

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI
590018**



Project Report on
“Brick Smash: A Classic Java Arcade Game”

By

Khushi B Chilakawad (1BF24CS137) Madhav V Nair (1BF24CS159)
Asif (1BM25CS420) Madhireddy Sathvik Reddy (1BF24CS160)

Under the Guidance of
Tejaswini
Professor, Department of CSE
BMS College of Engineering

Work carried out at



Department of Computer Science and Engineering
BMS College of Engineering
(Autonomous college under VTU)

P.O. Box No.: 1908, Bull Temple Road, Bangalore-560 019
2025-2026

BMS COLLEGE OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING



CERTIFICATE

This is to certify that the OOPS with JAVA project titled "**Brick Smash: A Classic Java Arcade Game**" has been carried out by Khushi B Chilakawad (1BF24CS137), Madhav V Nair (1BF24CS159), Asif (1BM25CS420), Madhireddy Sathvik Reddy (1BF24CS160) during the academic year 2025-2026.

Signature of the guide
Tejaswini
Assistant Professor,
Department of Computer Science and Engineering
BMS College of Engineering, Bangalore

BMS COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND

ENGINEERING



DECLARATION

We, Khushi B Chilakawad (1BF24CS137), Madhav V Nair (1BF24CS159), Asif (1BM25CS420), Madhireddy Sathvik Reddy (1BF24CS160), students of 3rd Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this project work entitled "ATM Withdrawal System" has been carried out by us under the guidance of Monisha H M , Professor, Department of CSE, BMS College of Engineering, Bangalore during the academic semester Sep-Dec 2025. We also declare that to the best of our knowledge and belief, the project reported here is not from part of any other report by any other students.

Signature of the Candidates

Khushi B Chilakawad (1BF24CS137)

Madhav V Nair (1BF24CS159)

Asif (1BM25CS420)

Madhireddy Sathvik Reddy (1BF24CS160)

INDEX

Sl.no	Title	Page number
1	Problem statement	
2	Introduction	
3	Overview of the project (Block diagram)	
4	Tools Used	
5	OOPs concept used & its explanation	
6	Implementation/code	
7	Result /snapshots	
8	References	

Problem statement

To design and implement a Java-based Brick Breaker game that uses keyboard controls, real-time graphics, and collision detection to break bricks, track score, and determine game over conditions using object-oriented programming principles.

Explanation:

The aim of this project is to create a Brick Breaker game using Java where the player interacts with the game through keyboard controls to move a paddle left and right. The paddle is used to bounce a moving ball toward the bricks displayed on the screen.

The game uses real-time graphics, which means the ball, paddle, and bricks are continuously drawn and updated on the screen to provide smooth gameplay. Collision detection is implemented to check when the ball hits the walls, paddle, or bricks. When a collision occurs, the ball changes direction and the brick is removed.

The system also tracks the player's score, which increases every time a brick is broken. The game continuously checks for game over conditions, such as when the ball falls below the paddle or when all bricks are destroyed.

Throughout the project, object-oriented programming (OOP) principles like classes, objects, inheritance, encapsulation, and polymorphism are used to organize the code, improve readability, and make the game easier to manage and extend.

Introduction

Computer games have become an important medium for learning and entertainment, especially in the field of computer science and software development. Developing simple games helps students understand core programming concepts in an interactive and practical way. One such popular arcade game is Brick Breaker, which involves real-time interaction, graphics handling, and logical decision-making.

The Brick Breaker game is a classic 2D game where the player controls a paddle to bounce a ball and break a set of bricks arranged at the top of the screen. The objective of the game is to destroy all the bricks without letting the ball fall below the paddle. The game requires continuous monitoring of ball movement, collision detection with different objects, and timely response to user inputs.

This project focuses on the design and implementation of a Brick Breaker game using Java, making use of Object-Oriented Programming (OOP) principles. Java is a platform-independent, robust programming language that provides strong support for graphical user interfaces through AWT and Swing libraries. These libraries are used to create windows, draw shapes, display colors, and handle keyboard events necessary for interactive gameplay.

The game is controlled using keyboard inputs, allowing the player to move the paddle left and right. The ball moves automatically across the screen, and collision detection logic is implemented to handle interactions between the ball, walls, paddle, and bricks. When the ball hits a brick, the brick is removed and the player's score is updated accordingly. The game also checks for win and game-over conditions, providing appropriate messages to the player.

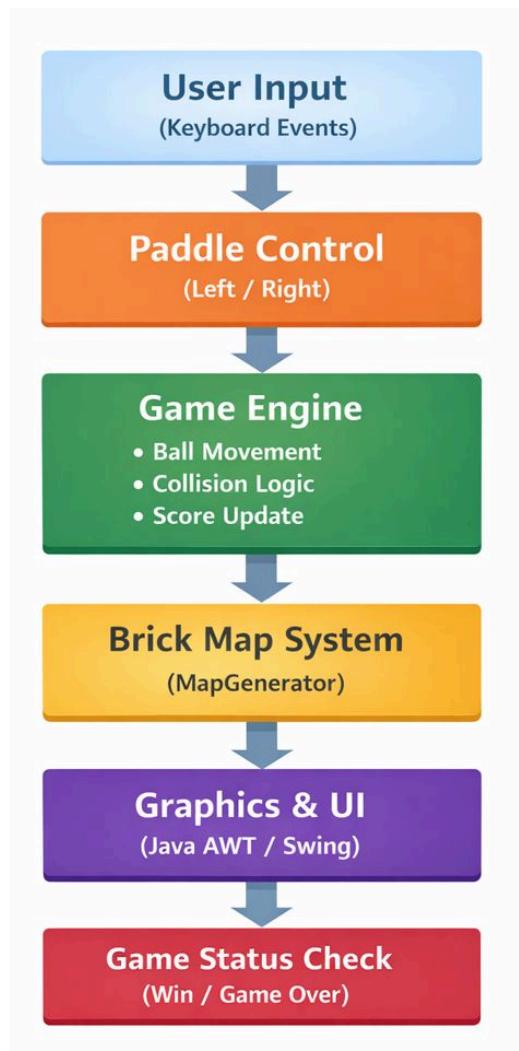
Through this project, important programming concepts such as classes and objects, inheritance, encapsulation, polymorphism, event handling, and basic game physics are applied in a practical manner. The project not only enhances logical thinking and problem-solving skills but also provides a foundation for understanding basic game development techniques using Java.

Overview of the project (Block diagram)

The Brick Breaker game is a Java-based interactive application developed using object-oriented programming principles. The game allows the user to control a paddle using keyboard inputs to bounce a ball and break bricks displayed on the screen. The application continuously updates game elements in real time, ensuring smooth gameplay and immediate response to user actions.

The system is divided into multiple functional modules such as user input handling, game logic processing, brick generation, graphical rendering, and game status evaluation. These modules work together to manage ball movement, collision detection, score calculation, and win or game-over conditions.

Block Diagram (Conceptual Representation)



Block Diagram Explanation

The block diagram represents the overall working flow of the Brick Breaker game system. Each block performs a specific function and passes control to the next block in sequence.

1. User Input (Keyboard Events)

This block captures keyboard inputs from the user. The player uses arrow keys to move the paddle left or right during gameplay.

2. Paddle Control

Based on the keyboard input, this module updates the paddle's position on the screen while ensuring it stays within the game boundaries.

3. Game Engine

The game engine is the core component of the system. It controls ball movement, detects collisions between the ball, paddle, walls, and bricks, and updates the game state accordingly.

4. Brick Map System (MapGenerator)

This module is responsible for creating and managing the arrangement of bricks. When the ball collides with a brick, the corresponding brick is removed from the map.

5. Graphics and User Interface (AWT & Swing)

This block handles rendering of all game elements such as the paddle, ball, bricks, and score on the screen using Java AWT and Swing libraries.

6. Game Status Check

This final block continuously checks for win or game-over conditions. The game ends when all bricks are destroyed or when the ball falls below the paddle.

Tools Used

1. Java Programming Language

Java is the core programming language used to develop the Brick Breaker game. It follows object-oriented programming principles, which help in organizing the code into classes and objects. Java provides reliability, platform independence, and strong support for game logic and graphics handling.

2. Visual Studio Code (VS Code)

Visual Studio Code is used as the code editor to write, edit, and run the Java program. It provides features such as syntax highlighting, error detection, extensions for Java support, and an integrated terminal, which makes development easier and more efficient.

3. Java Development Kit (JDK)

The Java Development Kit is required to compile and execute Java programs. It includes essential tools such as the Java compiler (`javac`) and Java Virtual Machine (JVM), which allow the Brick Breaker game to run on the system.

4. Java AWT (Abstract Window Toolkit)

Java AWT is used to create the graphical components of the game. It provides classes for drawing shapes, handling colors, fonts, and managing basic user interface elements required for the game screen.

5. Java Swing

Java Swing is used to build the graphical user interface of the game. It provides components like `JFrame` and `JPanel` that help in creating the game window and rendering game objects smoothly.

6. Keyboard Event Handling

Keyboard event handling is used to capture user inputs such as left and right arrow key presses. These inputs allow the player to control the paddle movement during the game.

7. Graphics Class

The Graphics class is used to draw game elements like the ball, paddle, bricks, and score on the screen. It enables real-time rendering and updates of the game visuals.

OOPs concept used & its explanation

The major OOP concepts used in this project are explained below:

1. Class and Object

A class is a blueprint used to define properties and behaviors, while an object is an instance of a class. In this project, classes such as `Main`, `Gameplay`, and `MapGenerator` are used to represent different parts of the Brick Breaker game. Objects created from these classes manage game elements like the paddle, ball, and bricks.

2. Encapsulation

Encapsulation refers to binding data and methods together within a class and restricting direct access to data. In the Brick Breaker game, variables related to score, ball position, and game state are kept private and accessed through class methods, which ensures data security and controlled modification.

3. Inheritance

Inheritance allows one class to acquire the properties and methods of another class. The `Gameplay` class extends `JPanel`, which allows it to inherit graphical and event-handling features required to display the game interface and respond to user actions.

4. Polymorphism

Polymorphism allows methods to behave differently based on the object that calls them. In this project, method overriding is used for the `paint()` method to customize how game components such as bricks, ball, and paddle are drawn on the screen.

5. Abstraction

Abstraction focuses on hiding complex implementation details and showing only essential features. Complex operations like collision detection, ball movement, and brick generation are implemented through methods, so the user interacts only with the game functionality and not the internal logic.

6. Event Handling

Event handling allows the program to respond to user actions such as key presses. The project implements interfaces like `KeyListener` and `ActionListener` to handle keyboard input and timer-based events, making the game interactive and responsive.

Implementation/code

The Brick Breaker game is implemented using Java by applying object-oriented programming principles and graphical libraries such as AWT and Swing. The entire application is divided into multiple classes, each responsible for a specific task, which makes the system modular and easy to understand.

The program execution begins from the **Main** class, which creates the main game window using **JFrame**. This class initializes the game environment and adds the gameplay panel to the frame.

The **Gameplay** class contains the core logic of the game. It extends **JPanel** and implements **KeyListener** and **ActionListener** interfaces to handle keyboard input and timer-based events. This class manages ball movement, paddle movement, collision detection, score calculation, and game state updates. A timer is used to repeatedly update the game screen, enabling real-time animation.

The **MapGenerator** class is used to create and manage the brick layout. It generates a matrix of bricks and draws them on the screen. When the ball collides with a brick, the corresponding brick is removed, and the score is updated.

Keyboard events are captured to move the paddle left or right, ensuring it stays within the game boundaries. Collision detection logic checks interactions between the ball, paddle, walls, and bricks to change the ball's direction accordingly.

The **Graphics** class is used to draw all game components such as the paddle, ball, bricks, borders, and score. The **paint()** method is overridden to ensure smooth and continuous rendering of the game elements.

The game continuously checks for win and game-over conditions. The game ends when all bricks are destroyed or when the ball falls below the paddle, and appropriate messages are displayed to the player.

CODE

Main Class

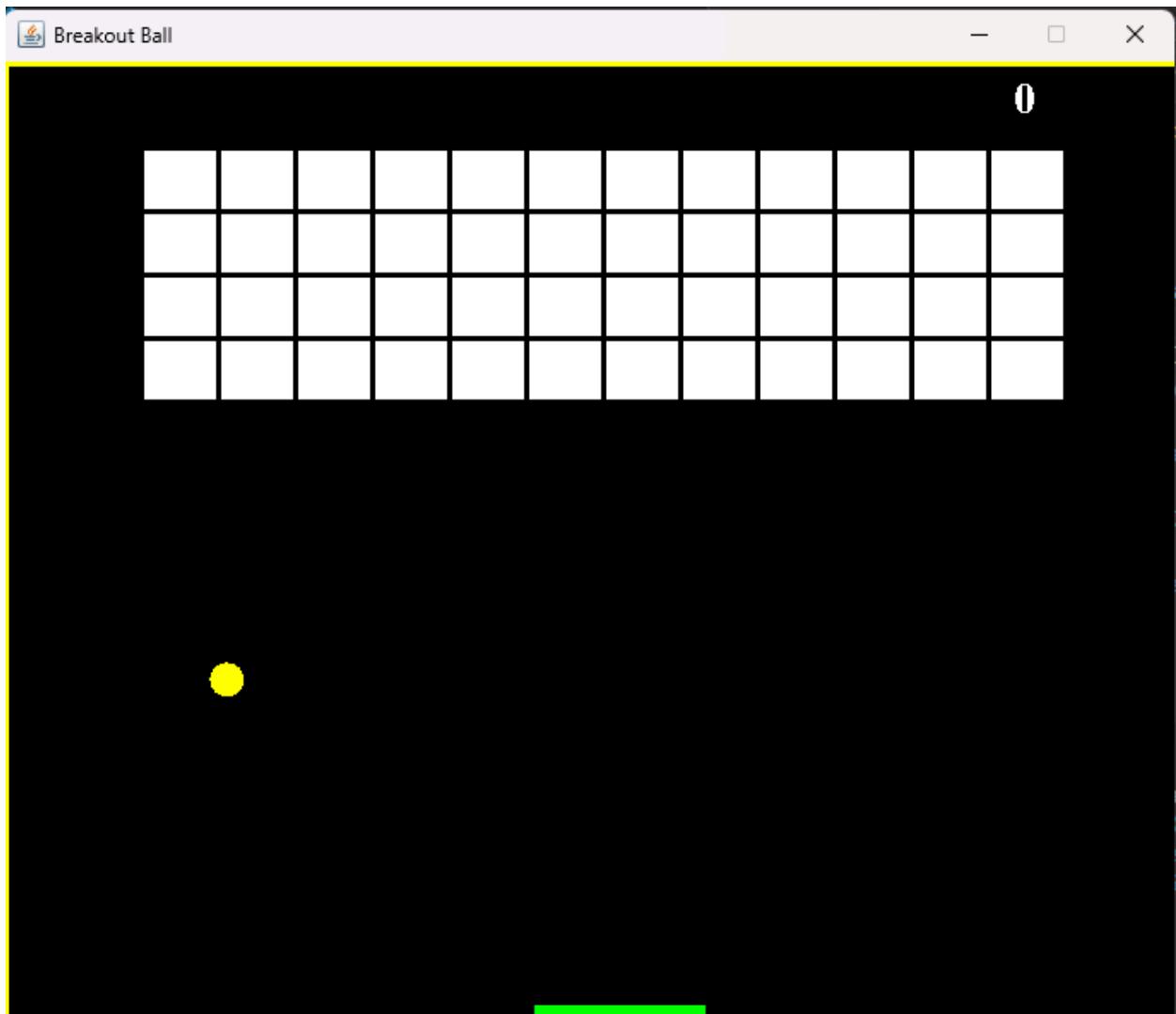
```
import java.awt.Color;
import javax.swing.JFrame;
public class Main {
    public static void main(String[] args) {
        JFrame obj=new JFrame();
        Gameplay gamePlay = new Gameplay();
        obj.setBounds(10, 10, 700, 600);
        obj.setTitle("Breakout Ball");
        obj.setResizable(false);
        obj.setVisible(true);
        obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        obj.add(gamePlay);
        obj.setVisible(true);
    }
}
```

MapGenerator

```
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 q)
    {
        for(int i = 0; i<map.length; i++)
        {
            for(int j =0; j<map[0].length; j++)
            {
                if(map[i][j] > 0)
                {
                    q.setColor(Color.white);
                    q.fillRect(j * brickWidth + 80, i * brickHeight + 50,
brickWidth, brickHeight);
                    // this is just to show separate brick, game can still
run without it
                    q.setStroke(new BasicStroke(3));
                    q.setColor(Color.black);
                    q.drawRect(j * brickWidth + 80, i * brickHeight + 50,
brickWidth, brickHeight);
                }
            }
        }
    }
    public void setBrickValue(int value, int row, int col)
    {
        map[row][col] = value;
    }
}
```

Results



References

1. https://chatgpt.com/backend-api/estuary/content?id=d5830d80e561513%23file_0000000375072099176552f371378ab%23md&ts=490808&p=fs&cid=1&sig=d7a685b398fa3cd74865244889a54089122a2ace551fc195309e99327d9c799&v=0
2. <https://share.google/iSVJJhzQdZXygfHZ>
3. Class Notes / PPTs