdgeHit) {
            xVel = -xVel; // Bounce off vertical edges
        }
        else {
            yVel = -yVel; // Standard bounce off the paddle's top surface
        }
    }
```

## Code 6: Brick collisions handling



```
// Brick collision
ArrayList<double[]> updatedBrickLocations = new ArrayList<>();
ArrayList<Color> updatedBrickColors = new ArrayList<>();
boolean yVelocityChanged = false;
boolean xVelocityChanged = false;
for (int j = 0; j < brickLocations.length; j++) {
    double xBrick = brickLocations[j][0];
    double yBrick = brickLocations[j][1];

    if (xBall + 8 >= xBrick - brickHalfwidth &&
            xBall - 8 <= xBrick + brickHalfwidth &&
            yBall + 8 >= yBrick - brickHalfheight &&
            yBall - 8 <= yBrick + brickHalfheight) {
        // Check where did the ball hit the brick from
        boolean cornerHit = (Math.abs(xBall - xBrick) > brickHalfwidth) && (Math.abs(yBall - yBrick) > brickHalfheight);
        boolean horizontalEdgeHit = xBall >= xBrick - brickHalfwidth && xBall <= xBrick + brickHalfwidth;
        boolean verticalEdgeHit = yBall >= yBrick - brickHalfheight && yBall <= yBrick + brickHalfheight;
        if (cornerHit) {
            double[] normal = normalizeVector( x: xBall - xBrick,  y: yBall - yBrick);
            double[] newVelocity = calculateNewVelocity(xVel, yVel, normal);
            if (!xVelocityChanged) {
                xVel = newVelocity[0];
                xVelocityChanged = true;
            }
            // Change y_vel only if it hasn't changed yet in this loop
            if (!yVelocityChanged) {
                yVel = newVelocity[1];
                yVelocityChanged = true; // Mark that we changed y_vel
            }
        }
        else { // Edge hit
            // Change y_vel only once, preventing double inversion in intersections
            if (horizontalEdgeHit && !yVelocityChanged) { // Bounce off horizontal edges
                yVel = -yVel;
                yVelocityChanged = true;
            }
            if (verticalEdgeHit && !xVelocityChanged) { // Bounce off vertical edges
                xVel = -xVel;
                xVelocityChanged = true;
            }
        }

        newScore += 10; // Increase score
        continue; // Skip adding this brick (removal)
    }
    updatedBrickLocations.add(brickLocations[j]);
    updatedBrickColors.add(brickColors[j]);
}
```

## Code 7: Updated game variables



```
// Canvas properties, scale and set the canvas with the given parameters
double xScale = 800.0, yScale = 400.0;
StdDraw.setCanvasSize( canvasWidth: 800,  canvasHeight: 400);
StdDraw.setXscale(0.0, xScale);
StdDraw.setYscale(0.0, yScale);


// Color array for bricks
// Updated Colors for Next Level
Color[] colors = new Color[]{
        new Color( r: 204,  g: 0,  b: 0),      // Dark Red
        new Color( r: 255,  g: 221,  b: 51),   // Golden Yellow
        new Color( r: 255,  g: 51,  b: 51),    // Light Red
        new Color( r: 255,  g: 179,  b: 0),    // Yellow Orange
        new Color( r: 153,  g: 0,  b: 0),      // Deep Red
        new Color( r: 255,  g: 179,  b: 51),   // Warm Yellow
        new Color( r: 255,  g: 102,  b: 102),  // Soft Red
        new Color( r: 255,  g: 204,  b: 51),   // Bright Yellow
        new Color( r: 204,  g: 51,  b: 51),    // Crimson Red
        new Color( r: 255,  g: 204,  b: 102)   // Light Golden Yellow
};

// Game Components (Subtle updates for next level)
double ballVelocity = 10; // Slightly increased ball velocity for a higher challenge
Color ballColor = new Color( r: 15,  g: 82,  b: 186); // Keeping the same ball color for consistency
double[] initialBallPos = { 400, 18 }; // Same initial position
double[] paddlePos = { 400, 5 }; // Same paddle position
double paddleSpeed = 20; // Same paddle speed
Color paddleColor = new Color( r: 75,  g: 0,  b: 130); // Keeping the paddle color consistent

// 2D array to store center coordinates of bricks in the format {x, y}
// Slightly increased the number of rows of bricks
double[][] brickCoordinates = new double[][]{
        {250, 320}, {350, 320}, {450, 320}, {550, 320}, {650, 320},
        {150, 300}, {250, 300}, {350, 300}, {450, 300}, {550, 300}, {650, 300},
        {50, 280}, {150, 280}, {250, 280}, {350, 280}, {450, 280}, {550, 280}, {650, 280},
        {50, 260}, {150, 260}, {250, 260}, {350, 260}, {450, 260}, {550, 260}, {650, 260},
        {150, 240}, {250, 240}, {350, 240}, {450, 240}, {550, 240},
        {250, 220}, {350, 220}, {450, 220}
};

// Brick colors (Adding a bit more variety to the pattern)
Color[] brickColors = new Color[]{
        colors[0], colors[1], colors[2], colors[3], colors[4],
        colors[5], colors[0], colors[1], colors[2], colors[3],
        colors[4], colors[5], colors[0], colors[1], colors[2],
        colors[3], colors[4], colors[5], colors[0], colors[1],
        colors[2], colors[3], colors[4], colors[5], colors[0],
        colors[1], colors[2], colors[3], colors[4], colors[5],
        colors[0], colors[1], colors[2]
};

double ballRadius = 8; // Keeping the same ball radius
double paddleHalfwidth = 40; // Keeping the same paddle size
double paddleHalfheight = 5; // Keeping the same paddle size
double brickHalfwidth = 50; // Same brick size
double brickHalfheight = 10; // Same brick height
```