**Visvesvaraya Technological University**

**JNANA SANGAMA**

**Belagavi-590018, Karnataka**



**Computer Graphics Mini-Project Report On**

**“SNAKE GAME”**

**Submitted in partial fulfilment of requirements for the award of**

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**Submitted By**

**Mr.Narayan.G.P USN:2BU20CS035**

**Mr.Deepak.S.B USN:2BU21CS405**

**Ms.Gouri.S.R USN:2BU21CS406**

**Ms.Mrunal.P.M USN:2BU21CS413**

Under the Guidance of

**Mrs.Sushiladevi.V**

**Asst. Prof, Dept. Of CSE**

 **SUBJECT CODE: 18CSL67**

**Department of Computer Science & Engineering**

**Accredited by NBA**

**S.G. BALEKUNDRI INSTITUTE OF TECHNOLOGY**

***An ISO 21001:2018 certified institution***

**Shivabasava Nagar, Belagavi-590010**

**Academic Year: 2022-2023**

**S.S.E.T’S**

**S.G. BALEKUNDRI INSTITUTE OF TECHNOLOGY**

***An ISO 21001:2018 certified institution***

**SHIVABASAVA NAGAR, BELAGAVI-590010**



**DEPARTMENT OF**

**COMPUTER SCIENCE & ENGINEERING**

**Accredited by NBA**

**CERTIFICATE**

This is to certify that the project work titled **SNAKE GAME** is a Bonafide work satisfactorily completed by **Mr.Narayan.G.P (2BU20CS035), Mr.Deepak.S.B(2BU21CS405), Ms.Gouri.S.R(2BU21CS406) and Ms.Mrunal.P.M(2BU21CS413)** in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering under Visvesvaraya Technological University, Belagavi, for the year 2022-2023. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the Report deposited in the departmental library. The Mini project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

Guide and Course Coordinator HOD

**Mrs.Sushiladevi.V Dr. B. S. Halakarnimath**

Principal

**Dr. B. R. Patagundi**

**External Viva**

Name of the Examiner’s Signature with Date

1)

2)

**Abstract**

The Snake Game is a classic arcade game that involves controlling a snake to navigate and consume food items while avoiding collision with walls or its own tail. This project aims to implement the Snake Game using the OpenGL (Open Graphics Library) library for graphics rendering.

The game is a classic arcade game where the player controls a snake that moves around a grid, eating food items and growing longer. The objective is to eat as much food as possible without colliding with the boundaries of the grid or the snake's own body.

The implementation of the Snake game using GLUT involves utilizing the features provided by the library for creating a window, handling keyboard input, and rendering graphics. The game consists of several key components, including the snake, food items, and game logic.

# Table of Contents

**ACKNOWLEDGEMENT I**

**ABSTRACT II**

**TABLE OF CONTENTS III**

**LIST OF FIGURES I**

|  |  |  |
| --- | --- | --- |
| **Chapter**  **No.** | **Title** | **Page.No** |
| 1 | Introduction | 1 |
| 2 | System Requriments | 2 |
| 3 | Design | 3 |
| 4 | Implementation | 5-15 |
| 5 | Snapshots | 16-17 |
| 6 | Conclusion | 18-19 |

|  |
| --- |
|  |
|  |
|  |

# List of Figures

|  |  |  |
| --- | --- | --- |
| **Figure No** | **Figure Name** | **Page No.** |
| Figure 3.1 | Flow Chart | 3 |
| Figure 5.1 | Game Start | 16 |
| Figure 5.2 | Pause | 16 |
| Figure 5.3 | Winning Game | 17 |
| Figure 5.4 | Game Over | 17 |

**Chapter 1**

**INTRODUCTION**

The Snake Game is a classic video game that has entertained players for decades. In this project report, we present the implementation details of a Snake Game, a simple and addictive game where the player controls a snake that moves around the screen, consuming food and growing in length. The objective is to achieve the highest score possible by strategically maneuvering the snake without colliding with walls or its own body.

**1.1 The main objectives of this project are as follows:**

1. Develop a functional Snake Game with intuitive gameplay.
2. Implement the core mechanics of the game, including snake movement, food generation.
3. Create a visually appealing game environment using appropriate graphics and design elements.
4. Provide user-friendly controls for the player to interact with the game.
5. Implement a scoring system to track and display the player's progress.
6. Enhance the game with additional features to make it more engaging and entertaining.

### 1.2 Introduction to OpenGL:

* OpenGL is an open specification for an applications program interface for defining 2D and 3D objects. The specification is cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. It renders 3D objects to the screen, providing the same set of instructions on different computers and graphics adapters. Thus it allows us to write an application that can create the same effects in any operating system using any OpenGL-adhering graphics adapter.
* Computer graphics, a 3-dimensional primitive can be anything from a single point to an n- sided polygon. From the software standpoint, primitives utilize the basic 3-dimensional rasterization algorithms such as Bresenham's line drawing algorithm, polygon scan line fill, texture mapping and so forth. OpenGL's basic operation is to accept primitives such as points, lines and polygons, and convert them into pixels. This is done by a graphics pipeline known as the OpenGL state machine. Most OpenGL commands either issue primitives to the graphics pipeline, or configure how the pipeline processes these primitives.

**Chapter 2**

**SYSTEM REQURIMENTS**

**2.1 Hardware Requirements:**

* Processor: Dual-core processor or higher
* RAM: 2 GB or more
* Graphics Card: OpenGL 2.0 compatible graphics card
* Display: Minimum resolution of 1024x768 pixels
* Input: Keyboard or game controller

**2.2 Software Requirements:**

* Operating System: Windows, macOS, or Linux
* OpenGL: The game relies on OpenGL for graphics rendering, so an OpenGL library must be installed on the system.
* Development Environment: A C++ compiler and development environment such as Visual Studio, Xcode, or Code::Blocks to compile and run the code.
* GLUT Library: The code utilizes the GLUT library (OpenGL Utility Toolkit) for window management and input handling. The GLUT library must be installed and configured properly.

**Chapter 3**

**DESIGN**

**3.1 Flow Chart:**

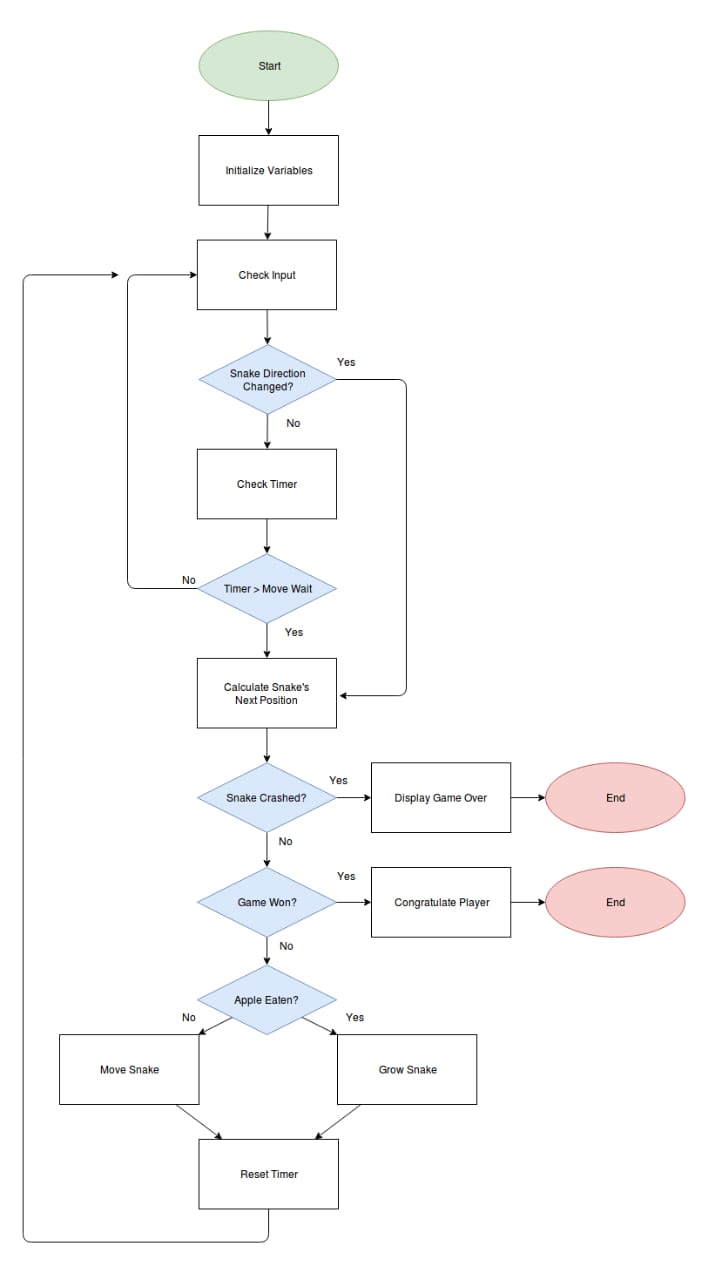


Figure 3.1 Flowchart Of Snake Game

**3.2: Working Principle:**

1. Initialization: The game initializes the snake’s starting position, generates the initial food position, and sets the score and level to their initial values. It also sets the initial game state variables.

2. Drawing the Scene: The game uses OpenGL to draw the game scene on the screen. It draws the snake, food, score, and level using different colors and font styles.

3. Update Function: The update function is responsible for updating the game state. It moves the snake according to the current direction, checks for collisions with the food, updates the score and level, and checks for game over conditions (snake collision or reaching 100 points).

4. Input Handling: The game handles keyboard input from the user. The arrow keys and ‘W’, ‘S’, ‘A’, ‘D’ keys are used to change the snake’s direction. The spacebar is used to toggle the pause state.

5. Game Loop: The main game loop continuously redraws the game scene, updates the game state, and handles user input. It uses a timer to control the speed of the snake and the game’s overall difficulty.

6. Game Over and Win States: When the game detects a collision or the player reaches 100 points, it sets the corresponding game state variables (isGameOver or hasWon) to true. These states are used to display the “Game Over” or “Congratulations” messages on the screen.

7. Initialization and Main Function: The game initializes OpenGL settings, sets up the display and input functions, and starts the main loop.

Overall, the code implements the logic for controlling the snake, detecting collisions, managing the game state, and rendering the game scene using OpenGL. It provides a basic framework for a playable Snake game.

**Chapter 4**

**IMPLEMENTATION**

**4.1 Coding:**

#include <GL/glut.h>

#include <cstdio>

#include <cstdlib>

#include <string>

// Game constants

const int WIDTH = 800; // Width of the game window

const int HEIGHT = 600; // Height of the game window

const int GRID\_SIZE = 20; // Size of each grid cell

const int GRID\_WIDTH = WIDTH / GRID\_SIZE;

const int GRID\_HEIGHT = HEIGHT / GRID\_SIZE;

// Snake variables

int snakeX[100], snakeY[100]; // Arrays to store snake segments

int snakeLength = 3;

int direction = 3; // 1: Up, 2: Down, 3: Left, 4: Right

// Food variables

int foodX, foodY;

// Score and level variables

int score = 0;

int level = 1;

// Game state variables

bool isPlaying = true;

bool isPaused = false;

bool isGameOver = false;

bool hasWon = false;

// Initialize the game

void initGame() {

// Initialize snake position

snakeX[0] = GRID\_WIDTH / 2;

snakeY[0] = GRID\_HEIGHT / 2;

// Generate initial food position

foodX = rand() % GRID\_WIDTH;

foodY = rand() % GRID\_HEIGHT;

// Reset score and level

score = 0;

level = 1;

// Reset game state variables

isPlaying = true;

isPaused = false;

isGameOver = false;

hasWon = false;

}

// Draw the game scene

void drawScene() {

glClear(GL\_COLOR\_BUFFER\_BIT);

if (isPlaying) {

// Draw snake

glColor3f(0.0, 1.0, 0.0); // Green color for the snake

for (int i = 0; i < snakeLength; i++) {

glRectf(snakeX[i] \* GRID\_SIZE, snakeY[i] \* GRID\_SIZE,

(snakeX[i] \* GRID\_SIZE) + GRID\_SIZE,

(snakeY[i] \* GRID\_SIZE) + GRID\_SIZE);

}

// Draw food

glColor3f(1.0, 0.0, 0.0); // Red color for the food

glRectf(foodX \* GRID\_SIZE, foodY \* GRID\_SIZE,

(foodX \* GRID\_SIZE) + GRID\_SIZE,

(foodY \* GRID\_SIZE) + GRID\_SIZE);

// Draw score and level

glColor3f(1.0, 1.0, 1.0); // White color for the score and level

glRasterPos2f(10, HEIGHT - 30);

std::string scoreString = "Score: " + std::to\_string(score);

for (char c : scoreString) {

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, c);

}

glRasterPos2f(WIDTH - 90, HEIGHT - 30);

std::string levelString = "Level: " + std::to\_string(level);

for (char c : levelString) {

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, c);

}

if (isPaused) {

// Draw "Paused" message

glColor3f(1.0, 1.0, 1.0);

glRasterPos2f((WIDTH / 2) - 30, HEIGHT / 2);

std::string pausedString = "Paused";

for (char c : pausedString) {

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, c);

}

}

}

else {

// Game over or won state

if (isGameOver) {

// Draw "Game Over" message

glColor3f(1.0, 1.0, 1.0);

glRasterPos2f((WIDTH / 2) - 40, HEIGHT / 2);

std::string gameOverString = "Game Over";

for (char c : gameOverString) {

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, c);

}

}

else if (hasWon) {

// Draw "Congratulations" message

glColor3f(1.0, 1.0, 1.0);

glRasterPos2f((WIDTH / 2) - 60, HEIGHT / 2);

std::string congratsString = "Congratulations! You Won!";

for (char c : congratsString) {

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, c);

}

}

// Draw score and level

glColor3f(1.0, 1.0, 1.0); // White color for the score and level

glRasterPos2f(10, HEIGHT - 30);

std::string scoreString = "Score: " + std::to\_string(score);

for (char c : scoreString) {

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, c);

}

glRasterPos2f(WIDTH - 90, HEIGHT - 30);

std::string levelString = "Level: " + std::to\_string(level);

for (char c : levelString) {

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, c);

}

}

glutSwapBuffers();

}

// Update the game state

void update(int value) {

if (!isGameOver && isPlaying && !isPaused) {

// Move the snake and update the game state

for (int i = snakeLength - 1; i > 0; i--) {

snakeX[i] = snakeX[i - 1];

snakeY[i] = snakeY[i - 1];

}

// Update the snake's head position based on the direction

switch (direction) {

case 1: // Up

snakeY[0]++;

break;

case 2: // Down

snakeY[0]--;

break;

case 3: // Left

snakeX[0]--;

break;

case 4: // Right

snakeX[0]++;

break;

}

// Wrap the snake around the grid boundaries

if (snakeX[0] < 0)

snakeX[0] = GRID\_WIDTH - 1;

else if (snakeX[0] >= GRID\_WIDTH)

snakeX[0] = 0;

if (snakeY[0] < 0)

snakeY[0] = GRID\_HEIGHT - 1;

else if (snakeY[0] >= GRID\_HEIGHT)

snakeY[0] = 0;

// Check if the snake has collided with the food

if (snakeX[0] == foodX && snakeY[0] == foodY) {

// Increase the snake's length

snakeLength++;

// Generate new food position

foodX = rand() % GRID\_WIDTH;

foodY = rand() % GRID\_HEIGHT;

// Increase the score

score++;

// Check if level needs to be increased

if (score % 10 == 0) {

level++;

}

}

// Check if the snake has collided with itself

for (int i = 1; i < snakeLength; i++) {

if (snakeX[0] == snakeX[i] && snakeY[0] == snakeY[i]) {

// Game over

isGameOver = true;

isPlaying = false;

break;

}

}

// Check if the player has reached 100 points

if (score == 100) {

hasWon = true;

isPlaying = false;

}

}

// Calculate the timer interval based on the current level \*increase or decrease speed of snake(100) lower to increase speed\*

int timerInterval = 100 - (level \* 10);

// Update the game with the adjusted timer interval

glutTimerFunc(timerInterval, update, 0);

// Trigger redrawing of the game scene

glutPostRedisplay();

}

// Handle keyboard input

void handleKeypress(unsigned char key, int x, int y) {

switch (key) {

case 'w': case 'W': // Move up

if (direction != 2)

direction = 1;

break;

case 's': case 'S': // Move down

if (direction != 1)

direction = 2;

break;

case 'a': case 'A': // Move left

if (direction != 4)

direction = 3;

break;

case 'd': case 'D': // Move right

if (direction != 3)

direction = 4;

break;

case ' ': // Spacebar to toggle pause state

isPaused = !isPaused;

break;

}

}

// Handle arrow key input

void handleSpecialKeypress(int key, int x, int y) {

switch (key) {

case GLUT\_KEY\_UP: // Move up

if (direction != 2)

direction = 1;

break;

case GLUT\_KEY\_DOWN: // Move down

if (direction != 1)

direction = 2;

break;

case GLUT\_KEY\_LEFT: // Move left

if (direction != 4)

direction = 3;

break;

case GLUT\_KEY\_RIGHT: // Move right

if (direction != 3)

direction = 4;

break;

}

}

// Initialize the game and OpenGL settings

void init() {

glClearColor(0.0, 0.0, 0.0, 1.0);

gluOrtho2D(0, WIDTH, 0, HEIGHT);

initGame();

}

int main(int argc, char\*\* argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(WIDTH, HEIGHT);

glutCreateWindow("Snake Game");

glutDisplayFunc(drawScene);

glutKeyboardFunc(handleKeypress);

glutSpecialFunc(handleSpecialKeypress);

init();

glutTimerFunc(100, update, 0);

glutMainLoop();

return 0;

}

**Chapter 5**

**RESULTS**

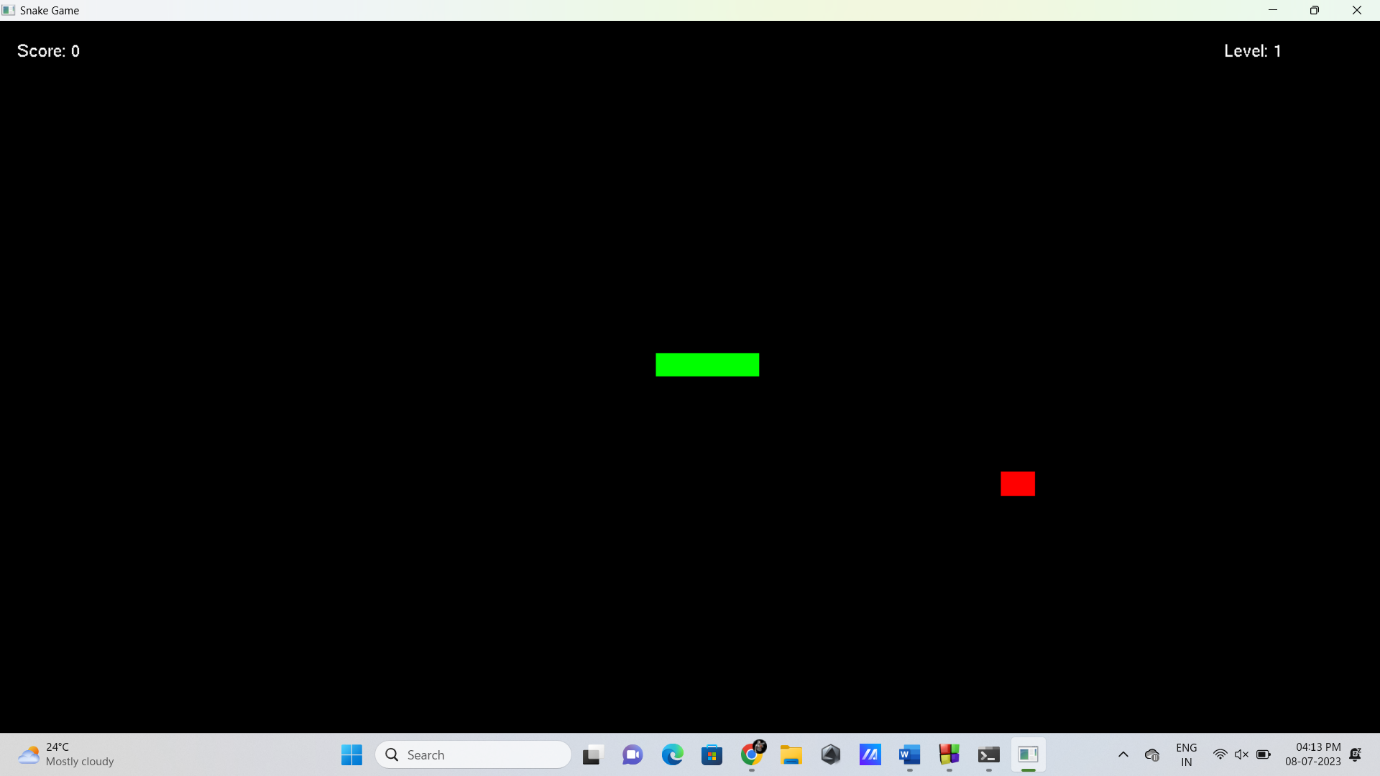
****

Figure5.1: Game Start (Level 1)

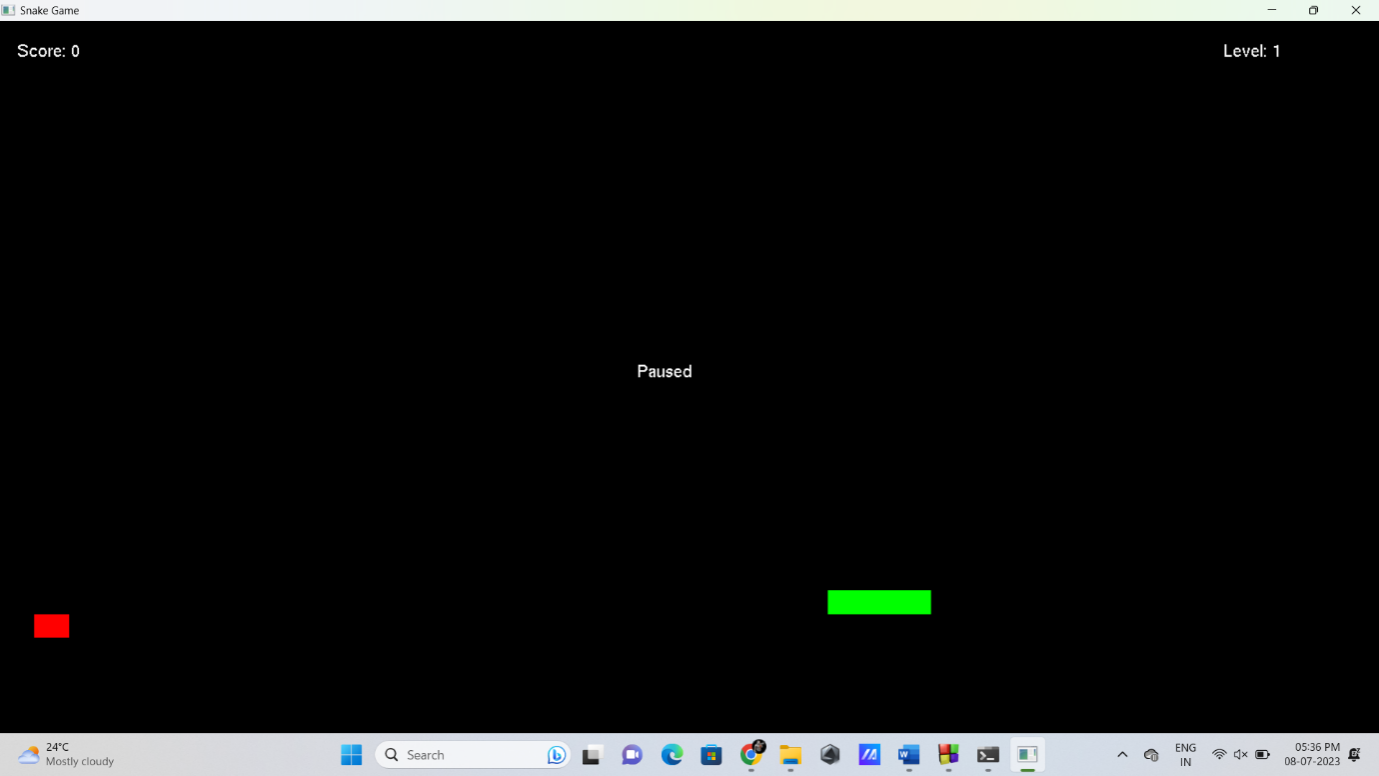


Figure5.2: Pause

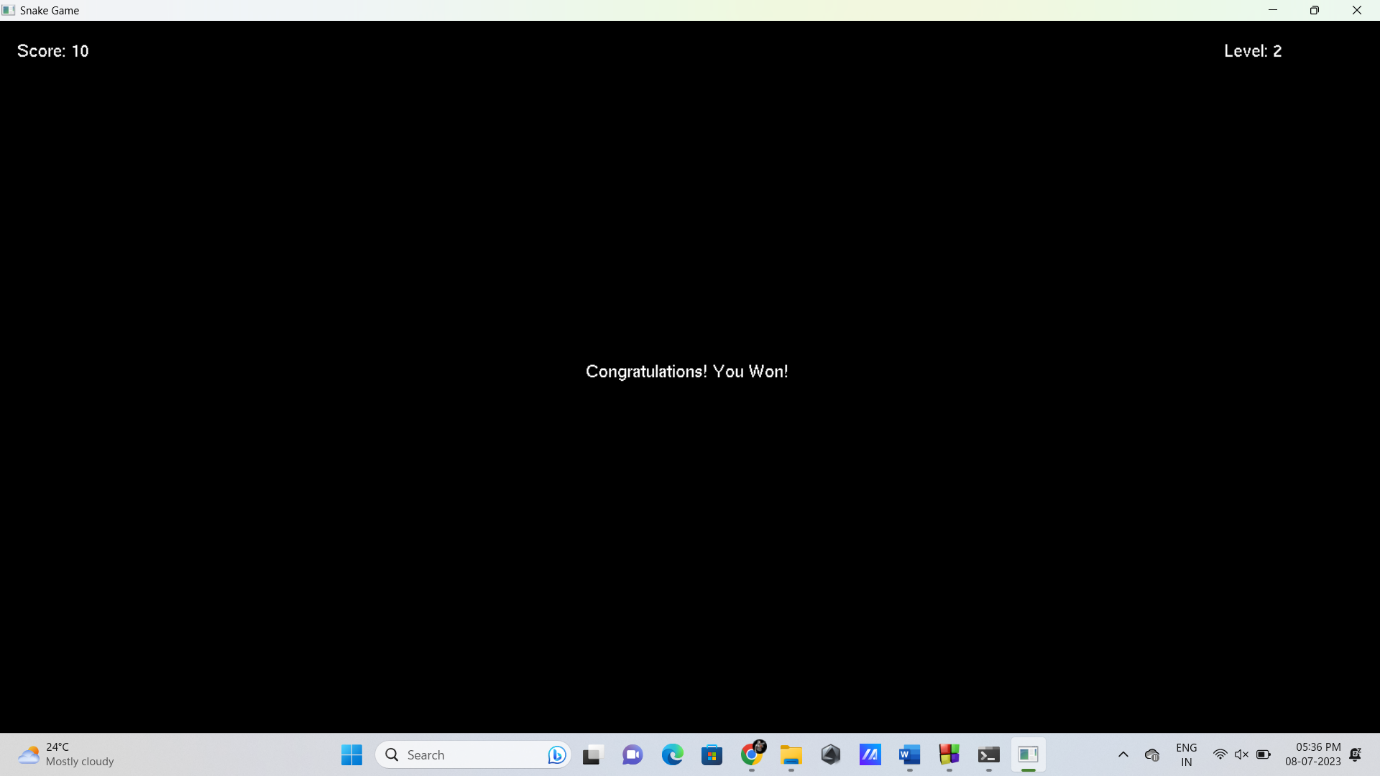


Figure5.3: Winning Game

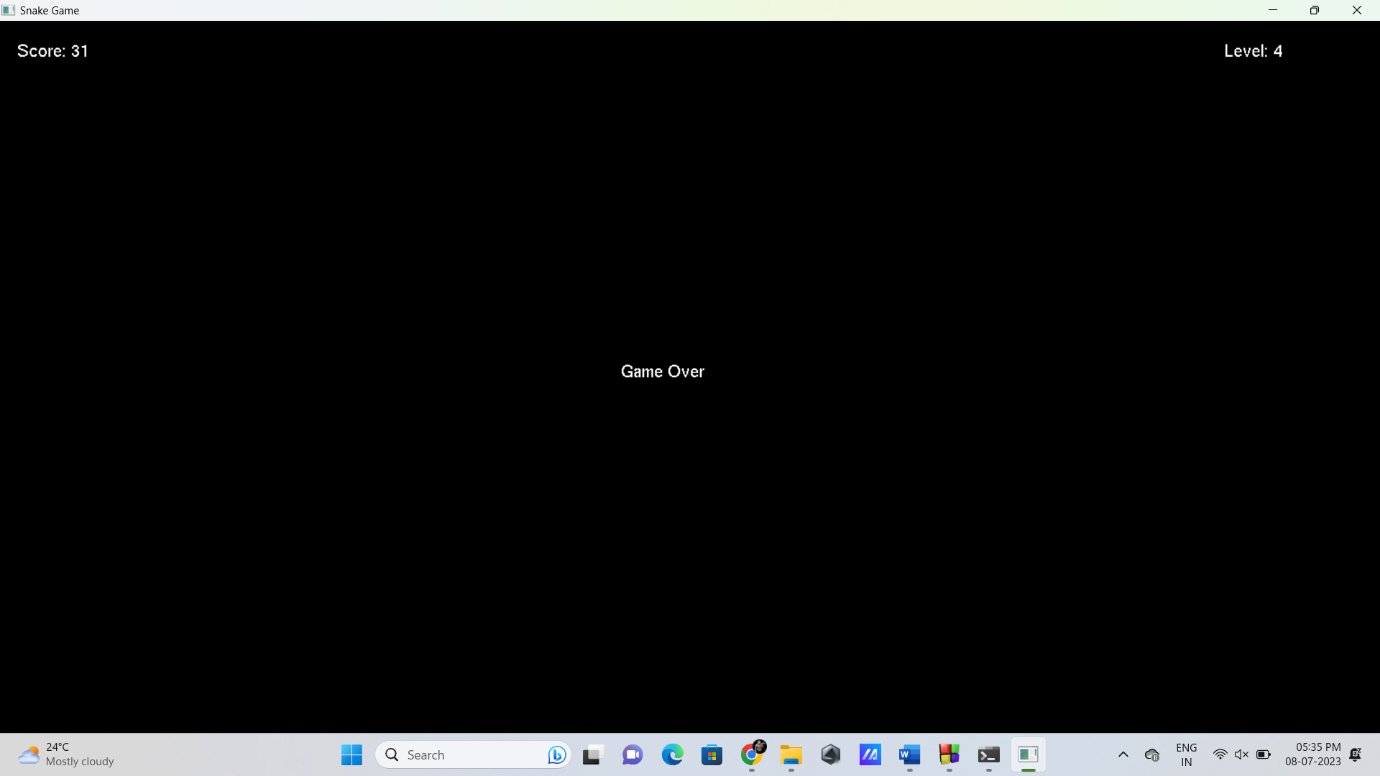


Figure5.4: Game over

**Chapter 6**

**CONCLUSION**

In conclusion, the provided Snake game code demonstrates a simple implementation of the classic Snake game using the OpenGL library. It showcases key game development concepts such as game state management, user input handling, collision detection, and rendering. The code allows the player to control a snake that moves around the game grid and consumes food to grow longer. The game features a scoring system and level progression based on the player's performance. The game ends when the snake collides with itself or when the player reaches a score of 100, displaying the appropriate "Game Over" or "Congratulations" messages.

While the provided code offers a functional Snake game, it can be further enhanced and expanded. Additional features such as sound effects, power-ups, different game modes, and improved graphics could be incorporated to enhance the gameplay experience. Overall, this implementation serves as a starting point for building more complex and feature-rich Snake games, and it provides a valuable learning resource for understanding game development concepts and OpenGL programming.

**REFERENCES**

1] "Computer Graphics: Principles and Practice" by James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes: This book is a comprehensive guide to computer graphics, covering both fundamental concepts and advanced techniques. It provides a solid foundation in computer graphics theory and practical implementation.

2] "OpenGL Programming Guide: The Official Guide to Learning OpenGL" by Dave Shreiner, Graham Sellers, John M. Kessenich, Bill M. Licea-Kane: This guide focuses on using the OpenGL library for computer graphics programming. It covers topics such as creating graphics contexts, rendering techniques, shaders, and more.