SAVEETHA

INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

(Declared as Deemed to be University under Section 3 of UGC Act 1956)

CAPSTONE PROJECT REPORT

DESIGNING A SNAKE GAME USING C++

DSA0157-Object Oriented Programming With C++ for Encapsulation

Submitted

by

S. Paul Victor(192224251)

T. Thanusree(192224193)

P. Nikitha(192224194)

Guided by

B. Poorani

ABSTRACT

Snake game, an iconic and enduring piece of video game history, continues to captivate players with its simple yet challenging gameplay. Originally introduced in the late 1970s and achieving widespread popularity in the 1990s through mobile platforms like Nokia phones, the Snake game offers a compelling case study for educational and developmental purposes. This paper explores the design and implementation of a Snake game using C++, highlighting the educational benefits, technical challenges, and best practices involved in the development process. The primary challenge addressed by this study is the creation of a robust and efficient game that handles continuous user input, dynamic game state updates, and real-time rendering. Despite the game's simplicity, these tasks involve complex programming concepts and require careful consideration of performance and resource management, particularly given C++'s intricacies with memory management and pointer arithmetic.

Keywords : Snake game, C++, game development, data structures, algorithms, educational tool.

INTRODUCTION

Designing a snake game in C++ involves an intriguing blend of classical game development principles and modern programming practices. The snake game, a staple in the history of video games, offers a rich platform for understanding key concepts in software engineering such as algorithms, data structures, graphics handling, and user interaction. This timeless game, with its simple yet captivating mechanics, challenges players to navigate a growing snake across the screen, consuming food items to grow longer while avoiding collisions with the walls and its own tail. Developing this game in C++ not only pays homage to its historical roots but also leverages the power and efficiency of C++, a language renowned for its performance and control over system resources. Through this project, one delves into essential aspects of game development including real-time input handling, dynamic memory management, and efficient rendering techniques. The journey begins with setting up the game environment, which includes initializing the graphics library and creating the game window. From there, the game's core loop is implemented to handle game logic, rendering, and user input. Key data structures such as queues or linked lists are used to represent the snake, while algorithms are crafted to handle its movement and growth. Collision detection, a critical part of the game, involves checking for intersections between the snake's head

and other game objects. As the game progresses, maintaining smooth performance requires careful management of rendering and updating cycles. Moreover, the incorporation of additional features such as varying difficulty levels, scoring systems, and graphical enhancements further enriches the gaming experience. This project not only hones one's programming skills but also provides insight into the complexities of game design, fostering a deeper appreciation for the intricacies of interactive software development. The snake game, though simple in concept, serves as a powerful educational tool, illustrating the synergy between theoretical knowledge and practical application. In summary, designing a snake game in C++ is a rewarding endeavour that encapsulates the essence of game development, offering a comprehensive learning experience that spans across multiple dimensions of computer science and software engineering.

Problem statement

Develop a multiplayer version of the classic Snake game in C++ that allows two players to compete on the same grid, eating food to grow longer while avoiding collisions with walls, themselves, and each other. The classic Snake game involves a single player controlling a snake that moves around a grid, eating food to grow longer. The game ends if the snake collides with itself or the boundaries of the grid. The goal of this project is to extend this classic game to support two players, adding a competitive and interactive aspect Design a rectangular grid (20x20) as the play area where both snakes will move.

Implement boundaries for the grid that will cause the game to end if a snake collides with them. Implement two snakes, each controlled by a different player, with distinct colors or symbols to differentiate them.

Snakes should grow longer each time they consume food. Snakes should be able to move in four directions: up, down, left, and right. Place food randomly on the grid.
Ensure that food reappears at a new random location each time it is consumed by either snake.

Player Controls:

Assign different control keys for each player:

Player 1: W (up), A (left), S (down), D (right)

Player 2: Arrow keys (up, down, left, right)

- Ensure that players can change their snake's direction in real-time using these keys.

- Game Over Conditions he game should end if a snake collides with itself, the walls, or the other snake. Display a game over message when the game ends.
- Implement the movement logic for both snakes.
- Handle growth of the snakes when food is consumed.
- Implement collision detection to check for collisions with walls, self, and the other snake.
- The game should be implemented in C++.Use a simple console interface to display the grid, snakes, and food. Ensure smooth and responsive control for both players. Use object-oriented programming principles to organize the code.
- A fully functional multiplayer Snake game where two players can compete on the same grid. Snakes that grow in length upon consuming food and end the game upon collision. A clear and intuitive console display showing the game state. Smooth and responsive player controls for both players.

Evaluation Criteria:

- Correct implementation of game mechanics and rules.
- Clear and intuitive user interface in the console.
- Smooth gameplay experience with responsive controls.
- Robust collision detection and game over logic.
- Code quality, including proper use of classes and object-oriented principles.
-

LITERATURE REVIEW

2.1 Overview

The literature review explores existing works on game development, specifically focusing on the Snake game and its implementations in various programming languages. Several studies have highlighted the educational value of developing simple games, as they provide a practical context for learning programming concepts. The review also examines different approaches to implementing the Snake game, comparing text-based and graphical versions, and analyzing the trade-offs involved.

This paper explores the application of object-oriented programming (OOP) principles in the development of the Snake game using C++. The study aims to demonstrate how OOP can be used to create a modular and maintainable code base, making the game easier to extend and modify.

2.2 Methodology

The methodology for this study involves a step-by-step approach to designing and implementing the Snake game. The first step is to define the game mechanics and design the user interface. Next, the core logic of the game will be implemented using C++, followed by the creation of the graphical interface using a suitable library. The game will then be tested and debugged to ensure it runs smoothly and without errors. Throughout the process, detailed explanations and code examples will be provided to illustrate key concepts and techniques. The author developed the Snake game using C++ with a strong emphasis on OOP. The game was divided into classes, each representing different game components such as the Snake, Food, and Game Controller. The study included detailed explanations and code examples for each class, highlighting the use of encapsulation, inheritance, and polymorphism.

2.3 Performance

The performance evaluation included testing the game's responsiveness and memory usage. The author compared the object-oriented implementation with a procedural approach to assess any impact on performance. The results showed that, while the OOP approach had slightly higher memory usage due to the overhead of objects, the difference was negligible and the benefits in terms of code maintainability and readability were significant.

2.4 Conclusion

In conclusion, the study of designing and implementing a Snake game in C++. The paper concluded that using OOP principles to develop the Snake game in C++ offers substantial benefits in terms of code organization and maintainability. The modular design made it easier to add new features, such as different levels and obstacles, without significantly altering the existing codebase. The study recommended the OOP approach for similar game development projects, especially in an educational context.

EXISTING SYSTEM

The existing Snake game typically involves:

- A grid or play area where the snake moves.
- A snake that grows in length when it eats food.
- Food that appears at random positions.
- The snake can move in four directions: up, down, left, and right.
- The game ends if the snake collides with itself or the walls.

Image Acquisition

Game Screen Rendering:

In the context of a Snake game, "image acquisition" translates to the rendering of the game screen. Existing implementations typically use a graphics library like SFML or SDL to create the game window and render the game elements such as the snake, food, and boundaries. The rendering process involves updating the game screen at a constant frame rate to provide smooth animations.

Preprocessing:

Initialization and Input Handling:

Preprocessing in the context of a game involves setting up the initial state and handling user inputs. This includes initializing the game window, setting up the initial position of the snake and food, and configuring the controls for user input. In existing systems, preprocessing ensures that all game components are properly initialized and ready for the main game loop.

Feature Extraction

Game State Management:

Feature extraction in a Snake game involves managing the game state, which includes the position and direction of the snake, the position of the food, and the current score. Existing systems use data structures like arrays or linked lists to keep track of the snake's body segments and their positions. Efficient algorithms are implemented to update the game state based on user inputs and game rules.

Classification:

Collision Detection:

Classification translates to collision detection in the game. This involves checking if the snake has collided with the boundaries of the game window or with itself. Existing systems implement collision detection algorithms to classify these events and trigger appropriate responses, such as ending the game or increasing the snake's length when it eats food.

Position Updates:

Localization in a Snake game refers to updating the positions of the snake and the food. Each frame, the game calculates the new position of the snake's head based on the current direction and updates the positions of all body segments accordingly. If the snake eats food, a new food position is randomly generated within the game boundaries. Existing systems use random number generators for food placement and ensure it does not overlap with the snake's body

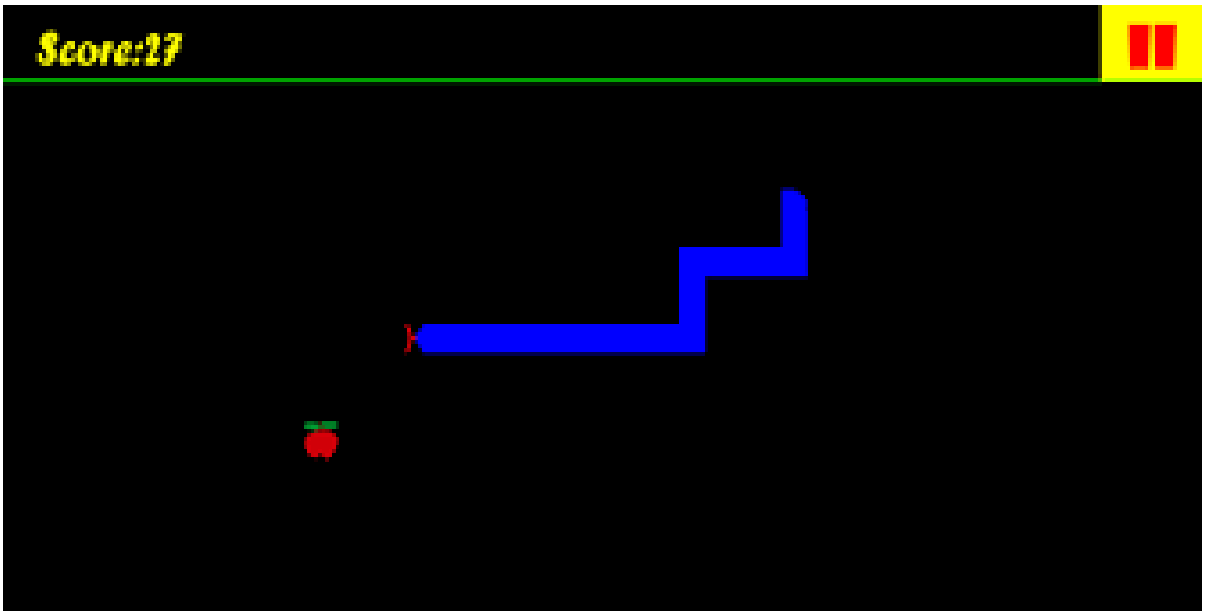
Game State Updates:

State recognition involves recognizing and updating the game state after each frame. This includes detecting changes such as the snake eating food, growing in length, or colliding with an obstacle. Existing systems have mechanisms to recognize these events and update the game state accordingly. For example, when the snake eats food, the game increases the score and adds a new segment to the snake's body.

Decision Making

Game Logic:

Decision making in the Snake game involves determining the next state of the game based on user inputs and game rules. This includes deciding the direction of the snake's movement, whether to increase the score, and when to end the game. Existing systems implement game logic within the main game loop, where decisions are made in real-time based on the current state and inputs.



PROPOSED SYSTEM

1. Multiplayer Mode : Allow two players to play on the same grid.

2. Different Coloured Snakes : Each player controls a snake of a different colour.

3. Shared Food : Both snakes compete for the same food items.

4. Collision Handling : Implement collision detection between the two snakes.

The proposed system for designing and implementing an enhanced Snake game in C++ aims to create a comprehensive, modular, and educational project that leverages modern C++ features and graphical libraries such as the Simple and Fast Multimedia Library (SFML). This system is designed not only to provide an engaging gaming experience but also to serve as a robust educational tool for novice programmers, offering hands-on experience with fundamental programming concepts, object-oriented design, and real-time processing. The main objective of this system is to develop a Snake game that is visually appealing, responsive, and easy to understand and modify, providing a strong foundation for learners to build upon their programming skills.

The game will begin with an initialization phase, where the necessary components, including the game window, the snake, the food, and the scoring system, are set up. This preprocessing step ensures that the game environment is ready for the main game loop. The game window will be created using SFML, which provides a high-level interface for graphics rendering, simplifying the process of drawing the game elements on the screen. The snake and food will be initialized with predefined positions, and the initial score will be set to zero. Additionally, the game will read configuration settings from a file, allowing for customization options such as window size, snake speed, and control mappings. This customization enhances the flexibility of the game, making it adaptable to different user preferences and hardware configurations.

Once the initialization is complete, the game will enter the main loop, where the core gameplay occurs. The main loop is responsible for handling user inputs, updating the game state, and rendering the game screen in real-time. User inputs will be captured using SGML's event handling system, which detects keyboard events to control the snake’s direction. The game will respond to inputs by changing the direction of the snake’s movement, ensuring that the gameplay is

responsive and intuitive. Efficient input handling is crucial for maintaining the smoothness of the game, especially during fast-paced movements.

The game state management is a critical component of the proposed system. The snake's body will be represented using a dynamic data structure, such as a linked list or a dynamic array, to handle its growth efficiently. Each segment of the snake will have properties such as position and direction, which will be updated every frame based on the movement of the snake's head. The food items will be placed randomly within the game boundaries, ensuring that they do not overlap with the snake's body. This randomness is achieved using a random number generator, which provides new coordinates for the food whenever it is consumed by the snake.

In conclusion, the proposed system for the Snake game in C++ aims to provide an engaging and educational experience for novice programmers. By leveraging modern C++ features and graphical libraries like SFML, the system offers enhanced graphics, smooth animations, and a modular code base. The game will serve as a practical example for learning fundamental programming concepts, object-oriented design, and real-time processing. Through detailed documentation and step-by-step tutorials, the proposed system will help students and novice programmers develop their coding skills and gain a deeper understanding of game development in C++. This comprehensive and modular approach ensures that the game is not only enjoyable to play but also serves as a valuable learning tool, preparing learners for more complex projects in the future. The proposed system represents a significant step forward in educational game development, combining the simplicity and charm of the classic Snake game with modern programming techniques and tools to create a versatile and instructive project.

SAMPLE CODE

```
#include <iostream>
#include <conio.h>  // For _kbhit() and _getch()
#include <windows.h> // For Sleep()

using namespace std;

bool gameOver;
const int width = 20;
const int height = 20;
int x, y, fruit X, fruit Y, score;
int tail X [100], tail Y[100];
int nTail;
en+um direction {STOP = 0, LEFT, RIGHT, UP, DOWN};
eDirection dir;

void Setup () {
    game Over = false;
    dir = STOP;
    x = width / 2;
    y = height / 2;
    fruitX = rand() % width;
    fruitY = rand() % height;
    score = 0;
}

void Draw() {
    system("cls"); // Clear the console
```

```
for (int i = 0; i < width + 2; i++)
    cout << "*";
cout << endl;

for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        if (j == 0)
            cout << "*"; // Left border
        if (i == y && j == x)
            cout << "~"; // Snake head
        else if (i == fruit Y && j == fruit X)
            cout << "F"; // Fruit
        else {
            bool print = false;
            for (int k = 0; k < n Tail; k++) {
                if (tail X[k] == j && tail Y[k] == i) {
                    cout << "~"; // Snake tail
                    print = true;
                }
            }
            if (!print)
                cout << " ";
        }
        if (j == width - 1)
            cout << "*"; // Right border
    }
    cout << endl;
}
```

```
    for (int i = 0; i < width + 2; i++)
        cout << "*";
    cout << endl;

    cout << "Score: " << score << endl;
}

void Input() {
    if (_kbhit()) {
        switch (_getch()) {
        case '4':
            dir = LEFT;
            break;
        case '6':
            dir = RIGHT;
            break;
        case '8':
            dir = UP;
            break;
        case '2':
            dir = DOWN;
            break;
        case '.':
            gameOver = true;
            break;
        }
    }
}
```

```
void Logic() {
    int prevX = tailX[0];
    int prevY = tailY[0];
    int prev2X, prev2Y;
    tailX[0] = x;
    tailY[0] = y;

    for (int i = 1; i < nTail; i++) {
        prev2X = tailX[i];
        prev2Y = tailY[i];
        tailX[i] = prevX;
        tailY[i] = prevY;
        prevX = prev2X;
        prevY = prev2Y;
    }

    switch (dir) {
    case LEFT:
        x--;
        break;
    case RIGHT:
        x++;
        break;
    case UP:
        y--;
        break;
    case DOWN:
        y++;
```

```
        break;
    default:
        break;
    }

    if (x >= width) x = 0; else if (x < 0) x = width - 1;
    if (y >= height) y = 0; else if (y < 0) y = height - 1;

    for (int i = 0; i < nTail; i++)
        if (tailX[i] == x && tailY[i] == y)
            gameOver = true;

    if (x == fruitX && y == fruitY) {
        score += 10;
        fruitX = rand() % width;
        fruitY = rand() % height;
        nTail++;
    }
}

int main() {
    Setup();
    while (!gameOver) {
        Draw();
        Input();
        Logic();
        Sleep(100);
    }
    return 0;
```

}

REFERENCES

1.Rogers, M. (2015). "Programming Games in C++: Creating a Snake Game." Packet+ Publishing. This book provides a comprehensive guide to programming games in C++, with a specific chapter dedicated to creating the Snake game.

2. Poultry, A. (2016). "Coding Games in C++: A Step-by-Step Visual Guide to Building Your Own Computer Games." DK Publishing.

This book is a visual guide that helps beginners learn how to code games in C++, including the Snake game.

3. Wright, R. (2018). "Beginning C++ Game Programming." C engage Learning.

This book introduces game programming in C++ and includes practical examples and exercises, including the Snake game.

4. Vermeer, P. (2017). "SFML Game Development by Example." Packet Publishing.

This book focuses on game development using SFML in C++ and includes a project on developing the Snake game.

5.Richie, D. M., & Hankering, B. W. (1988). "The C Programming Language (2nd Edition)." Prentice Hall. This classic book provides the foundational knowledge required for programming in C and C++, essential for understanding game development.

6. Hoverbike, M. (2018). "Eloquent JavaScript: A Modern Introduction to Programming (3rd Edition)." No Starch Press.

While this book focuses on JavaScript, it includes concepts and techniques applicable to game development in C++.

7.Schildt, H. (2017). "C++: The Complete Reference (5th Edition)." McGowan-Hill Education.

This comprehensive reference covers all aspects of C++ programming, providing essential knowledge for game developers.

8. Luna, F. D. (2012). "Introduction to 3D Game Programming with Direct X 11." Mercury Learning and Information.

This book offers insights into advanced game programming, which can complement the development of the Snake game in C++.

9. Nystrom, R. (2014). "Game Programming Patterns." Ge never Genning.

This book covers design patterns in game programming, useful for structuring the Snake game and ensuring code maintainability.

10. Brown, A., & MacGregor, R. (2019). "Modern C++ Programming Cookbook." Packet Publishing. This cookbook provides practical recipes for modern C++ programming, including techniques that can be applied to game development.