**FlappyBirdFX.java**

import javafx.animation.AnimationTimer;  
import javafx.application.Application;  
import javafx.geometry.Pos;  
import javafx.scene.Scene;  
import javafx.scene.canvas.Canvas;  
import javafx.scene.canvas.GraphicsContext;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.PasswordField;  
import javafx.scene.control.TextField;  
import javafx.scene.image.Image;  
import javafx.scene.input.KeyCode;  
import javafx.scene.layout.GridPane;  
import javafx.scene.layout.StackPane;  
import javafx.scene.paint.Color;  
import javafx.scene.text.Font;  
import javafx.stage.Stage;  
import java.io.\*;  
import java.util.ArrayList;  
import java.util.Random;  
  
public class FlappyBirdFX extends Application {  
  
 private static final int *BOARD\_WIDTH* = 1280;  
 private static final int *BOARD\_HEIGHT* = 640;  
  
 private Image backgroundImg;  
 private Image birdImg;  
 private Image topPipeImg;  
 private Image bottomPipeImg;  
 private Image backgroundStage2Img;  
 private Image backgroundStage3Img;  
  
 private double birdX = *BOARD\_WIDTH* / 8.0;  
 private double birdY = *BOARD\_HEIGHT* / 2.0;  
 private static final int *BIRD\_WIDTH* = 51;  
 private static final int *BIRD\_HEIGHT* = 36;  
  
 private double velocityY = 0;  
 private static final double *GRAVITY* = 0.4;  
 private static final int *PIPE\_WIDTH* = 64;  
 private static final int *PIPE\_HEIGHT* = 512;  
 private double velocityX = -2;  
  
 private boolean gameOver = false;  
 private double score = 0;  
 private double highScore = 0;  
 private int currentStage = 1;  
 private boolean inMenu = true;  
 private boolean loggedIn = false;  
 private char currentLetter = 'A'; // The current letter to match  
 private int gameMode = 1; // Default to 1: Spacebar Jump  
  
 private final ArrayList<Pipe> pipes = new ArrayList<>();  
 private final Random random = new Random();  
  
 private static final String *USER\_DATA\_FILE* = "userData.txt";  
 private String currentUsername;  
  
  
 @Override  
 public void start(Stage stage) {  
 Canvas canvas = new Canvas(*BOARD\_WIDTH*, *BOARD\_HEIGHT*);  
 GraphicsContext gc = canvas.getGraphicsContext2D();  
 loadImages();  
 setupInput(canvas);  
 generateRandomLetter(); // Generate the first letter  
 if (!loggedIn) {  
 showLoginPage(stage);  
 return;  
 }  
  
 AnimationTimer timer = new AnimationTimer() {  
 @Override  
 public void handle(long now) {  
 if (inMenu) {  
 drawMenu(gc);  
 } else if (!gameOver) {  
 update();  
 draw(gc);  
 } else {  
 drawGameOver(gc);  
 }  
 }  
 };  
  
 timer.start();  
  
 Scene scene = new Scene(new StackPane(canvas));  
 stage.setScene(scene);  
 stage.setTitle("Flappy Bird");  
 stage.show();  
 }  
  
 private boolean authenticateUser(String username, String password) {  
 try (BufferedReader reader = new BufferedReader(new FileReader(*USER\_DATA\_FILE*))) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 String[] parts = line.split(",");  
 if (parts[0].equals(username) && parts[1].equals(password)) {  
 highScore = Double.*parseDouble*(parts[2]);  
 return true;  
 }  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return false;  
 }  
  
 private boolean registerUser(String username, String password) {  
 try (BufferedReader reader = new BufferedReader(new FileReader(*USER\_DATA\_FILE*))) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 String[] parts = line.split(",");  
 if (parts[0].equals(username)) {  
 return false;  
 }  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 try (BufferedWriter writer = new BufferedWriter(new FileWriter(*USER\_DATA\_FILE*, true))) {  
 writer.write(username + "," + password + ",0\n");  
 return true;  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return false;  
 }  
  
 private void saveHighScore() {  
 File tempFile = new File("temp.txt");  
 File userDataFile = new File(*USER\_DATA\_FILE*);  
  
 try (BufferedReader reader = new BufferedReader(new FileReader(userDataFile));  
 BufferedWriter writer = new BufferedWriter(new FileWriter(tempFile))) {  
  
 String line;  
 while ((line = reader.readLine()) != null) {  
 String[] parts = line.split(",");  
 if (parts[0].equals(currentUsername)) {  
 // Update the high score for the current user  
 double updatedHighScore = Math.*max*(Double.*parseDouble*(parts[2]), score);  
 writer.write(currentUsername + "," + parts[1] + "," + updatedHighScore + "\n");  
 highScore = updatedHighScore; // Update local high score  
 } else {  
 writer.write(line + "\n");  
 }  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
  
 private void loadImages() {  
 backgroundImg = new Image("flappybirdbg.png");  
 birdImg = new Image("flappybird.png");  
 topPipeImg = new Image("toppipe.png");  
 bottomPipeImg = new Image("bottompipe.png");  
 backgroundStage2Img = new Image("flappybirdbg\_stage2.png");  
 backgroundStage3Img = new Image("flappybirdbg\_stage3.png");  
 }  
  
 private void setupInput(Canvas canvas) {  
 canvas.setFocusTraversable(true);  
 canvas.setOnKeyPressed(event -> {  
 if (inMenu) return; // Ignore key events in the menu  
  
 switch (gameMode) {  
 case 1: // Default Controls  
 if (event.getCode() == KeyCode.*SPACE*) {  
 if (gameOver) {  
 saveHighScore();  
 restartGame();  
 } else {  
 velocityY = -8; // Bird jumps  
 }  
 }  
 break;  
  
 case 2: // Letter Matching  
 if (event.getCode() == KeyCode.*SPACE* && gameOver) {  
 saveHighScore();  
 restartGame();  
 } else if (event.getText().equalsIgnoreCase(String.*valueOf*(currentLetter))) {  
 velocityY = -8; // Bird jumps on correct letter  
 generateRandomLetter(); // Refresh the letter  
 }  
 break;  
 default:  
 break;  
 }  
 });  
  
 canvas.setOnMouseClicked(event -> {  
 if (inMenu) {  
 handleMenuClick(event.getX(), event.getY());  
 } else if (gameOver) {  
 handleGameOverClick(event.getX(), event.getY());  
 } else if (gameMode == 3) {  
 velocityY = -8; // Bird jump  
 }  
 });  
 }  
  
  
  
 private void handleMenuClick(double clickX, double clickY) {  
 if (clickX >= *BOARD\_WIDTH* / 2.0 - 100 && clickX <= *BOARD\_WIDTH* / 2.0 + 100) {  
 if (clickY >= *BOARD\_HEIGHT* / 3.0 && clickY <= *BOARD\_HEIGHT* / 3.0 + 40) {  
 gameMode = 1; // Default Controls  
 inMenu = false; // Start the game  
 } else if (clickY >= *BOARD\_HEIGHT* / 3.0 + 100 && clickY <= *BOARD\_HEIGHT* / 3.0 + 140) {  
 gameMode = 2; // Letter Matching  
 inMenu = false; // Start the game  
 } else if (clickY >= *BOARD\_HEIGHT* / 3.0 + 160 && clickY <= *BOARD\_HEIGHT* / 3.0 + 200) {  
 gameMode = 3; // Mouse Only  
 inMenu = false; // Start the game  
 }  
 else if (clickY >= *BOARD\_HEIGHT* / 3.0 + 220 && clickY <= *BOARD\_HEIGHT* / 3.0 + 260) {  
 gameMode = 4; // Mouse Only  
 inMenu = false; // Start the game  
 }  
 }  
 }  
  
 private void handleGameOverClick(double clickX, double clickY) {  
 // Restart button bounds  
 if (clickX >= *BOARD\_WIDTH* / 2.0 - 50 && clickX <= *BOARD\_WIDTH* / 2.0 + 50) {  
 if (clickY >= *BOARD\_HEIGHT* / 2.0 && clickY <= *BOARD\_HEIGHT* / 2.0 + 40) {  
 restartGame(); // Restart the game  
 } else if (clickY >= *BOARD\_HEIGHT* / 2.0 + 60 && clickY <= *BOARD\_HEIGHT* / 2.0 + 100) {  
 inMenu = true; // Return to menu  
 restartGame();  
 }  
 }  
 }  
  
  
 private void update() {  
 velocityY += *GRAVITY*;  
 birdY += velocityY;  
  
 ArrayList<Pipe> toRemove = new ArrayList<>();  
 for (Pipe pipe : pipes) {  
 pipe.x += velocityX;  
  
 if (!pipe.passed && birdX > pipe.x + *PIPE\_WIDTH*) {  
 score += 0.5;  
 pipe.passed = true;  
 }  
  
 if (pipe.x + *PIPE\_WIDTH* < 0) {  
 toRemove.add(pipe);  
 }  
  
 if (checkCollision(pipe)) {  
 if (!gameOver) {  
 saveHighScore(); // Save high score on first collision  
 }  
 gameOver = true;  
 }  
 }  
  
 pipes.removeAll(toRemove);  
  
 if (birdY > *BOARD\_HEIGHT* || birdY < 0) {  
 if (!gameOver) {  
 saveHighScore(); // Save high score if bird hits bounds  
 }  
 gameOver = true;  
 }  
  
 if (pipes.isEmpty() || pipes.get(pipes.size() - 1).x < *BOARD\_WIDTH* - 350) {  
 placePipes();  
 }  
  
 updateStage();  
 }  
  
  
 private void updateStage() {  
 if (score > 20 && currentStage == 1) {  
 currentStage = 2;  
 velocityX = -3; // Increase difficulty  
 } else if (score > 50 && currentStage == 2) {  
 currentStage = 3;  
 velocityX = -4; // Further increase difficulty  
 }  
 }  
  
 private void draw(GraphicsContext gc) {  
 switch (currentStage) {  
 case 1:  
 gc.drawImage(backgroundImg, 0, 0, *BOARD\_WIDTH*, *BOARD\_HEIGHT*);  
 break;  
 case 2:  
 gc.drawImage(backgroundStage2Img, 0, 0, *BOARD\_WIDTH*, *BOARD\_HEIGHT*);  
 break;  
 case 3:  
 gc.drawImage(backgroundStage3Img, 0, 0, *BOARD\_WIDTH*, *BOARD\_HEIGHT*);  
 break;  
 }  
  
 gc.drawImage(birdImg, birdX, birdY, *BIRD\_WIDTH*, *BIRD\_HEIGHT*);  
  
 for (Pipe pipe : pipes) {  
 if (pipe.isTop) {  
 gc.drawImage(topPipeImg, pipe.x, pipe.y, *PIPE\_WIDTH*, *PIPE\_HEIGHT*);  
 } else {  
 gc.drawImage(bottomPipeImg, pipe.x, pipe.y, *PIPE\_WIDTH*, *PIPE\_HEIGHT*);  
 }  
 }  
  
 // Display score and stage  
 gc.setFill(Color.*WHITE*);  
 gc.fillText("Score: " + (int) score, 10, 20);  
 gc.fillText("Stage: " + currentStage, 10, 40);  
  
// Display the current letter to type  
 gc.setFont(javafx.scene.text.Font.*font*(24));  
 gc.setFill(Color.*YELLOW*);  
 gc.fillText("Press: " + currentLetter, *BOARD\_WIDTH* / 2.0 - 30, 50);  
  
 }  
 private void showLoginPage(Stage stage) {  
 GridPane grid = new GridPane();  
 grid.setAlignment(Pos.*CENTER*);  
 grid.setVgap(10);  
 grid.setHgap(10);  
  
 // Adding the background image  
 StackPane root = new StackPane();  
 root.setStyle("-fx-background-image: url('flappybirdbg.png'); -fx-background-size: cover;");  
  
 Label messageLabel = new Label();  
 messageLabel.setTextFill(Color.*WHITE*); // White text for contrast  
  
 TextField usernameField = new TextField();  
 usernameField.setPromptText("Username");  
 PasswordField passwordField = new PasswordField();  
 passwordField.setPromptText("Password");  
  
 Button loginButton = new Button("Login");  
 Button switchToSignupButton = new Button("Sign Up");  
  
 // Styling buttons  
 loginButton.setStyle("-fx-background-color: #4CAF50; -fx-text-fill: white;");  
 switchToSignupButton.setStyle("-fx-background-color: #2196F3; -fx-text-fill: white;");  
  
 grid.add(new Label("Login"), 0, 0, 2, 1);  
 grid.add(usernameField, 0, 1, 2, 1);  
 grid.add(passwordField, 0, 2, 2, 1);  
 grid.add(loginButton, 0, 3);  
 grid.add(switchToSignupButton, 1, 3);  
 grid.add(messageLabel, 0, 4, 2, 1);  
  
 root.getChildren().add(grid); // Add the grid to the root with background  
  
 loginButton.setOnAction(e -> {  
 String username = usernameField.getText();  
 String password = passwordField.getText();  
 messageLabel.setText("");  
  
 if (username.isEmpty() || password.isEmpty()) {  
 messageLabel.setText("Fields cannot be empty.");  
 return;  
 }  
  
 if (authenticateUser(username, password)) {  
 loggedIn = true;  
 currentUsername = username;  
 start(stage);  
 } else {  
 messageLabel.setText("Invalid credentials!");  
 }  
 });  
  
 switchToSignupButton.setOnAction(e -> showSignupPage(stage));  
  
 Scene scene = new Scene(root, *BOARD\_WIDTH*, *BOARD\_HEIGHT*);  
 stage.setScene(scene);  
 stage.setTitle("Login");  
 stage.show();  
 }  
  
 private void showSignupPage(Stage stage) {  
 GridPane grid = new GridPane();  
 grid.setAlignment(Pos.*CENTER*);  
 grid.setVgap(10);  
 grid.setHgap(10);  
  
 // Adding the background image  
 StackPane root = new StackPane();  
 root.setStyle("-fx-background-image: url('flappybirdbg.png'); -fx-background-size: cover;");  
  
 Label messageLabel = new Label();  
 messageLabel.setTextFill(Color.*WHITE*); // White text for contrast  
  
 TextField usernameField = new TextField();  
 usernameField.setPromptText("Username");  
 PasswordField passwordField = new PasswordField();  
 passwordField.setPromptText("Password");  
 PasswordField confirmPasswordField = new PasswordField();  
 confirmPasswordField.setPromptText("Confirm Password");  
  
 Button signupButton = new Button("Sign Up");  
 Button switchToLoginButton = new Button("Login");  
  
 // Styling buttons  
 signupButton.setStyle("-fx-background-color: #4CAF50; -fx-text-fill: white;");  
 switchToLoginButton.setStyle("-fx-background-color: #2196F3; -fx-text-fill: white;");  
  
 grid.add(new Label("Sign Up"), 0, 0, 2, 1);  
 grid.add(usernameField, 0, 1, 2, 1);  
 grid.add(passwordField, 0, 2, 2, 1);  
 grid.add(confirmPasswordField, 0, 3, 2, 1);  
 grid.add(signupButton, 0, 4);  
 grid.add(switchToLoginButton, 1, 4);  
 grid.add(messageLabel, 0, 5, 2, 1);  
  
 root.getChildren().add(grid); // Add the grid to the root with background  
  
 signupButton.setOnAction(e -> {  
 String username = usernameField.getText();  
 String password = passwordField.getText();  
 String confirmPassword = confirmPasswordField.getText();  
 messageLabel.setText("");  
  
 if (username.isEmpty() || password.isEmpty() || confirmPassword.isEmpty()) {  
 messageLabel.setText("Fields cannot be empty.");  
 return;  
 }  
  
 if (!password.equals(confirmPassword)) {  
 messageLabel.setText("Passwords do not match.");  
 return;  
 }  
  
 if (registerUser(username, password)) {  
 messageLabel.setText("User registered successfully! You can now log in.");  
 } else {  
 messageLabel.setText("User already exists.");  
 }  
 });  
  
 switchToLoginButton.setOnAction(e -> showLoginPage(stage));  
  
 Scene scene = new Scene(root, *BOARD\_WIDTH*, *BOARD\_HEIGHT*);  
 stage.setScene(scene);  
 stage.setTitle("Sign Up");  
 stage.show();  
 }  
  
 private void drawMenu(GraphicsContext gc) {  
 // Load the background image  
 Image backgroundImg = new Image("flappybirdbg.png");  
  
 // Draw the background image to cover the entire canvas  
 gc.drawImage(backgroundImg, 0, 0, *BOARD\_WIDTH*, *BOARD\_HEIGHT*);  
  
 // Title  
 gc.setFill(Color.*WHITE*);  
 gc.setFont(Font.*font*("Verdana", 32));  
 gc.fillText("Marvel Comicon", *BOARD\_WIDTH* / 2.0 - 140, *BOARD\_HEIGHT* / 6.0);  
  
 // Buttons  
 double buttonWidth = 200;  
 double buttonHeight = 50;  
 double buttonX = *BOARD\_WIDTH* / 2.0 - buttonWidth / 2;  
  
 // Button 1: Start  
 drawButton(gc, buttonX, *BOARD\_HEIGHT* / 3.0, buttonWidth, buttonHeight, "#4CAF50", "white", "Start");  
  
 // Button 2: Letter Matching  
 drawButton(gc, buttonX, *BOARD\_HEIGHT* / 3.0 + 80, buttonWidth, buttonHeight, "#FFC107", "black", "Letter Matching");  
  
 // Button 3: Mouse Only  
 drawButton(gc, buttonX, *BOARD\_HEIGHT* / 3.0 + 160, buttonWidth, buttonHeight, "#2196F3", "white", "Mouse Only");  
  
 //Advance Game  
 drawButton(gc, buttonX, *BOARD\_HEIGHT* / 3.0 + 240, buttonWidth, buttonHeight, "#1ECBE1", "gray", "Iron Passer");  
 }  
  
  
 private void drawButton(GraphicsContext gc, double x, double y, double width, double height, String bgColor, String textColor, String text) {  
 // Draw button background with rounded corners  
 gc.setFill(Color.*web*(bgColor));  
 gc.fillRoundRect(x, y, width, height, 20, 20);  
  
 // Draw button text  
 gc.setFill(Color.*web*(textColor));  
 gc.setFont(Font.*font*("Verdana", 18));  
 gc.fillText(text, x + width / 4.0, y + height / 1.7);  
 }  
  
  
  
 private void drawGameOver(GraphicsContext gc) {  
 gc.setFill(Color.*color*(0.1, 0.1, 0.1, 0.6));  
 gc.drawImage(backgroundImg, 0, 0, *BOARD\_WIDTH*, *BOARD\_HEIGHT*);  
 gc.fillRect(0, 0, *BOARD\_WIDTH*, *BOARD\_HEIGHT*);  
  
 gc.setFill(Color.*WHITE*);  
 gc.setFont(javafx.scene.text.Font.*font*(24));  
 gc.fillText("GAME OVER", *BOARD\_WIDTH* / 2.0 - 70, *BOARD\_HEIGHT* / 3.0);  
 gc.fillText("Score: " + (int) score, *BOARD\_WIDTH* / 2.0 - 50, *BOARD\_HEIGHT* / 2.5);  
 gc.fillText("High Score: " + (int) highScore, *BOARD\_WIDTH* / 2.0 - 70, *BOARD\_HEIGHT* / 2.2);  
  
 gc.setFill(Color.*GRAY*);  
 gc.fillRect(*BOARD\_WIDTH* / 2.0 - 50, *BOARD\_HEIGHT* / 2.0, 100, 40);  
 gc.fillRect(*BOARD\_WIDTH* / 2.0 - 50, *BOARD\_HEIGHT* / 2.0 + 60, 100, 40);  
 gc.setFill(Color.*WHITE*);  
 gc.fillText("RESTART", *BOARD\_WIDTH* / 2.0 - 50, *BOARD\_HEIGHT* / 2.0 + 25);  
 gc.fillText("MENU", *BOARD\_WIDTH* / 2.0 - 30, *BOARD\_HEIGHT* / 2.0 + 85);  
 }  
  
 private void placePipes() {  
 double gap = *BOARD\_HEIGHT* / 3.0;  
 double topPipeY = -*PIPE\_HEIGHT* / 4.0 - random.nextDouble() \* (*PIPE\_HEIGHT* / 2.0);  
  
 pipes.add(new Pipe(*BOARD\_WIDTH*, topPipeY, true));  
 pipes.add(new Pipe(*BOARD\_WIDTH*, topPipeY + *PIPE\_HEIGHT* + gap, false));  
 }  
  
 private boolean checkCollision(Pipe pipe) {  
 double birdRight = birdX + *BIRD\_WIDTH*;  
 double birdBottom = birdY + *BIRD\_HEIGHT*;  
 double pipeRight = pipe.x + *PIPE\_WIDTH*;  
 double pipeBottom = pipe.y + *PIPE\_HEIGHT*;  
  
 return birdX < pipeRight && birdRight > pipe.x && birdY < pipeBottom && birdBottom > pipe.y;  
 }  
  
 private void restartGame() {  
 birdY = *BOARD\_HEIGHT* / 2.0;  
 velocityY = 0;  
 pipes.clear();  
 score = 0; // Reset current score  
 gameOver = false;  
 currentStage = 1;  
 velocityX = -2;  
 }  
 // Generate a random letter from A-Z  
 private void generateRandomLetter() {  
 currentLetter = (char) ('A' + random.nextInt(26)); // Random letter A-Z  
 }  
 public class Pipe {  
 double x, y;  
 boolean passed;  
 boolean isTop;  
  
 Pipe(double x, double y, boolean isTop) {  
 this.x = x;  
 this.y = y;  
 this.isTop = isTop;  
 this.passed = false;  
 }  
 }  
  
  
 public static void main(String[] args) {  
 *launch*();  
 }  
}

**IronPasser.java**

**import javafx.animation.AnimationTimer;  
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.canvas.Canvas;  
import javafx.scene.canvas.GraphicsContext;  
import javafx.scene.image.Image;  
import javafx.scene.input.KeyCode;  
import javafx.scene.layout.StackPane;  
import javafx.stage.Stage;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Random;  
  
public class IronPasser extends Application {  
  
 private static final int *BOARD\_WIDTH* = 360;  
 private static final int *BOARD\_HEIGHT* = 640;  
  
 // Bird properties  
 private Image birdImg;  
 private double birdX = *BOARD\_WIDTH* / 8.0;  
 private double birdY = *BOARD\_HEIGHT* / 2.0;  
 private static final int *BIRD\_WIDTH* = 51;  
 private static final int *BIRD\_HEIGHT* = 36;  
  
 // Movement speed  
 private static final double *MOVE\_SPEED* = 20;  
  
 // Pipe properties  
 private Image leftPipeImg;  
 private Image rightPipeImg;  
 private static final int *PIPE\_HEIGHT* = 40;  
 private static final int *PIPE\_WIDTH* = 80;  
 private static final int *ROW\_SPACING* = 130;  
 private double pipeSpeed = 2;  
  
 private List<HorizontalPipeRow> pipes = new ArrayList<>();  
 private Random random = new Random();  
  
 // Score and level properties  
 private int score = 0;  
 private int highScore = 0;  
 private boolean gameOver = false;  
 private int level = 1;  
  
 // Background properties  
 private Image backgroundDayImg;  
 private Image backgroundNightImg;  
 private boolean isDay = true;  
  
 @Override  
 public void start(Stage stage) {  
 // Load images  
 birdImg = new Image("flappybird.png");  
 leftPipeImg = new Image("left.png");  
 rightPipeImg = new Image("right.png");  
 backgroundDayImg = new Image("flappybirdbg.png");  
 backgroundNightImg = new Image("flappybirdbg\_stage3.png");  
  
 // Create canvas and GraphicsContext  
 Canvas canvas = new Canvas(*BOARD\_WIDTH*, *BOARD\_HEIGHT*);  
 GraphicsContext gc = canvas.getGraphicsContext2D();  
  
 // Set up input actions for movement  
 setupInput(canvas);  
  
 // Add initial pipes  
 addInitialPipes();  
  
 // Animation loop  
 new AnimationTimer() {  
 @Override  
 public void handle(long now) {  
 if (!gameOver) {  
 updatePipes();  
 checkCollisions();  
 updateLevel();  
 draw(gc);  
 } else {  
 drawGameOver(gc);  
 }  
 }  
 }.start();  
  
 // Set up the scene and stage  
 StackPane root = new StackPane(canvas);  
 Scene scene = new Scene(root, *BOARD\_WIDTH*, *BOARD\_HEIGHT*);  
  
 stage.setScene(scene);  
 stage.setTitle("Enhanced Flappy Bird");  
 stage.show();  
 canvas.requestFocus();  
 }  
  
 private void setupInput(Canvas canvas) {  
 canvas.setOnKeyPressed(event -> {  
 if (event.getCode() == KeyCode.*UP*) birdY -= *MOVE\_SPEED*;  
 if (event.getCode() == KeyCode.*DOWN*) birdY += *MOVE\_SPEED*;  
 if (event.getCode() == KeyCode.*LEFT*) birdX -= *MOVE\_SPEED*;  
 if (event.getCode() == KeyCode.*RIGHT*) birdX += *MOVE\_SPEED*;  
 if (event.getCode() == KeyCode.*SPACE* && gameOver) resetGame();  
 constrainBirdWithinBounds();  
 });  
 }  
  
 private void addInitialPipes() {  
 for (int i = 0; i < 5; i++) {  
 addPipeRow(-i \* *ROW\_SPACING*);  
 }  
 }  
  
 private void addPipeRow(double y) {  
 int leftGapSize = random.nextInt(101) + 50;  
 int rightGapSize = random.nextInt(101) + 50;  
 pipes.add(new HorizontalPipeRow(y, leftGapSize, rightGapSize));  
 }  
  
 private void constrainBirdWithinBounds() {  
 if (birdX < 0) birdX = 0;  
 if (birdX > *BOARD\_WIDTH* - *BIRD\_WIDTH*) birdX = *BOARD\_WIDTH* - *BIRD\_WIDTH*;  
 if (birdY < 0) birdY = 0;  
 if (birdY > *BOARD\_HEIGHT* - *BIRD\_HEIGHT*) birdY = *BOARD\_HEIGHT* - *BIRD\_HEIGHT*;  
 }  
  
 private void updatePipes() {  
 List<HorizontalPipeRow> crossedPipes = new ArrayList<>();  
  
 // Iterate through pipes  
 for (HorizontalPipeRow pipe : new ArrayList<>(pipes)) {  
 pipe.setY(pipe.getY() + pipeSpeed);  
  
 if (pipe.getY() > *BOARD\_HEIGHT* / 2 && !pipe.isCrossed()) {  
 pipe.setCrossed(true);  
 score++;  
 if (score > highScore) highScore = score;  
 addPipeRow(-*ROW\_SPACING*);  
 }  
  
 if (pipe.getY() > *BOARD\_HEIGHT*) {  
 crossedPipes.add(pipe); // Collect pipes to be removed  
 }  
 }  
  
 // Remove crossed pipes  
 pipes.removeAll(crossedPipes);  
 }  
  
  
 private void checkCollisions() {  
 for (HorizontalPipeRow pipe : pipes) {  
 if (birdY < pipe.getY() + *PIPE\_HEIGHT* && birdY + *BIRD\_HEIGHT* > pipe.getY()) {  
 if (birdX < pipe.getLeftGapStartX() || birdX + *BIRD\_WIDTH* > pipe.getRightGapEndX()) {  
 gameOver = true;  
 }  
 }  
 }  
 }  
  
 private void updateLevel() {  
 if (score % 10 == 0 && score != 0) {  
 level = score / 10 + 1;  
 pipeSpeed += 0.5;  
 }  
 }  
  
 private void draw(GraphicsContext gc) {  
 // Draw background  
 isDay = (score / 10) % 2 == 0;  
 Image backgroundImg = isDay ? backgroundDayImg : backgroundNightImg;  
 gc.drawImage(backgroundImg, 0, 0, *BOARD\_WIDTH*, *BOARD\_HEIGHT*);  
  
 // Draw bird  
 gc.drawImage(birdImg, birdX, birdY, *BIRD\_WIDTH*, *BIRD\_HEIGHT*);  
  
 // Draw pipes  
 for (HorizontalPipeRow pipe : pipes) {  
 gc.drawImage(leftPipeImg, 0, pipe.getY(), pipe.getLeftGapStartX(), *PIPE\_HEIGHT*);  
 gc.drawImage(rightPipeImg, pipe.getRightGapEndX(), pipe.getY(), *BOARD\_WIDTH* - pipe.getRightGapEndX(), *PIPE\_HEIGHT*);  
 }  
  
 // Draw score and level  
 gc.fillText("Score: " + score, 10, 20);  
 gc.fillText("High Score: " + highScore, 10, 40);  
 gc.fillText("Level: " + level, 10, 60);  
 }  
  
 private void drawGameOver(GraphicsContext gc) {  
 gc.fillText("Game Over! Press SPACE to Restart", *BOARD\_WIDTH* / 4.0, *BOARD\_HEIGHT* / 2.0);  
 gc.fillText("Final Score: " + score, *BOARD\_WIDTH* / 4.0, *BOARD\_HEIGHT* / 2.0 + 20);  
 }  
  
 private void resetGame() {  
 score = 0;  
 gameOver = false;  
 birdX = *BOARD\_WIDTH* / 8.0;  
 birdY = *BOARD\_HEIGHT* / 2.0;  
 pipeSpeed = 2;  
 pipes.clear();  
 addInitialPipes();  
 }  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
  
 private static class HorizontalPipeRow {  
 private double y;  
 private double leftGapSize;  
 private double rightGapSize;  
 private boolean crossed;  
  
 public HorizontalPipeRow(double y, double leftGapSize, double rightGapSize) {  
 this.y = y;  
 this.leftGapSize = leftGapSize;  
 this.rightGapSize = rightGapSize;  
 this.crossed = false;  
 }  
  
 public double getY() {  
 return y;  
 }  
  
 public void setY(double y) {  
 this.y = y;  
 }  
  
 public double getLeftGapStartX() {  
 return leftGapSize;  
 }  
  
 public double getRightGapEndX() {  
 return *BOARD\_WIDTH* - rightGapSize;  
 }  
  
 public boolean isCrossed() {  
 return crossed;  
 }  
  
 public void setCrossed(boolean crossed) {  
 this.crossed = crossed;  
 }  
 }  
}**