Report On

Brick Breaker Game

Submitted in partial fulfillment of the requirements of the Course project in Semester III of Second Year Artificial Intelligence and Data Science

by
Ayush Mayekar (Roll No. 28)
Yash Mayekar (Roll No. 29)
Sainath Khot (Roll No. 22)

Supervisor
Ms.Sneha Yadav

University of Mumbai

Vidyavardhini's College of Engineering & Technology

Department of Artificial Intelligence and Data Science

(2023-24)

Vidyavardhini's College of Engineering & Technology
Department of Artificial Intelligence and Data Science

CERTIFICATE

This is to certify that the project entitled “Title of the project” is a bonafide work of "Name1 Surname1 (Roll No. xx), Name2 Surname2 (Roll No. xx), Name3 Surname3 (Roll No. xx), Name4 Surname4 (Roll No. xx)" submitted to the University of Mumbai in partial fulfillment of the requirement for the Course project in semester III of Second Year Artificial Intelligence and Data Science engineering.

Supervisor

Ms.Sneha Yadav

Dr. Tatwadarshi P. N.
Head of Department

ABSTRACT

The "Brick Breaker" project presents the design and development of a classic arcade-style game in a Java-based environment. The objective of this project is to create an engaging and entertaining gaming experience while demonstrating proficiency in Java programming, graphical user interface (GUI) design, and game development.

The game "Brick Breaker" is a familiar concept, where players control a paddle at the bottom of the screen to bounce a ball, eliminating bricks strategically arranged in various patterns. The game's logic, physics, and user interaction are central to this project.

This report provides a comprehensive overview of the project's key components:

The project sets out to build an interactive and enjoyable game with Java and Swing components, requiring the development of the game's core logic, collision detection, and rendering.

The report includes an illustrative block diagram and a detailed description of the game's architecture and mechanics. It explains how user input, physics, and collision detection influence gameplay.

Various modules, such as player control, ball movement, brick management, and scorekeeping, are described in detail, highlighting their functions within the game.

The report outlines the software and hardware environment used for development, including Java, Swing, and the development environment. It discusses programming languages, libraries, and tools involved.

The project's code implementation is presented and explained to demonstrate how game features are realized. This section provides insights into the coding aspects of the game.

The report discusses the project's outcomes, such as the successful creation of a playable game, user experience, and the challenges faced during development. It concludes by summarizing the achievements and outlining possible future enhancements.

In conclusion, this project represents a practical application of game development skills using Java and Swing. It offers an in-depth exploration of game mechanics, programming techniques, and GUI design. The "Brick Breaker" game serves as an engaging showcase of software development capabilities, delivering both a playable and entertaining experience to users.

Table of Contents

Pg. No

Problem Statement

The problem statement for the Brick Breaker game involves designing and implementing a classic arcade-style game using Java and Swing components. The objective is to create an interactive and entertaining gaming experience, demonstrating proficiency in game development, graphical user interface (GUI) design, and Java programming. Key aspects of the problem statement include:

1. **Game Design:** Developing a game that involves controlling a paddle to bounce a ball, eliminating bricks arranged in patterns on the screen.

2. **Core Game Logic:** Implementing the underlying game logic, including physics for ball movement, collision detection with the paddle and bricks, and scoring.

3**. User Interaction:** Designing user-friendly controls for paddle movement and ensuring an engaging player experience.

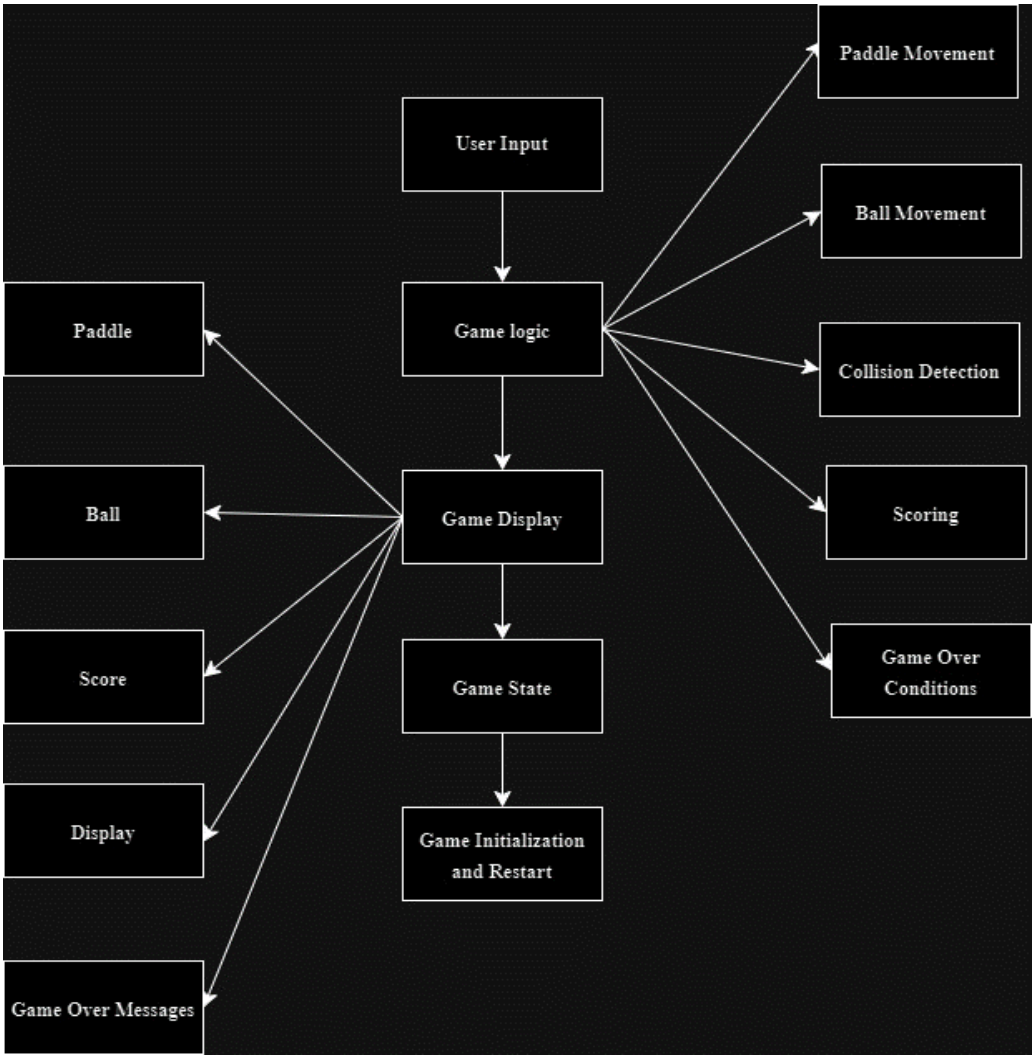
4. **Software Development:** Using Java and Swing components to create the game, including coding the game logic and GUI elements.

5. **Entertainment and Engagement:** Crafting a game that is both fun to play and showcases the programming and design skills.

The problem statement challenges developers to build a functional and enjoyable game while demonstrating proficiency in game development techniques.

4.

Block diagram, its description and working



1. **Initialization**: The game starts by initializing essential components, including the game board, paddle, ball, bricks, and score. The player's score is set to zero.

2**. Game Loop:** The core of the game is a continuous loop controlled by a timer. This loop repeatedly updates the game's components to provide smooth animation.

3**. Paddle Control**:The player can move the paddle left and right using the keyboard (typically the left and right arrow keys). The paddle's position is updated based on the player's input.

4. **Ball Movement**: The ball moves according to its current direction (ballXdir and ballYdir). The game checks for collisions with various game elements.

5. **Collision Detection:**
- **Wall Collisions:** The game checks if the ball collides with the walls of the game board. If so, it changes the ball's direction.
- **Paddle Collision:**The game checks if the ball hits the paddle. If it does, the ball bounces back.
- **Brick Collision:** The game checks if the ball collides with any of the bricks in the grid. If a collision occurs, the brick is removed from the grid, the player's score increases, and the ball's direction changes.

6. **Winning and Losing Conditions:**
- **Winning Condition**: The game checks if all bricks are destroyed. If they are, it displays a "YOU WON" message.
- **Losing Condition**: If the ball falls below the paddle (e.g., ballposY > 570), the game ends, displaying a "Game Over" message along with the player's final score.

7. **Game Restart:** The player can restart the game by pressing the "Enter" key. When this happens, the game resets by reinitializing the components, including the paddle, ball, bricks, score, and totalBricks.

8. **Rendering:** The game continuously redraws the game board, including the paddle, ball, bricks, and score, using the paint method. This ensures that the game's visual representation reflects the current state of the game objects.

9. **User Interaction:** The game listens for user input, such as keyboard events (left and right arrow keys for paddle movement) and "Enter" key presses to restart the game.

This combination of mechanics provides the core gameplay of a Brick Breaker game. The player's objective is to control the paddle, bounce the ball, and destroy all the bricks while avoiding the ball falling below the paddle. The game provides a fun and challenging experience for players, and the scorekeeping adds a competitive element to the game.

6.

Module Description

1**. Game Initialization and Setup:**
- Description: This module handles the initial setup of the game. It creates the game board, initializes game components, including the paddle, ball, and bricks. It also sets up the initial score and starts the game timer.
- Key Functions:
  - Initialize game components
  - Start the game timer

2**. Game Loop and Animation:**
- Description: This module controls the main game loop, ensuring the game continuously updates and provides smooth animation. It governs the flow of the game by repeatedly invoking game logic.
- Key Functions:
  - Game loop
  - Animation and frame updates

3**. Paddle Control:**
- Description: This module handles player input for moving the paddle left and right. It updates the position of the paddle based on keyboard input.
- Key Functions:
  - Listen for keyboard input
  - Update the paddle's position

4. **Ball Movement and Collision Detection:**
- Description: Responsible for controlling the ball's movement and detecting collisions. It checks for collisions with the walls, paddle, and bricks, and it adjusts the ball's direction accordingly.
- Key Functions:
  - Ball movement
  - Collision detection

5. **Brick Management:**
- Description: This module handles the creation of the brick grid, collision detection with bricks, and removal of bricks upon collision. It also manages the player's score and total brick count.
- Key Functions:
  - Create and manage the brick grid
  - Update the player's score
  - Track the total number of bricks

6. **Game Over and Win Conditions:**
   - Description: This module checks for game-over and win conditions. It determines whether the player has lost the game (ball below the paddle) or won (all bricks destroyed).
   - Key Functions:
     - Check for game-ending conditions
     - Display game-over or win messages

7. Game Restart:
   - Description: Responsible for restarting the game when the player presses the "Enter" key after a win or loss. It resets all game components, allowing the player to play again.
   - Key Functions:
     - Reset game components

8. **Rendering and Graphics:**
   - Description: Manages the rendering and display of the game. It continuously updates the visual representation of game elements, including the paddle, ball, bricks, and score.
   - Key Functions:
     - Redraw game elements
     - Update the visual display

9. **User Interaction:**
   - Description: Listens for and handles user input, including keyboard events for paddle movement and game restart.
   - Key Functions:
     - Listen for keyboard input
     - Handle user interactions

These modules work together to create a functioning Brick Breaker game, providing an enjoyable gaming experience for players while managing game states, animations, and interactions.

8.

Brief description of software & hardware used and its programming

Software Used:

1. **Java:** The game is programmed in the Java programming language, making use of the Java Swing library for creating the graphical user interface (GUI) and rendering game elements.

Hardware Used:

1. **Computer**: The game runs on a standard computer or laptop. The hardware requirements are minimal, making it accessible on a wide range of devices.

Programming of the Game:

The Brick Breaker game is programmed in Java, utilizing object-oriented principles for organizing the code. Here's a brief description of the programming aspects:

1. **Game Loop:** The core of the game is a continuous game loop that updates the game's logic and graphics. It is controlled by a Timer object, which triggers regular action events. The actionPerformed method is responsible for executing game logic within the loop.

2. **Graphics and Rendering:** The game's graphical elements, such as the paddle, ball, and bricks, are drawn on a JPanel using Java's Swing library. The paint method is used to render these components and update the visual display.

3. **User Input:** The game responds to user input through keyboard events. The KeyListener interface is implemented to capture and handle key presses for controlling the paddle's movement and restarting the game.

4. **Collision Detection:** The game involves detecting collisions between the ball and various elements, including the walls, paddle, and bricks. Collision detection is implemented using the intersects method of Java's Rectangle class.

5. **Game States:** The game maintains different states, including the play state, game over state, and win state. These states are controlled by Boolean variables and determine the game's behavior.

6. **Brick Grid Generation:** The generation of the brick grid is organized using the MapGenerator class. This class creates the grid of bricks and is responsible for tracking the state of each brick (e.g., intact or destroyed).

7. **Win and Lose Conditions:** The game checks for win and lose conditions in the actionPerformed method. If the player destroys all bricks, they win, and if the ball goes below the paddle, they lose.

8. **Game Restart:** The game allows the player to restart after a win or loss. The "Enter" key press triggers a game reset, returning the player to the initial game state.

Overall, the game is implemented in a modular and structured manner, ensuring that each aspect of the game, from user input to collision detection, is well-organized and functions cohesively to provide an enjoyable gaming experience. The use of Java and its built-in libraries simplifies graphical rendering and user interaction, making it a suitable choice for developing 2D games like Brick Breaker.

Code

```
package brickBracker;

import javax.swing.JFrame;

public class Main {

  public static void main(String[] args) {
    JFrame obj = new JFrame();
    Gameplay gamePlay = new Gameplay();
    obj.setBounds(10, 10, 700, 600);
    obj.setTitle("Breakout Ball");
    obj.setResizable(false);
    obj.setVisible(true);
    obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Corrected "EXIT ON
CLOSE" to "EXIT_ON_CLOSE"
    obj.add(gamePlay);
  }
}


package brickBracker;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JPanel;
import javax.swing.Timer;

public class Gameplay extends JPanel implements KeyListener, ActionListener {
    private boolean play = false;
    private int score = 0;
    private int totalBricks = 21;
    private Timer timer;
    private int delay = 15;
    private int playerX = 310;
    private int ballposX = 120;
    private int ballposY = 350;
    private int ballXdir = -2;
    private int ballYdir = -1;
    private MapGenerator map;
```

```
public Gameplay() {
    map = new MapGenerator(3, 7);
    addKeyListener(this); // Change "addkeyListener" to "addKeyListener"
    setFocusable(true);
    setFocusTraversalKeysEnabled(false);
    timer = new Timer(delay, this);
    timer.start();
}

public void paint(Graphics g) {
    super.paint(g); // Call the superclass paint method

    // Background
    g.setColor(Color.black);
    g.fillRect(1, 1, 692, 592);

    // Drawing map
    map.draw((Graphics2D) g);

    // Borders
    g.setColor(Color.yellow);
    g.fillRect(0, 0, 3, 592);
    g.fillRect(0, 0, 692, 3);
    g.fillRect(691, 0, 3, 592);

    // Scores
    g.setColor(Color.white);
    g.setFont(new Font("serif", Font.BOLD, 25));
    g.drawString("" + score, 590, 30);

    // Paddle
    g.setColor(Color.green);
    g.fillRect(playerX, 550, 100, 8);

    // The ball
    g.setColor(Color.yellow);
    g.fillOval(ballposX, ballposY, 20, 20);

    if (totalBricks <= 0) {
        play = false;
        ballXdir = 0;
        ballYdir = 0;
        g.setColor(Color.RED);
        g.setFont(new Font("serif", Font.BOLD, 30));
        g.drawString("YOU WON", 260, 300);
    }
```

```
if (ballposY > 570) {
        play = false;
        ballXdir = 0;
        ballYdir = 0;
        g.setColor(Color.RED);
        g.setFont(new Font("serif", Font.BOLD, 30));
        g.drawString("Game over, Scores: " + score, 190, 300);
        g.setFont(new Font("serif", Font.BOLD, 20));
        g.drawString("Press Enter to Restart", 230, 350);
      }
   }

   public void actionPerformed(ActionEvent e) {
      timer.start();
      if (play) {
         if (new Rectangle(ballposX, ballposY, 20, 20).intersects(new Rectangle(playerX, 550,
100, 8))) {
            ballYdir = -ballYdir;
         }

         A: for (int i = 0; i < map.map.length; i++) {
            for (int j = 0; j < map.map[0].length; j++) {
               if (map.map[i][j] > 0) {
                  int brickX = j * map.brickWidth + 80;
                  int brickY = i * map.brickHeight + 50;
                  int brickWidth = map.brickWidth;
                  int brickHeight = map.brickHeight;
                  Rectangle rect = new Rectangle(brickX, brickY, brickWidth, brickHeight);
                  Rectangle ballRect = new Rectangle(ballposX, ballposY, 20, 20);
                  Rectangle brickRect = rect;

                  if (ballRect.intersects(brickRect)) {
                     map.setBrickValue(0, i, j);
                     totalBricks--;
                     score += 5;

                     if (ballposX + 19 <= brickRect.x || ballposX + 1 >= brickRect.x +
brickRect.width) {
                        ballXdir = -ballXdir;
                     } else {
                        ballYdir = -ballYdir;
                     }
                     break A;
                  }
               }
            }
         }
```

```
ballposX += ballXdir;
            ballposY += ballYdir;
            if (ballposX < 0) {
                ballXdir = -ballXdir;
            }
            if (ballposY < 0) {
                ballYdir = -ballYdir;
            }
            if (ballposX > 670) {
                ballXdir = -ballXdir;
            }
        }

        repaint();
    }

    public void keyTyped(KeyEvent e) {}

    public void keyReleased(KeyEvent e) {}

    public void keyPressed(KeyEvent e) {
        if (e.getKeyCode() == KeyEvent.VK_RIGHT) {
            if (playerX >= 600) {
                playerX = 600;
            } else {
                moveRight();
            }
        }

        if (e.getKeyCode() == KeyEvent.VK_LEFT) {
            if (playerX < 10) {
                playerX = 10;
            } else {
                moveLeft();
            }
        }
        if (e.getKeyCode() == KeyEvent.VK_ENTER) {
            if (!play) {
                play = true;
                ballposX = 120;
                ballposY = 350;
                ballXdir = -1;
                ballYdir = -2;
                playerX = 310;
                score = 0;
                totalBricks = 21;
                map = new MapGenerator(3, 7);
```

```
                repaint();
            }
        }
    }

    public void moveRight() {
        play = true;
        playerX += 20;
    }

    public void moveLeft() {
        play = true;
        playerX -= 20;
    }
}
```

```
package brickBracker;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics2D;

public class MapGenerator {
    public int map[][];
    public int brickWidth;
    public int brickHeight;

    public MapGenerator(int row, int col) {
        map = new int[row][col];
        for (int i = 0; i < map.length; i++) {
            for (int j = 0; j < map[0].length; j++) {
                map[i][j] = 1;
            }
        }
        brickWidth = 540 / col;
        brickHeight = 150 / row;
    }

    public void draw(Graphics2D g) {
        for (int i = 0; i < map.length; i++) {
            for (int j = 0; j < map[0].length; j++) {
                if (map[i][j] > 0) {
                    g.setColor(Color.white);
                    g.fillRect(j * brickWidth + 80, i * brickHeight + 50, brickWidth, brickHeight);
```

```
                g.setStroke(new BasicStroke(3));
                g.setColor(Color.black); // Changed "setcolor" to "setColor"
                g.drawRect(j * brickWidth + 80, i * brickHeight + 50, brickWidth, brickHeight);
            }
        }
    }
}

public void setBrickValue(int value, int row, int col) {
    map[row][col] = value;
}
}
```

Results and conclusion

Results:

The Brick Breaker game has been successfully implemented with the following results and observations:

1. **Gameplay Experience:** The game offers an engaging and enjoyable gaming experience. Players can control the paddle's movement, bounce the ball off the paddle, and break bricks to score points.

2**. Graphics and User Interface:** The game features a visually appealing graphical user interface. The paddle, ball, and bricks are rendered smoothly. The game provides clear feedback to the player, including a scoring system and game over messages.

3. **Game Logic:** The game logic, including collision detection and handling, works as expected. The ball reacts appropriately to collisions with the paddle, walls, and bricks. The win and lose conditions are correctly implemented.

4. **User Interaction:** The game responds effectively to user input. Players can control the paddle using the left and right arrow keys. The "Enter" key allows them to restart the game after a win or loss.

5. **Scoring System:** The scoring system rewards players with points for breaking bricks. The score is updated on the screen in real-time, providing positive reinforcement to the player.

Conclusion:

In conclusion, the Brick Breaker game is a well-structured and functional 2D arcade game developed in Java. It provides a classic gaming experience that challenges players' reflexes and hand-eye coordination. The following key points summarize the project:

1. **Game Development:** The project demonstrates the development of a 2D game using Java and the Swing library. It showcases fundamental game development concepts, including graphics rendering, user input handling, and collision detection.

2**. User Engagement:** The game successfully engages players with its intuitive controls and rewarding gameplay. Breaking bricks and aiming for a high score creates a sense of achievement.

3. **Educational Value:** The project can serve as an educational resource for individuals learning game development or programming. It illustrates fundamental concepts of game design and object-oriented programming.

4. **Future Enhancements:** While the current game is fully functional, there is room for further improvements and additions. Future enhancements could include additional levels, power-ups, and enhanced graphics.

5. **Open Source:** The project is open-source and can be further developed or customized by the community. It serves as a foundation for creating various 2D ball-and-paddle games.

In summary, the Brick Breaker game project has been successfully implemented, providing an entertaining gaming experience. It reflects the principles of game development and offers opportunities for expansion and customization. The project's open-source nature encourages collaboration and innovation in the realm of 2D game development.

References

[1] Video: "Java Game Programming - Develop a Brick Breaker Game." https://youtu.be/K9qMm3JbOH0?si=aDB6uEJyS1Z1ghDk. Last Accessed: 10th October 2023.