Flappy Bird using GUI in java

A PROJECT REPORT

Submitted by
Vaibhav Kumar
(23BCS13386)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

Chandigarh University

April, 2025

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

1.1 Introduction to Project

Flappy Bird is a 2D side-scrolling arcade-style game in which the player takes control of a bird that must navigate through an endless series of pipes without colliding with them. The challenge lies in maintaining the bird's altitude by tapping or pressing a key, which causes the bird to flap and ascend momentarily before gravity pulls it downward again. The simplicity of the gameplay, paired with the increasing difficulty, makes it both addictive and instructive for learning game development.

This project aims to replicate the classic Flappy Bird experience using the Java programming language and its Swing GUI framework. Through this implementation, we cover key areas of computer science such as event-driven programming, real-time graphics rendering, basic physics simulation, and object-oriented programming (OOP) principles. The game features animated motion, user input handling, score tracking, collision detection, and game state management.

By developing this project from scratch using only core Java libraries, students gain hands-on experience in designing and building a complete desktop application. It serves as an excellent introduction to game development, highlighting the logical structure behind game loops, component rendering, and user interaction in GUI-based software applications.

1.2 Identification of Problem

While there are countless tutorials available for building basic applications in Java, very few provide insight into developing full-fledged games. Most introductory courses emphasize console applications or simple form-based GUIs. This creates a gap in understanding how Java can be leveraged to build interactive, visually dynamic applications such as games.

Furthermore, students often find it difficult to transition from static programs to real-time systems that require constant updates and user interactions.
Developing Flappy Bird from scratch helps address this gap by introducing learners to the fundamental concepts of real-time event handling, rendering, and continuous game loop updates using core Java tools.

CHAPTER 2. BACKGROUND STUDY

2.1 Existing Solutions

Flappy Bird has been recreated on multiple platforms including Unity, JavaScript, Python (Pygame), and C++. Each of these implementations varies in approach depending on the platform and language. Unity, for instance, offers built-in physics and animation systems, making it easier to create game-like experiences with minimal code. JavaScript-based versions run on browsers and are typically lightweight.

Despite the popularity of these implementations, Java-based versions remain underrepresented, particularly ones that use only Swing without external libraries. Swing offers a powerful but often underutilized set of tools for creating graphical applications in Java. Through this project, we explore how Swing can be used effectively to deliver a rich user experience in game development, which can be valuable for both academic and practical learning.

2.2 Problem Definition

The core problem is to design and implement a real-time 2D side-scrolling game using Java's standard library without relying on external game development engines. The game must simulate physics-based bird movement, randomly generate pipe obstacles, detect collisions accurately, and maintain an ongoing scoring system.

In doing so, the solution should focus on performance efficiency, responsiveness to user input, and visual appeal. It should also support restartability after a game over, allowing the player to try again without restarting the program. These goals will collectively ensure that the project achieves both educational and entertainment value.

2.3 Goals/Objectives

- Understand and implement Java Swing components for graphics and user input.
- Apply object-oriented programming principles to structure game logic efficiently.
- Develop custom game loop and physics logic (gravity, jump mechanics).

- Design and manage multiple game objects like pipes, birds, and background.
- Implement collision detection between the bird and pipes.
- Create an intuitive and interactive user experience.
- Support game restart and scoring system.
- Optimize the code for better frame rate and responsiveness.

CHAPTER 3. DESIGN FLOW/PROCESS

3.1 Evaluation & Selection of Features

To ensure the gameplay is close to the original Flappy Bird experience, we evaluated key elements:

- Bird flight controlled by spacebar (jump mechanism)
- Gravity for natural falling motion
- Random generation of top and bottom pipes
- Constant leftward movement of pipes
- Collision detection and scoring logic

Each feature was assessed for its feasibility in Java Swing. Once validated, it was implemented step-by-step, ensuring that it integrates smoothly into the main game loop and GUI components.

3.2 Analysis and Finalization of Features

The finalized list of features includes:

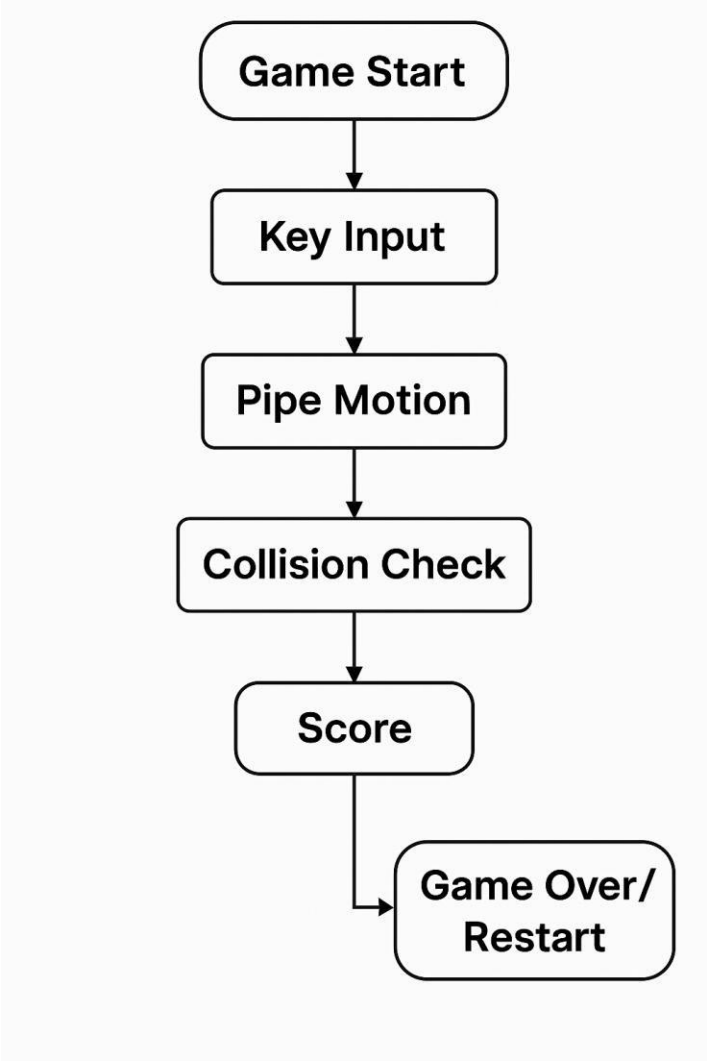
- **Game Initialization**: Using JFrame and JPanel for rendering
- **Asset Loading**: Images for background, pipes, and bird using ImageIcon
- **Animation Loop**: Timer class used to create smooth game loop at 60 FPS

- **Event Handling**: KeyListener for jump control
- **Collision Detection**: Coordinate-based logic with bounding rectangles
- **Game State Management**: Game over state, restart logic, and scoring

All features were tested iteratively to ensure stability, responsiveness, and an engaging user experience.

3.3 Design Flow

- Initialize the GUI using JFrame and set up the drawing surface using JPanel.
- Load the required image assets (bird, pipes, and background).
- Add a KeyListener to handle the jump action when the spacebar is pressed.
- Start a Timer to continuously update game state and repaint graphics.
- Dynamically generate pipes at regular intervals using another Timer.
- Continuously move pipes leftward to simulate bird movement.
- Apply gravity and velocity logic to control bird position.
- Check for collisions between the bird and pipes or the ground.
- Increment score when bird passes through pipe pairs.
- Display game over screen and reset game state upon spacebar input.



CHAPTER 4. RESULTS ANALYSIS AND VALIDATION

4.1 Implementation of Solution

The project uses two core classes: App.java and FlappyBird.java

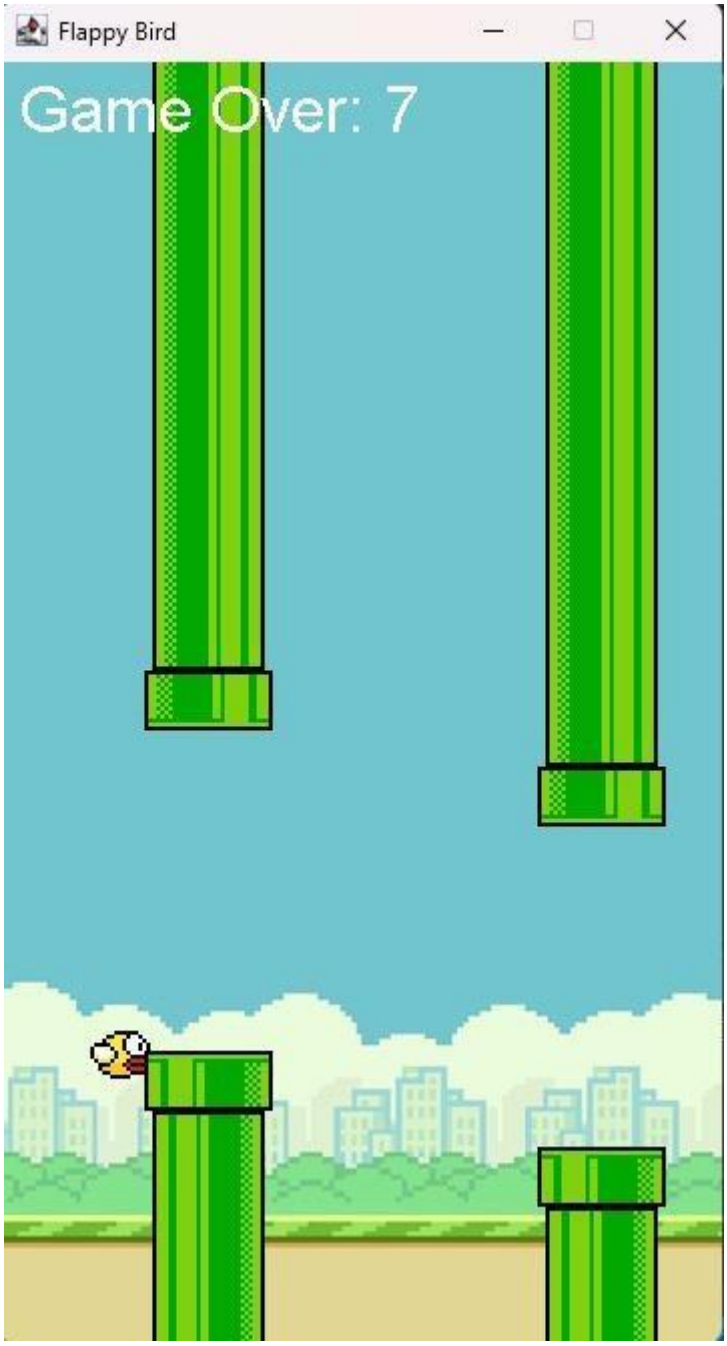
App.java (Main Launcher)

```java
public class App {
    public static void main(String[] args) throws Ex
        JFrame frame = new JFrame("Flappy Bird");
        frame.setSize(360, 640);
        frame.setLocationRelativeTo(null);
        frame.setResizable(false);
        frame.setDefaultCloseOperation(JFrame.EXIT_C

        FlappyBird flappyBird = new FlappyBird();
        frame.add(flappyBird);
        frame.pack();
        flappyBird.requestFocus();
        frame.setVisible(true);
    }
}
```

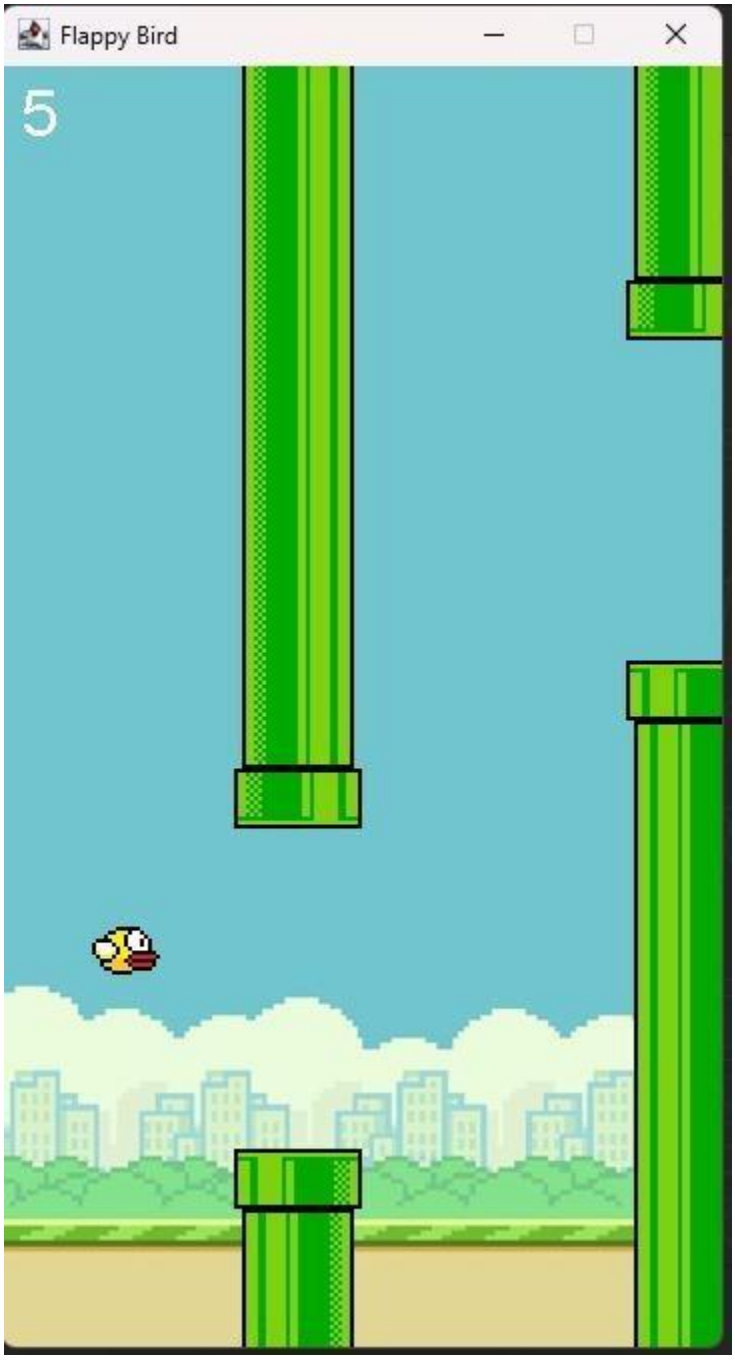
FlappyBird.java (Game Logic)

Includes implementation of bird mechanics, pipe generation, game loop, rendering and input:

```java
public class FlappyBird extends JPanel implements ActionListene
    int velocityY = 0;
    int gravity = 1;
    boolean gameOver = false;

    public void move() {
        velocityY += gravity;
        bird.y += velocityY;
        // ... update pipe positions and check collisions
    }

    public void keyPressed(KeyEvent e) {
        if (e.getKeyCode() == KeyEvent.VK_SPACE) {
            velocityY = -9;
            // ... reset if game over
        }
    }

    public void actionPerformed(ActionEvent e) {
        move();
        repaint();
        if (gameOver) {
            gameLoop.stop();
        }
    }
}
```



Output for the Flappy Bird Game



Output

CHAPTER 5. CONCLUSION AND FUTURE WORK

5.1 Conclusion

The Flappy Bird project demonstrates how a classic arcade game can be developed using core Java tools and Swing. Through this project, we gained in-depth understanding of game loops, graphical rendering, object-oriented architecture, and user input handling. It provided a hands-on opportunity to apply theoretical concepts learned in class to a practical and entertaining software application.

The project also highlighted the flexibility and power of Java's built-in libraries, proving that even a simple GUI toolkit like Swing is capable of producing responsive and visually engaging games. Overall, it has been a valuable learning experience that combined creativity with technical problem-solving.

5.2 Future Work

- Although the current implementation is functional and meets its primary objectives, several improvements can enhance the game:
- **Sound Integration**: Add background music and sound effects for jumping and collisions.
- **Visual Enhancements**: Incorporate sprite animations and smoother transitions.
- **Difficulty Scaling**: Gradually increase pipe speed or reduce gaps over time.
- **High Score System**: Save and display highest scores using file I/O.
- **Mobile Compatibility**: Adapt the game for Android using Java frameworks like LibGDX.
- **Power-Ups**: Introduce items like slow-motion or shields for gameplay variety.
- Implementing these features will make the game more dynamic and appealing while offering further learning opportunities.