FlappyBird Game Documentation

CIS 150 project: A flappy-bird clone built with SFML in C++.

Team Code:

GEN_156

Team Members:

Mohamed Hesham Salah Abdelhamid
Seat Num.: 20201701245

Mazen Mohsen Ibrahim Hussien
Seat Num.: 20201701229

Omar Yasser Hamed Abdelaziz
Seat Num.: 20201701195

Yehia Edries Mokhtar Sediek
Seat Num.: 20201700982

Abdullah Hussien Mohamed Bahr
Seat Num.: 20201701244

Table of Contents

- Software Development Life Cycle (SDLC).
- Planning Phase.
- Define Requirements Phase.
- Design & Prototyping Phase.
- Software Development Phase.
- Testing Phase.
- Deployment Phase.
- Operation & Maintenance Phase.

What is the Software Development Life Cycle (SDLC)?

Software Development Life Cycle (SDLC) is the application of standard business practices to building software applications. SDLC is a way to measure and improve the development process. It allows a fine-grain analysis of each step of the process.

It is typically divided into six to eight steps: Planning, Requirements, Design, Build, Document, Test, Deploy, Maintain.

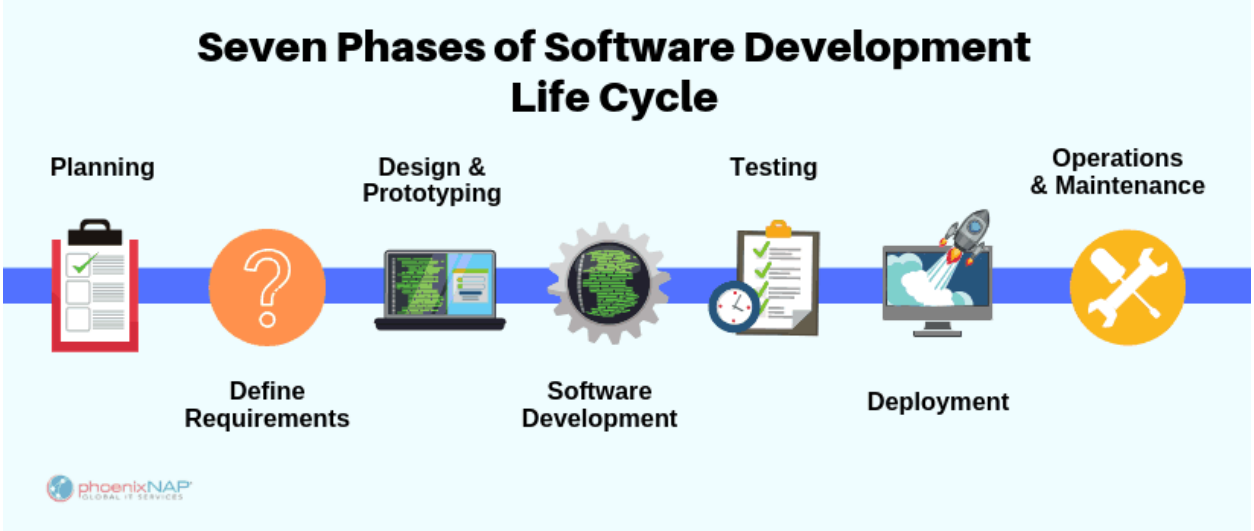

Figure (1): Seven Phases of Software Development Life Cycle.

Planning:

In the planning phase, the terms of the project are evaluated. This includes calculating workforce (5 people) and material costs (totally free), creating a timetable (within 2 weeks) with target goals (the project requirements which is specified below), and creating the project's team and leadership structure.

Team Structure:

- Mohamed Hesham *(Team Leader) (Coder).*
- Mazen Saad *(Designer) (Coder).*
- Omar Yasser *(Documenter) (Coder).*
- Yahya E-elkhdiwy *(Researcher) (Coder).*
- Abdullah Bahr *(Researcher) (Coder).*

Define Requirements:

Defining requirements is considered part of planning to determine what the application is supposed to do and its requirements.

Project Description:

Flappy Bird is an arcade-style game in which the player controls the bird Faby, which moves persistently to the right. The player is tasked with navigating Faby through pairs of pipes that have equally sized gaps placed at random heights. Faby automatically descends and only ascends when the player taps the touchscreen. Each successful pass through a pair of pipes awards the player one point. Colliding with a pipe or the ground ends the gameplay.

Figure (2): FlappyBird.

Deliverables:

Implementing a description with:

1- Simple GUI.
2- Save the highest score.

Must be done:

1- Use arrays, structs and functions.
2- Use C++.
3- Apply the concepts you study through the course.

Bonus:

1- Sounds.
2- More than one level difficulty.
3- Can stop and resume the game.
4- More functionality.

Design and Prototyping:

The Design phase models the way a software application will work. Some aspects of the design include:

Architecture – Specifies programming language, industry practices, overall design, and use of any templates or boilerplate. As it was specified earlier in the project requirements, the programming language is C++ using the Simple Fast Multimedia Library (SFML).

Figure (3): C++ & SFML.

User Interface – Defines the ways customers interact with the software, and how the software responds to input. That in which is achieved using the efficient, responsive, keyboard-driven and clear GUI.

Platforms – Defines the platforms on which the software will run, such as Apple, Android, Windows version, Linux, or even gaming consoles. The project so far is on Windows and Linux, tested and secure.

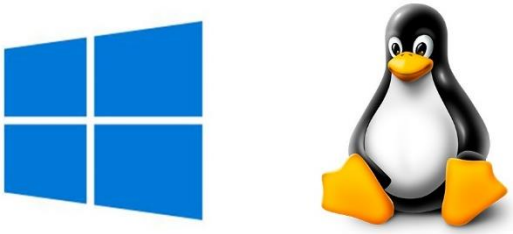

Figure (4): Windows & Linux.

Communications – Defines the methods that the application can communicate with other assets, such as a central server or other instances of the application. These methods include Discord, a VoIP, instant messaging and digital distribution platform designed for creating communities, and GitHub Git Version Control service, which allows users to create repositories and let contributors to cooperate using Add, Push, and Pull commands to maintain an up-to-date workflow.

Figure (5): Discord & GitHub.

Software Development:

This phase includes the actual writing of the program and the approach that was taken to fulfill the design requirements.

Pseudocode –

Main Menu:

```
// Render Window
lvlGame = 0;
while(window.is.open())
    if(keyPressed == E)
        lvlGame = 1;
    else if(keyPressed == M)
        lvlGame = 500;
    else if(keyPressed == H)
        lvlGame = 999;
```

Gameplay:

```
if(lvlGame == 1) // Easy Level
        displayEasyMode();
else if(lvlGame == 500) // Medium Level
    displayMediumMode();

else if(lvlGame == 999) // Hard Level
```

```
        displayHardMode();
else
        displayMainMenu();
birdState = 1; // bird is alive
while(gameLoop)
        if(!Pause) // if the game not paused
                bird.moveDown();
                if(birdCollided())
                        birdState = 0;
                        if(!Mute) // if the game sounds not muted
                                birdCrashSound.play();
                if(keyPressed == Up)
                        bird.moveUp();
                        if(!Mute)
                                birdUpSound.play();
                if(birdState == 0) // bird died
                        lvlGame = 0;
                        if(!Mute)
                                gameOverSound.play();
```

Scoring:

```
// if the bird position on the x-axis is bigger than pipe position
// on x-axis and the pipe is not counted before we increase the
// score by one.
while(gameLoop)
    if(bird.PosX > pipe.PosX && marked[pipe.Index] == false)
        score++;
        marked[pipe.Index] = true;
```

High Score:

```
if(birdState == 0)
    if(score > highScore)
        highScore = score;
        if(!Mute)
            highScoreSound.play();
```

Pause/Resume:

```
while(gameLoop)
    if(keyPressed == P)
        Pause = !Pause;
```

Flowcharts –

Main Menu:

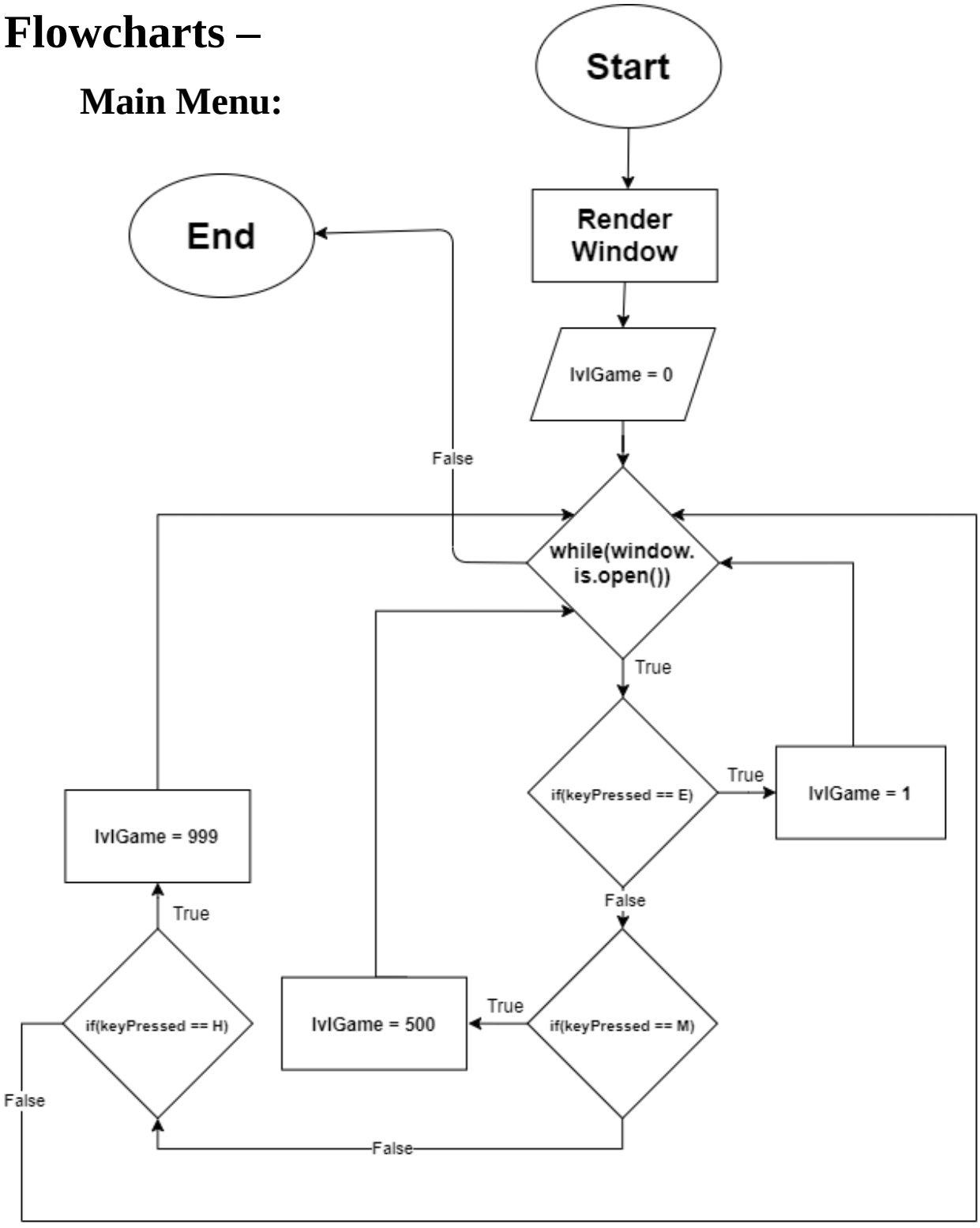

Figure (6): Main Menu Flowchart.

Gameplay:

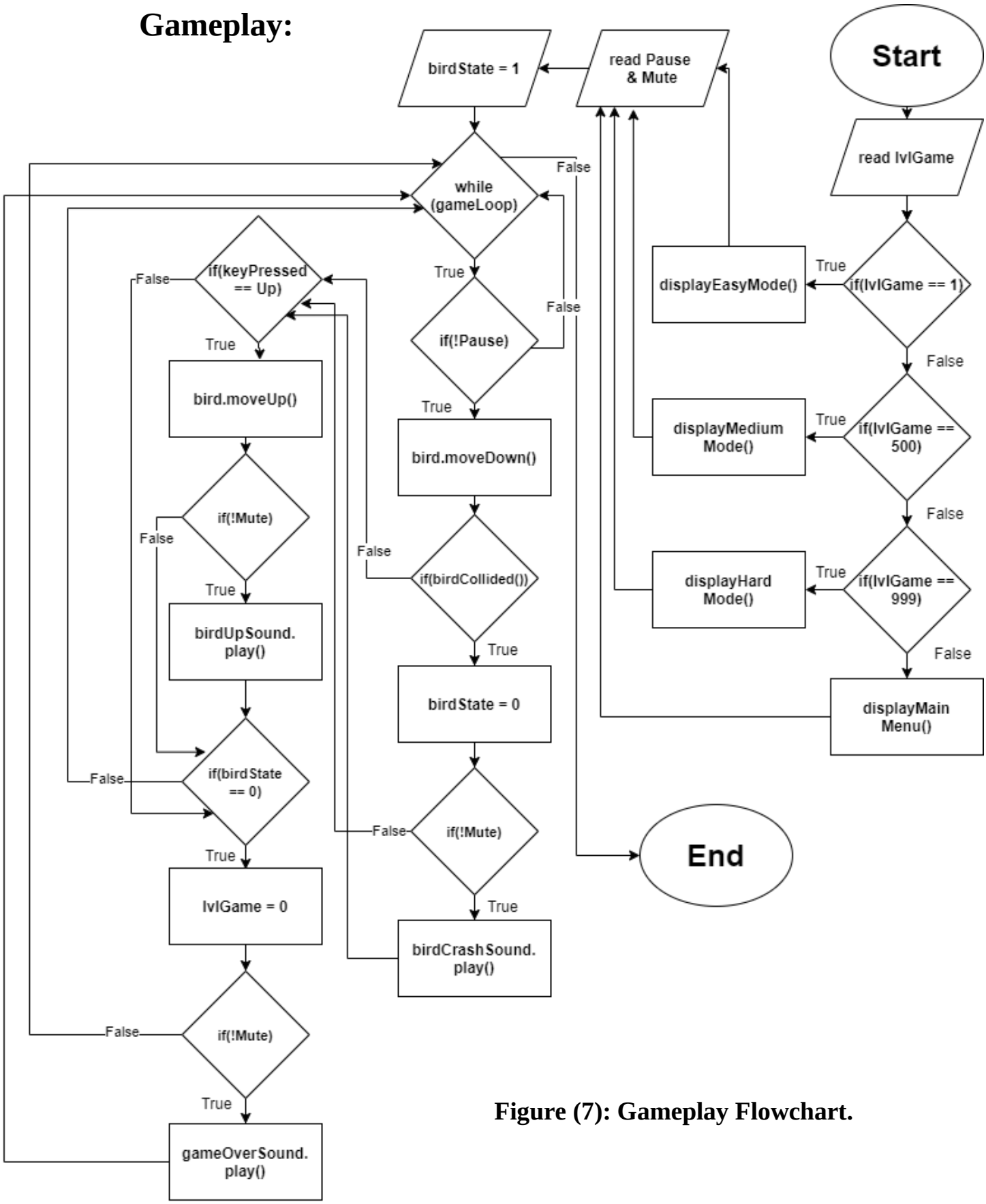

Figure (7): Gameplay Flowchart.

Scoring:

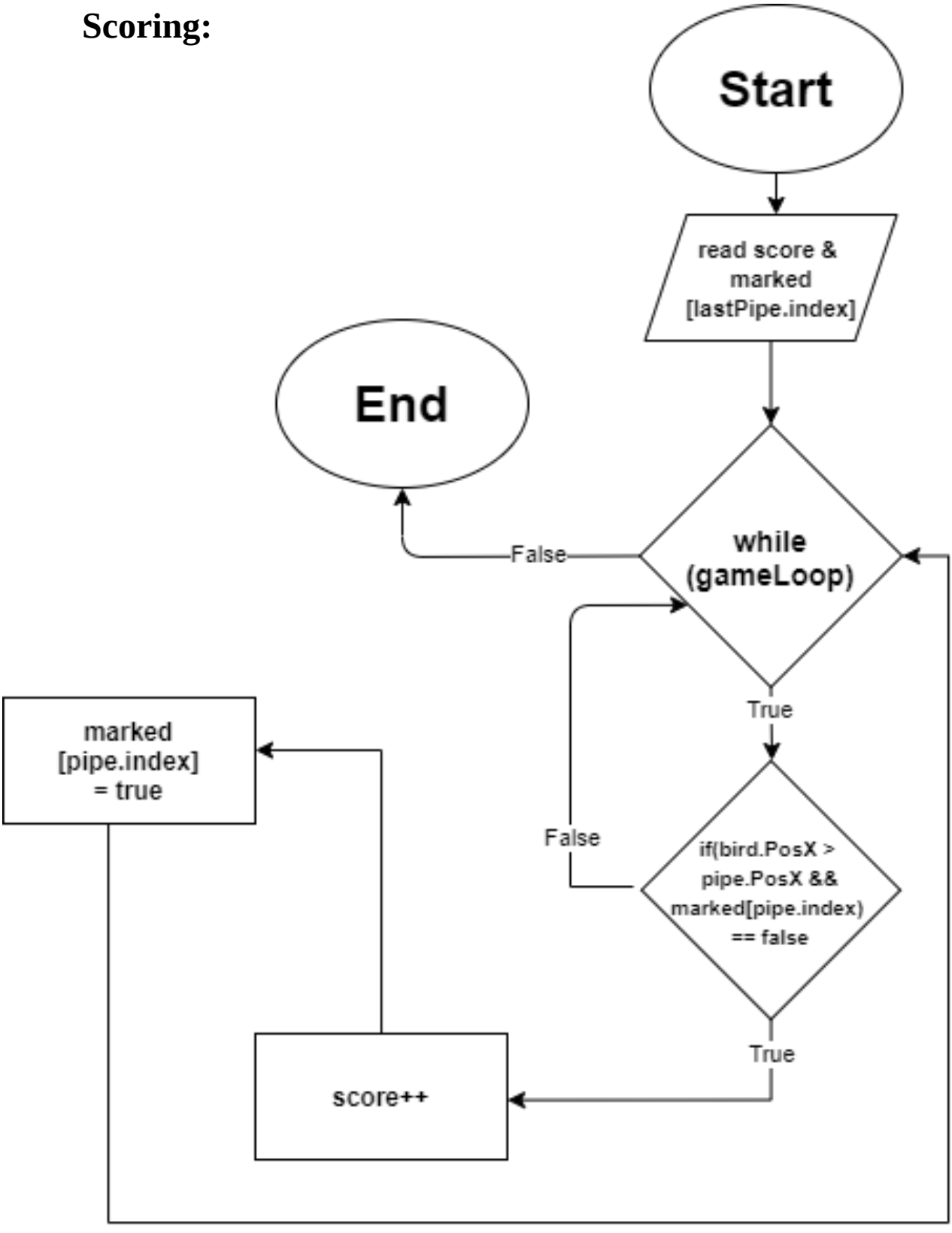

Figure (8): Scoring Flowchart.

High Score:

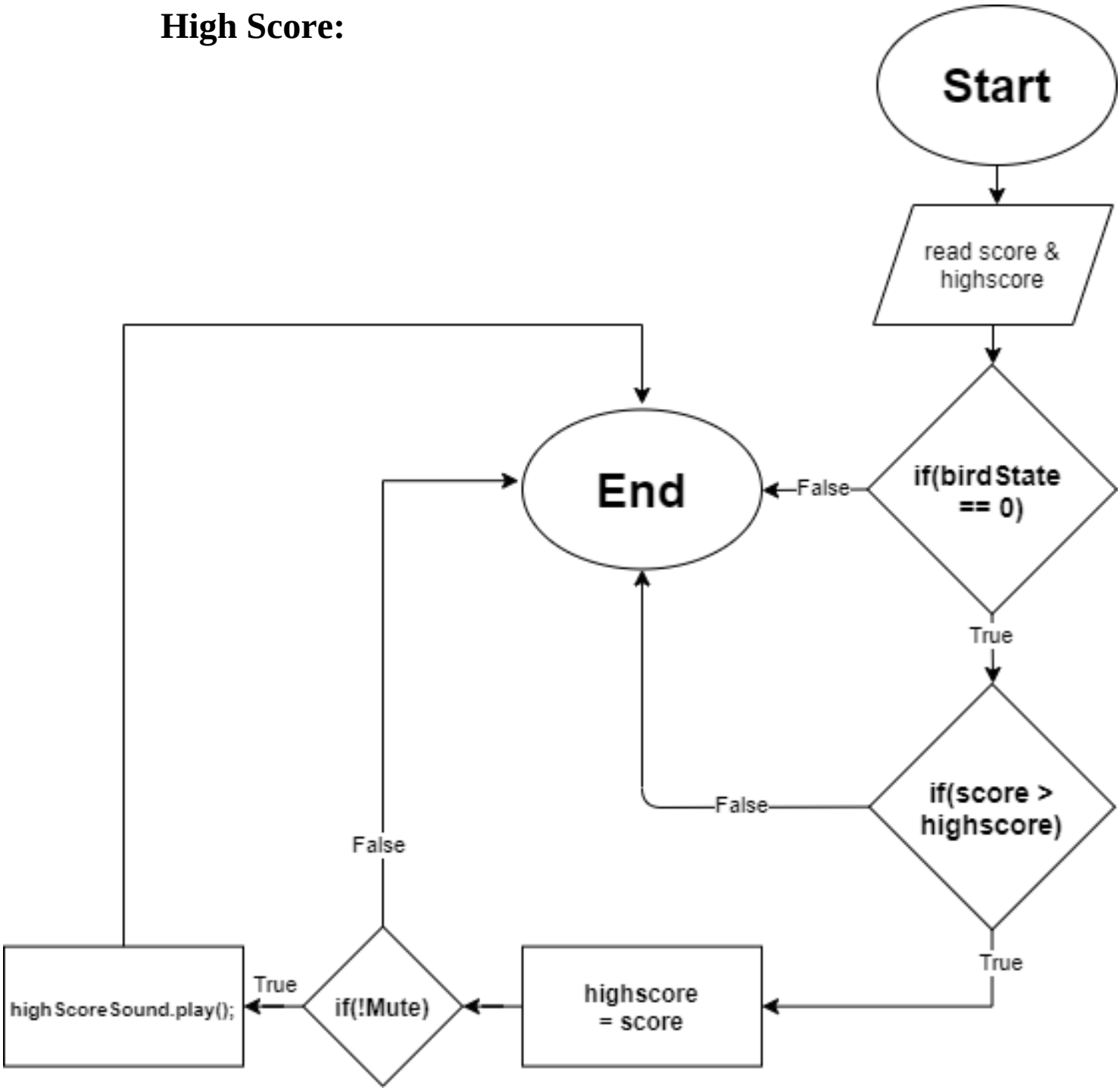

Figure (9): High Score Flowchart.

Pause/Resume:

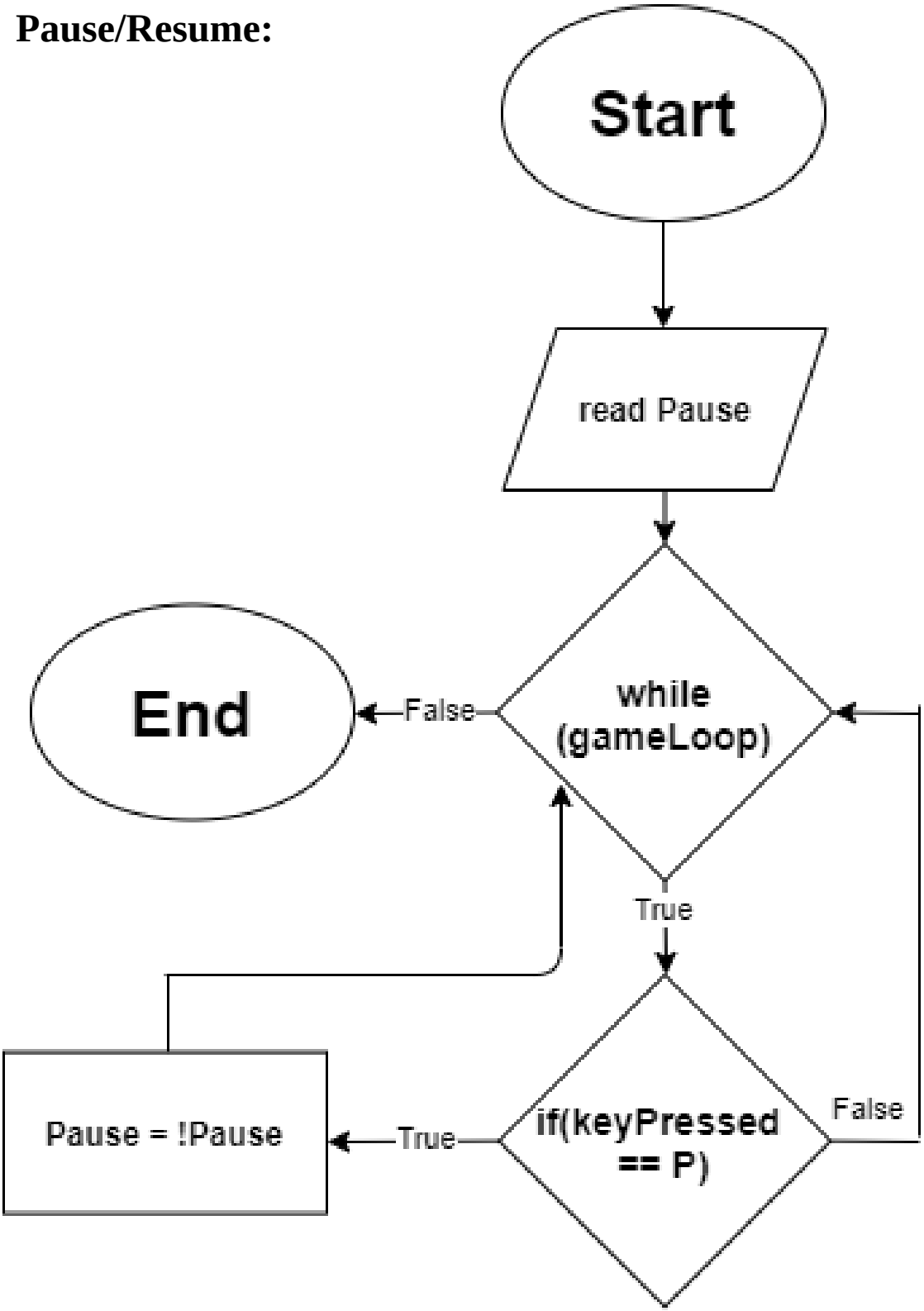

Figure (10): Pause/Resume Flowchart.

Algorithm –

Functions:

bird.hpp

isColliding. A function in which continuously checking if the bird did hit any of the pipes as it returns true if it did and if not it returns false.

```
bool isColliding(float x1, float y1, float w1, float h1, float x2, float y2, float w2, float h2)
{
    if (x1 + w1 >= x2 && x1 <= x2 + w2 && y1 + h1 >= y2 && y1 <= y2 + h2)
        return true;
    return false;
}
```

Figure (11): isColliding Function.

hitGround. A function in which continuously checking if the bird did hit the ground as it returns true if it did and if not it returns false.

```
bool hitGround(float y)
{
    if (y >= 692.0)
        return true;
    return false;
}
```

Figure (12): hitGround Function.

displayingEvents.hpp

handleEvent. A function in which handles the events that is happening while the game loop returns true.

```
// Function to handle Events
void handleEvent(RenderWindow &window, Event &event, bool &isPaused, bool &isGameover)···
```

Figure (13): handle Event Function.

displayTitle. A function in which it is called whenever it is needed to display the game title in the main menu.

```
// Fuction to display the Title in the Main Menu
void displayTitle(RenderWindow &window)
{
    Font font;
    font.loadFromFile("assets/font.ttf");
    Text title;
    title.setFont(font);
    title.setOutlineThickness(5);
    title.setOutlineColor(Color::Black);
    title.setString("FLAPPY BIRD");
    title.setCharacterSize(130);
    title.setFillColor(Color::White);
    title.setPosition(WID - 365, HEI - 300);
    window.draw(title);
}
```

Figure (14): displayTitle Function.

displayBackground. A function that is called whenever it is needed to display the background in the game loop.

```
// Function to display Background
void displayBackground(RenderWindow &window)
{
    Texture backgroundImage;
    backgroundImage.loadFromFile("assets/background.png");
    Sprite background(backgroundImage);
    window.draw(background);
}
```

Figure (15): displayBackground Function.

displayIntroBird. A function that is called whenever it is needed to display the bird in the main menu.

```
// Function to display the bird in the main menu
void displayIntroBird(RenderWindow &window)
{
    Texture bird;
    bird.loadFromFile("assets/gnu.png");
    Sprite flappyBird(bird);
    flappyBird.setPosition(WID - 50, HEI + 200);
    flappyBird.setScale(0.5f, 0.5f);
    window.draw(flappyBird);
}
```

Figure (16): displayIntroBird Function.

displayMenu. A function that is called whenever it is needed to display the main menu.

.

```
// Function to display Menu and reset score
void displayMenu(RenderWindow &window, int &score)···
```

Figure (17): displayMenu Function.

displayBoard. A function that is called whenever it is needed to display the board in which the score and high score is written on it.

```
void displayBoard(RenderWindow &window)
{
    Texture boardImage;
    boardImage.loadFromFile("assets/board.png");
    Sprite board(boardImage);
    board.setPosition(WID - 270, HEI - 100);
    window.draw(board);
}
```

Figure (18): displayBoard Function.

displayGameOver.
A function that is called whenever it is needed to display the game over splash screen.

```
// Function to display Game over Screen
void displayGameover(RenderWindow &window, int score, int usrHighScore, bool isHighscore)···
```

Figure (19): displayGameOver Function.

highscore.hpp

txtHighscore. A function in which it creates 3 text files for each difficulty level, if not created and write 0 in it as the user opens the game and if created, leave their values as it is.

```
// Create 3 txt files for each lvl if not created and write 0 in it, if created leave their values as it is
void txtHighscore()···
```

Figure (20): txtHighscore Function.

Highscore. A function that reads and writes high score form a file after every game over.

```
// Function to Read/Write highscore from file for each game lvl
void Highscore(int score, int &userHighScore, string gameLevel, bool &isHighscore)···
```

Figure (21): Highscore Function.

displayScore. A function that displays the score while playing the game.

```
// Function to display while playing the Game
void displayScore(RenderWindow &window, int score)···
```

Figure (22): displayScore Function.

pause.hpp

displayPause. A function that displays the pause screen if the game is paused.

```
// Function to display Pause Screen
void displayPause(RenderWindow &window)
{
    Font font;
    font.loadFromFile("assets/font.ttf");
    Text pause;
    pause.setFont(font);
    pause.setOutlineThickness(5);
    pause.setOutlineColor(Color::Black);
    pause.setString("Game Paused");
    pause.setCharacterSize(130);
    pause.setFillColor(Color::White);
    pause.setPosition(WID - 410, HEI - 300);
    window.draw(pause);
}
```

Figure (23): displayPause Function.

pipe.hpp

loadPipe. A function that loads the pipes texture.

```
void loadPipe()
{
    pipeUp.texture.loadFromFile("assets/pipe.png");
    pipeDown.texture.loadFromFile("assets/pipe.png");
    pipeUp.sprite.setTexture(pipeUp.texture);
    pipeDown.sprite.setTexture(pipeDown.texture);
}
```

Figure (24): loadPipe Function.

positionPipe. A function that positions the pipes whenever it is called.

```
void positionPipe(int random, int gap)
{
    pipeUp.sprite.setPosition(1000, random);
    pipeUp.sprite.setScale(0.5, -0.5);
    pipeDown.sprite.setPosition(1000, random + gap);
    pipeDown.sprite.setScale(0.5, 0.5);
}
```

Figure (25): positionPipe Function.

Definitions:

definitions.hpp

const unsigned int WIDTH, HEIGHT. Indicating the window width and height.

int frames, pipeRate, gap, score, userHighScore. These variables are used to determine the number of frames that is used at the moment, the pipe rate at which it shows, the gap between every two pipes, the current user score, and the user overall high score.

bool isGameover, isPaused, isMuted, isHighscore, highscoreSoundPlayed, pauseSoundPlayed, checkedHighscore, firstTime. These variables are used to determine if the gamed ended or not, the game is paused or not, the sound is muted or not, the player achieved a new high score or not, the high score sound should be played or not, the pause sound should be played or not, the high score is already checked or not, and a first-time player or not.

string gameStates[4]. An array in which contains the 4 game states which are easy, medium, hard, and over.

Testing:

It is critical to test an application before making it available to users. Testing should ensure that each function works correctly. Different parts of the application should also be tested to work seamlessly together—performance test, to reduce any hangs or lags in processing. The testing phase helps reduce the number of bugs and glitches that users encounter. This leads to a higher user satisfaction and a better usage rate.

As shown in the figure, as individuals, the game was tested over and over on different platforms, Windows, and Linux, to address bugs, compatibility issues, and glitches.

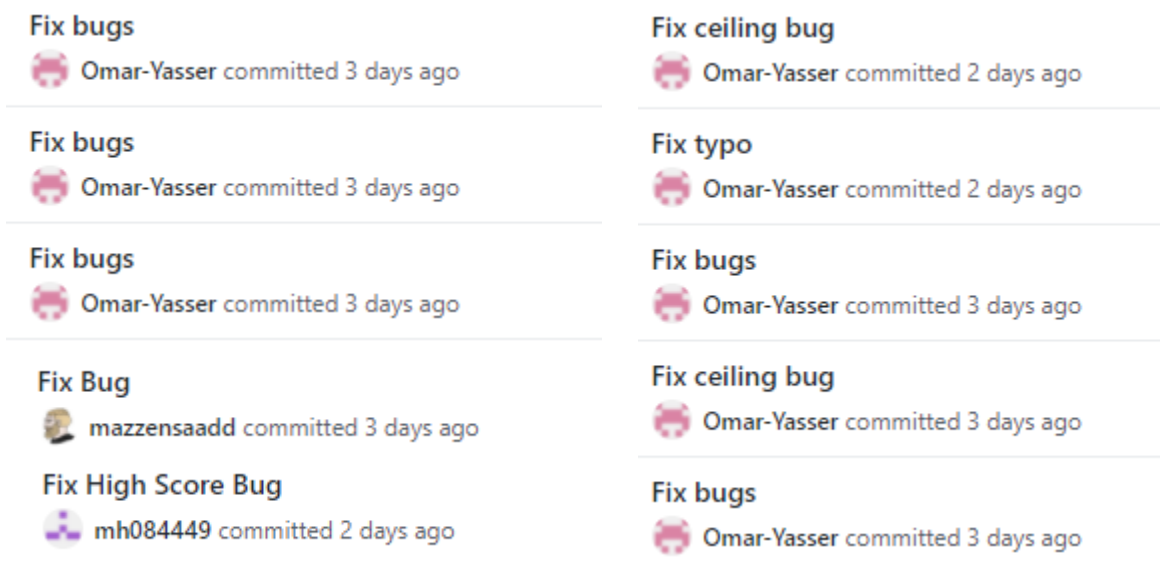

Figure (26): Fix Bugs Commits on GitHub.

Deployment:

In the deployment phase, the application is made available to users. This can be as simple as uploading it to Google Drive and spreading the download link to those who would give it a try.

Here is a glimpse of the gameplay.

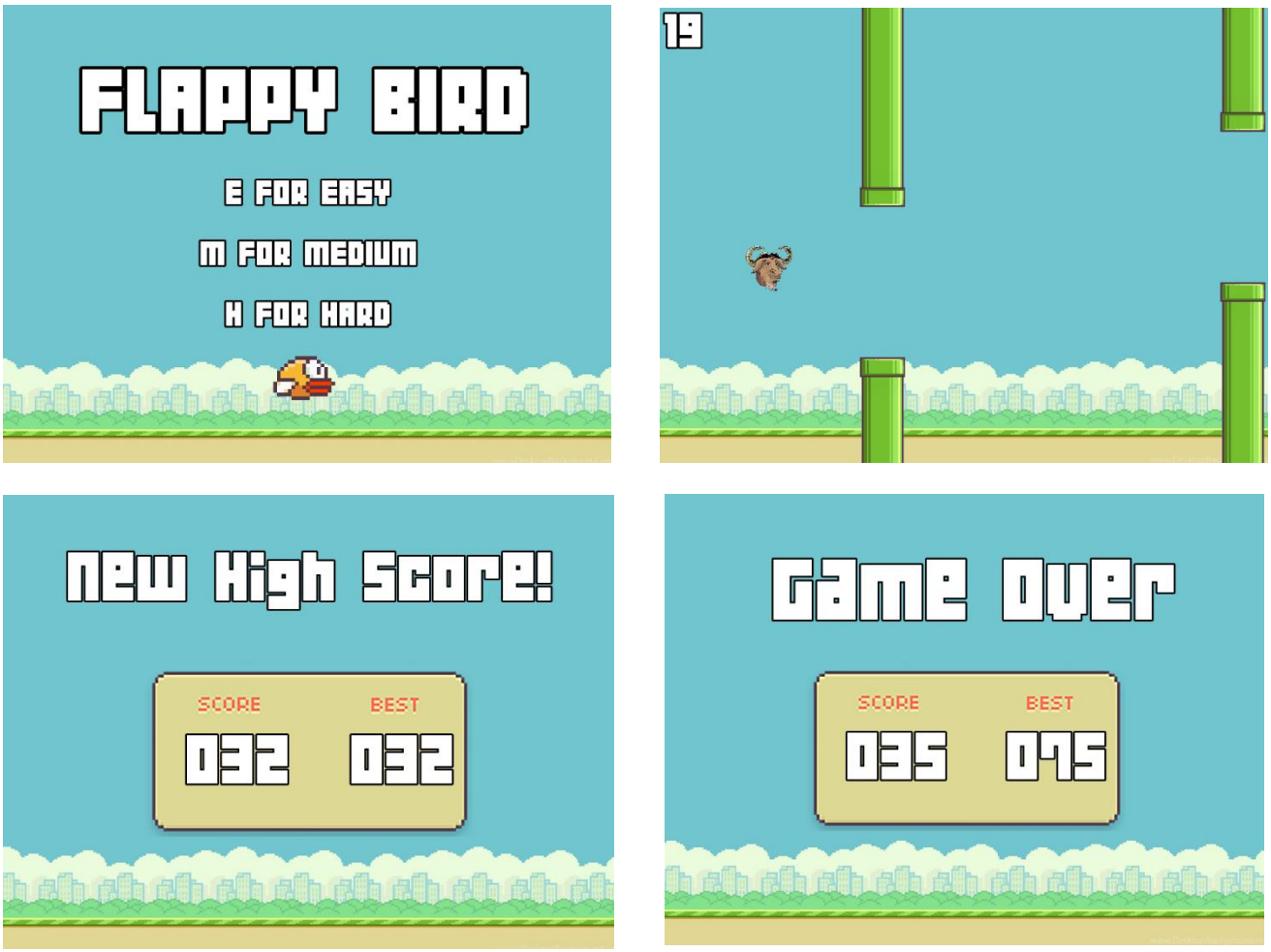

Figure (27): FlappyBird Gameplay.

Operation and Maintenance:

At this point, the development cycle is almost finished. The application is done and being used in the field. The Operation and Maintenance phase is still important, though. In this phase, users discover bugs that were not found during testing. These errors need to be resolved, which can spawn new development cycles.

Here is how people reacted to the game on a scale of 1 to 5.

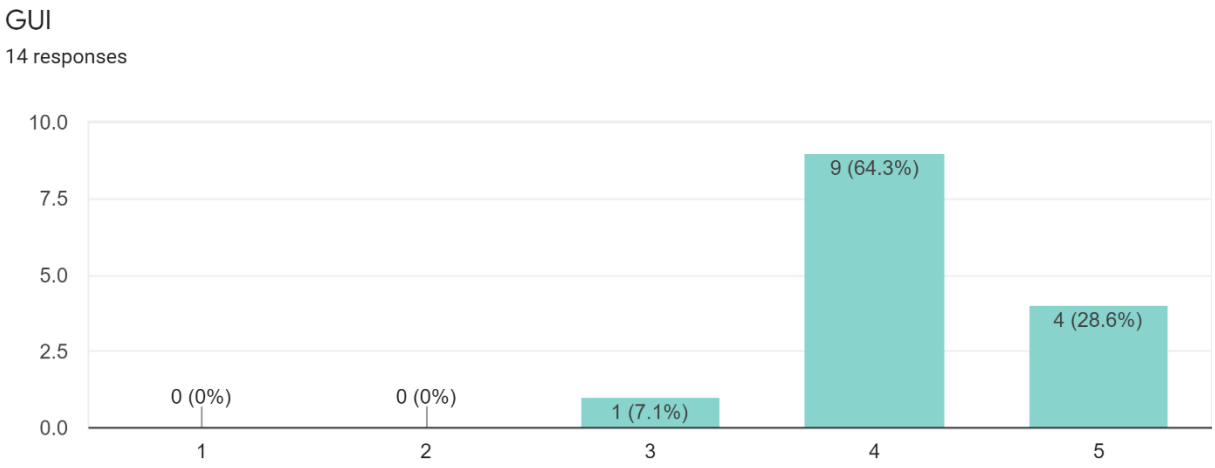

Figure (28): GUI Rating chart.

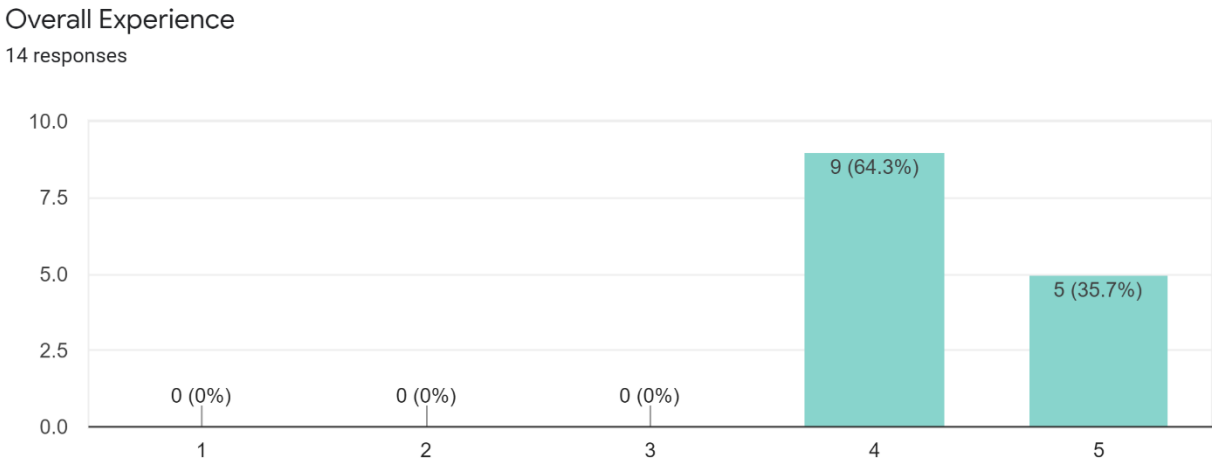

Figure (29): Overall Experience Rating chart.