Capital University of Science & Technology

Term Project Proposal

Department of Electrical and Computer Engineering

Project Title	Snake Game
Course Title	Application of Information and Communication Technologies lab

Sr. No.	**Student Name**	**Registration Number**
01.	Abdul Raffay	BCPE243029
02.	Abdullah Khan	BCPE243016

Contents

1. Introduction

The Snake game is one of the most iconic and enduring video games, originally developed in the late 1970s and gaining widespread popularity on mobile phones in the late 1990s. In the game, the player controls a snake that moves around the screen, consuming food items that randomly appear. Each time the snake eats, it grows longer, making the game progressively more challenging as the player must avoid crashing into the walls or its own body. The game ends when either of these collisions occurs, and the player’s score is determined by the number of food items consumed before the game ends.

This project aims to develop a simplified version of the classic Snake game using the C++ programming language. The main goal is to implement a functional and interactive game that replicates the key features of the original, such as snake movement, collision detection, food generation, and score tracking. By doing so, this project provides an opportunity to apply fundamental programming concepts like loops, arrays, functions, conditionals, and random number generation in a practical setting.

The game is designed to be both simple and enjoyable, focusing on core mechanics and user interaction. The player's ability to control the snake’s movement with the arrow keys, navigate through the grid, and consume food to grow the snake, offers a basic yet entertaining gameplay experience. In addition to these features, the game includes a real-time score tracker that updates as the player consumes food, providing an additional layer of challenge and motivation to keep playing.

This project not only showcases the application of basic programming techniques but also presents an opportunity to explore game design principles. The following sections of this report outline the problem statement, the methodology employed during development, the code implementation, results from the game’s testing, and conclusions drawn from the project’s completion.

2. Problem Statement

The goal of this project was to develop a fully functional version of the Snake game, incorporating the following key requirements and additional features:

1. **Snake Movement**: The snake should move in a continuous direction across the grid, with the player being able to control its movement using the arrow keys (up, down, left, and right). The snake's movement should be fluid and responsive to user inputs, with each movement updating the position of the snake's head and body segments.
2. **Food Generation**: The game should randomly generate food items at different locations within the grid. Each time the snake's head eats a food item, it should grow longer by one segment, and the player's score should increase accordingly. The food should never appear at the same position as the snake's body to avoid immediate collisions.
3. **Collision Detection**: The game must detect two types of collisions:
   - **Wall Collision**: If the snake's head touches any of the walls that form the boundaries of the grid, the game should end immediately.
   - **Self Collision**: If the snake's head collides with any part of its own body (i.e., a segment of the tail), the game should end. This feature is essential for maintaining the challenge and gameplay balance.
4. **Score Tracking**: The score should increase by 10 points each time the snake eats food. This score should be clearly visible on the screen throughout the gameplay. Additionally, the score should reset when the game is restarted, and the game should keep track of the highest score achieved during a session.
5. **Game Controls**: The game should provide the player with basic controls to enhance the gameplay experience:
   - **Pause Functionality**: The game should allow the player to pause the game during gameplay by pressing a designated key (e.g., the "P" key).
   - **Restart Option**: The player should be able to restart the game after it ends, resuming from the initial conditions with a reset score and snake size.
   - **Exit Option**: The game should also offer the ability to exit the game at any point by pressing a designated key (e.g., the "X" key).
6. **Game Over Mechanism**: The game should have an intuitive and clear game-over screen that shows the final score, as well as options to restart or exit the game. This adds to the overall user experience and ensures smooth gameplay transitions.

3. Methodology

The game was developed using C++, which offers both efficiency and control over system resources. Below are the detailed steps involved in the methodology of the game development:

3.1 Game Grid Design

The game screen is represented by a 2D grid. The grid's boundaries act as walls, and the snake is confined within this grid. The food items also appear randomly within this grid, ensuring they don't overlap with the snake's body. The grid design allows simple collision detection and smooth movement tracking.

3.2 Snake Movement

The snake's movement is controlled by the arrow keys. Initially, the snake moves to the right, and the direction can be changed using the corresponding arrow key. The snake's body and movement are tracked by storing the positions of the snake's head and its body.

3.3 Food Generation

Food items are generated randomly on the grid. Each time the snake's head reaches the food, it grows longer, and the player's score increases. The random generation of food ensures that the game has an element of unpredictability and challenge.

3.4 Collision Detection

The game incorporates two primary types of collision detection:

1. **Wall Collision:** If the snake's head crosses the boundary of the grid, the game ends.
2. **Self Collision:** If the snake's head collides with any part of its own body, the game ends.

These conditions are checked each time the snake moves.

3.5 Score Tracking

The score is displayed at the top of the screen. The score increases by 10 points each time the snake eats food. The game maintains the score in real-time, allowing the player to see their progress throughout the game.

3.6 Libraries and Tools Used

- **C++ Standard Library:** For essential functions like input/output operations and random number generation.
- **conio.h:** Used for capturing keyboard input in real-time (arrow key movement).
- **system("cls"):** Clears the screen between frames to refresh the game grid.
- **time.h:** Used for random number generation and ensuring food appears in different grid locations.

4. Code Implementation

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <stdlib.h>
#include <windows.h>
#include <time.h>
#include <string.h>

#define LEFT 1
#define RIGHT 2
#define UP 3
#define DOWN 4

// Define colors
const char *colors[] = {"Black", "Blue", "Green", "Aqua", "Red", "Purple", "Yellow", "White", "Gray",
                        "LightBlue", "LightGreen", "LightAqua", "LightRed", "LightPurple", "LightYellow",
"BrightWhite"};

int getColorCode(const char *colorName) {
    for (int i = 0; i < 16; i++) {
        if (_stricmp(colors[i], colorName) == 0) {
            return i;  // Return the color code
        }
    }
    return 7;  // Default to White if no match
}

void setTextColor(int foreground, int background = -1) {
    if (foreground < 0 || foreground > 15) return;

    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    if (background >= 0 && background < 16) {
        SetConsoleTextAttribute(hConsole, foreground | background * 16);  // Combining foreground
and background colors
    } else {
        SetConsoleTextAttribute(hConsole, foreground);  // Only foreground color
    }
}

void setTextColor(const char *foregroundColor, const char *backgroundColor = "") {
    int fgColor = getColorCode(foregroundColor);  // Get foreground color code
    int bgColor = (strlen(backgroundColor) > 0) ? getColorCode(backgroundColor) : 0;  // Default
background to black (0)

    setTextColor(fgColor, bgColor);  // Apply both foreground and background colors
}

void gotoxy(int x, int y) {
    COORD coord;
    coord.X = x;
    coord.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}
```

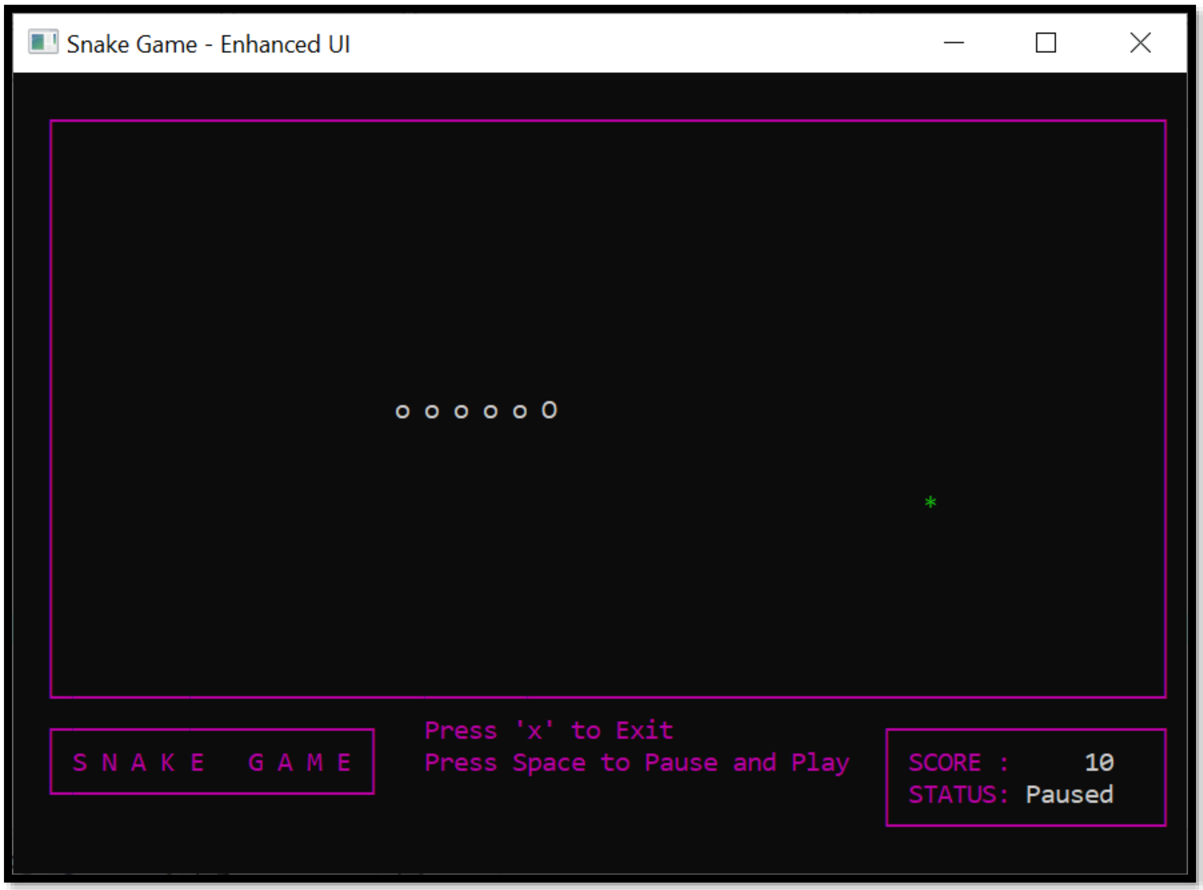
```
void getup() {
    HANDLE hout;
    CONSOLE_CURSOR_INFO cursor;
    hout = GetStdHandle(STD_OUTPUT_HANDLE);
    cursor.dwSize = 1;
    cursor.bVisible = false;
    SetConsoleCursorInfo(hout, &cursor);
    system("mode con:cols=80 lines=25");
    system("title Snake Game - Enhanced UI");
    system("cls");

    setTextColor("LightPurple");

    // Draw the game screen border
    printf("\n  %c", 218);
    for (int x = 0; x < 75; x++) printf("%c", 196);
    printf("%c  ", 191);

    for (int x = 0; x < 17; x++) {
        gotoxy(2, x + 2);
        printf("%c", 179);
        gotoxy(78, x + 2);
        printf("%c ", 179);
    }

    printf("  %c", 192);
    for (int x = 0; x < 75; x++) printf("%c", 196);
    printf("%c  ", 217);

    printf(" %c", 218);
    for (int x = 0; x < 21; x++) printf("%c", 196);
    printf("%c\n", 191);

    printf("  %c S N A K E   G A M E %c\n", 179, 179);
    printf("  %c", 192);
    for (int x = 0; x < 21; x++) printf("%c", 196);
    printf("%c", 217);

    // Draw the status panel
    gotoxy(59, 20);
    printf("%c", 218);
    for (int x = 0; x < 18; x++) printf("%c", 196);
    printf("%c", 191);

    gotoxy(59, 21);
    printf("%c SCORE : 100      %c", 179, 179);

    gotoxy(59, 22);
    printf("%c STATUS: Playing  %c", 179, 179);

    gotoxy(59, 23);
    printf("%c", 192);
    for (int x = 0; x < 18; x++) printf("%c", 196);
    printf("%c", 217);

    gotoxy(28, 20);
    printf("Press 'x' to Exit");
    gotoxy(28, 21);
    printf("Press Space to Pause and Play");
    gotoxy(10, 23);
    setTextColor("white", "blue");

    setTextColor("white");
}

void score(int sc) {
    gotoxy(69, 21);
    printf("%6d", sc * 10);
}
void status(const char *s, int c = 7) {
    gotoxy(69, 22);
    setTextColor(c);
    int x;
    for (x = 0; x < strlen(s); x++) {
        printf("%c", s[x]);
    }
    for (; x < 8; x++) {
        printf(" ");
    }
    setTextColor(7);
}
int main() {
    getup();
    register int flow, size, i, xb, yb;
    int speed, restart = 1, tmp, xpos[100], ypos[100], scr;
    srand(time(NULL));
    while (true) {
        if (restart) {
            status("Playing", 10);
            for (int k = 1; k < 75; k += 2) {
                for (int j = 0; j < 17; j++) {
                    gotoxy(k + 3, j + 2);
                    printf(" ");
                }
            }
            size = 5;
            speed = 200;
            scr = 0;
            score(scr);
            flow = RIGHT;
            xpos[0] = 20;
            for (i = 0; i < size; i++) {
                xpos[i] = xpos[0] - i * 2;
                ypos[i] = 10;
            }
            for (i = 0; i < size; i++) {
                gotoxy(xpos[i], ypos[i]);
                printf("o");
            }
            for (tmp = 1; true;) {
                do {
                    xb = rand() % 75 + 3;
                } while (xb % 2 != 0);
                yb = rand() % 17 + 2;
                for (i = 0; i < size; i++) {
                    if (xb == xpos[i] && yb == ypos[i]) {
                        tmp = 0;
                        break;
                    }
                }
                if (tmp) break;
            }
            gotoxy(xb, yb);
            setTextColor("lightgreen");
            printf("*");  // Improved food representation
            setTextColor(7);
            restart = 0;
        }
        while (!kbhit() && !restart) {
            if (xpos[0] == xb && ypos[0] == yb) {
                for (tmp = 1; true;) {
                    do {
                        xb = rand() % 75 + 3;
                    } while (xb % 2 != 0);
                    yb = rand() % 17 + 2;
                    for (i = 0; i < size; i++) {
                        if (xb == xpos[i] && yb == ypos[i]) {
                            tmp = 0;
                            break;
                        }
                    }
                    if (tmp) break;
                }
                gotoxy(xb, yb);
                setTextColor("lightgreen");
                printf("*");  // Improved food representation
                setTextColor(7);
                size++;
                scr++;
                speed -= 3;
                score(scr);
            }
            gotoxy(xpos[size - 1], ypos[size - 1]);
            for (i = size - 1; i > 0; i--) {
                xpos[i] = xpos[i - 1];
                ypos[i] = ypos[i - 1];
            }
            switch (flow) {
                case RIGHT: xpos[i] += 2; break;
                case LEFT: xpos[i] -= 2; break;
                case UP: ypos[i] -= 1; break;
                case DOWN: ypos[i] += 1;
            }
            tmp = 1;
            for (i = 1; i < size; i++) {
                if (xpos[i] == xpos[0] && ypos[i] == ypos[0]) {
                    tmp = 0;
                    break;
                }
            }
            if (xpos[0] > 76 || xpos[0] < 4 || ypos[0] < 2 || ypos[0] > 18) tmp = 0;
            if (tmp) {
                printf(" ");
                gotoxy(xpos[0], ypos[0]);
                printf("O");
                gotoxy(xpos[1], ypos[1]);
                printf("o");
            } else {
                setTextColor("LIGHTRED");
                printf("o");
                gotoxy(xpos[1], ypos[1]);
                printf("O");
                for (i = 2; i < size; i++) {
                    gotoxy(xpos[i], ypos[i]);
                    printf("o");
                }
                setTextColor(7);
                status("GameOver", 12);
                restart = 1;
                getch();
            }
            Sleep(speed);
        }
        char ch = getch();
        switch (tolower(ch)) {
            case 'x': system("cls");
                      return 0;
            case ' ': status("Paused");
                      while (true) {
                          char z = getch();
                          if (z == 'x')
                              return 0;
                          if (z == ' ')
                              break;
                      }
                      status("Playing", 10);
                      break;
            case -32: {
                char chh = getch();
                if (chh == 72 && flow != DOWN)
                    flow = UP;
                else if (chh == 80 && flow != UP)
                    flow = DOWN;
                else if (chh == 75 && flow != RIGHT)
                    flow = LEFT;
                else if (chh == 77 && flow != LEFT)
                    flow = RIGHT;
                break;
            }
        }
    }
}
```

5. Results and Discussion

The Snake game was successfully developed and tested. The results include the following:

1. **Gameplay:** The snake moves in real-time, controlled by the player using the arrow keys. The snake grows after consuming food, and the game ends when a collision occurs.
2. **Food Generation:** Food appears randomly within the grid. The snake consumes it, increasing its length.
3. **Collision Detection:** The game correctly detects wall and self-collisions, terminating the game as expected.
4. **Score Tracking:** The score is updated in real-time and displayed throughout the gameplay.

Output



```
Snake Game - Enhanced UI

                    o o o o o O

                                              *

S N A K E   G A M E    Press 'x' to Exit                      SCORE :      10
                       Press Space to Pause and Play           STATUS: Paused
```

6. Conclusion

In conclusion, the Snake game was successfully implemented using C++. The project successfully addressed the problem statement and met the required objectives, including snake movement, collision detection, score tracking, and food generation. This project provided hands-on experience with basic C++ programming concepts and game development.

Future work can focus on adding additional features such as sound effects, more complex game mechanics, or a graphical user interface for a more immersive experience.

7. References

1. C++ Programming Language Documentation – For basic syntax and functions used in the implementation.
2. Game Development Concepts – Online tutorials and resources for developing simple games in C++.
3. C++ Standard Library – Official documentation for using libraries such as <conio.h>, , and .