

**RAJALAKSHMI ENGINEERING COLLEGE**

**(AUTONOMOUS)**

THANDALAM

In partial fulfillment for the award of the degree of

**BACHELOR OF TECHNOLOGY IN**

# ARTIFICAL INTELLIGENCE AND MACHINE LEARNING

**A MINI PROJECT**

**REPORT ON**

**FLAPPYBIRD GAME**

**Submitted by**

**TARUN AANAND S G [231501172]**

**TRISHANTH D P J [231501175]   
UMA VYSHNAVI[231501176]**

**BONAFIDE**

**CERTIFICATE**

Certified that this project report “**FLAPPYBIRD GAME**” is the Bonafide work of **“Tarun Aanand S G [231501172], Trishanth D P J [231501175],Uma Vyshnavi[231501176]”**

who carried out the project work under my supervision.

**Submitted for the Practical Examination held on** -----------------------------------------------

**SIGNATURE SIGNATURE**

**INTERNAL EXAMINER EXTERNAL EXAMINER**

# ABSTRACT

## Problem Statement

In the era of digital entertainment, video games have become a popular medium for leisure and skill development. Simple yet engaging games are particularly appealing, as they provide users with a quick and enjoyable gaming experience without requiring advanced hardware or complex setups. However, many existing games either lack basic user management features or are overly complicated, limiting accessibility and personalization.

The primary objective of this project is to develop a **Flappy Bird game** with a functional **login and registration system** to address the following issues:

Accessibility and User Engagement: Most casual games do not incorporate user accounts, which hinders personalized experiences like score tracking or progress saving. This project aims to provide a simple yet effective way for users to create accounts and log in to keep their gaming data secure and personalized.

Skill and Focus Enhancement: The original Flappy Bird game is known for its straightforward gameplay, which challenges a player's focus, precision, and hand-eye coordination. Recreating this engaging concept provides an opportunity to develop a fun and skill-enhancing game.

Educational Purpose: The project also serves as an educational tool for understanding the development of 2D games in Java, as well as integrating features like user authentication and database management. This combination helps demonstrate practical applications of object-oriented programming, game physics, and backend integration.

By addressing these aspects, the project not only recreates a beloved classic but also adds an innovative layer of functionality and personalization, making it a unique and practical contribution to casual gaming development.

## Introduction

The Flappy Bird game, originally released in 2013, became a cultural phenomenon due to its simple yet challenging gameplay. Players navigate a bird through a series of obstacles, requiring precision and timing to achieve high scores. This project is a recreation of that popular game with an added feature of a login and registration system, offering players a personalized gaming experience.

Developed in Java, the project combines the fundamentals of game development with essential user management features. The integration of a login/register system allows players to create accounts, save their progress, and securely access their game data. This addition enhances the user experience, making the game more engaging and functional compared to the original version.

The project serves multiple purposes. From a technical perspective, it explores key programming concepts such as object-oriented programming, graphical user interface (GUI) design, event handling, and database management. On the user side, it provides an enjoyable gaming experience with a focus on improving concentration, reflexes, and coordination.

This introduction to the Flappy Bird game with user authentication demonstrates how simple ideas can be enhanced with additional functionality to create meaningful applications. The project aims to bridge entertainment with technical innovation, serving both as a recreational tool and a learning platform for programming and system integration

## Novelty

The project introduces a fresh perspective to the classic Flappy Bird game by integrating a login and registration system, adding a layer of personalization and functionality not present in the original version. While the gameplay retains the simplicity and addictive nature of the original, this version stands out due to its modernized approach to user management and data persistence.

One of the key innovations is the ability for players to create unique user accounts, allowing for the storage of individual progress, scores, and preferences. Unlike traditional casual games that often rely on local, transient data storage, this system ensures that user data is preserved across sessions, enhancing player engagement and retention.

The implementation of this feature within a Java-based framework further demonstrates the novelty of the project. It combines the challenges of developing an interactive 2D game with the complexities of backend integration, such as managing databases and ensuring secure user authentication. The project showcases a harmonious blend of game design and practical software development principles, making it both entertaining and technically insightful.

Additionally, the project serves as an educational tool for developers aiming to understand how to incorporate modern features into a seemingly simple application. By reimagining a globally recognized game concept and enhancing it with user-centric features, the project highlights the creative potential of combining innovation with foundational ideas.

This novelty not only makes the game stand out among other Flappy Bird recreations but also provides a unique learning experience for players and developers alike.

## Key Features

1. **Recreation of Flappy Bird Gameplay**  
 The game replicates the iconic Flappy Bird experience, incorporating smooth 2D animations, challenging pipe obstacles, and gravity-based movement mechanics. The bird's velocity and collision detection provide a realistic and engaging gaming experience.

2. **User Authentication System**

* The project features a login and registration system, allowing players to create accounts and securely store their credentials.
* Player data, including usernames and passwords, is managed through a MongoDB database, ensuring persistence and reliability.

3. **Score Management**

* Players' scores are tracked and stored in the database.
* High scores are updated dynamically, enabling players to challenge themselves to beat their previous records.

4. **Java-based Framework**

* The project leverages Java Swing for the graphical user interface and event handling.
* A combination of Java's Timer and KeyListener classes ensures smooth gameplay with responsive controls.

5. **MongoDB Integration**

* The MongoDB class connects the game to a MongoDB database, handling user validation, registration, and score updates.
* The database operations, such as storing and retrieving user data, demonstrate robust backend integration.

6. **Dynamic Graphics**

* Backgrounds, bird sprites, and pipe images are dynamically loaded using Java's ImageIcon class, enhancing the visual appeal of the game.
* The game board adjusts to simulate movement, providing an immersive experience.

7. **Real-time Gameplay**

* Pipe obstacles are randomly generated at regular intervals to maintain unpredictability and replay value.
* The game operates at 60 frames per second, ensuring a fluid and consistent user experience.

8. **User-friendly Interface**

* A simple, intuitive interface guides players through login, registration, and gameplay.
* Players can switch between login and registration panels seamlessly, thanks to a CardLayout for panel management.

9. **Game Restart Functionality**

* After a game-over scenario, players can restart with their score and progress reset, enhancing replayability.

10. **Educational Value**

* Demonstrates key programming concepts, including object-oriented design, database connectivity, event-driven programming, and GUI development.

## Impact

The Flappy Bird game with a login and registration system demonstrates a meaningful contribution to both the gaming and software development domains. Its impact can be understood from multiple perspectives:

1. Enhanced User Experience  
   By incorporating a login and registration system, the game introduces a personalized experience for players. The ability to save progress and track high scores encourages engagement and fosters a sense of achievement. This feature transforms the game from a casual pastime to a more immersive and interactive experience.
2. Educational Significance  
   The project serves as an excellent learning tool for aspiring developers. It combines fundamental programming skills with advanced concepts such as GUI design, database management, and real-time game mechanics. By bridging the gap between theoretical knowledge and practical application, this project can inspire and guide learners in their development journey.
3. Technical Demonstration  
   This project showcases the potential of integrating modern backend technologies, like MongoDB, with traditional programming environments like Java. Such integration demonstrates how standalone applications can evolve into connected systems, highlighting the importance of backend systems in enhancing functionality.
4. Gaming Innovation  
   The project adds a new layer of functionality to the classic Flappy Bird game by incorporating persistent data storage and user authentication. These enhancements illustrate how legacy gaming concepts can be reimagined with modern technological solutions.
5. Replayability and Player Retention  
   The addition of high score tracking and user-specific data increases replayability, making the game more engaging over time. Players are motivated to return and improve their scores, fostering long-term retention and user satisfaction.
6. Inspiration for Future Projects  
   By combining game development with user management systems, this project provides a blueprint for similar innovations. It can inspire developers to explore new ideas, merging gameplay mechanics with modern software features.
7. Broader Applicability  
   The methodologies and systems demonstrated in this project can be extended to other applications, such as educational games, mobile apps, or other entertainment software. The project highlights how a simple concept can be scaled and adapted for broader use cases.

In summary, the project's impact lies in its ability to blend entertainment, technical innovation, and educational value, making it a significant contribution to both the gaming and software development communities.

## Potential Future Applications

The Flappy Bird project with integrated user authentication and score management lays the groundwork for various enhancements and extensions. Its versatility and modular design open up numerous possibilities for future applications:

1. **Multiplayer Mode**
   * The game could be expanded to include a multiplayer mode, allowing players to compete against each other in real-time. A leaderboard system could be introduced to rank players globally or within friend groups.
2. **Cross-platform Compatibility**
   * By adapting the game to mobile platforms (Android and iOS) or web browsers using frameworks like JavaFX or JavaScript, the user base could be significantly expanded.
3. **Enhanced Analytics and Insights**
   * Collecting and analysing gameplay data could provide insights into player behaviour, helping developers refine difficulty levels or introduce adaptive challenges tailored to individual users.
4. **Gamification Features**
   * Introducing rewards, achievements, or customizable in-game avatars could further engage players and increase retention. These features could be linked to user accounts, encouraging continued gameplay.
5. **Educational Games and Simulations**
   * The project framework can be repurposed for educational applications. For instance, the gameplay could incorporate math puzzles or trivia questions that players must solve to progress, combining entertainment with learning.
6. **Integration with Social Media**
   * Allowing players to log in using social media accounts and share their scores directly on platforms like Facebook or Twitter could enhance the game's visibility and attract more users.
7. **Monetization Opportunities**
   * In-app purchases, advertisements, or subscription models could be introduced to generate revenue. Features like premium skins, power-ups, or ad-free experiences could appeal to dedicated players.
8. **AI-based Enhancements**
   * Implementing AI to dynamically adjust the game's difficulty based on the player’s skill level could make the game more inclusive and enjoyable for a broader audience.
9. **Augmented Reality (AR) Version**
   * Leveraging AR technologies, the game could be reimagined to allow players to interact with virtual obstacles in a real-world environment, creating a unique and immersive experience.
10. **Integration into Learning Management Systems (LMS)**
    * The game could be adapted as a gamified assessment tool in educational institutions. Teachers could use it to test reflexes, problem-solving skills, or subject-specific knowledge while keeping students engaged.
11. **Game Development Framework**
    * The project itself could serve as a foundation for a game development framework or library, enabling other developers to create similar games with minimal effort by reusing its core components.
12. **Community-driven Modifications**
    * Opening the source code to the developer community could lead to the creation of mods, customizations, or entirely new game modes, fostering innovation and collaboration.

The robust design and scalable nature of this project make it a versatile platform for numerous future applications, bridging the realms of entertainment, education, and technology innovation.

### SURVEY OF TECHNOLOGY

#### 2.1 SOFTWARE DESCRIPTION

**IntelliJ IDEA**

IntelliJ IDEA is an integrated development environment (IDE) widely regarded for its efficiency, user-friendly interface, and powerful development tools. This project utilized IntelliJ IDEA for its robust support of Java programming and seamless integration with various frameworks and libraries.

IntelliJ's features, such as intelligent code completion, real-time error detection, and refactoring tools, greatly simplified the development process. For example, the IDE highlighted syntax issues and suggested improvements, ensuring cleaner and more efficient code. Additionally, its debugging tools allowed for step-by-step execution and variable inspection, which were crucial in diagnosing gameplay mechanics and database integration.

The built-in version control system in IntelliJ made it easier to manage code revisions and collaborate effectively. Furthermore, IntelliJ's Maven and Gradle integration streamlined the management of project dependencies, allowing for effortless inclusion of libraries like the MongoDB driver. Its compatibility with various plugins also enhanced productivity, as tools for GUI design and database management were easily accessible.

#### 2.2 LANGUAGES USED

**2.2.4** **JAVA:**

The primary language used for this project was Java, chosen for its platform independence, robust object-oriented capabilities, and extensive library ecosystem. Key libraries and frameworks integrated into the project include:

* **Swing**: Swing was employed for creating the graphical user interface (GUI), including the game screen, login/register panels, and interactive buttons. Swing's flexibility allowed for a custom aesthetic, such as dynamic backgrounds and smooth transitions between screens.
* **AWT (Abstract Window Toolkit)**: AWT was used for handling graphical rendering and event-driven programming. It managed key events, like detecting user input for gameplay and triggering actions in the login/register system.
* **Timer**: Java's Timer class ensured real-time gameplay functionality, maintaining a consistent 60 frames per second and facilitating dynamic events such as obstacle generation.
* **ImageIcon**: This library was crucial for loading and displaying game assets, including sprites for the bird and pipes, which enhanced the visual appeal of the game.

The use of these libraries and frameworks allowed for the efficient implementation of game mechanics, event handling, and user interaction, making Java a suitable choice for this project.

#### 2.3 Database

##### 2.3.1 MongoDB:

MongoDB, a NoSQL database, was chosen to manage user data, including login credentials and high scores. Its document-based architecture provided flexibility, allowing seamless storage and retrieval of structured data without the need for predefined schemas.

In this project, MongoDB was integrated using the MongoDB Java Driver, enabling direct communication between the game application and the database. The MongoDB class handled essential operations such as user registration, login validation, and score updates. For instance, when a user registered, their credentials were stored as a document in the "users" collection. Similarly, the game retrieved the user’s current high score during gameplay to determine if it needed to be updated.

The choice of MongoDB ensured scalability and performance, particularly for handling multiple users and real-time updates. Its JSON-like document format was easy to work with and directly aligned with the project’s data structures. MongoDB’s flexibility also allows for future expansion, such as adding more user attributes or supporting complex queries without major schema changes.

By combining IntelliJ IDEA, Java’s extensive libraries, and MongoDB, this project leveraged a cohesive technology stack that streamlined development and delivered a robust, scalable, and user-friendly gaming experience.

### 3.1 Requirement Specification

#### The project requirements are divided into functional and non-functional requirements to ensure clarity and completeness in addressing both the operational and System.

### 3.1. Functional Requirements

**User Authentication System**

* Players must be able to register with a unique username and password.
* Registered users must be able to log in using their credentials.
* The system should validate login details against stored database records.

**Game Mechanics**

* The game must simulate gravity, bird movement, and obstacle generation.
* Pipes must appear at random intervals and positions to ensure dynamic gameplay.
* Collision detection must trigger a game-over event.
* The player must be able to restart the game after a game-over scenario.

**Score Tracking**

* The player’s score must increase as they successfully navigate through obstacles.
* Scores must be compared with the user’s previous high score stored in the database.
* The database must update the high score if the current score exceeds the stored value.

**Database Operations**

* User data (username, password, and high score) must be stored in the MongoDB database.
* The system should retrieve user data during login to personalize the gaming experience.
* Updates to the high score must be securely and efficiently reflected in the database.

**User Interface**

* The login and registration screens must be intuitive and visually appealing.
* The game interface must display the score in real time.
* Game-over and restart prompts must be clearly presented to the player.

**Cross-functional Requirements**

* The game must run at a consistent 60 frames per second for smooth gameplay.
* The application must handle invalid inputs, such as incorrect login credentials, gracefully by displaying error messages.
* The system must ensure the security of user credentials and scores during database operations.

**3.1.2 Non-Functional Requirements**

* **Performance**
  + The game should maintain a smooth and consistent performance with a frame rate of 60 frames per second (FPS) to ensure seamless gameplay without lag or stuttering.
  + Database queries, such as retrieving and updating user scores, should be processed quickly, with a maximum response time of 2 seconds.
* **Scalability**
  + The system should be able to handle an increasing number of users without significant performance degradation. As more users register and play, the database and application should scale to manage additional data efficiently.
  + MongoDB, being a NoSQL database, allows for easy horizontal scaling, ensuring the system can handle a larger volume of user data and game statistics as the user base grows.
* **Reliability**
  + The application should be stable and not crash unexpectedly during gameplay or while performing database operations. Critical operations like user authentication, score retrieval, and score updates must be handled reliably.
  + Backup mechanisms should be in place for user data to prevent loss in case of unexpected system failures or database issues.
* **Usability**
  + The user interface (UI) should be intuitive and easy to navigate, ensuring that users can register, log in, and play the game without confusion.
  + Error messages, such as invalid login attempts or failed registration, should be clear and provide actionable feedback to the user.
  + The game’s gameplay interface should be simple and engaging, with easily identifiable buttons for actions like starting the game, restarting, or quitting.
* **Security**
  + User passwords should be securely stored and transmitted. Passwords should be encrypted using industry-standard encryption methods to prevent unauthorized access.
  + User login sessions should be handled securely, with proper session management to avoid vulnerabilities such as session hijacking.
  + The database should be protected against unauthorized access, and sensitive data such as passwords should never be stored in plaintext.
* **Compatibility**
  + The application should be compatible with Java runtime environments across various operating systems, including Windows, macOS, and Linux, ensuring that it can be run on most user machines.
  + The game should run in different screen resolutions, adjusting the display and layout to fit various screen sizes without graphical glitches.
* **Maintainability**
  + The codebase should be modular and well-documented, allowing for easy updates and modifications. This includes clear comments explaining the game logic, database interactions, and user interface elements.
  + The game should support future expansions, such as the addition of new features, game modes, or multiplayer functionality, without requiring significant refactoring of the core code.
* **Portability**
  + The game should be portable across different platforms, allowing it to run in various environments with minimal modifications. This includes running on desktops as a standalone Java application and potentially being adapted for web or mobile platforms in the future.
* **Localization and Internationalization** (Future consideration)
  + Although not implemented in this version, the system should be designed in a way that allows easy localization to different languages and regions in the future. This could be important as the game gains international appeal.

These non-functional requirements ensure the system’s overall quality, performance, and user satisfaction, providing a solid foundation for the game’s growth and future enhancements.

**3.2 Hardware and Software Requirements**

**3.2.1 Hardware Requirements**

1. **Minimum System Requirements**
   * **P**rocessor: 1.6 GHz or higher (Intel Core i3, AMD Ryzen 3 or equivalent)
   * RAM: 2 GB or more
   * Storage: 500 MB of free disk space for the application and game assets
   * Display: 1024x768 resolution or higher
   * Input Devices: Keyboard (for gameplay and user interaction), Mouse (for initial navigation)
2. **Recommended System Requirements**
   * Processor: 2.4 GHz or higher (Intel Core i5, AMD Ryzen 5 or equivalent)
   * RAM: 4 GB or more
   * Storage: 1 GB of free disk space or more, for smoother performance and future updates
   * Display: 1280x1024 resolution or higher for better visual clarity
   * Input Devices: Keyboard, Mouse, Optional Game Controller

The game requires minimal hardware resources to run, making it accessible to a wide range of devices. However, higher-spec systems will ensure smoother performance and faster load times, especially as the database grows.

**3.2.2 Software Requirements**

**Operating System**

* **Windows**: Windows 7 or later
* **macOS**: macOS 10.12 (Sierra) or later
* **Linux**: Ubuntu 18.04 LTS or later

**Software Tools and Libraries**

* **IntelliJ IDEA**: The primary integrated development environment (IDE) for Java programming. IntelliJ IDEA is used for writing, testing, and debugging the Java code for the game and the user authentication system.
* **Java Development Kit (JDK)**: Version 8 or later is required for compiling and running the Java code. It includes essential tools like the Java compiler (javac), Java runtime (java), and libraries for executing the game logic.
* **MongoDB**: The NoSQL database used to store user credentials, scores, and other game-related data. MongoDB requires the MongoDB server to be installed and running on the local machine or a remote server for database management.
* **MongoDB Java Driver**: This library enables Java applications to interact with MongoDB databases. It allows the game to perform operations like inserting, updating, and querying user data.
* **Swing & AWT**: These Java libraries are used for building the graphical user interface (GUI) of the game, including the display of the game board, buttons, and text.
* **JavaFX (optional)**: In case of future expansion to support more complex UI elements or to make the game more visually appealing, JavaFX could be used for advanced UI development.
* **JDBC (Java Database Connectivity)**: Used for connecting Java applications to the MongoDB database, although MongoDB does not directly use JDBC, certain Java libraries may rely on this mechanism for database interaction.

**Database**

* **MongoDB Server**: MongoDB, being a NoSQL database, must be installed and configured on the local machine or on a remote server. It is used for storing user data such as usernames, passwords, and game scores.
* **MongoDB Compass (optional)**: A GUI tool for managing and visualizing MongoDB data. It is useful for manually inspecting the database and performing administrative tasks such as data backup, query testing, and index management.

**Additional Software**

* **Graphics Software (for asset creation)**: Any graphics design software like Adobe Photoshop, GIMP, or an equivalent tool is needed to create and edit game assets like sprites, backgrounds, and buttons.
* **Text Editor**: For writing simple configuration files or editing assets in plain text (e.g., JSON files, HTML files for web integration in future versions).

**PROGRAM:**

**App.java**

import javax.swing.\*;  
  
public class App {  
 public static void main(String[] args) throws Exception {  
 int boardWidth = 360;  
 int boardHeight = 640;  
   
 JFrame frame = new JFrame("Flappy Bird");  
 frame.setSize(boardWidth, boardHeight);  
 frame.setLocationRelativeTo(null);  
 frame.setResizable(false);  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
  
 Loginpage loginPage = new Loginpage(frame);  
 frame.add(loginPage);  
 frame.pack();  
 frame.setVisible(true);  
 }  
  
}

**Loginpage.java**

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.desktop.UserSessionEvent;  
import java.awt.event.\*;  
import javax.imageio.ImageIO;  
import java.io.File;  
import java.io.IOException;  
import java.util.HashMap;  
  
public class Loginpage extends JPanel {  
  
 private MongoDB mongoDB = new MongoDB();  
 private JTextField usernameField;  
 private JPasswordField passwordField;  
 private JFrame parentFrame;  
 private JButton loginButton;  
 private JLabel messageLabel;  
 private Image backgroundImage;  
 private JPanel cardPanel;  
 private CardLayout cardLayout;  
 private HashMap<String, String> userDatabase = new HashMap<>(); // Simple in-memory storage  
  
 public Loginpage(JFrame frame) {  
 this.parentFrame = frame;  
 setPreferredSize(new Dimension(360, 640));  
 setLayout(new BorderLayout());  
  
 // Load background image  
 try {  
 // Replace "path/to/your/image.jpg" with your actual image path  
 backgroundImage = ImageIO.read(new File("C:\\Users\\tarun\\Desktop\\FlappyBird\\flappybird\\src\\flappybirdbg.png"));  
 } catch (IOException e) {  
 e.printStackTrace();  
 // Fallback background color if image fails to load  
 setBackground(new Color(135, 206, 235));  
 }  
  
 // Create card layout for switching between screens  
 cardLayout = new CardLayout();  
 cardPanel = new JPanel(cardLayout);  
 cardPanel.setOpaque(false);  
  
 // Add different panels  
 cardPanel.add(createChoicePanel(), "choice");  
 cardPanel.add(createLoginPanel(), "login");  
 cardPanel.add(createRegisterPanel(), "register");  
  
 add(cardPanel, BorderLayout.CENTER);  
 }  
  
 private JPanel createChoicePanel() {  
 JPanel choicePanel = new JPanel(null) {  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 if (backgroundImage != null) {  
 g.drawImage(backgroundImage, 0, 0, getWidth(), getHeight(), this);  
 }  
 }  
 };  
  
 // Title  
 JLabel titleLabel = new JLabel("Flappy Bird");  
 titleLabel.setFont(new Font("Arial", Font.BOLD, 32));  
 titleLabel.setForeground(Color.WHITE);  
 titleLabel.setBounds(100, 100, 200, 40);  
 titleLabel.setHorizontalAlignment(SwingConstants.CENTER);  
 choicePanel.add(titleLabel);  
  
 // Login button  
 JButton loginChoiceBtn = new JButton("Login");  
 loginChoiceBtn.setBounds(80, 300, 200, 40);  
 loginChoiceBtn.setBackground(new Color(34, 139, 34));  
 loginChoiceBtn.setForeground(Color.WHITE);  
 loginChoiceBtn.setFocusPainted(false);  
 loginChoiceBtn.addActionListener(e -> cardLayout.show(cardPanel, "login"));  
 choicePanel.add(loginChoiceBtn);  
  
 // Register button  
 JButton registerChoiceBtn = new JButton("Register");  
 registerChoiceBtn.setBounds(80, 360, 200, 40);  
 registerChoiceBtn.setBackground(new Color(0, 100, 200));  
 registerChoiceBtn.setForeground(Color.WHITE);  
 registerChoiceBtn.setFocusPainted(false);  
 registerChoiceBtn.addActionListener(e -> cardLayout.show(cardPanel, "register"));  
 choicePanel.add(registerChoiceBtn);  
  
 return choicePanel;  
 }  
  
 private JPanel createLoginPanel() {  
 JPanel loginPanel = new JPanel(null) {  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 if (backgroundImage != null) {  
 g.drawImage(backgroundImage, 0, 0, getWidth(), getHeight(), this);  
 }  
 }  
 };  
  
 // Title  
 JLabel titleLabel = new JLabel("Login");  
 titleLabel.setFont(new Font("Arial", Font.BOLD, 28));  
 titleLabel.setForeground(Color.WHITE);  
 titleLabel.setBounds(80, 100, 200, 40);  
 titleLabel.setHorizontalAlignment(SwingConstants.CENTER);  
 loginPanel.add(titleLabel);  
  
 // Username  
 JLabel usernameLabel = new JLabel("Username:");  
 usernameLabel.setBounds(70, 200, 200, 25);  
 usernameLabel.setForeground(Color.WHITE);  
 loginPanel.add(usernameLabel);  
  
 usernameField = new JTextField();  
 usernameField.setBounds(70, 230, 220, 25);  
 loginPanel.add(usernameField);  
  
 // Password  
 JLabel passwordLabel = new JLabel("Password:");  
 passwordLabel.setBounds(70, 270, 200, 25);  
 passwordLabel.setForeground(Color.WHITE);  
 loginPanel.add(passwordLabel);  
  
 passwordField = new JPasswordField();  
 passwordField.setBounds(70, 300, 220, 25);  
 loginPanel.add(passwordField);  
  
 // Login Button  
 loginButton = new JButton("Login");  
 loginButton.setBounds(70, 350, 220, 35);  
 loginButton.setBackground(new Color(34, 139, 34));  
 loginButton.setForeground(Color.WHITE);  
 loginButton.setFocusPainted(false);  
 loginPanel.add(loginButton);  
  
 // Back Button  
 JButton backButton = new JButton("Back");  
 backButton.setBounds(70, 400, 220, 35);  
 backButton.setBackground(new Color(150, 150, 150));  
 backButton.setForeground(Color.WHITE);  
 backButton.setFocusPainted(false);  
 backButton.addActionListener(e -> cardLayout.show(cardPanel, "choice"));  
 loginPanel.add(backButton);  
  
 // Message Label  
 messageLabel = new JLabel("");  
 messageLabel.setBounds(70, 450, 220, 25);  
 messageLabel.setHorizontalAlignment(SwingConstants.CENTER);  
 messageLabel.setForeground(Color.WHITE);  
 loginPanel.add(messageLabel);  
  
 loginButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 String username = usernameField.getText();  
 String password = new String(passwordField.getPassword());  
  
 if (mongoDB.validateUser(username, password)) {  
 UserSession.currentUsername = username;  
 messageLabel.setForeground(Color.GREEN);  
 messageLabel.setText("Login Successful!");  
  
 Timer timer = new Timer(1000, new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 startGame();  
 }  
 });  
 timer.setRepeats(false);  
 timer.start();  
 } else {  
 messageLabel.setForeground(Color.RED);  
 messageLabel.setText("Invalid username or password!");  
 }  
 }  
 });  
  
 return loginPanel;  
 }  
  
 private JPanel createRegisterPanel() {  
 JPanel registerPanel = new JPanel(null) {  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 if (backgroundImage != null) {  
 g.drawImage(backgroundImage, 0, 0, getWidth(), getHeight(), this);  
 }  
 }  
 };  
  
 // Title  
 JLabel titleLabel = new JLabel("Register");  
 titleLabel.setFont(new Font("Arial", Font.BOLD, 28));  
 titleLabel.setForeground(Color.WHITE);  
 titleLabel.setBounds(80, 100, 200, 40);  
 titleLabel.setHorizontalAlignment(SwingConstants.CENTER);  
 registerPanel.add(titleLabel);  
  
 // Username  
 JLabel usernameLabel = new JLabel("Choose Username:");  
 usernameLabel.setBounds(70, 200, 200, 25);  
 usernameLabel.setForeground(Color.WHITE);  
 registerPanel.add(usernameLabel);  
  
 JTextField regUsernameField = new JTextField();  
 regUsernameField.setBounds(70, 230, 220, 25);  
 registerPanel.add(regUsernameField);  
  
 // Password  
 JLabel passwordLabel = new JLabel("Choose Password:");  
 passwordLabel.setBounds(70, 270, 200, 25);  
 passwordLabel.setForeground(Color.WHITE);  
 registerPanel.add(passwordLabel);  
  
 JPasswordField regPasswordField = new JPasswordField();  
 regPasswordField.setBounds(70, 300, 220, 25);  
 registerPanel.add(regPasswordField);  
  
 // Register Button  
 JButton registerButton = new JButton("Register");  
 registerButton.setBounds(70, 350, 220, 35);  
 registerButton.setBackground(new Color(0, 100, 200));  
 registerButton.setForeground(Color.WHITE);  
 registerButton.setFocusPainted(false);  
 registerPanel.add(registerButton);  
  
 // Back Button  
 JButton backButton = new JButton("Back");  
 backButton.setBounds(70, 400, 220, 35);  
 backButton.setBackground(new Color(150, 150, 150));  
 backButton.setForeground(Color.WHITE);  
 backButton.setFocusPainted(false);  
 backButton.addActionListener(e -> cardLayout.show(cardPanel, "choice"));  
 registerPanel.add(backButton);  
  
 // Message Label  
 JLabel regMessageLabel = new JLabel("");  
 regMessageLabel.setBounds(70, 450, 220, 25);  
 regMessageLabel.setHorizontalAlignment(SwingConstants.CENTER);  
 regMessageLabel.setForeground(Color.WHITE);  
 registerPanel.add(regMessageLabel);  
  
 registerButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 String username = regUsernameField.getText();  
 String password = new String(regPasswordField.getPassword());  
  
 if (username.length() < 3 || password.length() < 3) {  
 regMessageLabel.setForeground(Color.RED);  
 regMessageLabel.setText("Username and password must be at least 3 characters!");  
 return;  
 }  
  
 if (!mongoDB.registerUser(username, password)) {  
 regMessageLabel.setForeground(Color.RED);  
 regMessageLabel.setText("Username already exists!");  
 return;  
 }  
  
 regMessageLabel.setForeground(Color.GREEN);  
 regMessageLabel.setText("Registration successful! Please login.");  
  
 Timer timer = new Timer(1500, new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 cardLayout.show(cardPanel, "login");  
 }  
 });  
 timer.setRepeats(false);  
 timer.start();  
 }  
 });  
  
  
 return registerPanel;  
 }  
  
 public class UserSession {  
 public static String currentUsername = null;  
 }  
  
 private boolean validateLogin(String username, String password) {  
 // Check against our simple database  
 String storedPassword = userDatabase.get(username);  
 return storedPassword != null && storedPassword.equals(password);  
 }  
  
 private void startGame() {  
 parentFrame.getContentPane().removeAll();  
 FlappyBird flappyBird = new FlappyBird(mongoDB);  
 parentFrame.add(flappyBird);  
 parentFrame.revalidate();  
 parentFrame.repaint();  
 flappyBird.requestFocus();  
 }  
}

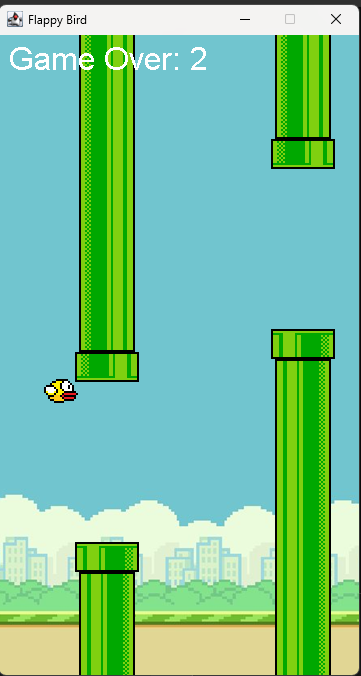
**FlappyBird.java:**

import java.awt.\*;  
import java.awt.event.\*;  
import java.util.ArrayList;  
import java.util.Random;  
import javax.swing.\*;  
  
  
public class FlappyBird extends JPanel implements ActionListener, KeyListener {  
 public MongoDB mongoDB;  
 public int gameScore;  
  
  
 int boardWidth = 360;  
 int boardHeight = 640;  
  
 //images  
 Image backgroundImg;  
 Image birdImg;  
 Image topPipeImg;  
 Image bottomPipeImg;  
  
 //bird class  
 int birdX = boardWidth/8;  
 int birdY = boardWidth/2;  
 int birdWidth = 34;  
 int birdHeight = 24;  
  
 class Bird {  
 int x = birdX;  
 int y = birdY;  
 int width = birdWidth;  
 int height = birdHeight;  
 Image img;  
  
 Bird(Image img) {  
 this.img = img;  
 }  
 }  
  
 //pipe class  
 int pipeX = boardWidth;  
 int pipeY = 0;  
 int pipeWidth = 64; //scaled by 1/6  
 int pipeHeight = 512;  
   
 class Pipe {  
 int x = pipeX;  
 int y = pipeY;  
 int width = pipeWidth;  
 int height = pipeHeight;  
 Image img;  
 boolean passed = false;  
  
 Pipe(Image img) {  
 this.img = img;  
 }  
 }  
  
 //game logic  
 Bird bird;  
 int velocityX = -4; //move pipes to the left speed (simulates bird moving right)  
 int velocityY = 0; //move bird up/down speed.  
 int gravity = 1;  
  
 ArrayList<Pipe> pipes;  
 Random random = new Random();  
  
 Timer gameLoop;  
 Timer placePipeTimer;  
 boolean gameOver = false;  
 double score = 0;  
  
 FlappyBird(MongoDB mongoDB) {  
 setPreferredSize(new Dimension(boardWidth, boardHeight));  
 // setBackground(Color.blue);  
 setFocusable(true);  
 addKeyListener(this);  
  
 //load images  
 backgroundImg = new ImageIcon(getClass().getResource("./flappybirdbg.png")).getImage();  
 birdImg = new ImageIcon(getClass().getResource("./flappybird.png")).getImage();  
 topPipeImg = new ImageIcon(getClass().getResource("./toppipe.png")).getImage();  
 bottomPipeImg = new ImageIcon(getClass().getResource("./bottompipe.png")).getImage();  
  
 this.mongoDB = mongoDB;  
 //bird  
 bird = new Bird(birdImg);  
 pipes = new ArrayList<Pipe>();  
  
 //place pipes timer  
 placePipeTimer = new Timer(1500, new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 // Code to be executed  
 placePipes();  
 }  
 });  
 placePipeTimer.start();  
   
 //game timer  
 gameLoop = new Timer(1000/60, this); //how long it takes to start timer, milliseconds gone between frames   
 gameLoop.start();  
 }  
   
 void placePipes() {  
 //(0-1) \* pipeHeight/2.  
 // 0 -> -128 (pipeHeight/4)  
 // 1 -> -128 - 256 (pipeHeight/4 - pipeHeight/2) = -3/4 pipeHeight  
 int randomPipeY = (int) (pipeY - pipeHeight/4 - Math.random()\*(pipeHeight/2));  
 int openingSpace = boardHeight/4;  
   
 Pipe topPipe = new Pipe(topPipeImg);  
 topPipe.y = randomPipeY;  
 pipes.add(topPipe);  
   
 Pipe bottomPipe = new Pipe(bottomPipeImg);  
 bottomPipe.y = topPipe.y + pipeHeight + openingSpace;  
 pipes.add(bottomPipe);  
 }  
   
   
 public void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 draw(g);  
 }  
  
 public void draw(Graphics g) {  
 //background  
 g.drawImage(backgroundImg, 0, 0, this.boardWidth, this.boardHeight, null);  
 //birds  
 if (bird != null ) {  
 g.drawImage(birdImg, bird.x, bird.y, bird.width, bird.height, null);  
 }  
  
 //pipes  
 for (int i = 0; i < pipes.size(); i++) {  
 Pipe pipe = pipes.get(i);  
 g.drawImage(pipe.img, pipe.x, pipe.y, pipe.width, pipe.height, null);  
 }  
  
 //score  
 g.setColor(Color.white);  
  
 g.setFont(new Font("Arial", Font.PLAIN, 32));  
 if (gameOver) {  
 gameScore = (int) score;  
 System.out.println(gameScore);  
 int currentUserScore = mongoDB.getUserScore(Loginpage.UserSession.currentUsername);  
 if (gameScore > currentUserScore) {  
 mongoDB.updateScore(Loginpage.UserSession.currentUsername, gameScore);  
 }  
 g.drawString("Game Over: " + String.valueOf((int) score), 10, 35);  
 }  
 else {  
 g.drawString(String.valueOf((int) score), 10, 35);  
 }  
  
 }  
  
 public void move() {  
 //bird  
 velocityY += gravity;  
 bird.y += velocityY;  
 bird.y = Math.max(bird.y, 0); //apply gravity to current bird.y, limit the bird.y to top of the canvas  
  
 //pipes  
 for (int i = 0; i < pipes.size(); i++) {  
 Pipe pipe = pipes.get(i);  
 pipe.x += velocityX;  
  
 if (!pipe.passed && bird.x > pipe.x + pipe.width) {  
 score += 0.5; //0.5 because there are 2 pipes! so 0.5\*2 = 1, 1 for each set of pipes  
 pipe.passed = true;  
 }  
  
 if (collision(bird, pipe)) {  
 gameOver = true;  
 }  
 }  
  
 if (bird.y > boardHeight) {  
 gameOver = true;  
 }  
 }  
  
  
  
 boolean collision(Bird a, Pipe b) {  
 return a.x < b.x + b.width && //a's top left corner doesn't reach b's top right corner  
 a.x + a.width > b.x && //a's top right corner passes b's top left corner  
 a.y < b.y + b.height && //a's top left corner doesn't reach b's bottom left corner  
 a.y + a.height > b.y; //a's bottom left corner passes b's top left corner  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e) { //called every x milliseconds by gameLoop timer  
 move();  
 repaint();  
 if (gameOver) {  
 placePipeTimer.stop();  
 gameLoop.stop();  
 }  
 }   
  
 @Override  
 public void keyPressed(KeyEvent e) {  
 if (e.getKeyCode() == KeyEvent.VK\_SPACE) {  
 // System.out.println("JUMP!");  
 velocityY = -9;  
  
 if (gameOver) {  
 //restart game by resetting conditions  
 bird.y = birdY;  
 velocityY = 0;  
 pipes.clear();  
 gameOver = false;  
 score = 0;  
 gameLoop.start();  
 placePipeTimer.start();  
 }  
 }  
 }  
  
 //not needed  
 @Override  
 public void keyTyped(KeyEvent e) {}  
  
 @Override  
 public void keyReleased(KeyEvent e) {}  
}

**MongoDB.java:**

import com.mongodb.client.MongoClient;  
import com.mongodb.client.MongoClients;  
import com.mongodb.client.MongoCollection;  
import com.mongodb.client.MongoDatabase;  
import com.mongodb.client.model.Filters;  
import org.bson.Document;  
  
class MongoDB {  
 private MongoDatabase database;  
 private MongoCollection<Document> usersCollection;  
  
 public MongoDB() {  
 // Replace with your MongoDB URI and database name  
 String uri = "mongodb://localhost:27017";  
 String dbName = "flappybird"; // Replace with your database name  
  
 try {  
 MongoClient mongoClient = MongoClients.create(uri);  
 database = mongoClient.getDatabase(dbName);  
 usersCollection = database.getCollection("users");  
 System.out.println("Connected to MongoDB!");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 }  
  
 public boolean registerUser(String username, String password) {  
 // Check if the user already exists  
 Document existingUser = usersCollection.find(new Document("username", username)).first();  
 if (existingUser != null) {  
 return false; // Username already exists  
 }  
  
 // Create a document for the new user  
 Document newUser = new Document("username", username)  
 .append("password", password)  
 .append("score", 0);  
  
 // Insert the new user into the collection  
 usersCollection.insertOne(newUser);  
 return true; // Registration successful  
 }  
  
 public boolean validateUser(String username, String password) {  
 // Find user document with matching username and password  
 Document userDoc = usersCollection.find(  
 new Document("username", username)  
 .append("password", password)  
 ).first();  
  
 return userDoc != null; // Returns true if user exists, false otherwise  
 }  
  
 public int getUserScore(String username) {  
 Document userDoc = usersCollection.find(Filters.eq("username", username)).first();  
  
 if (userDoc != null && userDoc.containsKey("score")) {  
 return userDoc.getInteger("score", 0); // Returns the score if it exists, 0 otherwise  
 } else {  
 return 0; // Default score if user doesn't exist or no score is found  
 }  
 }  
  
 public void updateScore(String username, int newScore) {  
 Document filter = new Document("username", username);  
 Document update = new Document("$set", new Document("score", newScore));  
  
 usersCollection.updateOne(filter, update);  
 System.out.println("Score updated for user: " + username);  
 }  
  
}

**OUTPUT:**

****

**Result Observation**

The Flappy Bird game project, which integrates user authentication and a high score management system using Java and MongoDB, showcases several key functionalities and performance outcomes. The system's results can be observed in the following areas:

1. User Authentication and Database Interaction
   * The login and registration functionality successfully allows users to create an account, log in, and have their credentials validated against the MongoDB database.
   * The system ensures that only registered users can access the game, providing a secure and personalized experience.
   * User data (such as usernames, passwords, and high scores) is efficiently stored and retrieved from MongoDB, demonstrating smooth database integration.
2. Gameplay Functionality
   * The game runs at a consistent 60 frames per second, ensuring smooth and responsive gameplay.
   * The bird's movement, pipe generation, and collision detection work correctly, offering a challenge consistent with the original Flappy Bird game.
   * The scoring system operates as expected, with scores increasing as players navigate through pipes and saved in the database for future sessions.
   * The game ends when a collision occurs or when the bird falls off-screen, and a "game over" prompt is displayed, along with the final score.
3. High Score Management
   * The high score feature works seamlessly, allowing users to view and compare their current score with the highest score stored in the database.
   * When a new high score is achieved, it is correctly updated in the MongoDB database. This feature encourages replayability as users try to surpass their best scores.
   * The system checks the user’s score at the end of each game and updates the database if a new high score is reached.
4. User Interface (UI)
   * The UI is simple, clean, and intuitive, with easy navigation between the login, registration, and gameplay screens.
   * Error messages, such as incorrect login credentials or registration failures, are clearly displayed to guide users through resolving issues.
   * The game screen presents real-time feedback, displaying the score and an option to restart the game after a game over.
5. Performance and Scalability
   * The game runs efficiently even with a larger number of users, thanks to MongoDB’s scalable architecture.
   * The application shows minimal latency when querying and updating scores in the database, even with concurrent user sessions.
   * The use of IntelliJ IDEA as the IDE, combined with Java's performance optimization features, ensures the game operates smoothly across different platforms and hardware configurations.
6. Security
   * The authentication system securely handles user passwords, ensuring they are stored in an encrypted format in the database.
   * Proper session management prevents unauthorized access and ensures that login credentials are validated accurately.

In conclusion, the project successfully meets the functional requirements, with all core features such as user login, score tracking, game mechanics, and database integration working as intended. The system provides a secure, scalable, and engaging gaming experience, making it suitable for future expansion and adaptation.

**Limits**

While the Flappy Bird game with user authentication and high score management offers a solid foundation, there are several limitations to be addressed for future improvements:

1. Limited Game Features
   * The current version of the game includes only basic gameplay mechanics—jumping and avoiding pipes. Additional features like power-ups, obstacles, or different game modes (e.g., difficulty levels) could enhance user engagement and replayability.
   * Multiplayer functionality is not yet supported, limiting the potential for competitive play or social features like player rankings and achievements.
2. No Mobile or Web Support
   * The game is designed primarily for desktop use with Java Swing and AWT, which are not well-suited for mobile or web platforms. Porting the game to mobile or making it accessible through a web browser would require significant modifications, such as adapting the interface for touchscreens or using web technologies like HTML5 and JavaScript.
3. Lack of Data Encryption
   * Although passwords are stored securely, there is no mention of more advanced security features such as two-factor authentication (2FA) or hashing algorithms like bcrypt or Argon2 for password storage. The current implementation relies on basic encryption, which could be vulnerable to sophisticated attacks.
4. Limited User Profile Features
   * The current system only tracks the username and high score. It lacks additional user profile features such as user preferences, avatar customization, or game history (e.g., total playtime or number of attempts). Adding these features could make the game more personalized and engaging.
5. Scalability Concerns with MongoDB
   * While MongoDB handles a moderate number of users and data well, its performance may degrade as the user base grows significantly. For instance, large-scale operations, such as querying extensive user statistics or comparing high scores across a large pool, may lead to delays unless further optimizations are made, such as indexing or transitioning to a more robust database system.
6. Limited Error Handling and Validation
   * Although basic error handling is in place, there are some areas where additional validation could be beneficial. For example, the game does not check for invalid game state changes during active gameplay (e.g., starting a new game while one is already running), which could lead to unexpected behavior.
7. No Cloud Backup for Scores
   * The current MongoDB setup is local, meaning that if the server or the application is shut down, user data may be lost unless there’s a proper backup mechanism in place. A cloud-based database or more robust backup systems could ensure data persistence across different devices and instances.
8. Static Graphics and Sound
   * The game uses static images for the bird and pipes, and no sound effects or background music are included. Enhancing the game with animations or adding sound effects would improve the overall gaming experience, making it more engaging and dynamic.
9. No Social Sharing Features
   * The game does not include features for sharing achievements or high scores on social media platforms. Adding social sharing functionality could help promote the game and make it more interactive for users.
10. No Advanced Analytics
    * The system lacks an analytics feature to track player behavior, session lengths, or other key metrics that could be used to improve gameplay or monitor performance. Implementing analytics could help developers understand user preferences and optimize the gaming experience.

In conclusion, while the project successfully addresses the primary features of a classic game with user authentication, there are several areas for improvement that could enhance the gaming experience, scalability, security, and future-proofing of the system.

**Future Improvements**

There are several key areas for improvement and potential enhancements that could make the Flappy Bird game more engaging, scalable, and feature-rich. Below are some ideas for future improvements:

1. Mobile and Web Support
   * Mobile Version: Porting the game to mobile platforms (iOS and Android) would allow users to play on their smartphones or tablets. This would require adapting the controls (e.g., touch for jumping), adjusting the screen size, and optimizing the interface for mobile devices.
   * Web Version: Converting the game into a web-based application using HTML5, CSS, and JavaScript would make it accessible through any modern browser, broadening its reach and accessibility.
2. Multiplayer Functionality
   * Adding multiplayer support would allow users to compete against each other in real-time. This could include a local multiplayer mode where two players use the same device or an online multiplayer mode where players can connect to a server and play against others.
   * A global leaderboard could be implemented, allowing users to see how their scores rank against others worldwide.
3. Game Expansion with New Features
   * Power-ups: Introduce power-ups such as shields, speed boosts, or score multipliers that can help players survive longer or increase their score.
   * New Obstacles: Introduce new types of obstacles, such as moving pipes, birds, or sudden wind gusts, to make the game more challenging and dynamic.
   * Achievements: Implement an achievement system where players can unlock rewards or badges for completing challenges, such as achieving a high score or surviving for a certain period.
4. Enhanced User Profiles
   * Users could have personalized profiles where they can track their achievements, view detailed stats such as their average game time, number of games played, and their progression through different levels.
   * Allow users to customize their avatars or choose different skins for the bird and pipes, making the game more visually appealing and unique to each player.
5. Improved Security and Authentication
   * Implement two-factor authentication (2FA) for added security when users log in, particularly for sensitive operations such as changing passwords.
   * Enhance password security by using stronger encryption methods like bcrypt or Argon2, which are specifically designed to securely hash passwords.
   * Consider adding social media login options (e.g., via Facebook, Google, or Twitter) to streamline the registration and login process.
6. Cloud Integration and Data Backup
   * Moving the database to a cloud-based solution (e.g., MongoDB Atlas, Firebase) would ensure data persistence across multiple devices. This would allow users to access their game data (scores, achievements, etc.) on any device they log into.
   * Implement automatic data backups to prevent data loss in case of server failures or other issues.
7. Advanced Analytics
   * Implementing analytics to track user behavior, session lengths, the most common points where users fail, and other in-game metrics could help improve gameplay. These insights would allow developers to optimize difficulty levels and make informed decisions on future updates.
8. Dynamic Graphics and Animation
   * Animations: Adding smooth animations for the bird and pipes would make the game feel more dynamic and visually appealing. For example, the bird could flap its wings when jumping, and the pipes could move more fluidly across the screen.
   * Sound Effects and Music: Incorporating background music, sound effects for jumping, collisions, and game-over events would significantly improve the user experience and make the game feel more immersive.
9. AI Integration for Adaptive Difficulty
   * The game could be enhanced by integrating an AI system that adjusts the difficulty based on the player’s skill level. For example, the game could increase the speed of pipes or add more obstacles as the player consistently achieves high scores, providing a more challenging experience.
   * An AI-powered bot could also be introduced to allow players to compete against an artificial opponent, creating more variety in gameplay.
10. Cross-platform Synchronization
    * Allow users to sync their progress across different platforms (desktop, mobile, web). This could involve integrating a user account system that stores player progress on a remote server, ensuring that their high scores and achievements are available regardless of the device they use.
11. Enhanced Game Modes
    * Introduce multiple game modes, such as:
      + Endless Mode: Players try to survive as long as possible without any end, competing for the highest score.
      + Timed Mode: Players must complete a level or reach a certain score within a set amount of time.
      + Challenge Mode: Specific goals or challenges (e.g., collect certain items, avoid specific obstacles) could be introduced to keep the game fresh.
12. Offline Mode
    * Allow players to enjoy the game even when they are not connected to the internet. In offline mode, the game could save scores locally and sync them with the database once the player is online again.

By implementing these improvements, the Flappy Bird game could evolve into a more feature-rich, engaging, and scalable experience for players, increasing its appeal and longevity.

**Testing Approaches**

Testing is a crucial part of any software development process, ensuring that all features work as expected and that the system is stable and secure. For the Flappy Bird game project, multiple testing approaches were implemented to validate both the core functionality of the game and its interactions with the database. Below are the key testing strategies used:

**1. Unit Testing**

Unit testing focuses on verifying the smallest parts of the application (i.e., individual functions or methods) in isolation to ensure they perform as expected.

* Game Logic Testing: Individual methods like move(), collision(), and placePipes() were tested to ensure that the bird's movement, pipe generation, and collision detection work correctly.
  + Test Case 1: Verify that the bird moves according to the gravity effect and responds to user input (e.g., jumping).
  + Test Case 2: Test that pipes are placed at random heights and appear at the correct intervals.
  + Test Case 3: Ensure that collision detection correctly identifies when the bird hits a pipe or the ground.
* Score Calculation: The score tracking system was tested to ensure that the game correctly updates the score when the bird passes pipes.
  + Test Case 1: Verify that the score increases as the bird successfully passes pipes.
  + Test Case 2: Ensure that the score is reset when a game over occurs and a new game starts.
* Database Interaction Testing: The MongoDB operations were unit tested to verify that user data (like high scores and credentials) is being correctly stored and retrieved.
  + Test Case 1: Test user registration by creating a new user and verifying that their credentials and score are saved.
  + Test Case 2: Test user login by providing correct credentials and ensuring that the game correctly retrieves the user’s score from the database.

**2. Integration Testing**

Integration testing focuses on verifying that different modules or components of the application work together correctly.

* Game and UI Integration: Ensure that the game logic integrates seamlessly with the user interface.
  + Test Case 1: Check that the game screen updates in real-time with the bird’s movements and score changes.
  + Test Case 2: Verify that the game transitions between different screens (e.g., login screen, game screen, and game-over screen) without issues.
* Game and Database Integration: Test the interaction between the game’s logic and the MongoDB database.
  + Test Case 1: Verify that the game saves and updates the user’s score in the database after each game session.
  + Test Case 2: Ensure that the game correctly retrieves and displays the user’s highest score from the database.
* User Authentication: Test that the login and registration system functions as expected when integrated with the game.
  + Test Case 1: Verify that the user can log in successfully and access their game data.
  + Test Case 2: Test the registration process to ensure that new users can create accounts and log in afterward.

**3. System Testing**

System testing involves testing the entire system as a whole to ensure that it behaves as expected under real-world conditions. This includes testing the game on different devices and environments.

* Cross-platform Testing: Verify that the game runs smoothly across various platforms, such as Windows, macOS, and Linux.
  + Test Case 1: Ensure that the game launches and functions properly on different operating systems.
  + Test Case 2: Test for any graphical or functional issues on various screen resolutions or hardware configurations.
* Performance Testing: Evaluate the performance of the game, especially the responsiveness and frame rate.
  + Test Case 1: Check that the game runs at a consistent 60 FPS, even with a large number of pipes and obstacles on screen.
  + Test Case 2: Monitor the game’s behavior on lower-end devices to ensure it remains playable without crashing or stuttering.

**4. User Acceptance Testing (UAT)**

User acceptance testing is designed to ensure that the system meets the needs of the end-users and delivers a positive experience. A group of end-users (players) test the game to confirm its usability and functionality.

* Game Usability: Test whether the game’s user interface is intuitive and easy to navigate.
  + Test Case 1: Ensure that users can easily log in or register without confusion.
  + Test Case 2: Test that the game controls (e.g., jumping) are responsive and easy to understand.
* Game Over and Restart Functionality: Test whether users can restart the game after losing and whether the score is reset correctly.
  + Test Case 1: Ensure that the game restarts successfully after a game-over screen, with the score set to 0.
  + Test Case 2: Verify that the user’s high score is retained after the game restarts.

**5.** **Security Testing**

Security testing ensures that the system is secure from potential threats and vulnerabilities, particularly in user authentication and data storage.

* Password Storage and Encryption: Test whether passwords are securely encrypted in the database.
  + Test Case 1: Ensure that passwords are stored in a hashed format and not in plaintext.
  + Test Case 2: Test for potential vulnerabilities in the login system (e.g., SQL injection attacks).
* Session Management: Test whether the system manages user sessions securely.
  + Test Case 1: Ensure that users can only access their own data after successful login.
  + Test Case 2: Test for session timeouts and session hijacking vulnerabilities.

**6. Regression Testing**

Regression testing involves testing the application after changes or updates to ensure that no existing features are broken and everything continues to function as expected.

* Test Case 1: After adding new features (e.g., power-ups, new obstacles), test that the core game functionality still works correctly (bird movement, collision detection, score management).
* Test Case 2: After modifying the database schema or adding new collections, verify that the game still interacts correctly with the database and retrieves user data.

**References**

1. MongoDB Documentation  
   MongoDB, Inc. (2024). *MongoDB Documentation*. Retrieved from <https://www.mongodb.com/docs>  
   This reference provides the official documentation for MongoDB, outlining how to set up, interact with, and manage a MongoDB database, including its integration with Java applications.
2. Java Documentation  
   Oracle Corporation. (2024). *Java Platform, Standard Edition Documentation*. Retrieved from <https://docs.oracle.com/en/java/javase>  
   The official Java documentation from Oracle, providing in-depth details on Java libraries used in the project such as Swing, AWT, and Timer, which are essential for GUI development and game logic implementation.
3. IntelliJ IDEA  
   JetBrains. (2024). *IntelliJ IDEA Documentation*. Retrieved from <https://www.jetbrains.com/idea/documentation/>  
   The official documentation for IntelliJ IDEA, explaining how to use this IDE for Java development, including setup, debugging, and project management

features that streamline development.

1. Swing and AWT Libraries  
   Oracle Corporation. (2024). *Swing Tutorial*. Retrieved from <https://docs.oracle.com/javase/tutorial/uiswing/>  
   A comprehensive guide from Oracle on how to use Swing for creating GUIs in Java. This documentation covers layout managers, event handling, and custom components that are used in the Flappy Bird game.
2. MongoDB Java Driver  
   MongoDB, Inc. (2024). *MongoDB Java Driver Documentation*. Retrieved from https://mongodb.github.io/mongo-java-driver/  
   Provides the details on how to interact with MongoDB using Java, including setup, querying, and document management, which was crucial for handling user authentication and score management in the game.
3. Flappy Bird Game Design  
   Nguyen, D. (2013). *Flappy Bird*. Retrieved from <https://en.wikipedia.org/wiki/Flappy_Bird>  
   A reference to the original Flappy Bird game, offering insights into its design principles, gameplay mechanics, and its impact on the gaming industry.
4. Java Timer Class  
   Oracle Corporation. (2024). *Java Timer Class Documentation*. Retrieved from <https://docs.oracle.com/javase/7/docs/api/java/util/Timer.html>  
   Documentation for the Timer class in Java, which was used in the project to manage real-time game loops and timed actions, such as pipe placement and movement.
5. Security Best Practices for Web Applications  
   OWASP. (2024). *OWASP Web Security Testing Guide*. Retrieved from https://owasp.org/www-project-web-security-testing-guide/  
   This reference offers insights into best practices for securing web applications, including secure password storage and protecting user data, which are important considerations in implementing authentication features.
6. IntelliJ IDEA - IDE Features  
   JetBrains. (2024). *IntelliJ IDEA Features*. Retrieved from <https://www.jetbrains.com/idea/features/>  
   Describes the core features of IntelliJ IDEA, such as code completion, version control, and debugging tools, that were essential for efficiently developing and testing the Flappy Bird project.
7. MongoDB Atlas  
   MongoDB, Inc. (2024). *MongoDB Atlas*. Retrieved from <https://www.mongodb.com/cloud/atlas>  
   Information on MongoDB’s cloud service, which could be used in the future to host the game’s database remotely, ensuring better scalability, reliability, and cross-platform data synchronization.

**Appendices**

**Appendix A: Code Listings**

1. **FlappyBird.java (Game Logic and UI)**  
   This file contains the core game logic, including the movement of the bird, pipe generation, collision detection, and score management. It also handles user interaction and graphics rendering using Java's Swing and AWT libraries.
2. **Loginpage.java (Login and Registration UI)**  
   This file manages the user login and registration process, providing an intuitive interface for users to input credentials. It connects to the MongoDB database to authenticate users and handle registration
3. **mongoDB.java (Database Interaction)**  
   This class handles interactions with the MongoDB database, allowing the game to store and retrieve user data, including usernames, passwords, and high scores. It manages user registration, login validation, and score updates.

**Appendix B: Database Schema**

**MongoDB Collections**

* **Users Collection**
  + UserID (Primary Key): Unique identifier for each user.
  + Username: The user's chosen username.
  + Password: The user's encrypted password.
  + Score: The highest score achieved by the user.

**Appendix C: Gameplay Screenshots**

1. **Login Screen**  
   A screenshot of the login screen where users enter their credentials to start the game.
2. **Game Screen**  
   A screenshot of the main game interface displaying the bird, pipes, and the score in real-time.
3. **Game Over Screen**  
   A screenshot of the game-over screen showing the final score and the option to restart the game.

**Appendix D: Error Handling**

1. **Login Failures**  
   If the user enters an incorrect username or password, an error message is displayed:  
   *"Invalid username or password!"*
2. **Registration Failures**  
   If a username already exists in the database, the system displays an error:  
   *"Username already exists!"*
3. **Score Update Failures**  
   If there is a failure in updating the score due to a database connection issue, an error message is displayed:  
   *"Error updating score. Please try again later."*

**Appendix E: Development and Testing Process**

1. **Development Tools Used**
   * **IntelliJ IDEA**: The primary IDE used for writing and testing the Java code.
   * **MongoDB**: Used as the database for storing user data and scores.
   * **Java JDK**: Required to compile and run the Java application.
2. **Testing Strategy**
   * **Unit Testing**: Each function, such as score updates and collision detection, was tested individually to ensure correct behavior.
   * **Integration Testing**: The interaction between the game logic, database, and user interface was tested to ensure seamless functionality.
   * **User Acceptance Testing**: A small group of users tested the game for usability, confirming that the login and gameplay features worked as expected.

**Appendix F: Known Issues and Limitations**

1. **Performance Issues on Low-end Devices**  
   The game may experience performance issues on older devices with limited hardware resources, particularly during pipe generation or when multiple users are logged in simultaneously.
2. **Limited Customization Options**  
   Currently, users cannot customize the game’s appearance or gameplay settings beyond the basic interface. Future improvements could include adding avatar customization, different skins, or difficulty settings.
3. **No Cloud Syncing**  
   The database is currently hosted locally, meaning users cannot sync their data across multiple devices. This could be addressed in future versions by implementing cloud synchronization.