

REPORT OF THE GAME

LINKS

<https://youtu.be/M-DFcY8MnxM> -> original game

<https://youtu.be/O5-MD6rRS0g> -> modified game

INTRODUCTION

DX_ball game is an old game of gaming history, and gameplay is as follows:

Gamer tries to destroy all the bricks by bouncing the ball from paddle, and controls the paddle by using left(<-) and right(>) arrows. Every brick and the ball collision adds 10 points to score. When the ball hits the ground, game is over. Moreover, gamer can adjust the initial shoot angle for his/her ease, and stop the game by using SPACE button.

StdDraw library is used for the construction of this game. This Java library can handle user interactions, and this is essential property for the gameplay. For instance, paddle movements are directed by keyboard inputs. Furthermore, StdDraw is able to draw simple shapes such as bricks, ball, and paddle; and interface and animation are created by this library.

GAME MECHANISM

- The paddle's movement is explained as follows:

Gamer controls the paddle by left and right arrows to keep the ball in the game, and the paddle's center moves depending on a constant speed.

- The ball's movement is explained as follows:

The magnitude of the ball's velocity is constant, but its direction changes according to collisions (How it changes is explained in collisions part.). The ball teleports the new location with velocity info for every movement as in *equation-1*, and the new game scene is drawn. In short, an animation is created by showing scenes in succession.

```
initialBallPos[0] = initialBallPos[0] + velocityX;  
initialBallPos[1] = initialBallPos[1] + velocityY;  
// New position of the ball (1)
```

- Collisions are explained as follows:

Collisions are categorized two different types: surface collisions and corner collisions.

Surface collisions are simple. If the ball collides to top of the brick (or paddle), only y component of the velocity changes as in *equation-2* (same way for other surface collisions), and 5 surface collisions are coded one by one (top, bottom, right, left of the bricks and top of

the paddle). Furthermore, surface collisions are handled at the end of the collision code part as shown *equation-3* with brick count which are collided to prevent bugs when the ball hits two bricks at the same time.

```

velocityX = -velocityX;
velocityY = -velocityY;
(2)

if (collisionCountBottom == 1 || collisionCountBottom == 2) {
    velocityY = -velocityY;

// Implement velocity changes for one time whether the ball collides with one or two
bricks (3)

```

Corner collisions are calculated by using dot product. Two vectors used in this product are velocity vector and normal vector. New velocity vector (which means with its direction) is determined as shown in *equation-4*, *equation-5* & *equation-6*, and 6 corner collisions are code one by one (top-left, bottom-left, top-right, bottom-right of the bricks / top-left and top-right of the paddle).

```

dot = velocityX * (dx / distance) + velocityY * (dy / distance); (4)
velocityX = velocityX - 2 * dot * (dx / distance); (5)
velocityY = velocityY - 2 * dot * (dy / distance); (6)

```

IMPLEMENTATION DETAILS

Firstly, general info is handled (canvas size, x and y scale, brick coordinates...) as shown in *code-1*. Existince info is added to bricks by a loop as shown in *code-2*, and this existince state is changed after collisions. Pressing SPACE ends the loop as shown in *code-3* and starts the game with determined shot angle. Main animation loop runs after it by controlling whether the game is paused or not by *code-4*. Then, all bricks are controlled whether collide with a loop *code-5*, and the game's result is handled at the *code-6*. At the end of the main while (*code-7*), the new scene of the game is drawn.

```

// Canvas properties, scale and set the canvas with the given parameters
double X_SCALE = 800.0, Y_SCALE = 400.0;
StdDraw.setCanvasSize(canvasWidth: 800, canvasHeight: 400);
StdDraw.setXscale(0.0, X_SCALE);
StdDraw.setYscale(0.0, Y_SCALE);
StdDraw.enableDoubleBuffering(); // to prevent flickering

// Color array for bricks (first import java.awt.Color)
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 BALL_RADIUS = 8; // Ball radius
double BALL_VELOCITY = 5; // Magnitude of the ball velocity
Color BALL_COLOR = 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 PADDLE_HALFWIDTH = 60; // Paddle half width
double PADDLE_HALFHEIGHT = 5; // Paddle half height
double PADDLE_SPEED = 20; // Paddle speed
Color PADDLE_COLOR = new Color(r: 128, g: 128, b: 128); // Paddle color
double BRICK_HALFWIDTH = 50; // Brick half width
double BRICK_HALFHEIGHT = 10; // Brick half height

// 2D array to store center coordinates of bricks in the format {x, y}
double[][] BRICK_COORDINATES = 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}
};

```

Code-1

```

// Brick coordinates whether being destroyed info
double[][] brickCoordinatesState = new double[BRICK_COORDINATES.length][3];

for (int i = 0; i < BRICK_COORDINATES.length; i++) { // add -1 to brick coordinates to show they did not encounter with a collision
    brickCoordinatesState[i][0] = BRICK_COORDINATES[i][0];
    brickCoordinatesState[i][1] = BRICK_COORDINATES[i][1];
    brickCoordinatesState[i][2] = -1;
}

```

Code-2

```

// Starting loop of the game
while (true) {
    if (StdDraw.isKeyPressed(keycode: 37)) { // move the arrow to the left with <- button
        if (angle < 180) {
            angle += 1; // one degree change
        }
    }
    if (StdDraw.isKeyPressed(keycode: 39)) { // move the arrow to the right -> button
        if (angle > 0) {
            angle -= 1; // one degree change
        }
    }
    if (StdDraw.isKeyPressed(keycode: 32)) { // throw the ball using SPACE button
        break; // end loop and start the game
    }

    // Draw the new scene of the game
    StdDraw.clear(StdDraw.WHITE); // clear the background

    StdDraw.setPenColor(StdDraw.BLACK);
    StdDraw.setFont(new Font(name: "Helvetica", Font.PLAIN, size: 18));

```

Code-3

```

// Main animation loop
while (true) {

    if (StdDraw.isKeyPressed( keycode: 32)) { // control of SPACE press
        long currentTime = System.currentTimeMillis(); // system time

        if (currentTime - lastSpaceTime >= 200) { // wait 200 ms
            spaceCount += 1;

            if (!spacePaused && spaceCount > 1) { // ignore first press
                spacePaused = true;
            } else if (spacePaused && spaceCount > 1) {
                spacePaused = false;
            }

            lastSpaceTime = currentTime; // change last time SPACE pressed
        }
    }

    if (spacePaused == false) { // continue game if it is not paused

        // Paddle moves
        if (StdDraw.isKeyPressed( keycode: 37)) { // move the paddle to the left
            if (paddlePos[0] - PADDLE_HALFWIDTH > 0) { // consider limitations
                paddlePos[0] = paddlePos[0] - PADDLE_SPEED;
            }
        }
        if (StdDraw.isKeyPressed( keycode: 39)) { // move the paddle to the right
            if (paddlePos[0] + PADDLE_HALFWIDTH < X_SCALE) { // consider limitations
                paddlePos[0] = paddlePos[0] + PADDLE_SPEED;
            }
        }
    }
}

```

Code-4

```

// for adjustment of double collisions
int collisionCountTop = 0;
int collisionCountBottom = 0;
int collisionCountRight = 0;
int collisionCountLeft = 0;

// Check all bricks one by one
for (int i = 0; i < BRICK_COORDINATES.length; i++) {

    if (brickCoordinatesState[i][2] == -1) { // the brick did not collide

        // Coordinates of the brick's edges
        double brickLeft = brickCoordinatesState[i][0] - BRICK_HALFWIDTH;
        double brickRight = brickCoordinatesState[i][0] + BRICK_HALFWIDTH;
        double brickBottom = brickCoordinatesState[i][1] - BRICK_HALFHEIGHT;
        double brickTop = brickCoordinatesState[i][1] + BRICK_HALFHEIGHT;

        // Coordinates of the ball's edges
        double ballLeft = initialBallPos[0] - BALL_RADIUS;
        double ballRight = initialBallPos[0] + BALL_RADIUS;
        double ballBottom = initialBallPos[1] - BALL_RADIUS;
        double ballTop = initialBallPos[1] + BALL_RADIUS;

        // Collisions with brick's 4 surfaces
        if (initialBallPos[0] >= brickLeft && initialBallPos[0] <= brickRight && ballTop >= br
    }
}

```

Code-5

```

if (allBricksDestroyed) { // end the game by victory

    StdDraw.clear(StdDraw.WHITE); // clear the background to get rid of the last brick

    StdDraw.setPenColor(BALL_COLOR); // arrange color for the ball
    StdDraw.filledCircle(initialBallPos[0], initialBallPos[1], BALL_RADIUS); // draw circle on the screen
    StdDraw.setPenColor(PADDLE_COLOR); // arrange color for the paddle
    StdDraw.filledRectangle(paddlePos[0], paddlePos[1], PADDLE_HALFWIDTH, PADDLE_HALFHEIGHT); // draw paddle on the screen

    // Victory display
    StdDraw.setPenColor(StdDraw.BLACK);
    StdDraw.setFont(new Font( name: "Helvetica", Font.BOLD, size: 30));
    StdDraw.textLeft( x: 328, y: 160, text: "VICTORY!" );
    StdDraw.setFont(new Font( name: "Helvetica", Font.PLAIN, size: 22));
    StdDraw.textLeft( x: 340, y: 132, text: "SCORE: " + score);

    StdDraw.setPenColor(StdDraw.WHITE);
    StdDraw.filledRectangle( x: 745, y: 375, halfWidth: 70, halfHeight: 10); // cover the scoreboard

    StdDraw.show(); // show the drawing on the screen
    StdDraw.pause( t: 20); // pause the drawing at each iteration
    break;

}

if (Math.abs(initialBallPos[1]) <= BALL_RADIUS) { // end the game by game over

    // Game Over display
    StdDraw.setPenColor(StdDraw.BLACK);
    StdDraw.setFont(new Font( name: "Helvetica", Font.BOLD, size: 30));
    StdDraw.textLeft( x: 308, y: 160, text: "GAME OVER!" );
    StdDraw.setFont(new Font( name: "Helvetica", Font.PLAIN, size: 22));

}

```

Code-6

```

// PAUSED sign at the top left
if (spacePaused == true) {
    StdDraw.setPenColor(StdDraw.BLACK);
    StdDraw.setFont(new Font( name: "Helvetica", Font.BOLD, size: 19));
    StdDraw.textLeft( x: 15, y: 370, text: "PAUSED");
    StdDraw.show(); // show the drawing on the screen
    StdDraw.pause( t: 20); // pause the drawing at each iteration
}

// Draw the new scene of the game
StdDraw.setPenColor(BALL_COLOR); // arrange color for the ball
StdDraw.filledCircle(initialBallPos[0], initialBallPos[1], BALL_RADIUS); // draw circle on the screen
StdDraw.setPenColor(PADDLE_COLOR); // arrange color for the paddle
StdDraw.filledRectangle(paddlePos[0], paddlePos[1], PADDLE_HALFWIDTH, PADDLE_HALFHEIGHT); // draw paddle on the screen

StdDraw.setPenColor(StdDraw.BLACK);
StdDraw.setFont(new Font( name: "Helvetica", Font.PLAIN, size: 19));
StdDraw.textLeft( x: 685, y: 375, text: "Score: " + score); // draw scoreboard

// Draw bricks with a loop
for (int i = 0; i < BRICK_COORDINATES.length; i++) {
    double brickX = BRICK_COORDINATES[i][0];
    double brickY = BRICK_COORDINATES[i][1];
    StdDraw.setPenColor(BRICK_COLORS[i]);
    if (brickCoordinatesState[i][2] == -1) { // bricks which exist
        StdDraw.filledRectangle(brickX, brickY, BRICK_HALFWIDTH, BRICK_HALFHEIGHT); // draw brick on the screen
    }
}

StdDraw.show(); // show the drawing on the screen
StdDraw.pause( t: 20); // pause the drawing at each iteration

```

Code-7

(Figure-1 is the original mode of the game)

In modified version, there are the following changes:

- Mode decision display-> It allows gamer to choose a mode. (as shown in *figure-2*)
- New ball color (as shown in *figure-3*)
- New brick colors (as shown in *figure-3*)
- New brick coordinates (as shown in *figure-3*)
- New brick color scheme (as shown in *figure-3*)
- New paddle speed-> It makes the game more difficult.
- New paddle color (as shown in *figure-3*)
- New background colors-> Background color change every time the ball hits the paddle's surface (as shown in *figure-3 & figure-4*)
- Reverse arrows-> Paddle shifts right by pressing left arrow and left by pressing right. This change makes the game more difficult to play.
- New PAUSED color (as shown in *figure-3*)

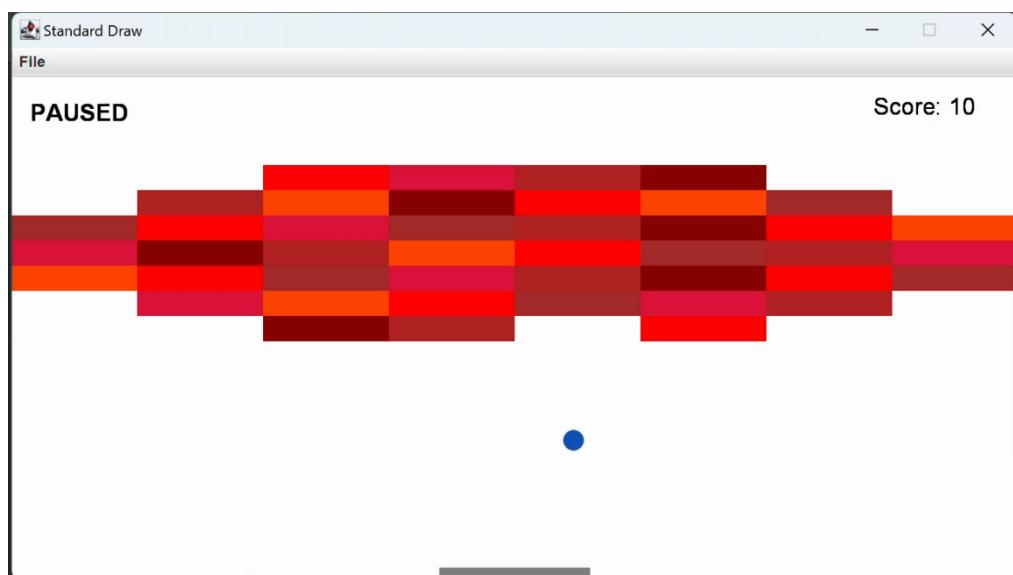


Figure-1

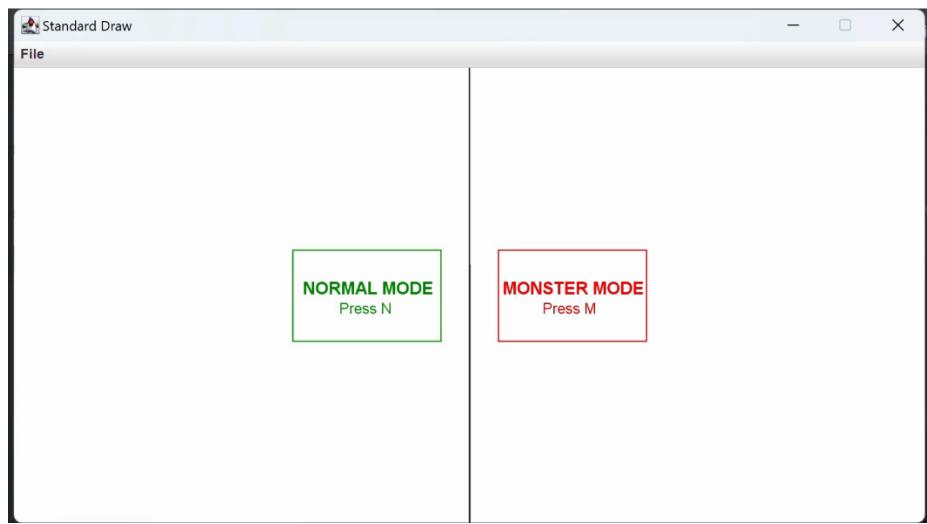


Figure-2

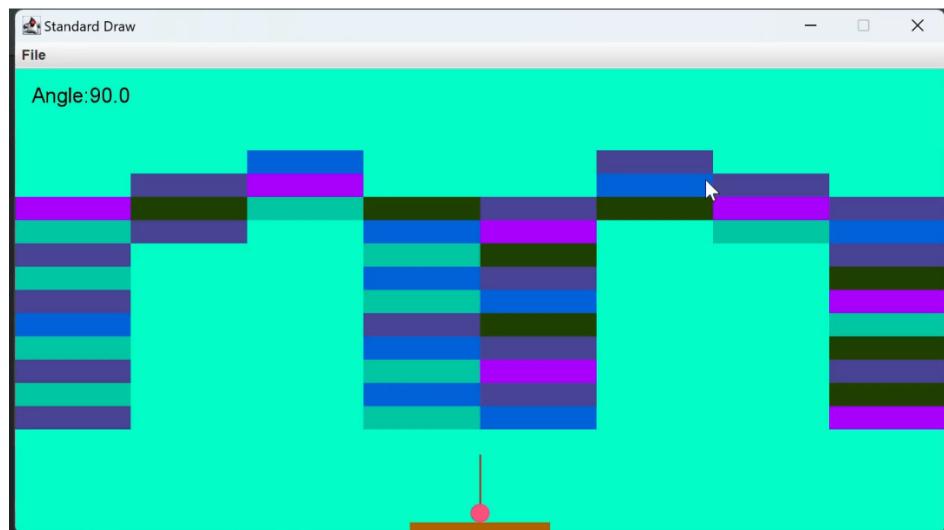


Figure-3



Figure-4