

# Flappy Bird

By

# Md Rohed

# ID- 163-35-163

**&**

**Ziad Hossain**

**ID-163-35-160**

# Department of Software Engineering

# DAFFODIL INTERNATIONAL UNIVERSITY

**Material Has Been Used**

Language – JAVA

Source Code Editor –Eclipse,Netbeans

UI Design-Adobe XD

**Abstract of the Project**

Flappy bird is a game in which a player controls a bird’s flight height to avoid obstacles. Pressing on the button for a longer period of time allows the bird to fly higher, while letting go causes the bird to fly lower.

**SRS of the Project**

**Objective**

This game aims to change the way people think of traditional flappy bird game.

The major objectives of this project are:

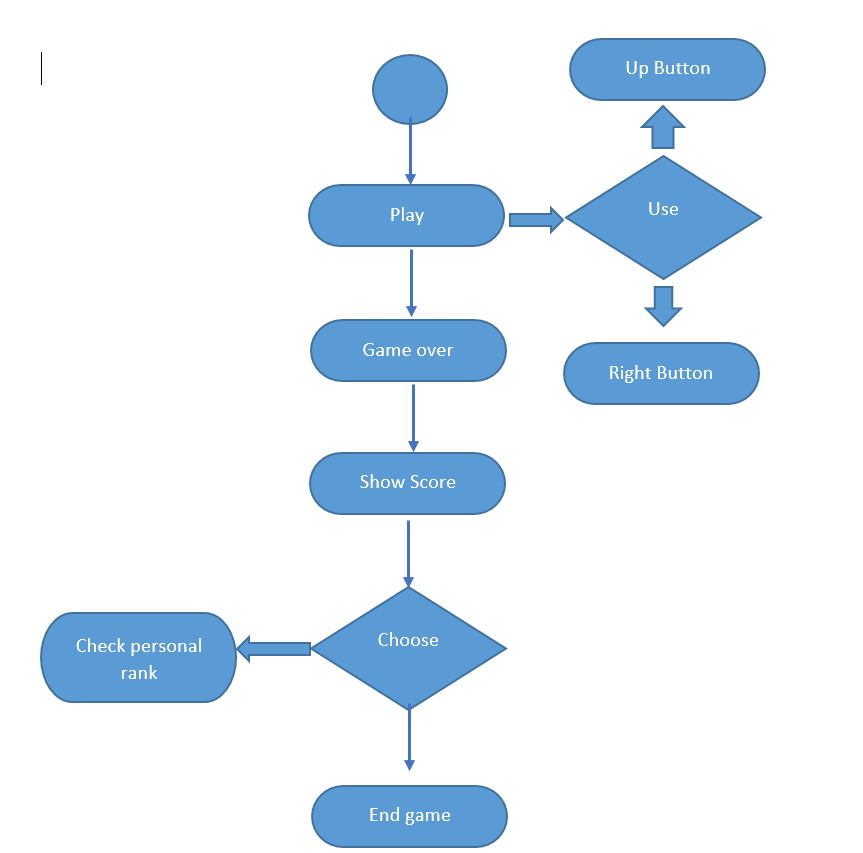
**Scope**

This document describes the software requirements for the flappy bird program. This document will be used by the users, tester, and developers of the game.

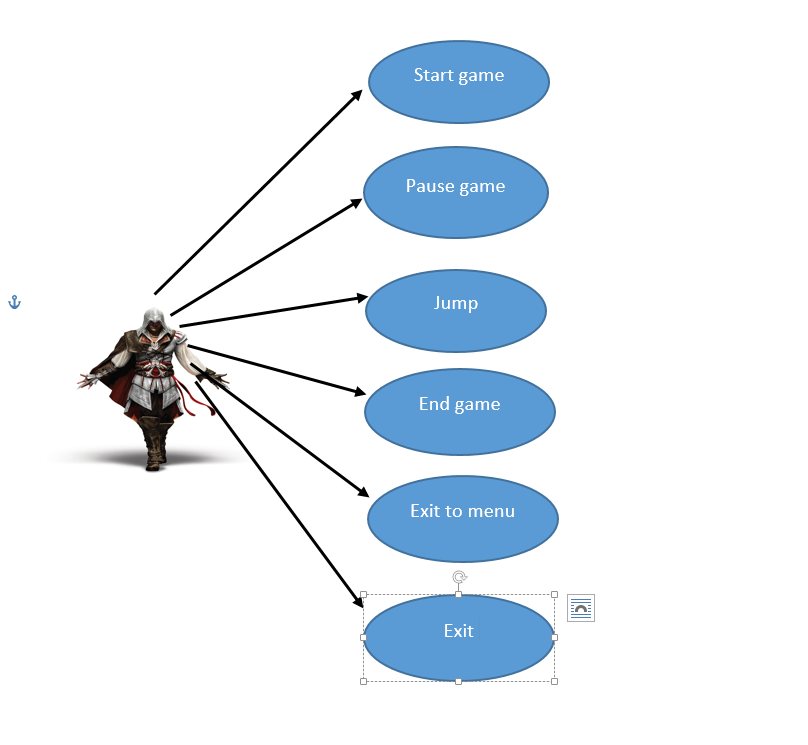
**Overview of Functional Requirements**

|  |  |  |
| --- | --- | --- |
| Requirement id | Statement | Priority |
| FR01 | The user can move a bird by clicking or pressing mouse and keyboard for avoiding obstacles | High |
| FR02 | When the player able to pass the every single obstacles user will receive points | HIGH |
| FR03 | When the player fail to avoid obstacle he loos the match | HIGH |

**Activity Diagram**

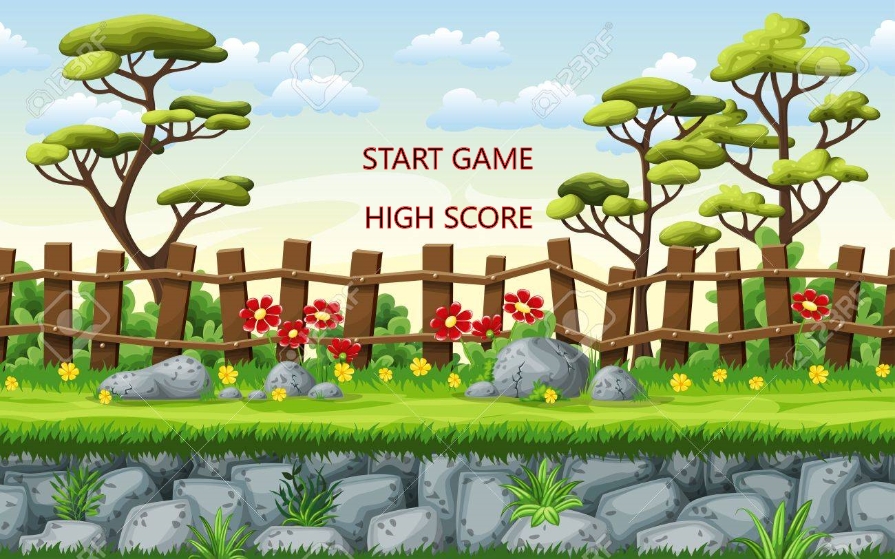


**Use Case Diagram**

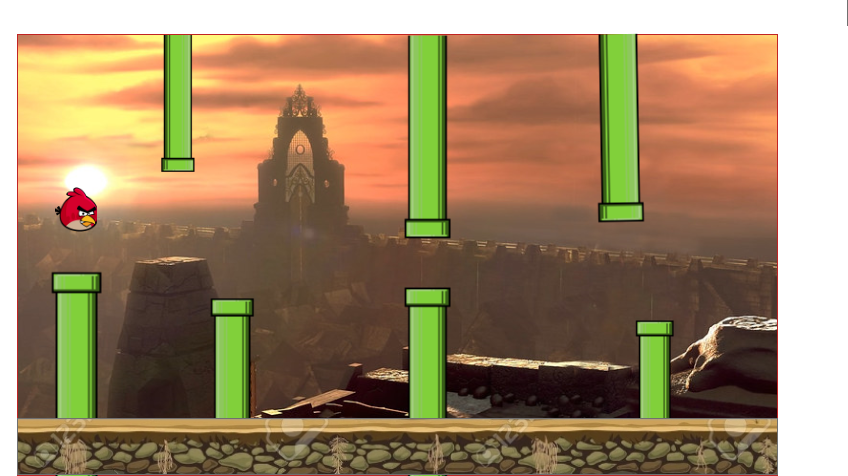


**UI Design**

**Start page**



**Main page**



**Source Code**

**App.java**

import javax.swing.\*;

public class App {

public static int *WIDTH* = 500;

public static int *HEIGHT* = 500;

public static void main(String[] args) {

JFrame frame = new JFrame();

frame.setVisible(true);

frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

frame.setSize(*WIDTH*, *HEIGHT*);

frame.setLocationRelativeTo(null);

Keyboard keyboard = Keyboard.*getInstance*();

frame.addKeyListener(keyboard);

GamePanel panel = new GamePanel();

frame.add(panel);

}

}

**Bird.java**

import java.awt.Image;

import java.awt.event.KeyEvent;

import java.awt.geom.AffineTransform;

public class Bird {

public int x;

public int y;

public int width;

public int height;

public boolean dead;

public double yvel;

public double gravity;

private int jumpDelay;

private double rotation;

private Image image;

private Keyboard keyboard;

public Bird() {

x = 100;

y = 150;

yvel = 0;

width = 45;

height = 32;

gravity = 0.5;

jumpDelay = 0;

rotation = 0.0;

dead = false;

keyboard = Keyboard.getInstance();

}

public void update() {

yvel += gravity;

if (jumpDelay > 0)

jumpDelay--;

if (!dead && keyboard.isDown(KeyEvent.VK\_SPACE) && jumpDelay <= 0) {

yvel = -10;

jumpDelay = 10;

}

y += (int)yvel;

}

public Render getRender() {

Render r = new Render();

r.x = x;

r.y = y;

if (image == null) {

image = Util.loadImage("lib/bird2.png");

}

r.image = image;

rotation = (90 \* (yvel + 20) / 20) - 90;

rotation = rotation \* Math.PI / 180;

if (rotation > Math.PI / 2)

rotation = Math.PI / 2;

r.transform = new AffineTransform();

r.transform.translate(x + width / 2, y + height / 2);

r.transform.rotate(rotation);

r.transform.translate(-width / 2, -height / 2);

return r;

}

}

**GamePanel.java**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.awt.geom.\*;

public class GamePanel extends JPanel implements Runnable {

private Game game;

public GamePanel() {

game = new Game();

new Thread(this).start();

}

public void update() {

game.update();

repaint();

}

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2D = (Graphics2D) g;

for (Render r : game.getRenders())

if (r.transform != null)

g2D.drawImage(r.image, r.transform, null);

else

g.drawImage(r.image, r.x, r.y, null);

g2D.setColor(Color.BLACK);

if (!game.started) {

g2D.setFont(new Font("TimesRoman", Font.PLAIN, 20));

g2D.drawString("Press SPACE to start", 150, 240);

} else {

g2D.setFont(new Font("TimesRoman", Font.PLAIN, 24));

g2D.drawString(Integer.toString(game.score), 10, 465);

}

if (game.gameover) {

g2D.setFont(new Font("TimesRoman", Font.PLAIN, 20));

g2D.drawString("Press R to restart", 150, 240);

}

}

public void run() {

try {

while (true) {

update();

Thread.sleep(25);

}

}

catch (Exception e) {

e.printStackTrace();

}

}

}

**Keyboard.java**

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

public class Keyboard implements KeyListener {

private static Keyboard instance;

private boolean[] keys;

private Keyboard() {

keys = new boolean[256];

}

public static Keyboard getInstance() {

if (instance == null) {

instance = new Keyboard();

}

return instance;

}

public void keyPressed(KeyEvent e) {

if (e.getKeyCode() >= 0 ) {

keys[e.getKeyCode()] = true;

}

}

public void keyReleased(KeyEvent e) {

if (e.getKeyCode() >= 0 && e.getKeyCode() < keys.length) {

keys[e.getKeyCode()] = false;

}

}

public void keyTyped(KeyEvent e) {}

public boolean isDown(int key) {

if (key >= 0 && key < keys.length) {

return keys[key];

}

return false;

}

}

**Pipe.java**

import java.awt.Image;

public class Pipe {

public int x;

public int y;

public int width;

public int height;

public int speed = 3;

public String orientation;

private Image image;

public Pipe(String orientation) {

this.orientation = orientation;

reset();

}

public void reset() {

width = 66;

height = 400;//400

x = App.*WIDTH* + 2;

if (orientation.equals("south")) {

y = -(int)(Math.*random*() \* 120) - height / 2;

}

}

public void update() {

x -= speed;

}

public boolean collides(int \_x, int \_y, int \_width, int \_height) {

int margin = 2;

if (\_x + \_width - margin > x && \_x + margin < x + width) {

if (orientation.equals("south") && \_y < y + height) {

return true;

} else if (orientation.equals("north") && \_y + \_height > y) {

return true;

}

}

return false;

}

public Render getRender() {

Render r = new Render();

r.x = x;

r.y = y;

if (image == null) {

image = Util.*loadImage*("lib/pipe-" + orientation + ".png");

}

r.image = image;

return r;

}

}

**Render.java**

import java.awt.Image;

import java.awt.geom.AffineTransform;

public class Render {

public int x;

public int y;

public Image image;

public AffineTransform transform;

public Render() {

}

public Render(int x, int y, String imagePath) {

this.x = x;

this.y = y;

this.image = Util.loadImage(imagePath);

}

}

**Util.java**

import java.awt.Image;

import javax.imageio.ImageIO;

import java.io.File;

import java.io.IOException;

import java.util.HashMap;

public class Util {

private static HashMap<String, Image> cache = new HashMap<String, Image>();

public static Image loadImage(String path) {

Image image = null;

if (cache.containsKey(path)) {

return cache.get(path);

}

try {

image = ImageIO.read(new File(path));

if (!cache.containsKey(path)) {

cache.put(path, image);

}

}

catch (IOException e) {

e.printStackTrace();

}

return image;

}

}