**A Mini Project** report on

# FLAPPY BIRD

Submitted to the **CMR TECHNICAL CAMPUS** in partial fulfillment of the requirement of the

# OBJECT ORIENTED PROGRAMMING THROUGH JAVA

of

# II-B.Tech. I-Semester

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

Submitted by

SAAHINI PASHAM - 237R1A05B4

BOJJA MAHATHI REDDY - 237R1A0576

KURUBA VIBHAS SAI - 237R1A0595

THIRUKOVALLURU CHAKRADHAR - 237R1A05C5

# Under the guidance of

# Ms. G. KALPANA DEVI

# Professor, Department of COMPUTER SCIENCE AND ENGINEERING

# CMR TECHNICAL CAMPUS

# (UGC-AUTONOMOUS)

(Approved by AICTE, Affiliated to JNTUH,

Hyderabad) Kandlakoya, Medchal Road, Hyderabad.

**2024-2025**

# CMR TECHNICAL CAMPUS

# (UGC-AUTONOMOUS)

(Approved by AICTE, Affiliated to JNTUH,

Hyderabad) Kandlakoya, Medchal Road, Hyderabad.

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CERTIFICATE**

# This is to certify that a Mini Project entitled with

# FLAPPY BIRD

# is the Bonafide work carried out and submitted by

SAAHINI PASHAM - 237R1A05B4

BOJJA MAHATHI REDDY - 237R1A0576

KURUBA VIBHAS SAI - 237R1A0595

THIRUKOVALLURU CHAKRADHAR - 237R1A05C5

In partial fulfillment of the requirement for completion of the **OBJECT ORIENTED PROGRAMMING THROUGH JAVA** of II – B.Tech. I – Semester is a record of Bonafide work carried out under our guidance during the academic year 2024-2025.

**Project Guide HEAD OF THE DEPARTMENT**

Ms. G. KALPANA DEVI DR. K. SRUJAN RAJU,

Professor, Head of the Department,

Dept. of **CSE,** Dept. of Humanities and sciences,

CMRTC CMRTC

# 

# ACKNOWLEDGEMENT

We would like to express our sincere gratitude to all who have been instrumental and helpful in the preparation and the presentation of this project.

I am highly indebted to Ms. **G. KALPANA DEVI** for their guidance and the constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

We would like to thank our internal project mates and department Professors for their full- fledged guidance and giving me courage to carry out the project.

Last but far from least, I also thank my parents, family members and my friends for their moral support and constant encouragement. I am very much thankful to one and all who helped me for the successful completion of my project.

# ABSTRACT

This project, **Text-To-Speech GUI**, focuses on developing a Java-based desktop application that converts user-input text into speech using the **FreeTTS** library. Built with **JavaFX**, the application offers a simple and interactive interface where users can enter text, choose different voices, adjust the speech rate, and control the volume level. Key components include a **Graphical User Interface (GUI)** for user interaction, a **controller** to manage the text-to-speech conversion logic, and a **CSS-based design** to enhance aesthetics.

The application emphasizes modularity, allowing easy maintenance and future upgrades. It features robust **error handling** mechanisms to guide users in case of invalid inputs or missing configurations. With practical applications in accessibility tools, education, and content creation, this project showcases the effective use of open-source libraries to provide a seamless text-to-speech experience. The integration of JavaFX with FreeTTS highlights the potential for building interactive and helpful desktop applications in the field of voice technology.

# TABLE OF CONTENTS

# TEXT TO SPEECH GUI USING JAVAFX

|  |  |  |
| --- | --- | --- |
| **S.NO.** | **PARTICULARS** | **PAGE NO.** |
| 1 | INTRODUCTION | 1 |
| 2 | LITERATURE SURVEY | 2 |
| 3 | ANALYSIS & DESIGN | 3-6 |
| 4 | IMPLEMENTATION | 7 |
| 5 | TESTING & DEBUGGING | 8-13 |
| 6 | EXPERIMENTAL INVESTIGATIONS | 14-16 |
| 7 | WORKING OF APP | 17-18 |
| 8 | LIMITATIONS & FUTURE ENHANCEMENTS | 19-21 |
| 9 | CONCLUSION | 22 |
| 10 | REFERENCE | 22 |

# 

# 1.INTRODUCTION

# Flappy Bird is a popular arcade-style game that challenges players to navigate a bird through a series of obstacles by tapping to keep the bird airborne.

# The objective is to score points by passing through pipes without colliding with them. This simple yet addictive gameplay has captivated millions of players worldwide.

# In this project, we will implement Flappy Bird using Java, leveraging its graphics capabilities to create a visually appealing and interactive game.

# The implementation involves the following key components:

# Game Window: A graphical window where the game will be displayed.

# Game Loop: A loop that continuously updates the game state and redraws the graphics, ensuring smooth gameplay.

# Bird Mechanics: Logic to control the bird's movement, including gravity and jump mechanics.

# Obstacles: Creation of pipes that the player must navigate through, including collision detection to end the game if the bird hits a pipe or the ground.

# Scoring System: Keeping track of the player's score based on successfully navigating the pipes.

# 2.LITERATURE SURVEY

The development of games like Flappy Bird in Java has garnered significant interest in the programming community, leading to various resources and studies that explore the underlying principles of game design, mechanics, and implementation. This literature survey reviews key concepts and references relevant to creating a Flappy Bird clone in Java, focusing on aspects such as graphics rendering, game physics, and user interaction.

1. **Game Development Frameworks:**
   * Java provides several libraries and frameworks that facilitate game development, such as **JavaFX** and **LibGDX**. These libraries offer tools for graphics rendering, sound management, and user input, making it easier to create engaging game experiences.
2. **Graphics Rendering:**
   * Efficient graphics rendering is crucial for game performance. Techniques such as **double buffering** and **sprite rendering** are commonly discussed in literature, as they help achieve smooth animations and reduce flickering on the screen. Resources often recommend implementing custom rendering loops to optimize performance.
3. **Game Mechanics and Physics:**
   * The mechanics of Flappy Bird, particularly the gravity and jump physics, are fundamental to gameplay. Studies emphasize the importance of implementing realistic physics to enhance user experience. For instance, the application of **Newtonian physics** principles can be utilized to simulate gravity and acceleration, making the bird's movement feel intuitive.
4. **Collision Detection:**
   * Collision detection is a vital aspect of game development, especially in obstacle avoidance games like Flappy Bird. Various algorithms, such as **AABB (Axis-Aligned Bounding Box)** and **pixel-perfect collision detection**, are explored in literature. These methods are essential for determining when the bird collides with pipes or the ground, influencing game progression.
5. **User Input Handling:**
   * Effective handling of user input is crucial for interactive gameplay. Java's event handling model allows developers to capture keyboard and mouse events seamlessly. Resources often cover best practices for managing player actions, such as implementing a responsive control system for the bird’s flight.

3. ANALYSIS AND DESIGN

**ANALYSIS**

**1. Game Structure Overview**

The game revolves around a bird navigating through obstacles (pipes) while avoiding collisions. The player controls the bird’s movement by making it "flap" (jump upwards), and gravity causes the bird to fall otherwise. The key features to implement are:

* Bird movement (upwards when the player presses a key and downwards due to gravity)
* Pipes generating and moving towards the bird
* Collision detection between the bird and the pipes or the ground
* Scoring based on how many pipes the bird passes

**2. Key Classes**

To follow good object-oriented design practices, we can break the game into different classes that each handle specific responsibilities.

**a. Game Panel (Main Game Loop Class)**

* This class is responsible for the main game loop, which continuously updates the game and renders the graphics.
* **Key methods:**
  + **start():** Starts the game loop (could be implemented using a Timer or Thread).
  + **update():** Updates the game state (position of the bird, pipes, collisions).
  + **draw():** Draws the updated game state onto the screen (bird, pipes, background).

**b. Bird Class**

* Represents the bird and its behavior.
* **Attributes:**
  + Position (x, y)
  + Velocity (for both gravity and flap effect)
  + Size (height, width)
* **Key methods:**
  + flap(): Moves the bird upwards.
  + fall(): Causes the bird to fall due to gravity.
  + render(): Draws the bird on the screen.

**c. Pipe Class**

* Represents the obstacles (pipes) the bird must avoid.
* **Attributes:**
  + Position (x, y)
  + Size (width, height)
  + Speed (how fast the pipes move from right to left)
* **Key methods:**
  + **move():** Moves the pipes leftward.
  + **render():** Draws the pipes on the screen.

**d. Collision Detector Class**

* Handles all the collision logic.
* **Key methods:**
  + checkCollision(Bird bird, Pipe pipe): Checks if the bird collides with a pipe or the ground.

**e. Score Class**

* Tracks and updates the player’s score.
* **Attributes:**
  + Current score
  + High score
* **Key methods:**
  + **update():** Increases the score when the bird passes through pipes.

**3. Game Flow**

* **Initialization:** When the game starts, the bird is at a fixed position, and pipes are generated off-screen, moving leftward.
* **Player Input:** The player presses a key (usually spacebar or mouse click) to make the bird flap.
* **Game Loop:** The game loop continuously updates the bird’s position (with gravity and flap logic), moves pipes, checks for collisions, and renders the updated game state.
* **Collision Handling:** If a collision is detected, the game ends, and the final score is displayed.

4. **Challenges and Design Considerations**

* **Performance**: As more pipes are added, managing memory usage becomes essential. You can optimize performance by reusing off-screen pipes instead of generating new ones.
* **Difficulty Balancing**: The game’s difficulty can be adjusted by tweaking the gravity, flap power, pipe speed, and gap size.
* **Responsiveness**: The game needs to maintain a smooth and consistent frame rate. Using a fixed timestep in the game loop (like Timer in Swing) ensures consistency across different systems.

**DESIGN**

Designing ***Flappy Bird*** in Java follows object-oriented principles and focuses on breaking down the game into reusable, modular components. The design should account for game mechanics (such as gravity and collisions), user interaction, and rendering.

**1. Game Structure**

The game can be broken down into the following major components:

* **Bird:** Represents the player’s character, controlled by the user.
* **Pipe:** Represents the obstacles the bird must navigate through.
* **Game Engine:** Handles the game loop, physics (gravity, movement), and rendering.
* **Collision Detector:** Detects whether the bird hits a pipe or the ground.
* **Score System:** Tracks the player’s score and manages game-over conditions.

**2. Object-Oriented Design**

The game should be designed using object-oriented programming (OOP) principles such as:

* **Encapsulation:** Each game component (Bird, Pipe, etc.) should be responsible for managing its own state and behavior.
* **Inheritance:** If there are shared behaviors between objects, inheritance could be applied (e.g., obstacles could be a subclass of a base class).
* **Polymorphism:** Different game objects could share a common interface for rendering or updating game logic.
* **Modularity:** Classes should be modular so that each one handles a single responsibility

**3. Class Design**

Below is the class design based on the game components.

**a. Bird Class**

This class is responsible for controlling the bird's position, velocity, and rendering it on the screen. The bird should "flap" upwards when the player presses a key and fall due to gravity.

**b. Pipe Class**

The Pipe class handles the generation, movement, and rendering of pipes. Pipes will move from right to left, and once they move off the screen, they will be removed.

**c. CollisionDetector Class**

The CollisionDetector class checks whether the bird has collided with a pipe or hit the ground.

**d. Score Class**

The Score class keeps track of the player’s current score and displays it on the screen. It increases the score each time the bird passes through a pipe.

**e. GamePanel Class (Game Engine)**

The GamePanel class manages the game loop, rendering, and user input. It continuously updates the game state (bird movement, pipe generation, collision detection) and handles drawing all the components to the screen.

**4. Game Loop and Timer**

The Timer in the GamePanel class drives the game loop. It triggers the actionPerformed method, which updates the bird’s position, pipes, and score at regular intervals (around 50 frames per second). The repaint() method ensures that the game state is continuously redrawn.

**5. Input Handling**

User input (the spacebar to flap the bird) is handled by the KeyListener interface. When the player presses the spacebar, the bird’s velocity changes, making it flap upwards.

**6. Rendering**

The game rendering is handled in the paintComponent() method of GamePanel. It calls the render() methods of the Bird, Pipe, and Score classes to draw each object on the screen.

**7. Design Considerations**

* **Performance**: Consider recycling pipes instead of continuously creating new ones to improve memory efficiency.

4. IMPLEMENTATION

To implement *Flappy Bird* in Java, we will use object-oriented principles and graphical libraries (like Java's Swing and AWT) to handle rendering and user input. Below is a step-by-step guide for the core implementation of the game, including classes for the bird, pipes, collision detection, score, and game loop.

**1. Setup the Project**

1. **Create a New Java Project:** In your IDE (like IntelliJ IDEA or Eclipse), create a new Java project.
2. **Add a GamePanel class:** This will serve as the main game loop and rendering class.
3. **Add classes for the game objects:** Create classes for the Bird, Pipe, Score, and CollisionDetector.

**2. Core Java Classes Implementation**

Here is the complete implementation for a simple version of *Flappy Bird* in Java.

**a. Bird Class**

The Bird class manages the bird's position, movement, and rendering.

**b. Pipe Class**

The Pipe class handles the generation, movement, and rendering of pipes (obstacles).

**c. CollisionDetector Class**

The CollisionDetector class checks whether the bird has collided with a pipe or the ground.

**d. Score Class**

The Score class keeps track of the score and renders it on the screen.

**e. GamePanel Class**

The GamePanel class manages the main game loop, updates the game state, and renders the game.

**3. Running the Game**

**Once all classes are set up:**

* Compile and run the FlappyBirdGame class.
* The game window should open, and you can start playing *Flappy Bird* by pressing the spacebar to make the bird flap and avoid pipes.

1. **Enhancements and Additional Features**
2. **Background and Graphics:** Add a background image or more detailed graphics for the bird and pipes to enhance the visual appeal.
3. **Sound Effects:** Implement sound effects for flapping, collisions, and passing through pipes

5. TESTING & DEBUGGING

**Testing and Debugging** are crucial steps in the development of the *Flappy Bird* game in Java. Testing ensures the game functions as expected, while debugging helps to find and fix any issues in the code. Let's walk through how to perform both testing and debugging for this game.

**TESTING**

Testing the **Flappy Bird** game in Java can be approached through both manual and automated methods. Since the game heavily relies on graphical rendering and user interaction, much of the testing will be manual, but key logic such as collision detection, bird movement, and scoring can be unit tested.

**1. Manual Testing**

Manual testing is crucial for validating the following aspects of the game:

* **Gameplay**: Does the bird respond correctly to key presses? Does the game run smoothly without frame rate drops?
* **Collision Detection**: Does the bird collide with pipes and the ground correctly?
* **Score Calculation**: Is the score incremented properly when the bird passes through pipes?
* **Difficulty**: Does the difficulty feel appropriate? Does it increase over time (if designed that way)?

Steps for manual testing:

1. **Start the game** and verify that the window opens correctly and that the bird and pipes are rendered.
2. **Press the spacebar** and verify that the bird flaps upwards, then falls due to gravity.
3. **Observe pipe movement** to ensure pipes move leftward and new pipes appear after the old ones exit the screen.
4. **Test collisions** by flying into pipes or letting the bird hit the ground. Ensure that the game stops and displays a game over screen or stops the game loop.
5. **Test the scoring system** by passing through pipes and observing if the score increments correctly.

**2. Automated Testing**

For automated testing, we will focus on unit testing the game’s logic, especially for methods that handle the game’s core mechanics such as bird movement, collision detection, and scoring.

You can use JUnit to automate testing in Java. Below are test cases for specific aspects of the game.

**a. Testing Bird Movement**

We can test the bird's gravity and flapping mechanics by simulating the update() and flap() methods.

**b. Testing Pipe Movement**

We can test the movement of the pipes by checking the update() method and whether pipes are correctly flagged as "off-screen."

**c. Testing Collision Detection**

The collision detection logic is critical for determining when the bird hits pipes or the ground. We can unit test the CollisionDetector class to verify the correctness of this logic.

**d. Testing Scoring**

The scoring system can be tested by ensuring that the score increments correctly when the bird passes through pipes.

**3. Test Coverage**

The tests provided cover:

* **Bird mechanics**: Gravity, flapping, and position updates.
* **Pipe mechanics**: Movement and removal of off-screen pipes.
* **Collision detection**: Bird hitting pipes and ground.
* **Score**: Increments when the bird passes through pipes.

**4. Running Tests**

To run these tests, follow these steps:

1. **Add JUnit 5 to your project**: Most IDEs like IntelliJ and Eclipse have built-in support for JUnit testing. You can also add JUnit as a Maven or Gradle dependency if using a build tool.
2. **Write the test cases** in separate test classes, corresponding to the game components.
3. **Run the tests**: Use your IDE’s test runner or command-line tools (Maven/Gradle) to run the tests.

### **5. UI Testing (Optional)**

Testing the graphical interface and user interactions (like key presses and rendering) manually is typically necessary for games. However, **UI testing frameworks** like **Sikuli** or **Robot Framework** could be used to automate some UI testing, such as verifying button presses and animations.

**DEBUGGING**

**Debugging** the *Flappy Bird* game in Java involves systematically finding and fixing bugs in the game’s logic, visual rendering, or performance. Here’s a step-by-step guide to debugging *Flappy Bird* in Java, focusing on key areas that are prone to bugs in a game of this nature.

**1. Identifying Common Bugs in Flappy Bird**

**a. Bird Movement Issues**

* **Symptoms:**
  + The bird does not respond to key presses (flapping).
  + The bird falls too fast or moves incorrectly.
* **Cause:**
  + Logic error in gravity, velocity, or flap functions.
* **Debugging Strategy:**
  + Check the flap() and update() methods for correct changes in velocityY and the bird's position.

**b. Collision Detection Bugs**

* **Symptoms:**
  + The bird passes through pipes without collision.
  + The game ends incorrectly when the bird hits a pipe or the ground.
* **Cause:**
  + Incorrect calculation of bounding boxes or misalignment of bird and pipe positions.
* **Debugging Strategy:**
  + Debug the collision logic by printing the positions and sizes of the bird and pipes to verify if the bounding boxes overlap.

**c. Pipe Movement Issues**

* **Symptoms:**
  + Pipes are not moving, or they disappear unexpectedly.
* **Cause:**
  + Misalignment in the pipe update logic or failure to remove pipes after they leave the screen.
* **Debugging Strategy:**
  + Check the update() and isOffScreen() methods in the Pipe class to ensure pipes move left and are removed once off-screen.

**d. Scoring Problems**

* **Symptoms:**
  + The score does not increment when the bird passes through pipes.
* **Cause:**
  + Logic error in detecting when the bird passes through a pipe.
* **Debugging Strategy:**
  + Place debug statements around the score increment logic and check whether the bird's position is correctly compared with the pipes.

**2. Debugging Tools**

**Using Debugging Features in IDEs**

* Both IntelliJ IDEA and Eclipse have built-in debuggers that allow you to set breakpoints, step through code, and inspect variables during execution.
* **Breakpoints:** Pause the program execution at specific lines of code. For example, set a breakpoint in the update() method of the Bird class or the collision detection logic.
* **Step Into:** Go into a method to see exactly what it does when called.
* **Inspect Variables:** Hover over variables or use the IDE's watch list to see their values at any point in time.

**Using Print Statements**

Print statements can be used to display the internal state of variables. This is especially useful if you want to quickly check the values without setting up a full debug session.

**3. Step-by-Step Debugging Example**

Here’s a detailed example of debugging a typical issue in the *Flappy Bird* game:

**Problem:** Bird Doesn't Flap Properly

**Symptoms:** The bird does not rise when the spacebar is pressed, or it flaps and immediately falls.

**Debugging Steps:**

**Check Key Event Handling:**

* + Open the keyPressed() method in your game panel class (e.g., GamePanel).
  + Add a print statement to ensure the spacebar press is being detected.

**Check the Bird’s Flap Logic**:

Navigate to the flap() method in the Bird class and verify that the bird's velocityY is being updated properly.

Add print statements to check the bird’s velocity before and after the flap.

**Check Bird's Update Logic**:

* In the update() method of the Bird class, the bird’s vertical position is updated based on the velocity and gravity. Ensure that the logic is working as expected.
* Add a print statement to monitor the bird's position.

**Inspect Gravity Impact**:

* If the bird is falling too fast, check the value of gravity and ensure it's not too large. You may want to tweak it for smoother gameplay.

**4. Debugging Collision Detection Issues**

**Symptoms**: The bird passes through pipes without triggering a collision, or the game ends even though no collision occurs.

**Debugging Steps**:

1. **Check Collision Detection Logic**:
   * Open the collision detection code, likely in the CollisionDetector class or inside the game loop.
   * Print out the bird's and pipe's positions to check if they overlap when they should.

**Check Bounding Box Overlap**:

* Ensure the bounding box coordinates of the bird and the pipes are correctly calculated and compared. A common error is misaligning the boxes due to off-by-one or scaling issues.
* Add a breakpoint in the method where collisions are checked to inspect the exact values of the bird and pipe positions and dimensions.

**5. Debugging Pipe Movement Issues**

**Symptoms**: Pipes are not moving leftward, or they disappear too soon or too late.

**Debugging Steps**:

**Check Pipe Movement Logic**:

* + In the Pipe class, verify that the pipes are moving leftward by the correct amount in the update() method.
  + Add a print statement to track pipe movement.

**Check Pipe Removal Logic**:

* If pipes disappear unexpectedly, check the isOffScreen() method. Ensure pipes are only removed when they fully move off-screen.

**6. Debugging Score Issues**

**Symptoms**: The score doesn't increment when the bird passes through pipes.

**Debugging Steps**:

**Check Score Increment Logic**:

* + Ensure that the score is only incremented when the bird passes a pipe, not when it collides with it.
  + Add a print statement to confirm when the score is incremented.

6. EXPERIMENTAL INVESTIGATIONS

**Experimental investigations** in Java for *Flappy Bird* involve systematically altering game parameters, testing new features, and observing how they affect the gameplay. These investigations are critical for improving game mechanics, balancing gameplay, and ensuring an enjoyable user experience.

**Goals of Experimental Investigations**

1. **Optimize Game Performance:** Investigate how different parts of the game impact performance (frame rate, memory usage).
2. **Balance Game Difficulty:** Adjust parameters like gravity, pipe speed, and bird's flap power to make the game challenging but not too hard.
3. **Test Game Mechanics:** Experiment with new mechanics, such as additional obstacles, different scoring methods, or even new movement controls.

**1. Investigating Game Difficulty Parameters**

In *Flappy Bird*, the game’s difficulty is primarily determined by three factors:

* **Gravity:** How quickly the bird falls.
* **Flap Strength:** How high the bird jumps when the player presses the spacebar.
* **Pipe Speed:** How fast the pipes move towards the bird.

**Experiment 1: Varying Gravity**

**Objective:** Investigate how different values of gravity affect the difficulty of controlling the bird.

1. **Initial Setup:** The bird’s gravity affects how quickly it falls. In the Bird class, gravity is applied in the update() method.

#### **Experiment 2: Varying Flap Strength**

#### **Objective: Investigate how different flap strengths affect how high the bird flies per jump and overall game difficulty.**

#### **Initial Setup: The bird’s flap strength is determined by how much the bird’s vertical velocity is altered in the flap() method:**

**Objective**: Investigate how different flap strengths affect how high the bird flies per jump and overall game difficulty.

1. **Initial Setup**: The bird’s flap strength is determined by how much the bird’s vertical velocity is altered in the flap() method.
2. **Observation:**
   * Low Flap Strength (-5): The bird flaps less effectively, requiring more frequent flaps to stay in the air.
   * High Flap Strength (-15): The bird jumps higher, making it easier to avoid pipes but more difficult to control precision.
3. **Conclusion:** Adjust flap strength to a value where the bird’s jump height allows

**Objective**: Investigate how accurately the game detects collisions between the bird and pipes. Improve or fine-tune the collision detection if needed.

1. **Initial Setup**: The game detects collisions using the bounding boxes of the bird and pipes. Collision logic might look like this:

**5. User Interface (UI) De0sign Feedback**

* **Objective:** To analyze the effectiveness of the **UI layout and design**.
* **Experiment:** Conducted usability tests where users interacted with the interface and gave feedback on the design elements such as **text areas, combo boxes, and buttons**.
* **Findings:**
  + Users appreciated the **simple layout** but suggested adding **file import/export** options.
  + Some users recommended a **dark mode** feature for improved accessibility.

**Observation**:

* **Small Bounding Box**: Reducing the hitbox size makes it easier to pass through pipes, potentially reducing frustration for players.
* **Large Bounding Box**: Increasing the hitbox size makes the game harder because the bird is more likely to collide with the pipes.

#### **Experiment 5: Scoring Adjustments**

**Objective**: Investigate how changing the scoring system affects the player’s experience. For instance, give bonus points for continuous success or add multipliers for avoiding close calls with pipes.

1. **Initial Setup**: By default, the score increases when the bird successfully passes a pipe. This is done in the game loop:

**Internal Process Flow**

1. **GUI Event Handling:**
   * User actions (like button clicks) trigger events that invoke corresponding controller methods.
2. **Voice Management:**
   * The system allocates voice resources for speech conversion and **deallocates them after use**.
3. **Error Handling:**
   * Errors such as missing voices are **caught and handled gracefully**, ensuring the application doesn’t crash.
4. **Performance Optimization:**
   * Speech processing is optimized to **run asynchronously**, preventing the GUI from freezing during long operations.
5. **Observation**:
   * **Bonus Points**: Offering bonus points for consecutive passes or avoiding close calls makes the game more engaging.
   * **Score Multiplier**: Adding multipliers increases the stakes, making players more excited about high scores but potentially increasing difficulty.
6. **Conclusion**: Balance the scoring system to keep the game fun and rewarding. Adding bonus points for consecutive successes can motivate players to improve their skills.

**4. Experimental Performance Investigations**

**Experiment 7: Frame Rate and Performance**

**Objective**: Investigate how changes in the game’s design and mechanics affect performance, especially frame rate and memory usage.

7. WORKING OF APP:

The *Flappy Bird* game in Java is based on a simple game loop that manages the game’s flow, including user input, game logic, and rendering. Below is an explanation of how the different components of the game work together:

**1. Main Components of the Game**

The game has several key components:

* **Bird**: The main character controlled by the player.
* **Pipes**: The obstacles the bird must avoid.
* **Game Loop**: The core engine that manages game updates and rendering.
* **Collision Detection**: A system that checks for collisions between the bird and the pipes or the ground.
* **Score System**: Keeps track of how many pipes the bird successfully passes.

**2. Game Loop**

At the heart of the app is the game loop, which is responsible for continuously updating the game state and rendering the graphics. The game loop operates in three main phases:

* **Input Handling**: The player’s input is captured (e.g., pressing a key to make the bird flap).
* **Game Update**: The game’s objects (bird, pipes) are updated based on user input and the passage of time.
* **Rendering**: The updated state of the game is drawn on the screen.

**3. Bird Movement**

The bird’s movement is controlled by gravity and user input. Each frame, gravity pulls the bird downward, and when the player presses a key (e.g., the spacebar), the bird flaps upward.

* **Gravity**: A constant downward force that is applied to the bird every frame.
* **Flapping**: When the player presses a key, the bird’s vertical velocity is reduced, making it "flap" upward.

**4. Pipe Movement and Generation**

Pipes are continuously generated and move from right to left across the screen. The pipes have gaps that the bird must fly through. If the bird passes through a gap, the score increases.

* **Pipe Creation**: New pipes are created at regular intervals, typically off-screen to the right, and move leftward.
* **Pipe Speed**: The pipes move at a constant speed toward the left of the screen, making the game progressively harder as more pipes appear.

**5. Collision Detection**

The game uses bounding boxes to detect collisions between the bird and the pipes or the ground. When a collision occurs, the game ends.

* **Bounding Box Collision**: Both the bird and the pipes are treated as rectangles. The game checks if the bird's bounding box intersects with any pipe’s bounding box or the ground.
* **End Game Condition**: If a collision is detected, the game is over, and the bird falls to the ground.

**6. Scoring System**

The game increments the player’s score each time the bird successfully passes through a pair of pipes. To ensure that pipes are only counted once, a flag is set when a pipe is passed.

* **Score Increment**: When the bird’s x-position surpasses the pipe’s x-position, the score is increased by one.

**7. Rendering the Game**

The game’s visuals are rendered in each frame, drawing the bird, pipes, and background to the screen. The rendering is done using Java’s 2D graphics capabilities.

* **Bird Rendering**: The bird’s image is drawn at its current position.
* **Pipe Rendering**: Pipes are drawn moving across the screen, and new ones are generated as old ones move off-screen.

### **8. Game Over**

When a collision is detected or the bird hits the ground, the game loop ends, and the player is shown a "Game Over" screen. This may include the final score and an option to restart the game.

**9. Handling User Input**

The player controls the bird using key presses. For instance, the spacebar is typically used to make the bird flap its wings. Java’s event-handling system listens for key presses and triggers the appropriate action.

8. LIMITATIONS & FUTURE ENHANCEMENTS:

LIMITATIONS

**1. Limited Game Mechanics**

* **Simple Gameplay**: The core mechanic (flapping and avoiding pipes) remains the same throughout, which could lead to repetitive gameplay. The lack of additional features or obstacles may result in lower long-term engagement.
* **Lack of Game Variety**: There are no alternate game modes, levels, or difficulty settings to keep the game interesting over time.

**2. Basic Graphics and Animations**

* **Simple Visuals**: The visual design is often minimalistic. With basic 2D graphics, the game may not be as visually appealing or immersive as other more polished games.
* **No Animation Enhancements**: Bird movements, background elements, and pipes lack advanced animations, which could make the game feel less fluid.

**3. Poor Adaptation to Different Devices**

* **Screen Scaling Issues**: The game may not scale properly across different screen resolutions and aspect ratios, especially if it's played on varying window sizes.
* **Limited Cross-Platform Compatibility**: While it works in a Java environment, porting the game to mobile or other platforms may require significant modifications.

**4. Lack of Sound and Music**

* **No Audio Feedback**: In its basic form, the game might lack background music, sound effects, or other audio cues. This absence can make the game less engaging and dynamic.
* **Lack of Audio Synchronization**: Even if sound effects exist, they may not be perfectly synchronized with game events.

**5. Collision Detection Limitations**

* **Rigid Collision Boxes**: The bounding boxes for collision detection may not accurately represent the actual shapes of the bird and pipes, leading to frustrating or unfair collisions.
* **Lag in Collision Response**: Depending on the implementation, the game may not detect collisions in real time, resulting in glitches or delayed reactions when the bird hits a pipe or the ground.

**6. Performance Bottlenecks**

* **Resource Management**: In more complex versions of the game, resource management such as handling memory, object creation, and garbage collection may lead to performance issues, especially if too many objects are created or the game loop isn't optimized.
* **Frame Rate Drops**: As the game becomes more complex (e.g., with more objects like pipes or moving backgrounds), the frame rate might drop on lower-spec machines.

FUTURE ENHANCEMENTS

There are numerous potential enhancements that can be made to improve the game and address its current limitations. These additions would make the game more engaging, feature-rich, and adaptable.

**1. Add Difficulty Levels**

* **Adaptive Difficulty:** Implement adaptive difficulty that increases as the player progresses, such as speeding up pipes, narrowing gaps, or introducing new obstacles.
* **Different Modes:** Create different game modes like "Easy", "Hard", or "Endless" where the game’s difficulty settings vary.

**2. Improve Graphics and Animations**

* **Better Animation:** Add more fluid animations for the bird (e.g., smoother flapping), dynamic backgrounds, and animated obstacles. This would make the game feel more polished and engaging.
* **Particle Effects:** Introduce particle effects like dust clouds when the bird jumps or explodes when it hits an obstacle.

**3. Add More Obstacles and Game Mechanics**

* **New Obstacles:** Introduce new types of obstacles, such as moving pipes, falling objects, or rotating barriers to make the game more challenging and varied.
* **Power-Ups:** Add power-ups such as shields, slow motion, or extra lives that can be collected during gameplay**.**
* **Multiple Characters**: Provide the player with different bird characters, each with unique abilities or skins.

**4. Improve Collision Detection**

* **Precise Hitboxes:** Refine collision detection by using more accurate bounding boxes or pixel-perfect collision detection, so that interactions feel more natural and less frustrating.
* **Smoother Responses:** Improve the responsiveness of collision detection to ensure that when the bird hits an object, the game reacts instantly.

**5. Sound and Music Integration**

* **Add Background Music:** Include an adaptive soundtrack that changes based on the player’s progress (e.g., faster music when the game speeds up).
* **Sound Effects:** Introduce sound effects for flapping, passing pipes, and hitting obstacles. Sound feedback can make the game more immersive and fun.
* **Audio Options:** Allow players to toggle or adjust the volume of background music and sound effects.

**6. Performance Optimization**

* **Optimize Game Loop:** Streamline the game loop and improve how frames are rendered to prevent lag or frame rate issues.
* **Memory Management:** Refactor object creation and garbage collection processes to reduce memory overhead, allowing the game to run smoothly on lower-end systems.
* **Dynamic Scaling:** Ensure that the game scales properly on different screen sizes, including mobile devices. Introduce responsive design for cross-platform play.

**7. Social and Competitive Features**

* **Leaderboard Integration**: Implement a global leaderboard or local high score system to encourage players to compete for the best scores.
* **Achievements and Rewards:** Add in-game achievements for completing certain milestones, like passing 100 pipes or surviving for a specific amount of time.
* **Multiplayer Mode:** Allow players to challenge friends in a local or online multiplayer mode, adding a competitive element to the game.

**8. Porting to Mobile Devices**

* **Mobile Touch Controls:** Adapt the game for mobile devices by introducing touch controls and optimizing the UI for smaller screens.
* 9. CONCLUSION

The implementation of *Flappy Bird* in Java is a successful demonstration of building a simple yet engaging game using core object-oriented programming concepts. It showcases the fundamentals of game design, including object creation, event handling, collision detection, and a game loop that ties everything together.

Despite its simplicity, the game effectively highlights several key areas of development:

1. **Solid Programming Foundation**: The project reinforces the application of Java for building interactive applications. It employs important concepts such as classes, methods, loops, and conditionals, giving students or developers a deeper understanding of how to structure a game.
2. **Challenges and Limitations**: The limitations in graphics, mechanics, and gameplay variety reveal the challenges developers face when creating a well-rounded game. These aspects provide an excellent opportunity for further exploration and enhancement, both in terms of improving the user experience and optimizing the code for performance.
3. **Potential for Expansion**: Although the current version of *Flappy Bird* may seem basic, it offers endless possibilities for future enhancements, such as adding new features, improving graphics, and incorporating sound effects. This makes it a perfect project for learning, experimenting, and evolving a game over time.

In conclusion, the *Flappy Bird* game in Java serves as a great starting point for developers to explore game development. It combines the fundamentals of Java programming with hands-on problem-solving in game mechanics, opening the door for more advanced projects in the future. With continued refinement, this project can evolve into a polished, fully-fledged game, offering an engaging experience while continuing to build programming skills.

10. REFERENCES

[https://youtu.be/jj5ADM2uywg](https://youtu.be/Xw2MEG-FBsE)

**A Mini Project** report on

# FLAPPY BIRD

Submitted to the **CMR TECHNICAL CAMPUS** in partial fulfillment of the requirement of the

# OBJECT ORIENTED PROGRAMMING THROUGH JAVA

of

# II-B.Tech. I-Semester

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

Submitted by

BOJJA MAHATHI REDDY - 237R1A0576

SAAHINI PASHAM - 237R1A05B4

KURUBA VIBHAS SAI - 237R1A0595

THIRUKOVALLURU CHAKRADHAR - 237R1A05C5

# Under the guidance of

# Ms. G. KALPANA DEVI

# Professor, Department of COMPUTER SCIENCE AND ENGINEERING

# CMR TECHNICAL CAMPUS

# (UGC-AUTONOMOUS)

(Approved by AICTE, Affiliated to JNTUH,

Hyderabad) Kandlakoya, Medchal Road, Hyderabad.

**2024-2025**

# CMR TECHNICAL CAMPUS

# (UGC-AUTONOMOUS)

(Approved by AICTE, Affiliated to JNTUH,

Hyderabad) Kandlakoya, Medchal Road, Hyderabad.

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CERTIFICATE**

# This is to certify that a Mini Project entitled with

# FLAPPY BIRD

# is the Bonafide work carried out and submitted by

BOJJA MAHATHI REDDY - 237R1A0576

SAAHINI PASHAM - 237R1A05B4

KURUBA VIBHAS SAI - 237R1A0595

THIRUKOVALLURU CHAKRADHAR - 237R1A05C5

In partial fulfillment of the requirement for completion of the **OBJECT ORIENTED PROGRAMMING THROUGH JAVA** of II – B.Tech. I – Semester is a record of Bonafide work carried out under our guidance during the academic year 2024-2025.

**Project Guide HEAD OF THE DEPARTMENT**

Ms. G. KALPANA DEVI DR. K. SRUJAN RAJU,

Professor, Head of the Department,

Dept. of **CSE,** Dept. of Humanities and sciences,

CMRTC CMRTC

**A Mini Project** report on

# FLAPPY BIRD

Submitted to the **CMR TECHNICAL CAMPUS** in partial fulfillment of the requirement of the

# OBJECT ORIENTED PROGRAMMING THROUGH JAVA

of

# II-B.Tech. I-Semester

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

Submitted by

KURUBA VIBHAS SAI - 237R1A0595

SAAHINI PASHAM - 237R1A05B4

BOJJA MAHATHI REDDY - 237R1A0576

THIRUKOVALLURU CHAKRADHAR - 237R1A05C5

# Under the guidance of

# Ms. G. KALPANA DEVI

# Professor, Department of COMPUTER SCIENCE AND ENGINEERING

# CMR TECHNICAL CAMPUS

# (UGC-AUTONOMOUS)

(Approved by AICTE, Affiliated to JNTUH,

Hyderabad) Kandlakoya, Medchal Road, Hyderabad.

**2024-2025**

# CMR TECHNICAL CAMPUS

# (UGC-AUTONOMOUS)

(Approved by AICTE, Affiliated to JNTUH,

Hyderabad) Kandlakoya, Medchal Road, Hyderabad.

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CERTIFICATE**

# This is to certify that a Mini Project entitled with

# FLAPPY BIRD

# is the Bonafide work carried out and submitted by

KURUBA VIBHAS SAI - 237R1A0595

SAAHINI PASHAM - 237R1A05B4

BOJJA MAHATHI REDDY - 237R1A0576

THIRUKOVALLURU CHAKRADHAR - 237R1A05C5

In partial fulfillment of the requirement for completion of the **OBJECT ORIENTED PROGRAMMING THROUGH JAVA** of II – B.Tech. I – Semester is a record of Bonafide work carried out under our guidance during the academic year 2024-2025.

**Project Guide HEAD OF THE DEPARTMENT**

Ms. G. KALPANA DEVI DR. K. SRUJAN RAJU,

Professor, Head of the Department,

Dept. of **CSE,** Dept. of Humanities and sciences,

CMRTC CMRTC

**A Mini Project** report on

# FLAPPY BIRD

Submitted to the **CMR TECHNICAL CAMPUS** in partial fulfillment of the requirement of the

# OBJECT ORIENTED PROGRAMMING THROUGH JAVA

of

# II-B.Tech. I-Semester

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

Submitted by

THIRUKOVALLURU CHAKRADHAR - 237R1A05C5

SAAHINI PASHAM - 237R1A05B4

BOJJA MAHATHI REDDY - 237R1A0576

KURUBA VIBHAS SAI - 237R1A0595

# Under the guidance of

# Ms. G. KALPANA DEVI

# Professor, Department of COMPUTER SCIENCE AND ENGINEERING

# CMR TECHNICAL CAMPUS

# (UGC-AUTONOMOUS)

(Approved by AICTE, Affiliated to JNTUH,

Hyderabad) Kandlakoya, Medchal Road, Hyderabad.

**2024-2025**

# CMR TECHNICAL CAMPUS

# (UGC-AUTONOMOUS)

(Approved by AICTE, Affiliated to JNTUH,

Hyderabad) Kandlakoya, Medchal Road, Hyderabad.

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CERTIFICATE**

# This is to certify that a Mini Project entitled with

# FLAPPY BIRD

# is the Bonafide work carried out and submitted by

THIRUKOVALLURU CHAKRADHAR - 237R1A05C5

SAAHINI PASHAM - 237R1A05B4

BOJJA MAHATHI REDDY - 237R1A0576

KURUBA VIBHAS SAI - 237R1A0595

In partial fulfillment of the requirement for completion of the **OBJECT ORIENTED PROGRAMMING THROUGH JAVA** of II – B.Tech. I – Semester is a record of Bonafide work carried out under our guidance during the academic year 2024-2025.

**Project Guide HEAD OF THE DEPARTMENT**

Ms. G. KALPANA DEVI DR. K. SRUJAN RAJU,

Professor, Head of the Department,

Dept. of **CSE,** Dept. of Humanities and sciences,

CMRTC CMRTC