Flappy Bird 2.0

introduction

**The Flappy Bird 2.0 is a game that has very simple mechanics. Simply tap or click to make the bird fly and avoid obstacles in the form of pipes that appear from the top and bottom of the screen. The main purpose of the game is to avoid these pipes for as long as possible to obtain a highscore, the game does not have an end goal in terms of “winning”. Players will aim to beat their own highscore or other players/ friends.**

project team roles

Okoyama Tantsi – Documentation

Lutho Sigodi – Graphical interface design

Abona Ntsuluba – Back ends

Tshephang Makgaloa – Intergrating the game

Software Development Life Cycle

requirements specification

Functional requirements: This game is required to have mechanics that make the game functional. Player control which allows the bird to move upwards and stay in the air by tapping the screen or pressing a key. Obstacle generation is also a requirement, this game should randomly generate a series of vertical pipes with the gaps that the bird has to navigate through. The game should detect when the bird hits one of the pipes or hits the ground. The game should detect when the bird hits a pipe or the ground. Score calculation is also a requirement of the game, the players score should increase by 1 each time the bird passes through a gap successfully. The game should also display game over on the screen when a bird hits an obstacle as well as a restart option to start the game over.

Non-Functional requirements: The game should respond instantly to the player pressing the key. Smooth animation is another requirement, the game should keep up and maintain a consistent frame rate. The usability of the game should be simple and reliable. The game should also increase in difficulty, in this game specifically the birds speed should increase as the player gains more points.

Technical requirements: the game should be able to run on mobile devices.

Acceptance criteria:

* Bird animation
* Pipes and background animation
* Bird control
* Obstacle generation
* Collision detection
* Scoring system
* Game over display

design

Risks

* The game might be slow and lag behind, inconveniencing the player.
* The game may not work properly across different devices or across various operating systems.
* There might be a lag in response to pressing the key that will lead to poor gaming experiences.

Technologies

* Programming language: Java
* Java Swing
* Java.awt
* ImageIcon
* Timers
* keyListener and ActionListener

Data design

Objects: The bird, pipes, and medals (bronze, silver, gold, platinum) are the key game objects, each having attributes such as position, size, and associated images.

Models:

* Bird class: Holds bird's position (X and Y coordinates) and size (width, height).
* Pipe class: Tracks each pipe's position and status (whether the player has passed it).
* Medals: Objects like Gold, Silver, Bronze, and Platinum have their positions, sizes, and associated images.

Interface representation

Storyboards:

* Start Screen: A button that says "PLAY" in the middle of the screen.
* Game Screen: Background image that consists of a sky and pipes), the bird flapping , and score at the top.
* End Screen: A display showing the final score and medals (bronze, silver, gold, platinum) based on performance.

Class diagrams

Background

- height: int

- width: int

- pipes: ArrayList<Pipe>

- bird: Bird

- bronze: Bronze

- silver: Silver

- gold: Gold

- platinum: Platinum

- score: double

- gameOver: boolean

+ Background()

+ pipeimport(): void

+ paintComponent(Graphics): void

+ acc(): void

+ Intersection(Bird, Pipe): bool

+ draw(Graphics): void

Bird Pipe

- X: int - Xp: int

- Y: int - Yp: int

- bwidth: int - Pwidth: int

- bheight: int - Pheight: int

- bmge: Image - succ: boolean

+ Bird(bmge: Image) + Pipe(pimg: Image)

Bronze Silver

- Xb: int - Xs: int

- Yb: int - Ys: int

- bswidth: int - swidth: int

- bsheight: int - sheight: int

- zmge: Image - smge: Image

+ Bronze(zmge: Image) + Silver(smge: Image)

Platinum

- Xm: int

- Ym: int

- mwidth: int

- mheight: int

- pmge: Image

+ Platinum(pmge: Image)

Gold

- Xg: int

- Yg: int

- gwidth: int

- gheight: int

- gmge: Image

+ Gold(gmge: Image)

App

- JFrame frame1

- JButton button

- JPanel pan

- JLabel lab

- Container con

- Font tite

+ App()

+ main(String[] args)

+ Gamescreen()

Inner Class

Title

+ actionPerformed(ActionEvent event) |

Implementation

A computer screen with many colorful text

Description automatically generated

Calling/importing the images

A screen shot of a computer code

Description automatically generated

The while loop to get the pipes, in order to import them to the game.

First if statement to check if the bird passed the pipes.

Detects when the bird hits a pipe or top and ground level.

Last IF statement is to check if the bird falls down.

A computer screen with many colorful text

Description automatically generated with medium confidence

Calculating the pipes

Importing them randomly

Calculating the space between them.

Creating objects for each, so that you can access them indirectly.

Then adding them to a list of pipes.

A screen shot of a computer program

Description automatically generated

Here we are using ActionListener in order to get the keys pressed.

e.g. Paused, Restarted, Resume, Play.

RUNNING THE APPLICATION:

1.Open visual studio code and download the extensions required for java programming.

2.Open the folder.

3.Navigate to “Background” then “src”.

4.Open the App.java Class and run it.

5.Then PLAY.

# References

Heilmann, T. A. (2014). “Tap, tap, flap, flap.” Ludic Seriality, Digitality, and the Finger.*Eludamos: Journal for Computer Game Culture, 8*(1), 33–46.

Sahin, A., Atici, E., & Kumbasar, T. (2016). Type-2 fuzzified flappy bird control system. Paper presented at the *2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE),*1578–1583.