

Project Documentation and Report

CSA0824– Python Programming – Slot A

Real-Time Weather Application Using Python and Tkinter



**Documentation by, Guided by,**

J.Peranbu T.Vincent Gnanaraj

192110056 (Course Faculty)

Department of CSE Python Programming

SSE, SIMATS SSE, SIMATS

**Bonafide Certificate**

This work is done by J.Peranbu(192110056) from Computer Science and Engineering Department during slot A Python Programming(CSA0824) in the duration from December 2023 to February 2024 under the guidance of T.Vincent Gnanaraj.

**table Of Contents**

|  |  |
| --- | --- |
| **Contents** | **Page Number** |

|  |  |
| --- | --- |
| Abstract | 04 |
| Introduction | 05 |
| Existing System | 06 |
| Proposed System | 07 |
| Requirements and Specification | 08 |
| Methodology | 09 |
| Analysis | 10 |
| Design | 11-12 |
| Coding | 13-15 |
| Testing | 16 |
| Implementation | 17-18 |
| Maintenance | 19 |
| Future Enhancement | 20 |
| Conclusion | 21 |
| Acknowledgment | 22 |
| Reference | 23 |

**Abstract**

The **Weather Application** project aims to develop a Python-based desktop application that provides real-time weather information for any specified city. Leveraging a graphical user interface (GUI) built with Tkinter, and utilizing various APIs such as OpenWeatherMap for weather data and Geopy for geolocation services, the application offers an intuitive and user-friendly platform for accessing comprehensive weather details.Extensive testing has been performed to ensure the application’s reliability and performance across various scenarios, including valid and invalid city inputs and different internet connectivity statuses.

***Keywords:***Python,Weather,Application,Tkinter,GUI(GraphicalUserInterface),

Geopy,Nominatim,TimezoneFinder,Pytz,Requests,OpenWeatherMap,API,Real-Time Weather Data

**I.Introduction**

The **Weather Application** is a modern, Python-based tool designed to provide real-time weather updates. Utilizing the power of various libraries and APIs, this application offers users an easy and efficient way to check weather conditions in any city around the globe. The intuitive graphical user interface (GUI) developed using Tkinter ensures a seamless user experience, while the integration with the OpenWeatherMap API guarantees accurate and up-to-date weather information.

However, despite its effectiveness, the Weather Application faces certain limitations. Its current scope is somewhat basic, offering only fundamental weather data without any advanced features like forecasts, alerts, or historical weather trends. Additionally, the application's user interface is designed for desktop use, which may not provide the best experience on different screen sizes and devices.

To enhance the application's functionality and user engagement, several improvements can be made. Introducing features such as multi-day weather forecasts, weather alerts, and the ability to display historical weather data can greatly enhance the application's utility. Moreover, implementing a responsive design will ensure that the application remains user-friendly across a wide range of devices, from desktop computers to mobile phones and tablets. These enhancements will not only make the Weather Application more versatile but also significantly improve the overall user experience.

**II.Existing System**

The current iteration of the **Weather Application** primarily operates by allowing users to enter the name of a city to fetch real-time weather data. The application utilizes the OpenWeatherMap API to retrieve essential weather information, such as temperature, humidity, wind speed, and atmospheric pressure. The graphical user interface (GUI) is constructed using Tkinter, providing a straightforward text entry box for user input and a search button to initiate the data retrieval process.

Despite its effectiveness, the existing system has several limitations. The user interface, although functional, is quite basic and may not offer the visual appeal required to engage all users. It lacks advanced features such as multi-day weather forecasts, weather alerts, and detailed weather statistics, which could significantly enhance the utility and user experience of the application. Moreover, the current design does not adapt well to different screen sizes and resolutions, limiting its usability on various devices such as tablets and smartphones.

To sum up, while the existing system serves its primary purpose of providing real-time weather data, there is ample room for improvement in terms of user interface design, feature set, and responsiveness to different device formats. These enhancements could make the application more versatile, engaging, and useful for a broader audience.

**III.Proposed System**

In the proposed system for the **Weather Application**, we aim to significantly enhance user engagement and utility by introducing a range of new features. The addition of multi-day weather forecasts will allow users to plan ahead more effectively, while weather alerts will ensure users are informed of severe weather conditions promptly. Detailed weather statistics such as UV index, visibility, and sunrise/sunset times will offer a more comprehensive understanding of the weather. These enhancements will transform the application from a simple weather-checking tool into a valuable resource for daily planning.

To improve the overall user experience, the application will be redesigned with a responsive layout that adapts to different screen sizes and resolutions. This will ensure that the application provides an optimal viewing experience on various devices, from desktops to mobile phones. Interactive elements will be introduced, allowing users to customize display settings, such as choosing between Celsius and Fahrenheit for temperature, and m/s and km/h for wind speed. The application will also use geolocation to automatically detect the user’s location and provide relevant weather information, making it more user-friendly.

**iv.Requirements**

We need the following requirements for developing, designing and Testing of a classical snake game project

**Hardware Requirements**

* **CPU**: AMD Ryzen 5 processor with 64-bit operating system, x64-based processor
* **RAM**: 16GB
* **SSD**: 512GB
* **KEYBOARD**: Membrane
* **MOUSE**: Optical
* **MONITOR**: LED
* **INTERNET**: Wifi

**Software Requirements**

* **DEVELOPMENT SDK**: Python 3.x
* **RUNTIME ENVIRONMENT**: Python Interpreter
* **OPERATING SYSTEM**: Windows 11 64-bit Processor
* **PACKAGES**: Tkinter, Geopy, TimezoneFinder, Pytz, Requests

**Human Requirements**

**TEAM SIZE:**1

**SUPERVISOR:**1

**TOTAL:**2

**Financial Requirements**

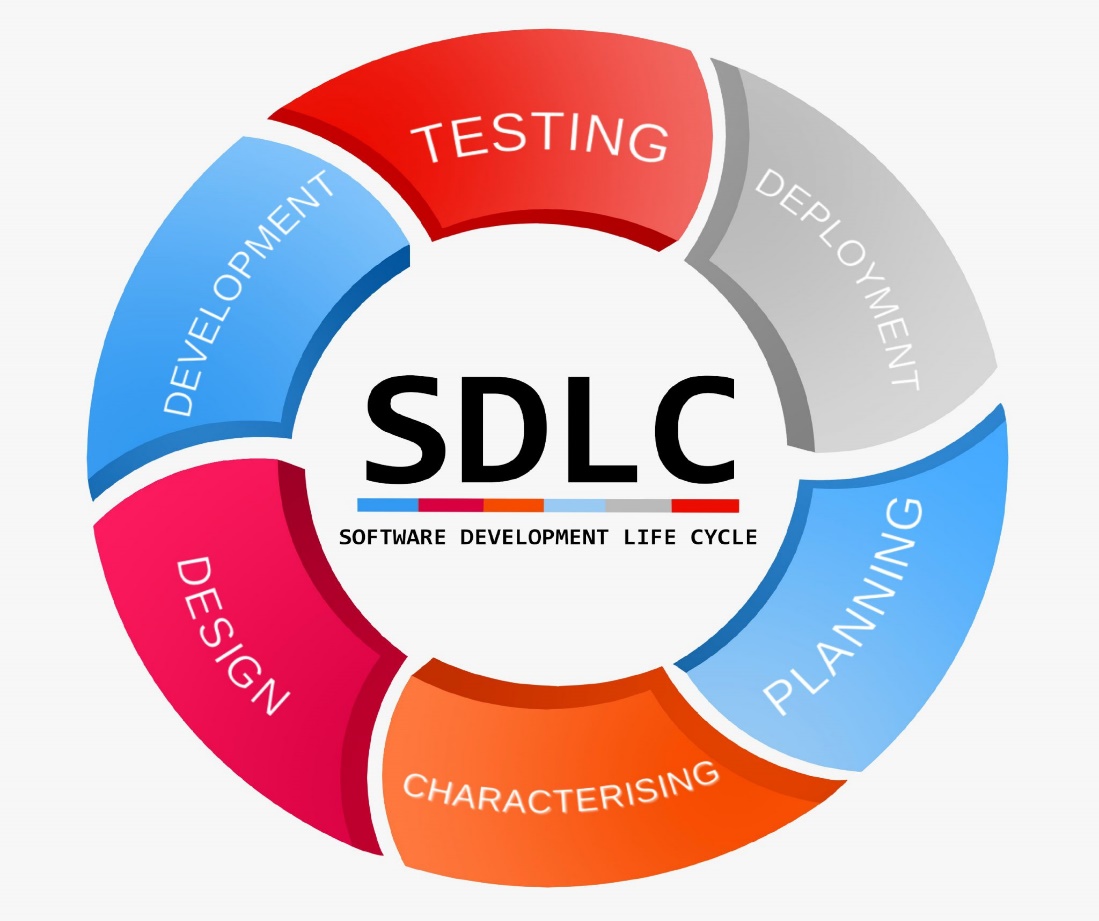
**BUDGET:** 1000 to 15000 depending on implementing and quality metrics

**v.Methodology**

**SDLC MODEL:** The Software Development Life Cycle (SDLC) model is a framework that describes the stages involved in the development of software, from initial concept to final deployment and maintenance. These stages typically include Requirement Analysis, Design, Implementation, Testing, Deployment, and Maintenance. Each stage is crucial and plays a significant role in the successful completion of the software. The SDLC model is versatile and can be adapted to various software development methodologies such as Waterfall, Agile, and more. It provides a structured approach to software development, ensuring that all necessary steps are followed, which leads to the production of high-quality software. However, it requires careful planning and execution, and changes in the later stages of the cycle can be costly and time-consuming. Despite these challenges, the SDLC model remains a fundamental process in the field of software development.

**SDLC MODEL FOR PYTHON WEATHER APPLICATION**

*  **Requirement Definition**:
* Define the application's requirements, such as weather data details (temperature, humidity, wind speed), user interface layout, and data retrieval frequency.
*  **System Design**:
* Design the application's architecture. This includes designing modules for the GUI, data fetching, and time zone handling. Also, plan how these modules will interact.
*  **Implementation**:
* Write the code for the application. This includes creating the GUI with Tkinter, implementing the functions to fetch weather data using the OpenWeatherMap API, and handling time zone conversions.
*  **Testing**:
* Test the application to ensure that it works as expected. This includes testing the data retrieval process, GUI functionality, and time zone accuracy.
*  **Deployment**:
* Deploy the application on the desired platform(s) and provide necessary documentation and user guides.
*  **Maintenance**:
* After the application is deployed, monitor it for any issues or bugs. If any are found, update the application accordingly.



**VI.Analysis**

* While Analyzing the classical snake game involves understanding its core mechanics and gameplay.
* The game is simple yet addictive, with the primary objective being to control a snake to consume food and grow in length.
* The challenge lies in avoiding collisions with the snake’s own body and the game area boundaries.
* The game’s simplicity and the increasing difficulty as the snake grows longer make it an interesting subject for analysis.

1. **DATA ANALYSIS**

* Data analysis in the context of the snake game could involve studying player behaviour, such as the average game duration, the typical length achieved by the snake, or the most common causes of game over.
* This data can provide insights into the game’s difficulty and player engagement levels.
* Additionally, the analyzing the scores achieved by players can help in understanding the skill levels required for the game and can be used to fine-tune the game’s difficulty.

1. **SOFTWARE ANALYSIS**

* Software analysis of the snake game would focus on the efficiency and quality of the code.
* This could involve examining the game loop’s performance, the effectiveness of collision detection algorithms, or the use of object-oriented principles in structuring the game’s classes and methods.
* The analysis could also include checking the responsiveness of the game, memory usage, and how well the game handles edge cases.

1. **HARDWARE ANALYSIS**

* Our Hardware is compatible with the software. There is enough space in the hardware to run the above mentioned software.

**MAINTANENCE**: The software is under good Maintenance.

1. **Human Resources**

* We have Gained enough experience in Java Programming to complete this Project. The Supervisor is capable of training this project.

1. **External Resources**

* On site training, Internships, Workshops, Funded Projects could be very helpful resources for this project.

**vii.Design**

For designing this Project we have developed a architecture diagram, Sequence diagram

**FRONTEND DESIGN:** Using Fonts, Layout and GUI (Graphical User Interface)

**BACKEND DESIGN:** No Backend Required for this project

**ARCHITECTURE DIAGRAM**

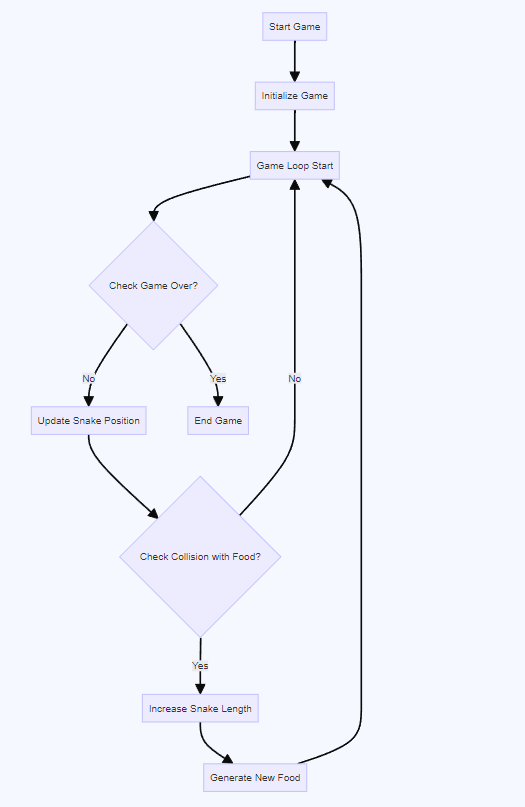
****

Figure 1.1-Architecture Diagram

**ACTIVITY DIAGRAM**

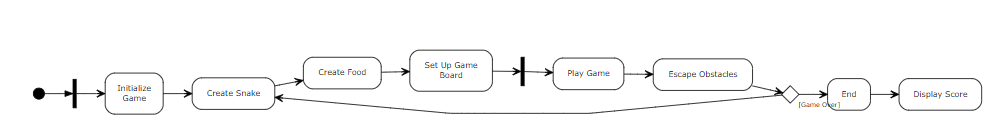
****

Figure 1.2-Activity Diagram

**CLASS DIAGRAM**

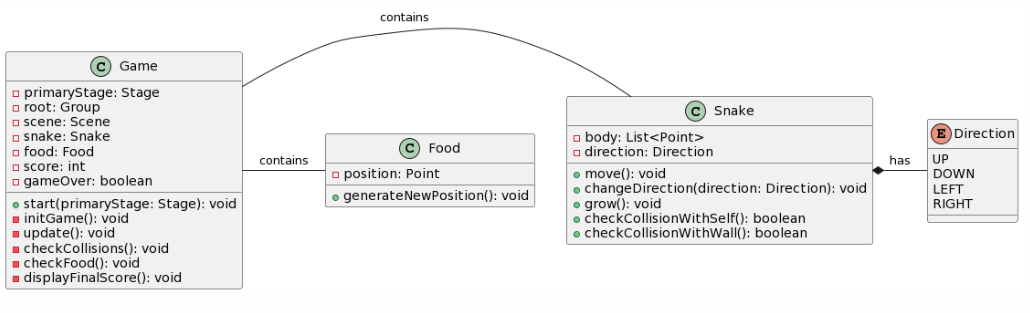
****

Figure 1 3-Class Diagram

**viii.Coding**

**SYNTAX**

Sure, here are the basic syntax elements in Java along with their explanations:

1. public class MyClass { } – This is a class declaration. MyClass is the name of the class.
2. public static void main(String[] args) { } – This is the main method that gets executed when you run your Java program.
3. int myVar = 5; – This is a variable declaration. myVar is a variable of type int and is assigned the value 5.
4. final double PI = 3.14; – This is a constant declaration. PI is a constant of type double and is assigned the value 3.14.
5. if (condition) { } – This is an if statement. The code inside the braces {} is executed if the condition is true.
6. if (condition) {} else {} – This is an if-else statement. If the condition is true, the code in the first block is executed; otherwise, the code in the else block is executed.
7. switch (variable) {case value: break; default: break;} – This is a switch statement. It allows a variable to be tested for equality against a list of values.
8. for (initialization; condition; increment) {} – This is a for loop. It is used to repeatedly execute a block of code until a certain condition is met.
9. while (condition) {} – This is a while loop. It repeatedly executes a block of code as long as a certain condition is true.
10. do {} while (condition); – This is a do-while loop. It is similar to a while loop, but the condition is tested after the execution of the block of code.
11. int[] myArray = new int[10]; – This is an array declaration. myArray is an array of int type with a size of 10.
12. public returnType functionName(parameters) {} – This is a function declaration. functionName is the name of the function, returnType is the data type of the value the function returns, and parameters are input to the function.
13. try {} catch (ExceptionType e) {} – This is a try-catch block. It is used to handle exceptions and errors that occur in a block of code.
14. MyClass obj = new MyClass(); – This is how to create an object. obj is an object of class MyClass.
15. obj.memberName; – This is how to access an object’s members. memberName is the name of a member (variable or method) of the object obj.

**Source Code**

import java.util.ArrayList;

import java.util.List;

import java.util.Random;

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.input.KeyCode;

import javafx.scene.input.KeyEvent;

import javafx.scene.layout.VBox;

import javafx.scene.paint.Color;

import javafx.scene.text.Font;

import javafx.stage.Stage;

public class main extends Application {

static int speed = 5;

static int foodcolor = 0;

static int width = 20;

static int height = 20;

static int foodX = 0;

static int foodY = 0;

static int cornersize = 25;

static List<Corner> snake = new ArrayList<>();

static Dir direction = Dir.left;

static boolean gameOver = false;

static Random rand = new Random();

public enum Dir {

left, right, up, down

public static class Corner {

int x;

int y;

public Corner(int x, int y) {

this.x = x;

this.y = y;

}

}

public void start(Stage primaryStage) {

try {

newFood();

VBox root = new VBox();

Canvas c = new Canvas(width \* cornersize, height \* cornersize);

GraphicsContext gc = c.getGraphicsContext2D();

root.getChildren().add(c);

new AnimationTimer() {

long lastTick = 0;

public void handle(long now) {

if (lastTick == 0) {

lastTick = now;

tick(gc);

return;

}

if (now - lastTick > 1000000000 / speed) {

lastTick = now;

tick(gc);

}

}

}.start();

public static Color getFoodColor() {

Color cc = Color.WHITE;

switch (foodcolor) {

case 0:

cc = Color.LIGHTCORAL;

break;

case 1:

cc = Color.CORNSILK;

break;

case 2:

cc = Color.MOCCASIN;

break;

case 3:

cc = Color.PALEVIOLETRED;

break;

case 4:

cc = Color.SALMON;

break;

}

return cc;

}

public static void newFood() {

start: while (true) {

foodX = rand.nextInt(width);

foodY = rand.nextInt(height);

for (Corner c : snake) {

if (c.x == foodX && c.y == foodY) {

continue start;

}

}

foodcolor = rand.nextInt(5);

speed++;

**ix.Testing**

**MODULE TESTING**

* This involves testing individual components of the game in isolation.
* For example, you might test the functionality that updates the snake’s position or the function that checks for collisions with the game boundaries or the snake itself.

**BLACKBOX TESTING**

* This is a high-level testing where you test the game functionality without knowing the internal workings.
* You could test scenarios like the behaviour of the game when the snake eats food, hits a wall, or collides with itself.

**WHITEBOX** **TESTING**

* This is a detailed testing where you test the internal workings of the game.
* For example, you might test the algorithm that generates the food in random positions, or the logic that increases the length of the snake.

**INTERGRATION TESTING**

* This involves testing the interaction between different components of the game.
* For instance, you might test how the game handles input from the user and how this affects the snake’s movement.

**SYSTEM TESTING**

* This is the final level of testing where you test the complete integrated game.
* It involves testing the game as a whole, ensuring all parts work together as expected.
* This includes testing the game performance, user interface and end-to-end gameplay.

**x.Implementation**

The classical snake game implemented in JavaFX involves creating a simple GUI application where the player controls a snake that grows in length as it consumes food. The game starts with a small snake moving in a grid. The user controls the direction of the snake’s movement using keyboard keys. A piece of food is randomly placed in the grid, and the objective is to guide the snake to the food. When the snake eats the food, it grows in length and the score increases. The game ends when the snake collides with the grid boundary or with itself. JavaFX is used for rendering the game graphics and handling user input. The game logic, such as the snake’s movement and collision detection, is implemented in Java. The game can be further enhanced by adding features like different levels of difficulty, power-ups, and high score tracking.

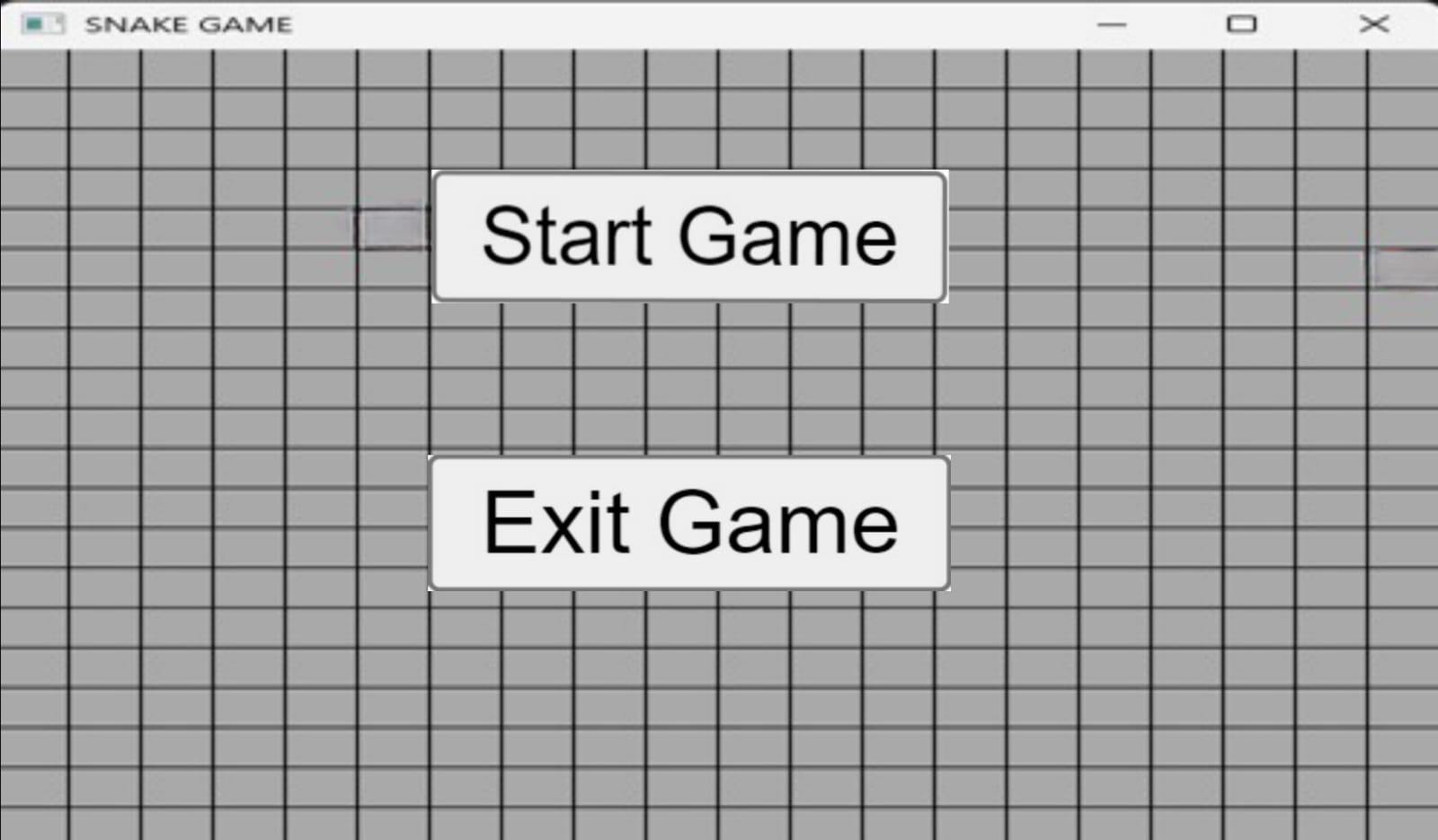


Figure 1.4 - Start Game

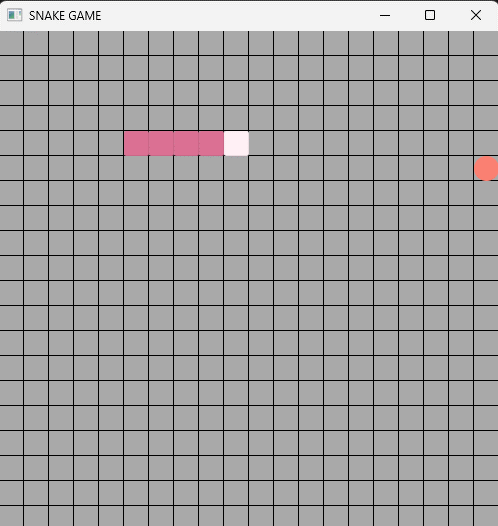


Figure 1.5 - Playing Game

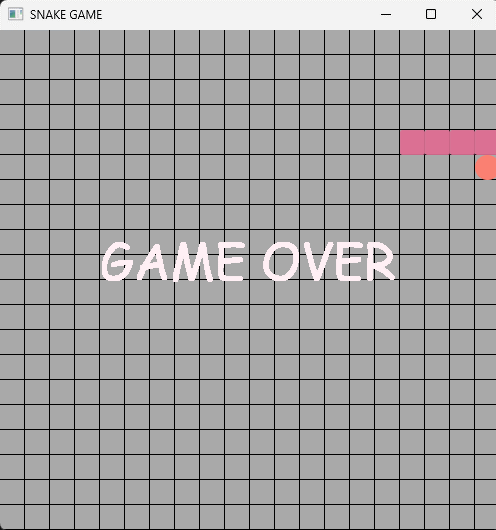


Figure 1.6 - End Game

**xi.Maintenance**

* This project requires maintenance according to software update and hardware changes
* Regularly update the game’s software to fix bugs, improve performance, and add new features. This could involve updating the game engine, libraries, or other dependencies.
* Ensure the game remains compatible with new hardware. This could involve testing the game on new devices or operating systems, and making necessary adjustments.
* Regularly check for and patch any security vulnerabilities. This is especially important if your game has online features.
* Regularly collect and review feedback from your players. This can help you identify areas for improvement, and understand what players like and dislike about your game.

**xii.Future Enhancement**

In future iterations of the Classical Snake game implemented in JavaFX, potential enhancements could include integrating advanced AI opponents with adjustable difficulty levels, offering customizable themes and settings through FXML for a personalized gaming experience, implementing multiplayer online functionality with networking classes and lobby interfaces, introducing game statistics tracking and leader boards to foster competition, and incorporating accessibility features like text-to-speech announcements and adjustable font sizes through classes and FXML files to ensure inclusivity for all players.

**xiii.Conclusion**

The game was successfully implemented using JavaFX, which provided a robust platform for creating the GUI and handling user interactions. The game logic was implemented in Java, which allowed for efficient handling of game dynamics such as the snake’s movement, collision detection, and score tracking. One of the main challenges was managing the game’s state, especially as the snake grew in length. Another challenge was handling user input in real-time to ensure smooth and responsive gameplay. Additionally, designing the GUI to be intuitive and user-friendly was also a significant task. While the core gameplay remained true to the classical snake game, some deviations were made to enhance the user experience. These include adding different difficulty levels, power-ups, and a high score tracking system. These additions aimed to make the game more engaging and enhancing the overall user experience.

**xiv.Acknowledgement**

We thank Oracle for providing java software. We thank our guide for technical support and thank mentor for moral support and we also thank our principal for accommodation support

**xv.Reference**

# [1] Md. Rafat Rahman Tushar, Shahnewaz Siddique, “A Memory Efficient Deep Reinforcement Learning Approach For Snake Game Autonomous Agents, January 27,2023.

(<https://arxiv.org/abs/2301.11977>)

[2] Sloan Kelly,” Game Project Snake”, Dec 24,2016

[(https://link.springer.com/chapter/10.1007/978-1-4842-2517-2\_16](https://link.springer.com/chapter/10.1007/978-1-4842-2517-2_16))

[3] Concept of Gamification in Adaptation of Snake Ladder Online Representation Education Covid-19By [Hani Dewi Ariessanti](https://ieeexplore.ieee.org/author/37087097339);[Agung Trisetyo](https://ieeexplore.ieee.org/author/37088562776);[Wayan Suparta](https://ieeexplore.ieee.org/author/37086346683); [Edi Abudurahman](https://ieeexplore.ieee.org/author/37088563240)

(<https://ieeexplore.ieee.org/document/9264961>)

[4] Jia-Fong Yeh**,** Pei-Hug Su, ”Snake game AI: Movement rating functions and evolutionary algorithm-based optimization”, November 2016

(<https://www.researchgate.net/publication/309390260_Snake_game_AI_Movement_rating_functions_and_evolutionary_algorithm-based_optimization>)

[5] S. Sharma, S. Mishra, N. Deodhar, A. Katageri and P. Sagar, "Solving The Classic Snake Game Using AI," 2019 IEEE Pune Section International Conference (PuneCon), Pune, India, 2019.

# (<https://ieeexplore.ieee.org/document/9105796>)

# [6] Akshay Katageri, Nachiket Deodhar,” Solving the Classic Snake Game Using AI”, Dec 2019

# (<https://www.researchgate.net/publication/341877231_Solving_The_Classic_Snake_Game_Using_AI>)

[7] Z. Wei, D. Wang, M. Zhang, A. -H. Tan, C. Miao and Y. Zhou, "Autonomous Agents in Snake Game via Deep Reinforcement Learning," 2018 IEEE International Conference on Agents (ICA), Singapore, 2018.

# (<https://ieeexplore.ieee.org/document/8460004>)

# [8] Fuad Hussain, “Snake game documentation”, Nov 2022.

# (<https://www.academia.edu/12695666/Snake_game_documentation>)

# [9] Z. Wei, D. Wang, M. Zhang, A. -H. Tan, C. Miao and Y. Zhou, “Autonomous agents in snake game via deep reinforcement learning”

# (<https://dr.ntu.edu.sg/handle/10356/89882>)

# [10] Snake (video game genre)

# (<https://en.wikipedia.org/wiki/Snake_(video_game_genre)>)