Flappy Bird Game

Basic of Programming 3

Final Project

Documentation

For Java Project

MŰEGYETEM 1782

Name: Mean Diamand

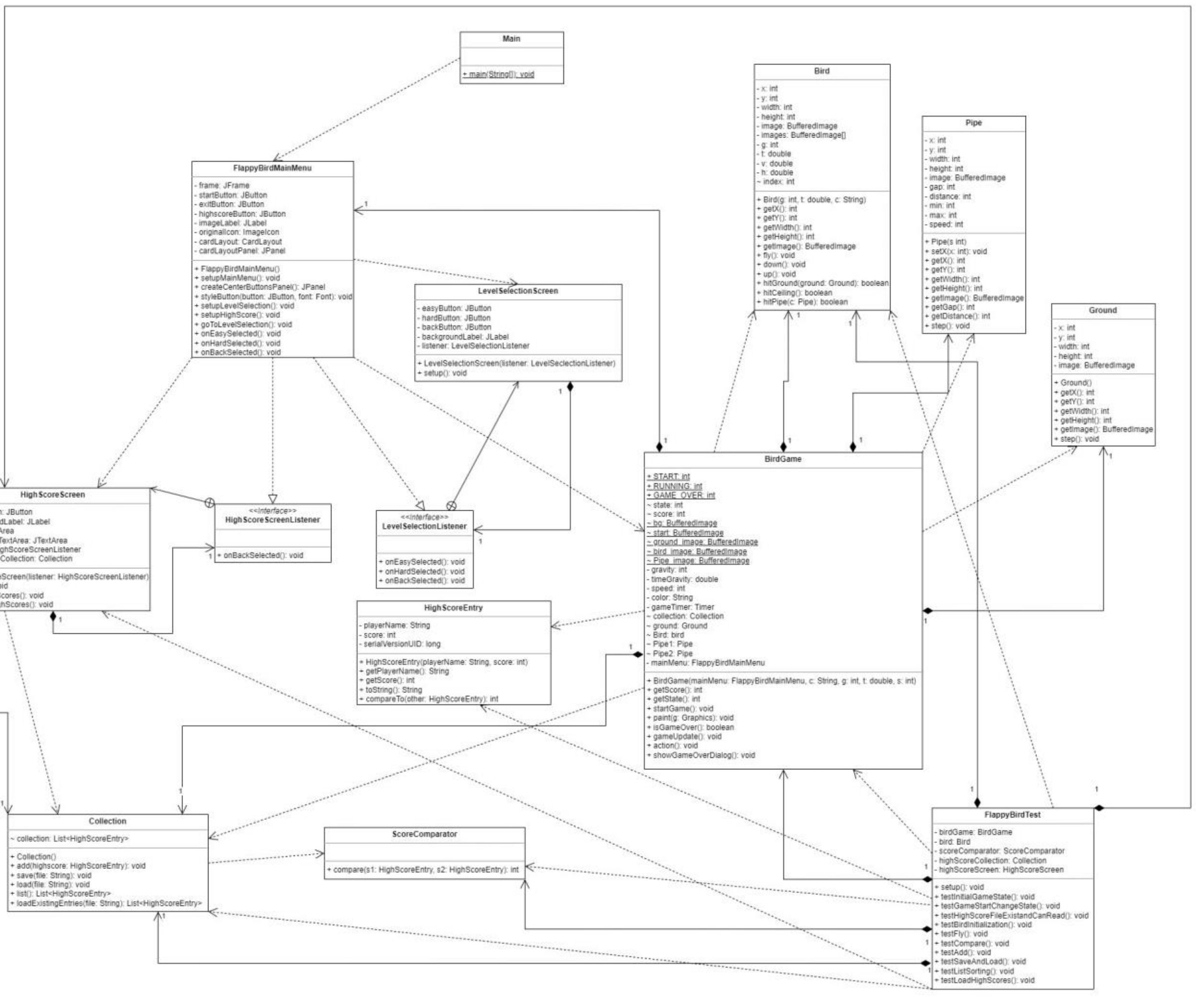
Neptun Code: KSG25Z

Name: Mean Diamand

Basic of Programming 3

Neptun Code: KSG25Z

UML Class Diagram



My project consists of 4 packages and within those packages, it contained 12 classes.

1. <u>Screen package:</u>
    a. <u>Main class</u>
        i. This Main class serves as the entry point for the Flappy Bird game.
        ii. It initialized the graphical user interface by invoking the FlappyBirdMainMenu class using the SwingUtilities.invokeLater() method which help to ensure the GUI components are created and updated.
        iii. The game starts by running this Main class, which in turn initializes the main menu for the game which allow the player to navigate through the different options that will be describe in the next class.
        iv. To run the Flappy Bird game, you need to execute the main method in this class.

b. <u>FlappyBirdMainMenu class</u>

   i. This FlappyBirdMainMenu class represents the main menu screen of the game.

   ii. It provides options for starting the game, checking the saved high scores of the previous player, and exiting the game.

   iii. This class uses a CardLayout to switch between different panels, such as main menu screen, level selection screen, and high score screen.

   iv. The main menu features a background image with transparent buttons for a visually design. There are "Start Game", "High Score", and "Exit" buttons for the players to navigate between panel by clicking it to trigger the action of those button according to the name of it.

   v. This class implements interfaces for LevelSelectionScreen and HighScoreScreen to handle user interactions related to level selection screen and high score screen.

   vi. This class can be instantiated to launch main menu of the game and interact with different components such as starting the game and checking the high score.

c. <u>LevelSelectionScreen class</u>

   i. This LevelSelectionScreen class represents the screen where the player can choose between different level of the game.

   ii. It provides buttons for selecting "Easy Mode", "Hard Mode", and "Back" for returning to the main menu.

   iii. This class extends JPanel and includes an inner interface called LevelSelectionListener which handle button click event.

   iv. This class can be instantiated and provide a LevelSelectionListener to handle button clicks.

2. <u>HighScore package:</u>

   a. <u>HighScoreScreen class</u>

      i. This HighScoreScreen class represents the screen where the player can check the saved scores of the game.

      ii. It provides a list of high score data including the player's username and their scores, and a "Back" button for returning to the main menu.

      iii. This class extends JPanel and includes an inner interface called HighScoreScreenListener which handle the back button click event.

      iv. This class can be instantiated and provide a HighScoreScreenListener to handle back button clicks.

   b. <u>HighScoreEntry class</u>

i. This HighScoreEntry class represents a single entry in a high score list.

ii. Each entry includes the player's name and their corresponding score that they obtained in the game.

iii. This class implements the Serializable interface to support serialization for saving and loading high score data.

c. <u>Collection class</u>

i. This Collection class represents a collection of high score entries for the game.

ii. It provides methods to add new high score entries, save the collection to a file, load the collection from a file, and list the high score entries in a descending order with the help of the ScoreComparator class which will be describe in the next class.

iii. This class can be instantiated for managing high score entries. Add new entries using the add method and save/load the collection to/from a file using the save and load methods. Retrieve the high score entries in descending order using the list method.

d. <u>ScoreComparator class</u>

i. This ScoreComparator class implements the Comparator interface for comparing HighScoreEntry objects based on their scores in descending order.

ii. This class is typically used to arrange high score entries from the highest to the lowest score.

3. <u>Game package:</u>

a. <u>Bird class</u>

i. This Bird class represents the bird character in the Flappy Bird game.

ii. It includes properties such as position, dimensions, images for animation, and methods to simulate the bird's flight, gravity, and interactions with the game environment.

iii. This class also provides methods to check if the bird hits the ground, ceiling, or pipes.

b. <u>Ground class</u>

i. This Ground class represents the ground in the game.

ii. It is responsible for providing information about the ground's position, dimensions, and image.

iii. The ground moves continuously to give the illusion of the scrolling in the game.

iv. This class can be instantiated to create the ground in the game and the provided methods can be used for retrieving information of the ground.

c. <u>Pipe class</u>

i. This Pipe class represents the pipes in the game.

ii. Pipes are obstacles that the bird must pass through.

iii. This class provides information about the pipe's position, dimensions, image, gap size, and movement speed that the pipe is moving on the screen.

iv. This class can be instantiated to create pipes in the game and the provided methods can be used for retrieving information about the pipe.

d. BirdGame class

i. This BirdGame class represents the main game logic for the game.

ii. It manages the game state, handles user input, updates the game elements, and displays the game graphics.

iii. This class includes methods that start the game, check for game over conditions, and interact with the player.

iv. It also integrates with a high score collection and provides a game over dialog for players to input their username to save the high score to the high score screen.

v. This class can be instantiated to create the Flappy Bird game and the provided methods can be used to start and control the game.

4. Test package

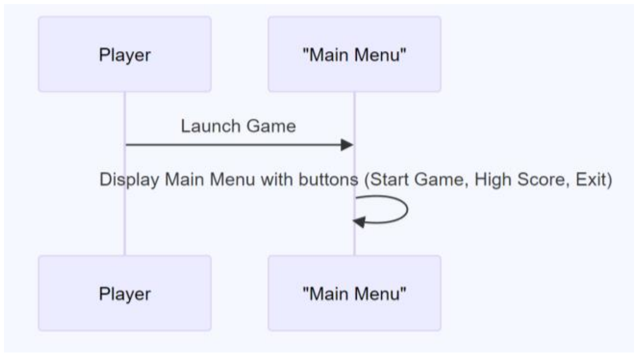
a. FlappyBirdTest class

i. This FlappyBirdTest class contains JUnit tests for various components of the game.

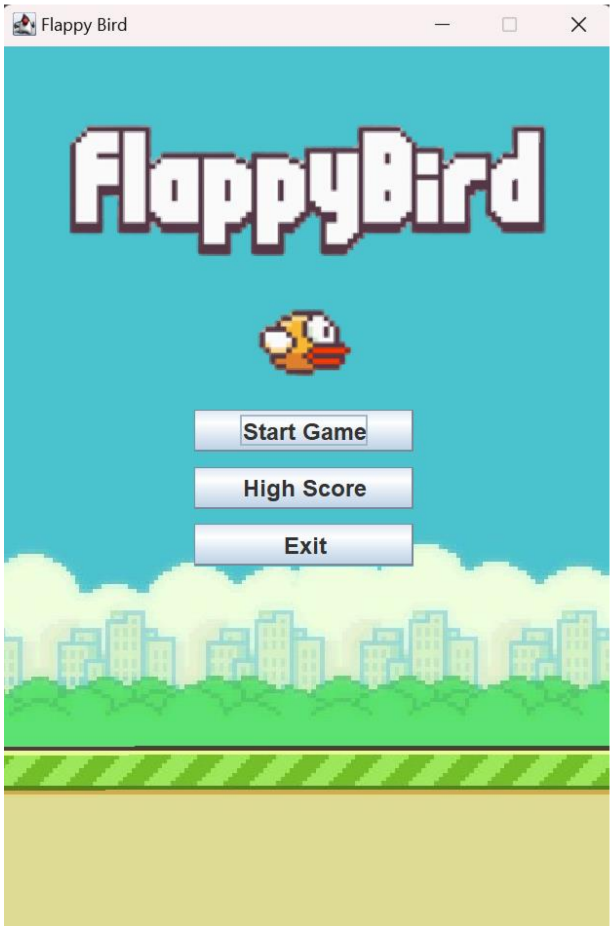
ii. These tests covered the game state, bird behavior, scoring system, high score collection, and related functionalities.

Sequential Diagram

1. Lauch Game and Main Menu

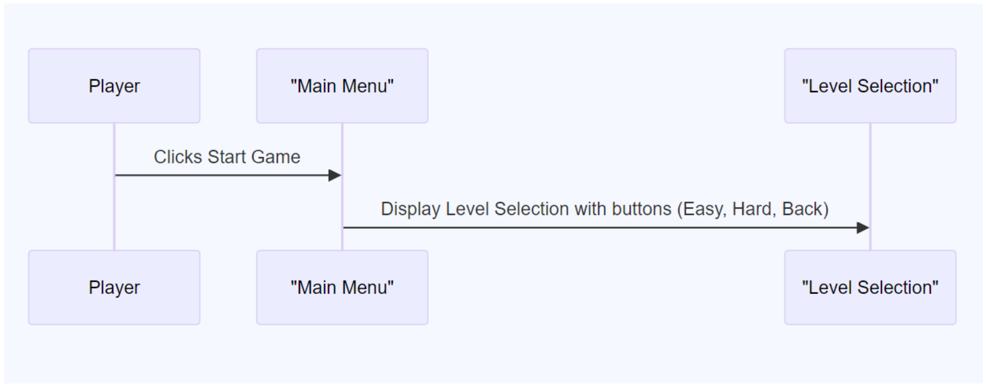

Player
"Main Menu"
Launch Game
Display Main Menu with buttons (Start Game, High Score, Exit)
Player
"Main Menu"

In User Interface

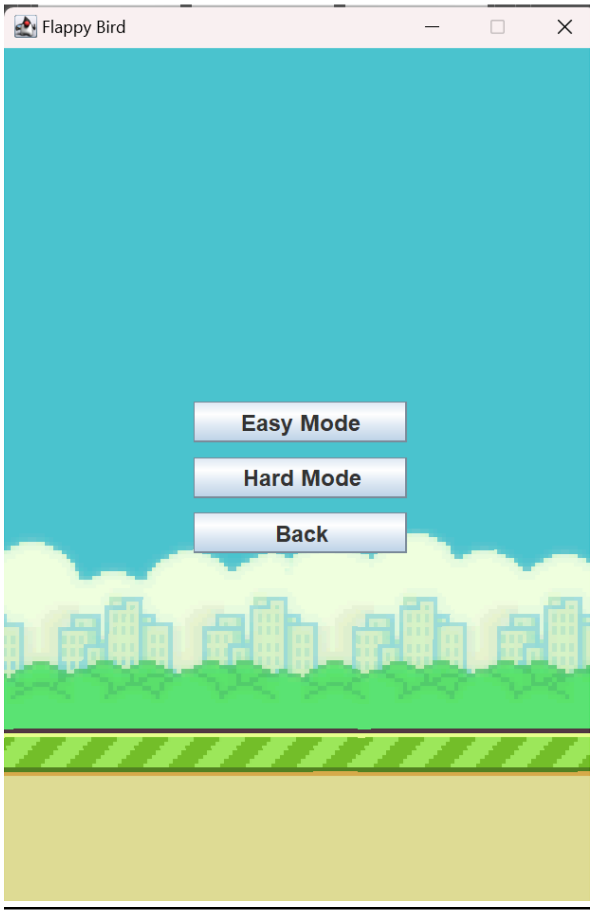


This is what the window panel shows when the player launch the game.

2. Start Game from Main Menu

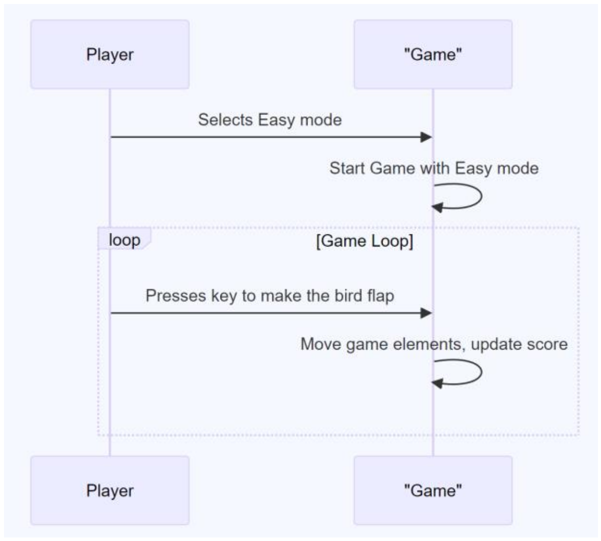


<u>In User Interface</u>

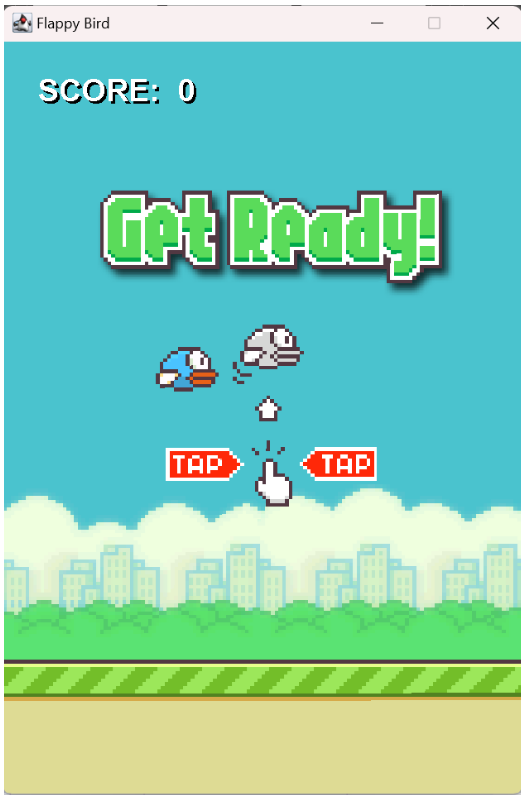


After selecting the Start Game button then it will display the Level Selection Screen.

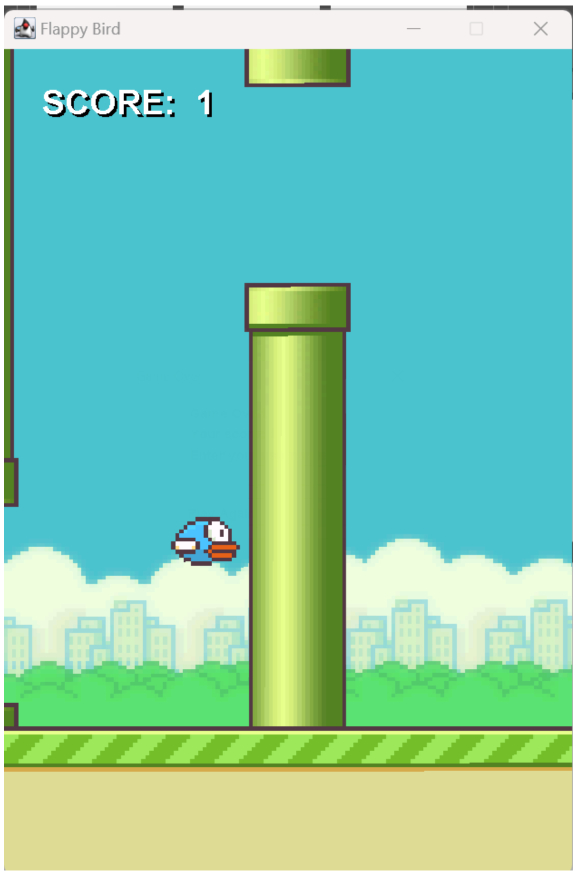
3. <u>Play the Game</u>


Player
"Game"
Selects Easy mode
Start Game with Easy mode
loop
[Game Loop]
Presses key to make the bird flap
Move game elements, update score
Player
"Game"

In User Interface

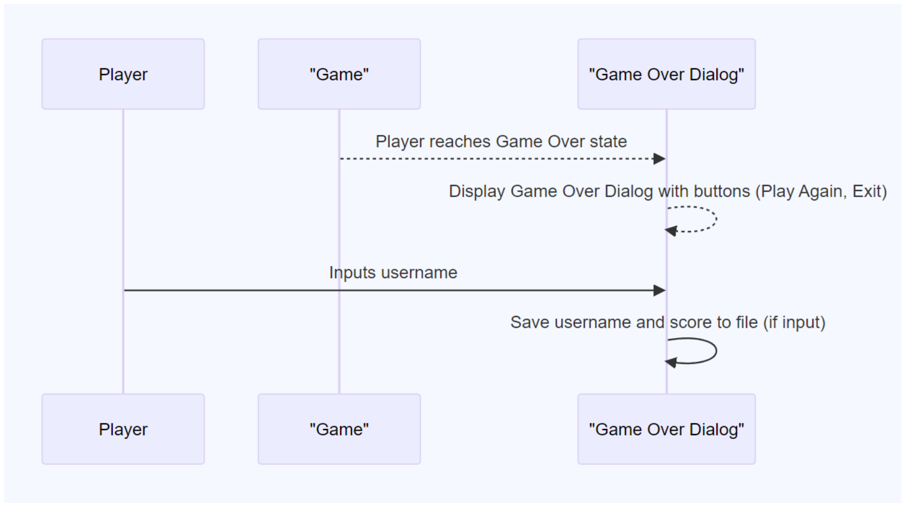


After clicking the Easy mode button, it will display the Game screen.

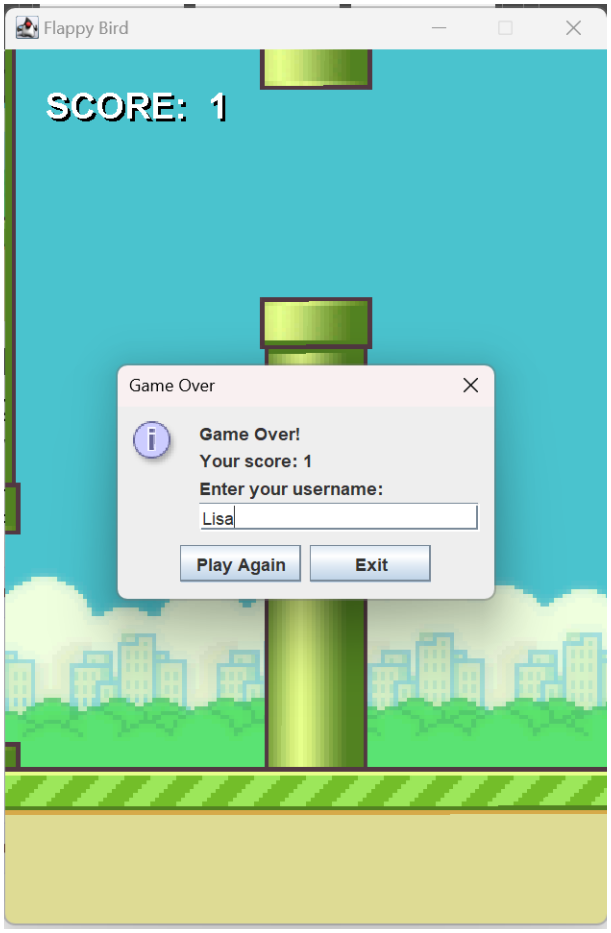


Then when the player press spacebar key, the bird will start moving and flapping the wing and every passes that the bird make through the each pipe, one point will be added to the score.

4. Game Over and Inputting the username of the player


Player
"Game"
"Game Over Dialog"
Player reaches Game Over state
Display Game Over Dialog with buttons (Play Again, Exit)
Inputs username
Save username and score to file (if input)
Player
"Game"
"Game Over Dialog"

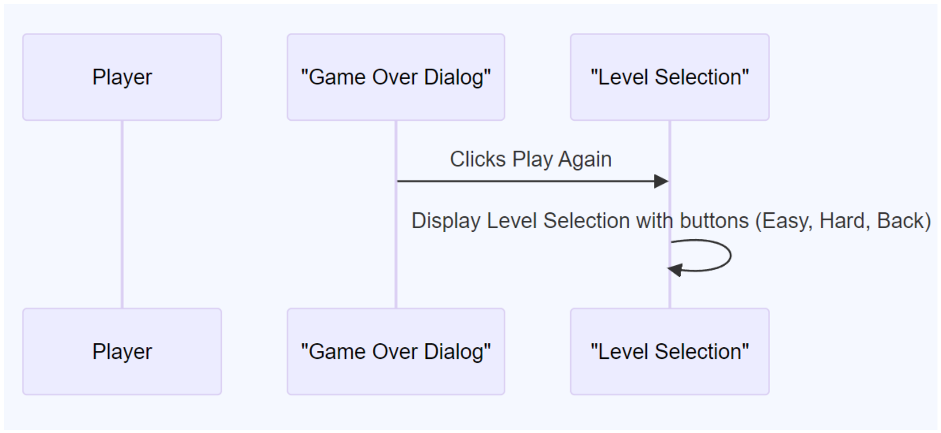
In User Interface



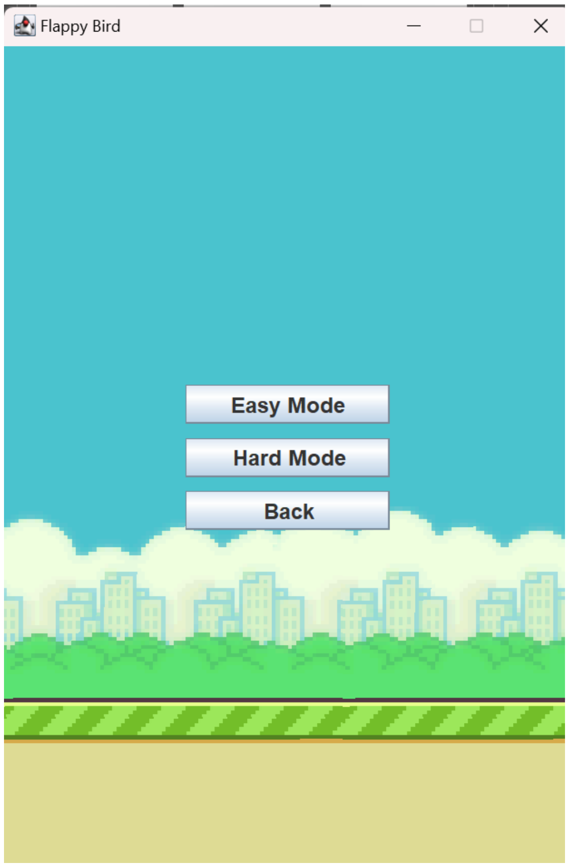
Game Over!
Your score: 1
Enter your username:
Lisa

The player can save their score by inputting their username.

5. Play Again or Exit after the Game Over

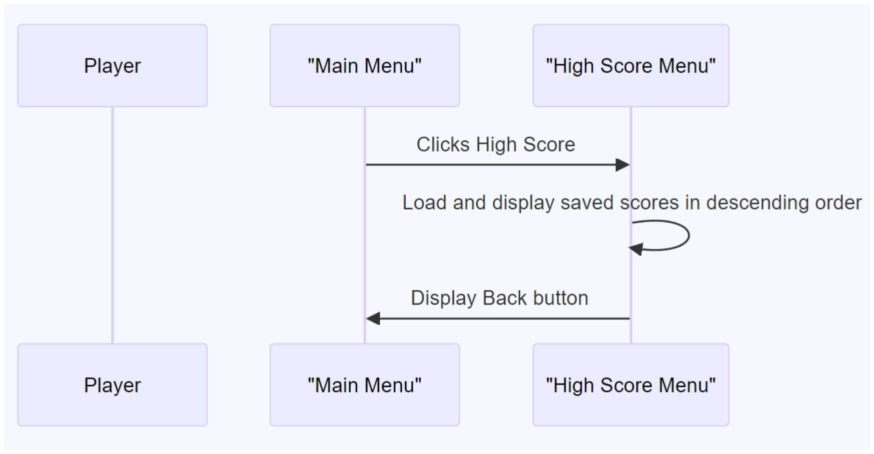


In User Interface

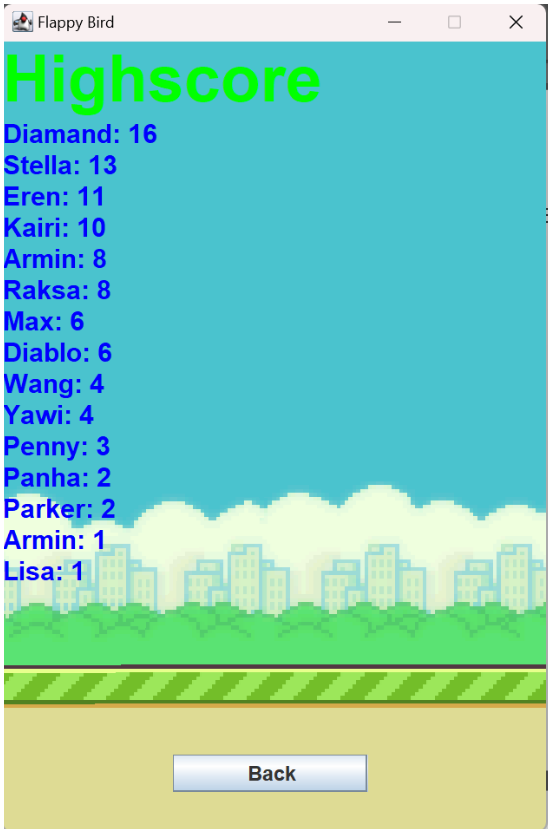


The players can click Play Again and it will display the level selection screen once again the pick the difficulty. And if the players click Exit, the game will be terminated.

6. High Score Display



In User Interface

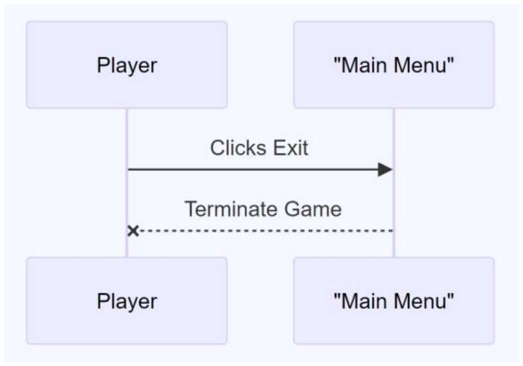


Highscore

Diamand: 16
Stella: 13
Eren: 11
Kairi: 10
Armin: 8
Raksa: 8
Max: 6
Diablo: 6
Wang: 4
Yawi: 4
Penny: 3
Panha: 2
Parker: 2
Armin: 1
Lisa: 1

If the player click High Score button in the main menu, it will display the list of all the High Score corresponding with the player's usernames that have been saved.

7. Exit Game



If the player click exit in the main menu then the game will terminate.

Neptun Code: KSG25Z

JUnit Testing

I have designed a collection of JUnit testing that can use to verify the functionality and correctness of various components within the Flappy Bird game.

1. BirdGame JUnit Tests
    a. testInitialGameState(): verifies if the game starts with the correct state and score.
    b. testGameStartChangesState(): tests if starting the game changes its state to "Running".
    c. testHighScoreFileExistandCanRead(): check if the Highscore.ser file exists and can be read.
2. Bird JUnit Tests
    a. testBirdInitialization(): tests if the bird is initialized with the correct position and image.
    b. testFly(): Checks if the fly() method of the bird results in a change of its image.
3. ScoreComparator JUnit Test
    a. testCompare(): tests the comparison logic of the ScoreComparator class by comparing HighScoreEntry instances.
4. Collection JUnit Tests
    a. testAdd(): verifies if the add() method adds a HighScoreEntry to the collection.
    b. testSaveAndLoad(): tests the save and load functionality of the Collection class.
    c. testListSorting(): checks if the list of high scores is sorted in descending order.
5. HighScoreScreen JUnit Test
    a. testLoadHighScores(): tests if the high scores can be loaded into the HighScoreScreen class.

Each test method is focusing on a specific aspect of the game or related classes and ensures that the behavior meets the expectations which help to ensure the correctness and reliability of the game implementation.