SNAKE GAME USING C++

Submitted by

Aaditya Jha RA2211003010818
Krish Mehta RA2211003010806
Arpit Raj RA2211003010813
Gautam Taneja RA2211003010812

Under the Guidance of

Dr. S Ramesh
Assistant Professor
COMPUTING TECHNOLOGIES

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR - 603203

MAY 2023

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that this Project Report titled **"C++ Snake Game"** is the bonafide work done by Aaditya Jha RA2211003010818 who completed the project under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

SIGNATURE **Dr. S Ramesh** **OODP – Course Faculty** Assistant Professor Department of Computing Technology SRMIST	**SIGNATURE** Dr. M. Pushpalatha **Head of the Department** Department of Computing Technology SRMIST

TABLE OF CONTENTS

Problem Statement

Creating a snake game using C++ and object-oriented programming concepts involves defining the game's rules and objectives, creating a game board using a two-dimensional array, defining the snake and its movements, defining the food and its appearance, implementing the game loop, allowing user input for controlling the snake, and implementing a scoring system. By following these steps, you can create a functional and engaging snake game that demonstrates your understanding of programming principles such as OOP, game mechanics, and user interactions.

CONCEPTS USED:

To make a snake game using C++ OODP (Object Oriented Design Principles), we use the following concepts:

1. Classes: We use classes to define the basic components of the game, such as the Snake, Game, and Food classes.

2. Encapsulation: We use encapsulation to hide the implementation details of each class and provide a public interface for accessing its methods and attributes.

3. Inheritance: We can use inheritance to create subclasses that inherit the attributes and methods of their parent classes. For example, we can create a subclass of the Snake class that has additional functionality, such as the ability to move faster.

4. Abstraction: We use abstraction to simplify the design of the game by representing complex systems with simpler models. For example, we can represent the snake as a series of connected line segments rather than a complex set of movements and collisions.

5. Modularity: We use modularity to break down the game into smaller, more manageable components that can be developed and tested independently. For example, we can develop the Snake class and the Game class separately and then integrate them into the final game.

METHODOLOGY:

The methodology for creating a C++ snake game typically involves several steps:

1. Planning: Define the game's rules and requirements, including game mechanics, user interactions, and game objectives. Determine the size of the game board and the resolution of the game window.
2. Create the game board: Use a two-dimensional array to represent the game board and initialize it with background characters. Define the size of the array based on the game window's resolution.
3. Define the snake: Define the snake's movements and behavior. This includes creating a class or structure to represent the snake, defining the snake's initial position, length, and direction, and creating methods to move the snake and check for collisions with food, walls, or the snake's body.
4. Define the food: Create a class or structure to represent the food and define its behavior. This includes generating food at random positions, detecting collisions with the snake, and increasing the snake's length and score when the food is eaten.
5. Implement the game loop: Create a loop that continuously updates the game's state, renders the game board, and checks for user input. Within the loop, update the position of the snake, check for collisions with food, walls, or the snake's body, and update the score and game status as needed.
6. Implement user input: Allow the user to control the snake using keyboard input. This includes detecting input using functions like "getch()" and updating the snake's direction accordingly.
7. Implement scoring: Create a scoring system that updates the score when the snake eats food. Display the score on the game board.
8. Add finishing touches: Add sound effects, graphics, and other features to enhance the game's aesthetics and playability.

MODULES USED:

This is a C++ program that implements the classic Snake game. Let's break down the code into its different parts:

1. Header Files:

The required header files are included at the beginning of the program, such as iostream for input/output, windows.h for Windows console functions, conio.h for console input/output, stdlib.h for standard library functions, and fstream for file input/output.

2. Struct:

The program defines a structure called "tailpos" to represent each segment of the snake. It includes two integer variables to store the x and y coordinates of the segment, and two pointers to the next and previous nodes of the snake.

3. Snake Class:

The snake class contains all the variables and functions required to handle the game mechanics. It has functions for inserting a new segment of the snake, drawing the snake and the walls, controlling the movement, checking for collisions, and displaying the game over message.

4. Variables:

The snake class has various member variables, including integer variables for the x and y coordinates of the walls, score, and food, handle for the console, and COORD struct for the current coordinate of the console cursor. It also has pointers to the head and tail of the snake.

5. Constructor:

The constructor initializes the variables and sets up the console screen buffer.

6. insert function:

This function inserts a new segment of the snake at the current food position when the snake eats the food.

7. move function:

This function contains the main logic to move the snake by incrementing and decrementing the x or y coordinates of each segment. It updates the position of the first segment and copies the positions of the other segments in reverse order, from the last segment to the first.

8. draw function:

This function draws the snake segments according to their positions. It traverses the linked list from the head to the tail and draws each segment on the console screen.

9. drawWall function:

This function draws the walls of the game board on the console screen.

10. collision function:

This function checks if the snake has collided with the walls or its own body. It returns true if there is a collision, false otherwise.

11. drawfood function:

This function draws the food at a random position on the console screen. If the parameter x is set to 1, it draws the food at a new position.

12. drawinit function:

This function initializes the game board by drawing the walls and the snake.

13. labelDead function:

This function displays the game over message when the snake dies.

14. menu function:

This function displays the main menu of the game.

15. help function:

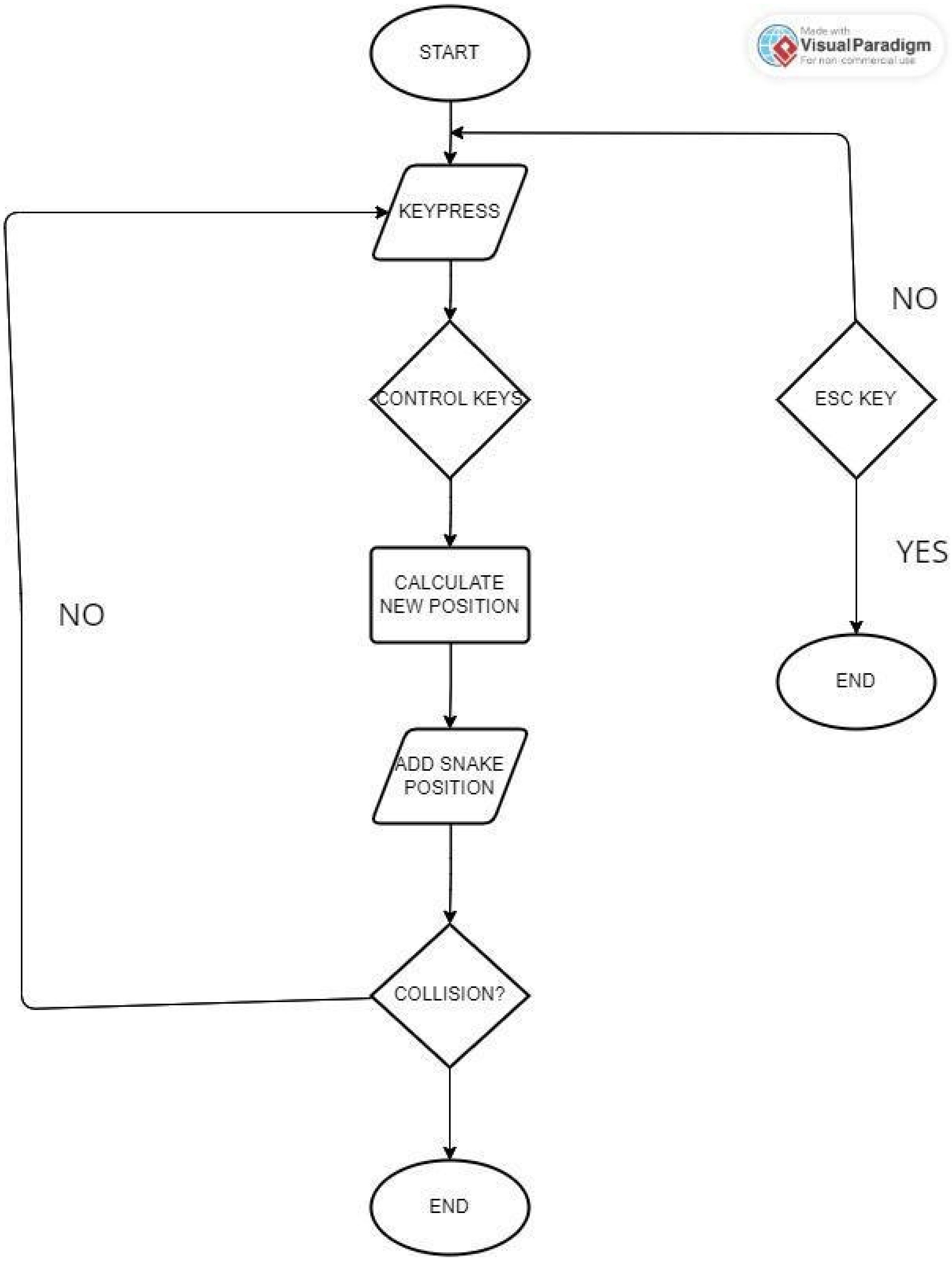
This function displays the instructions on how to play the game.

16. loop function:

This function contains the main game loop that updates the game board and checks for collisions. It also handles the keyboard input from the player to change the direction of the snake.

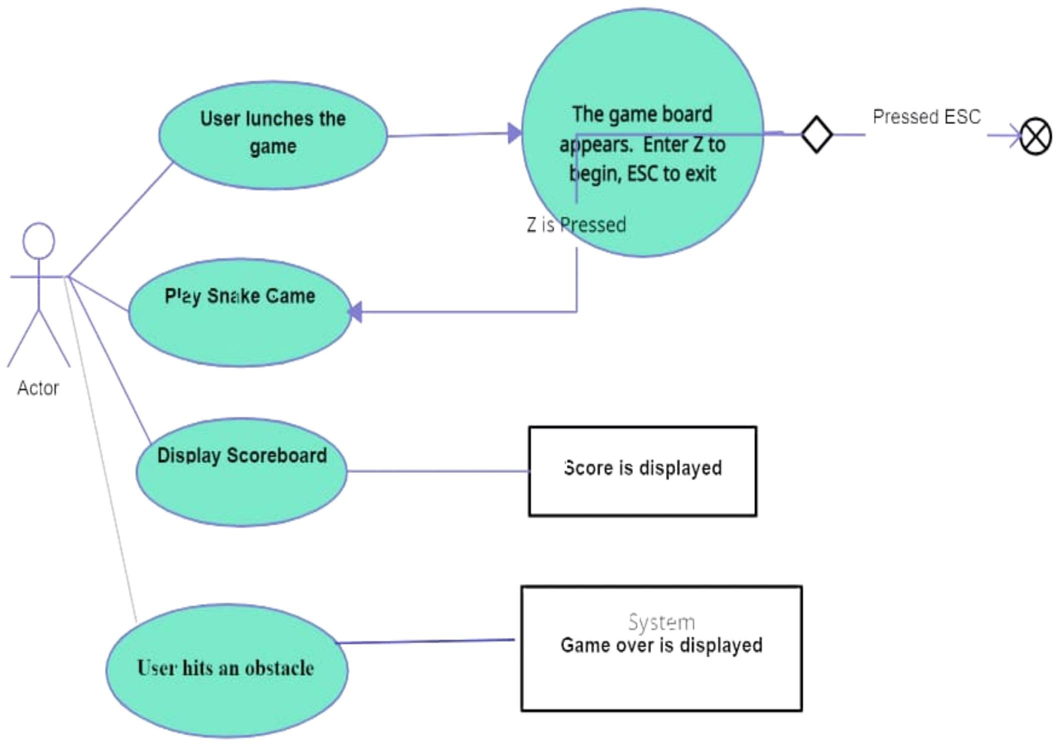
Overall, this code implements the classic Snake game using C++ and console input/output functions.

FLOW CHART:

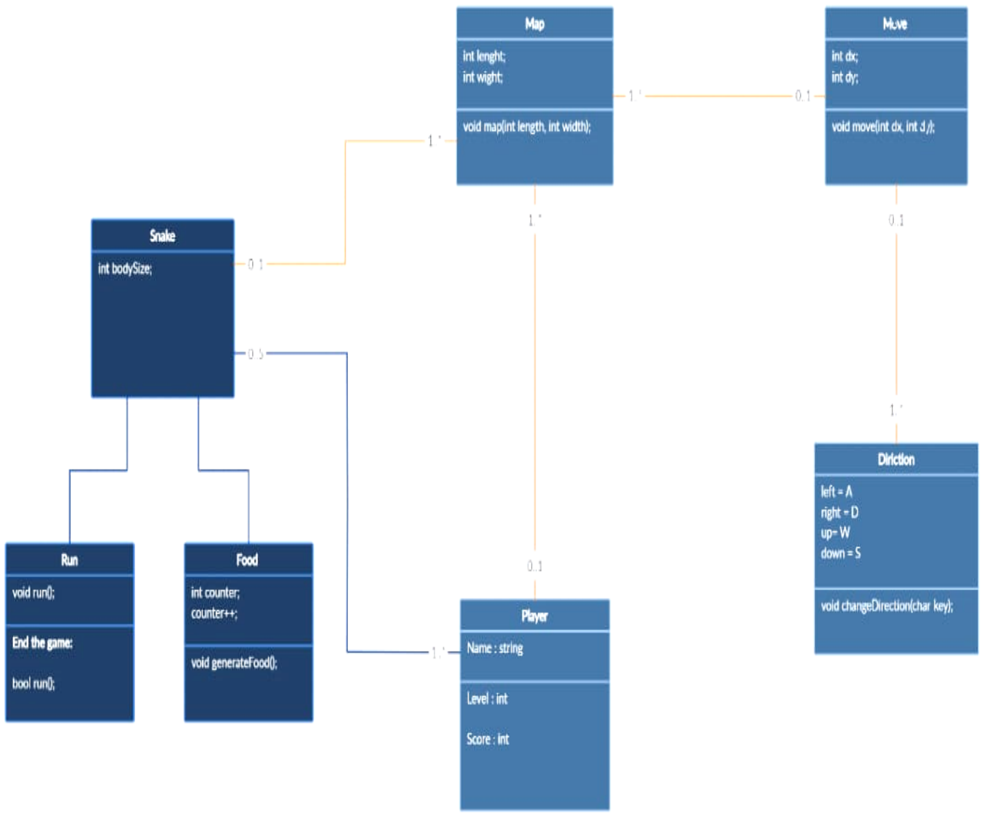

START
KEYPRESS
CONTROL KEYS
CALCULATE NEW POSITION
ADD SNAKE POSITION
COLLISION?
END
NO
ESC KEY
NO
YES
END

UML DIAGRAMS

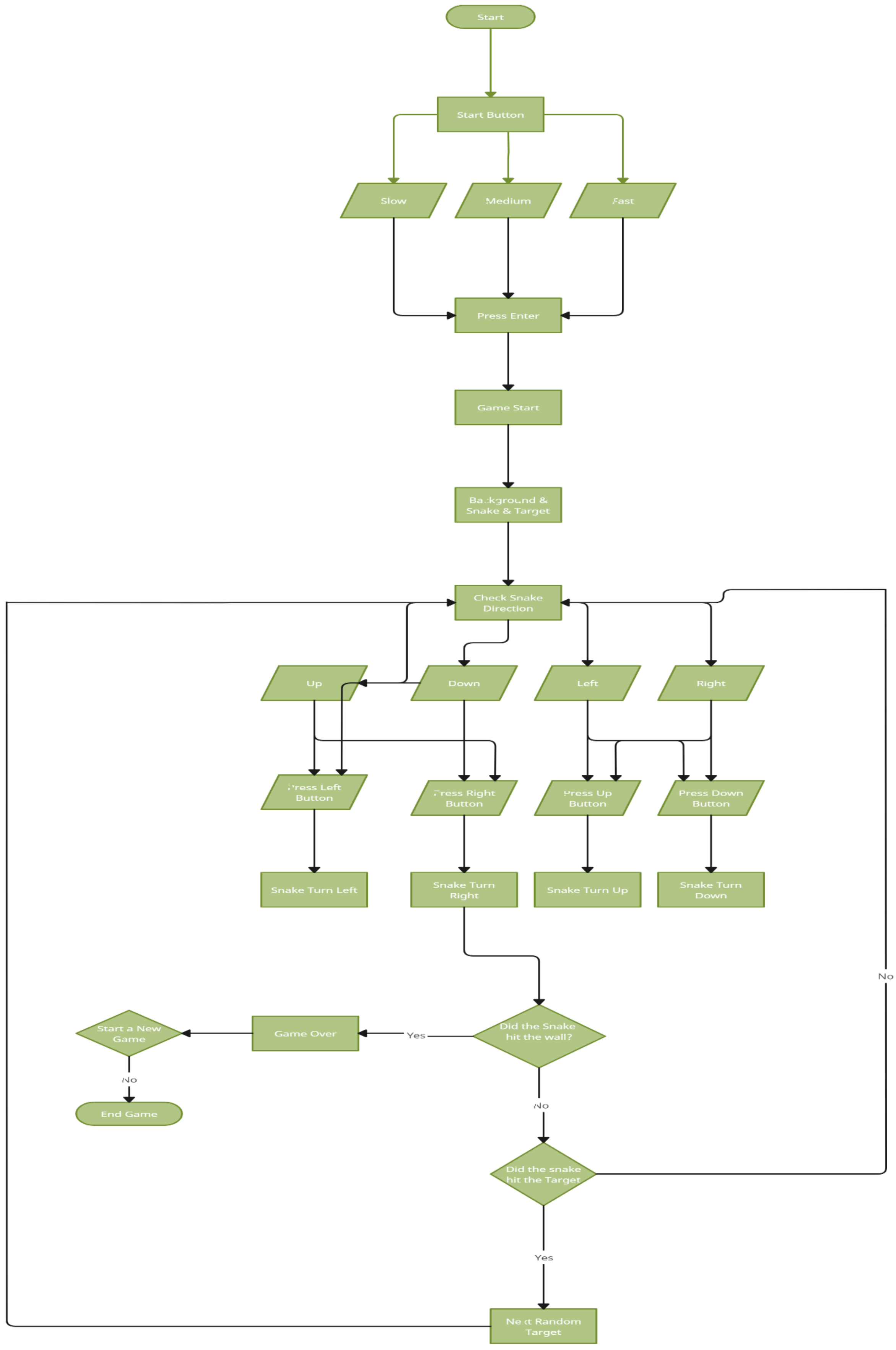
USE CASE DIAGRAM:



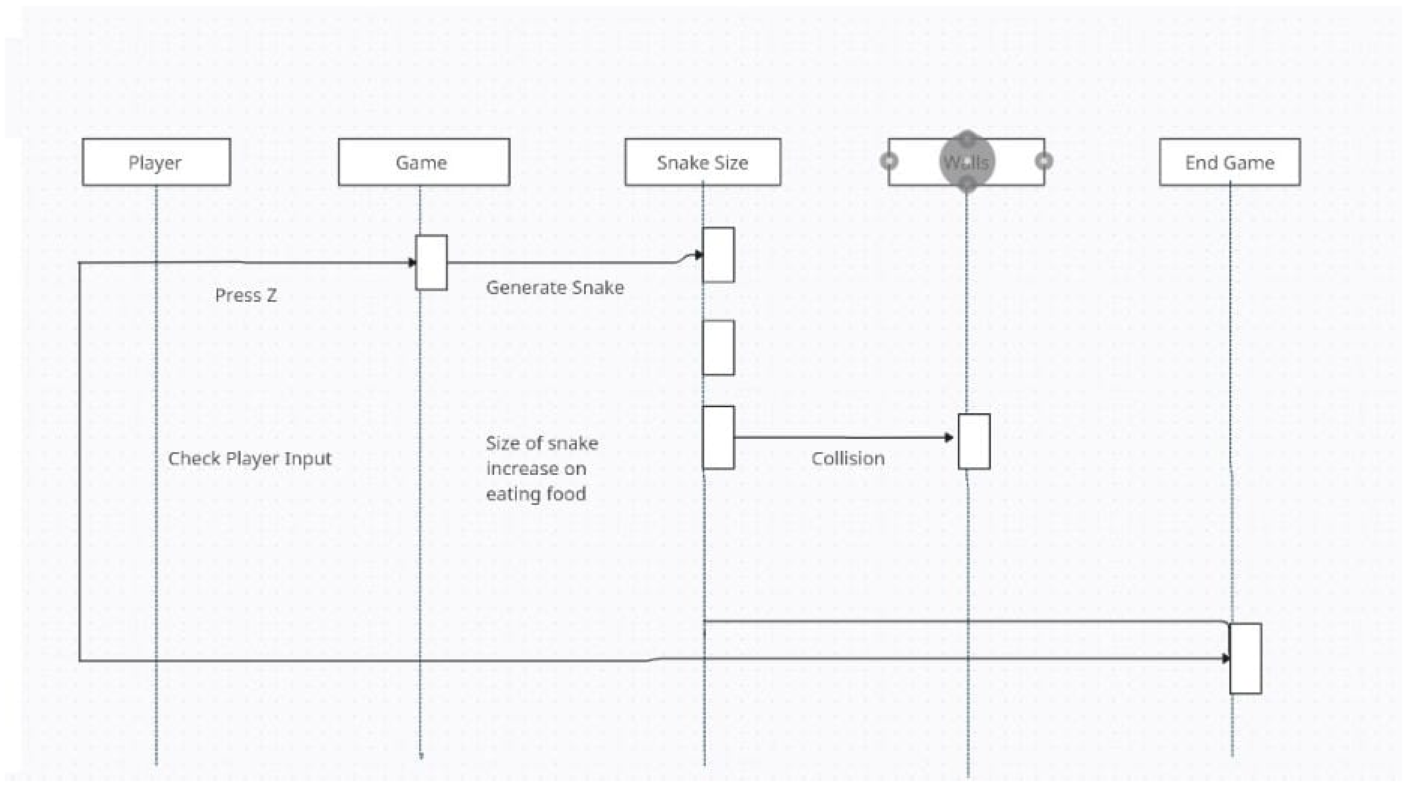
CLASS DIAGRAM:



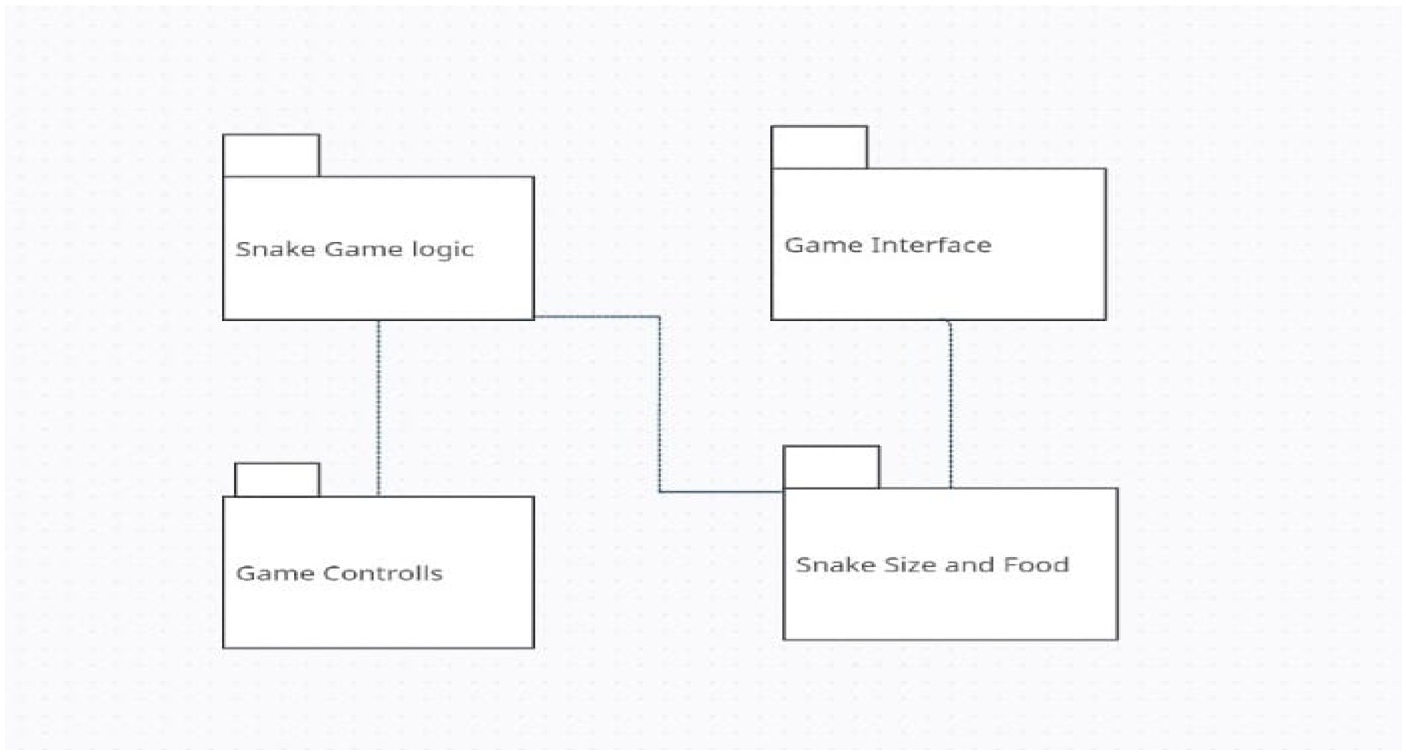
ACTIVITY DIAGRAM:-


Start
Start Button
Slow
Medium
Fast
Press Enter
Game Start
Background & Snake & Target
Check Snake Direction
Up
Down
Left
Right
Press Left Button
Press Right Button
Press Up Button
Press Down Button
Snake Turn Left
Snake Turn Right
Snake Turn Up
Snake Turn Down
Did the Snake hit the wall?
Yes
Game Over
Start a New Game
No
End Game
No
Did the snake hit the Target
No
Yes
Next Random Target

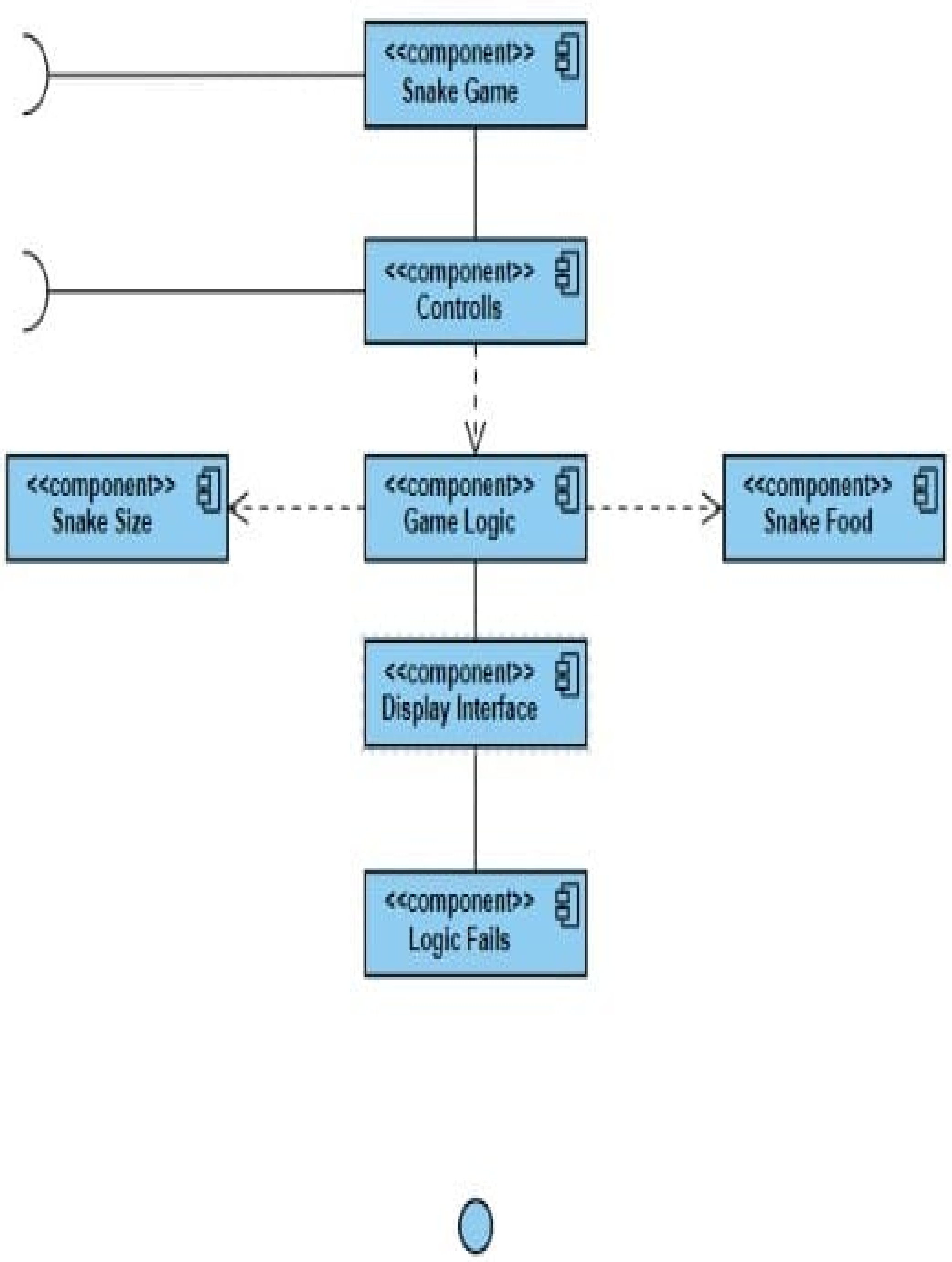
SEQUENCE DIAGRAM:



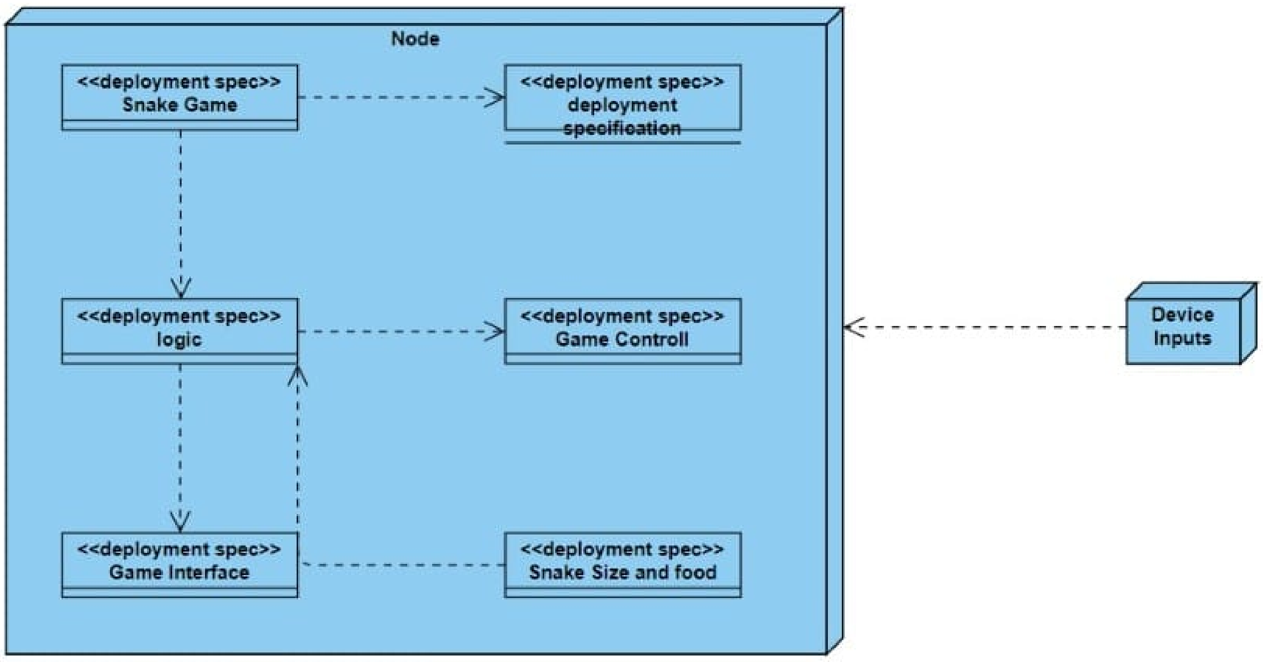
PACKAGE DIAGRAM:



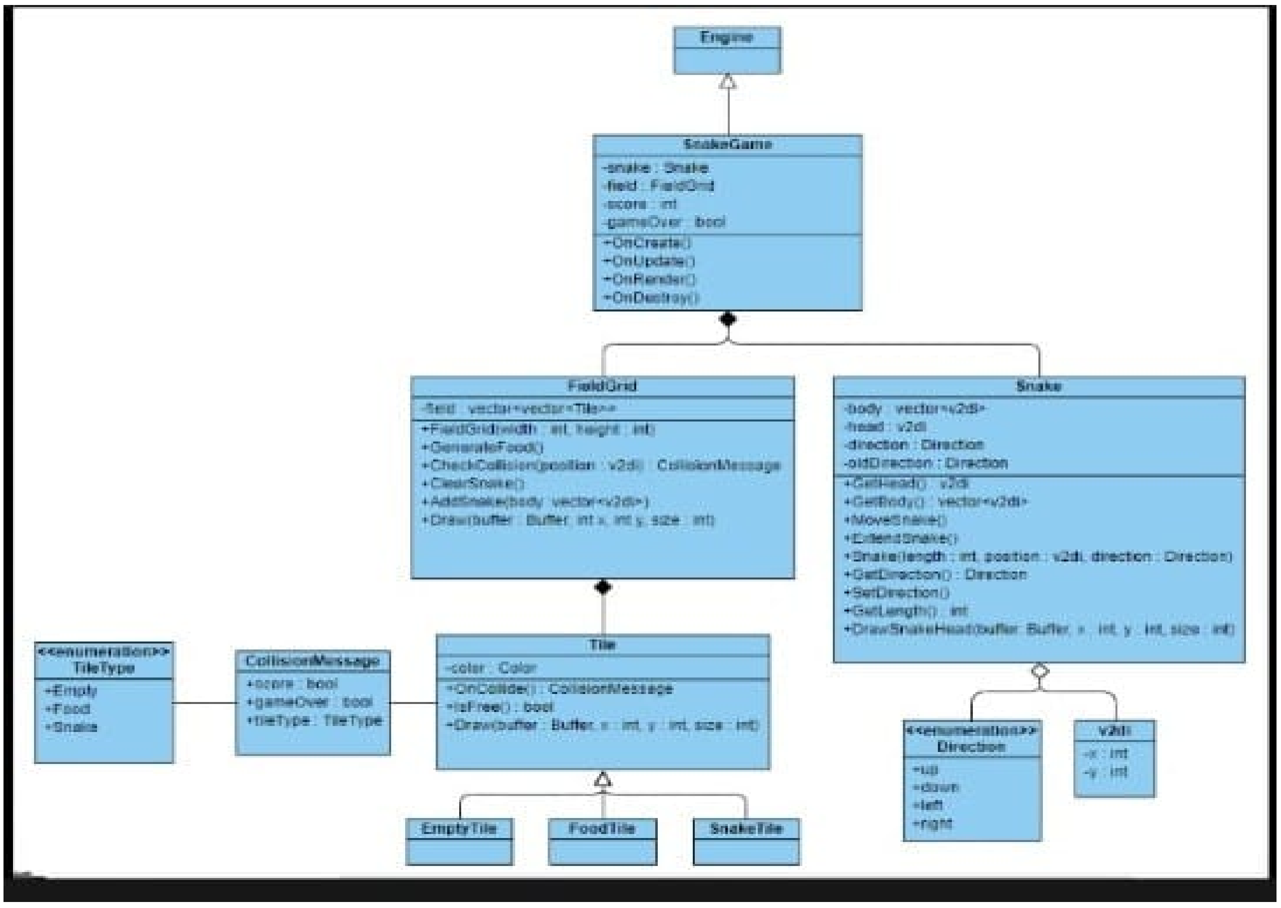
COMPONENT DIAGRAM:



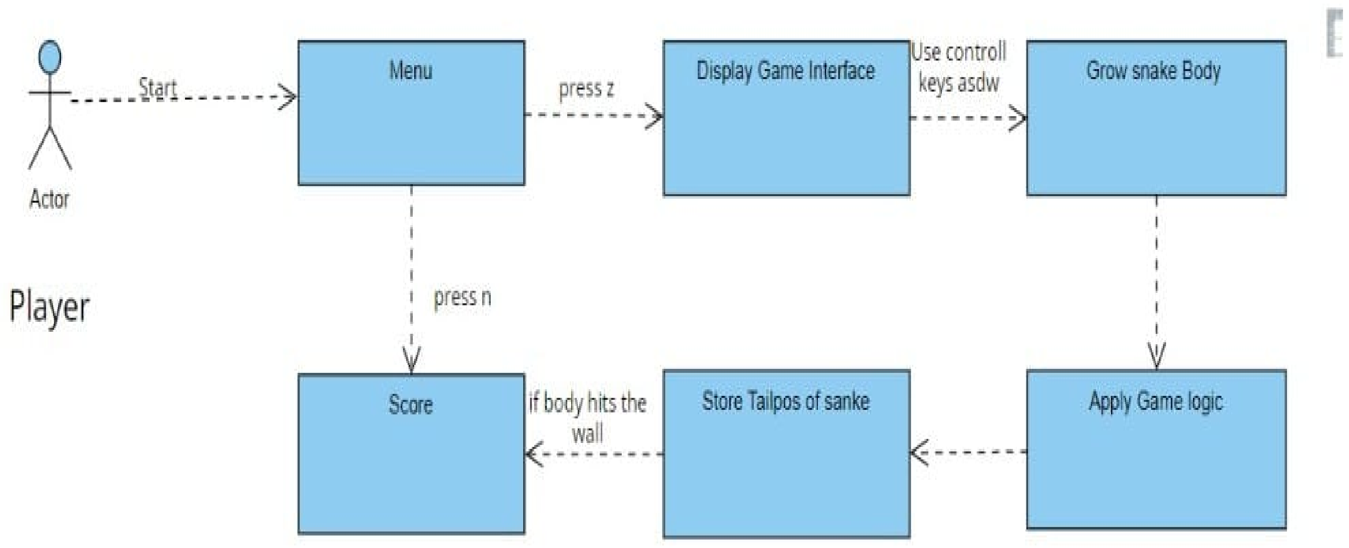
DEPLOYMENT DIAGRAM:



STATE CHART DIAGRAM:



COLLABORATION DIAGRAM:



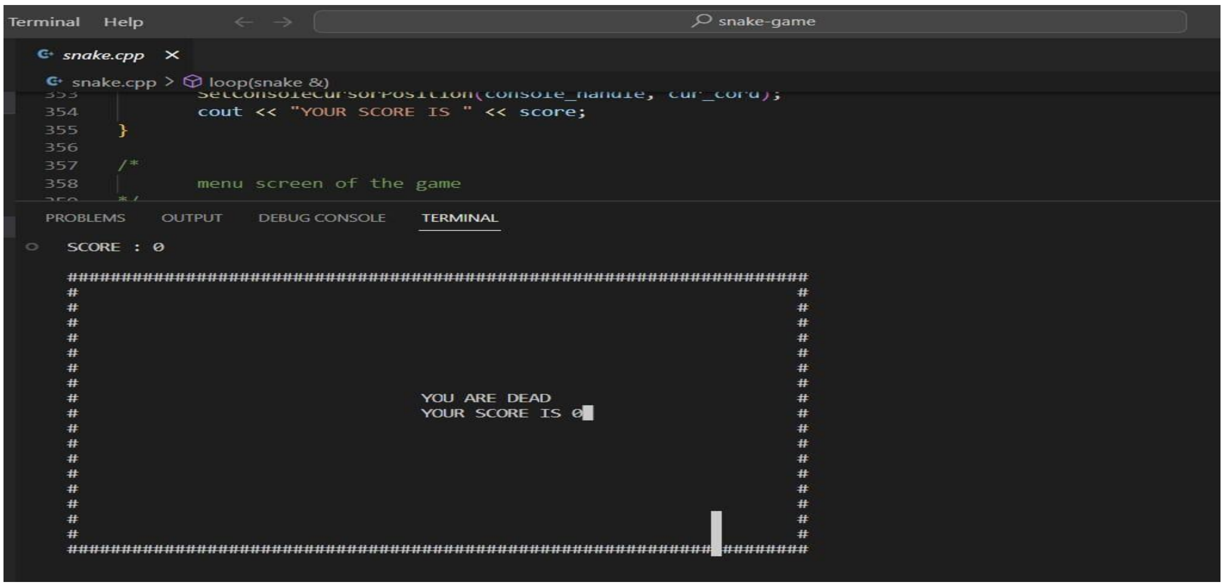
SOURCE CODE:

```
#include <iostream>
#include <windows.h>
#include <stdlib.h>
#include <conio.h>
#include <fstream>

using namespace std;
typedef struct tailpos
{
	int x;
	int y;
	struct tailpos *next;
	struct tailpos *prev;
} tail;

int d = 4;

class snake
{
public:
	int wallsX, wallsY;
	int walleX, walleY;

	int score;

	int foodx, foody;

	HANDLE console_handle;
	COORD cur_cord;
	tail *start, *current, *newtail;
	snake();
	void insert(int x, int y);
	void draw();
	void drawWall();
	void move();
	bool collision();
	void drawfood(int x = 0);
	void drawinit();
	void labelDead();
	void menu();
	void help();
};
void loop(snake &ob);

snake::snake()
{
	score = 0;
	start = NULL;
	current = NULL;
	newtail = NULL;
	console_handle = GetStdHandle(STD_OUTPUT_HANDLE);
	foodx = 12;
	foody = 14;
```

```
    wallsX = 2;
    wallsY = 2;
    walleX = 70;
    walleY = 20;

    cur_cord.X = 152;
    cur_cord.Y = 500;
    SetConsoleScreenBufferSize(console_handle, cur_cord);
}
void snake ::insert(int x, int y)
{
    if (start == NULL)
    {
        newtail = new tail;
        newtail->x = x;
        newtail->y = y;
        newtail->next = NULL;
        newtail->prev = NULL;
        start = newtail;
        current = newtail;
    }
    else
    {
        newtail = new tail;
        newtail->x = x;
        newtail->y = y;
        newtail->next = NULL;
        newtail->prev = current;
        current->next = newtail;
        current = newtail;
    }
}

void snake::move()
{
    tail *tmp, *cur;

    tmp = current;
    while (tmp->prev != NULL)
    {
        tmp->x = tmp->prev->x;
        tmp->y = tmp->prev->y;
        tmp = tmp->prev;
    }
    if (d == 1)
        start->y--;

    if (d == 2)
        start->y++;

    if (d == 3)
        start->x--;
    if (d == 4)
        start->x++;
}

void snake::draw()
{

cur_cord.X=10;
cur_cord.Y=5;

SetConsoleCursorPosition(console_handle,cur_cord);
cout << "use above statement to to set cursor position";


    cur_cord.X = 2;
    cur_cord.Y = 0;

    SetConsoleCursorPosition(console_handle, cur_cord);
    cout << "SCORE : " << score;

    tail *tmp, *last;
    tmp = start;
    last = current;

    while (tmp != NULL)
    {
        cur_cord.X = tmp->x;
        cur_cord.Y = tmp->y;
        SetConsoleCursorPosition(console_handle, cur_cord);
        cout << (char)219;
        tmp = tmp->next;
    }
    cur_cord.X = last->x;
    cur_cord.Y = last->y;
    SetConsoleCursorPosition(console_handle, cur_cord);
    cout << ' ';
    cur_cord.X = foodx;
    cur_cord.Y = foody;
    SetConsoleCursorPosition(console_handle, cur_cord);
    cout << (char)15;
}

void snake::drawWall()
{

    cur_cord.X = wallsX;
    for (int y = wallsY; y <= walleY; y++)
    {
        cur_cord.Y = y;
        SetConsoleCursorPosition(console_handle, cur_cord);
        cout << '#';
    }

    cur_cord.Y = wallsY;
    for (int x = wallsX; x <= walleX; x++)
    {
        cur_cord.X = x;
        SetConsoleCursorPosition(console_handle, cur_cord);
        cout << '#';
    }
    cur_cord.X = walleX;
    for (int y = wallsY; y <= walleY; y++)
    {
        cur_cord.Y = y;
        SetConsoleCursorPosition(console_handle, cur_cord);
        cout << '#';
    }


    cur_cord.Y = walleY;
    for (int x = wallsX; x <= walleX; x++)
    {
        cur_cord.X = x;
        SetConsoleCursorPosition(console_handle, cur_cord);
        cout << '#';
    }
}

void snake::drawfood(int x)
{
    tail *tmp;
    tmp = start;
    if (x == 1)
    {
        foodx = rand() % 2 + 39;
        foody = rand() % 2 + 16;

        while (tmp->next != NULL)
        {
            if (foodx == tmp->x && foody == tmp->y)
            {
                drawfood(1);
                cout << "drawn";
            }

            tmp = tmp->next;
        }
    }
}

void snake::drawinit()
{
    drawWall();
}
bool snake::collision()
{
    tail *tmp;
    tmp = start->next;
    while (tmp->next != NULL)
    {
        if (start->x == tmp->x && start->y == tmp->y)
            return true;

        tmp = tmp->next;
    }
    if (start->x == foodx && start->y == foody)
    {
        insert(foodx, foody);
        drawfood(1);
        score++;            }

    for (int x = wallsX; x <= walleX; x++)
    {
        if (start->x == x && start->y == wallsY)
        {
            return true;
        }
    }
    for (int y = wallsY; y <= walleY; y++)
    {
        if (start->x == wallsX && start->y == y)
        {
            return true;
        }
    }

    for (int y = wallsY; y <= walleY; y++)
    {
        if (start->x == walleX && start->y == y)
        {
            return true;
        }
    }
    for (int x = wallsX; x <= walleX; x++)
    {
        if (start->x == x && start->y == walleY)
        {
            return true;
        }
    }

    return false;
}
void snake::labelDead()
{

    cur_cord.X = (walleX / 2);
    cur_cord.Y = (walleY / 2);

    SetConsoleCursorPosition(console_handle, cur_cord);

    cout << "YOU ARE DEAD\n";
```

```
	cur_cord.Y = (walleY / 2) + 1;
	SetConsoleCursorPosition(console_handle, cur_cord);
	cout << "YOUR SCORE IS " << score;
}

void snake::menu()
{
	char word;
	ifstream iFile("menu.txt");
	word = iFile.get();
	while (iFile)
	{

		cout << word;
		word = iFile.get();
	}
	iFile.close();

}

void snake::help()
{
	char word;
	ifstream iFile("help.txt");
	word = iFile.get();
	while (iFile)
	{

		cout << word;
		word = iFile.get();
	}
	iFile.close();

	getch();
}
int main()
{	snake obc;
	obc.menu();
	switch (getch())
	{
	case 'z':
		system("CLS");
		loop(obc);
		break;
	case 'h':
		system("CLS");
		obc.help();
		system("CLS");
		main();
		break;
	case 'q':
		break;
	default:
        system("CLS");
        main();
    }

    return 0;
}
void loop(snake &ob)
{
    ob.insert(10, 6);
    ob.insert(10, 7);
    ob.insert(10, 8);
    ob.insert(10, 9);

    ob.drawinit();
    int dir = 1;
    while (1)
    {
        ob.draw();
        Sleep(200);

        if (ob.collision())
        {
            ob.labelDead();
            break;
        }

        if (kbhit())
        {
            switch (getch())
            {
            case 'w':
                d = 1;
                break;
            case 's':
                d = 2;
                break;
            case 'a':
                d = 3;
                break;
            case 'd':
                d = 4;
                break;
            case 'm':
                ob.insert(10, 7);
                break;
            }
        }

        ob.move();
    }
    int x;
    cin >> x;
}
```

OUTPUT



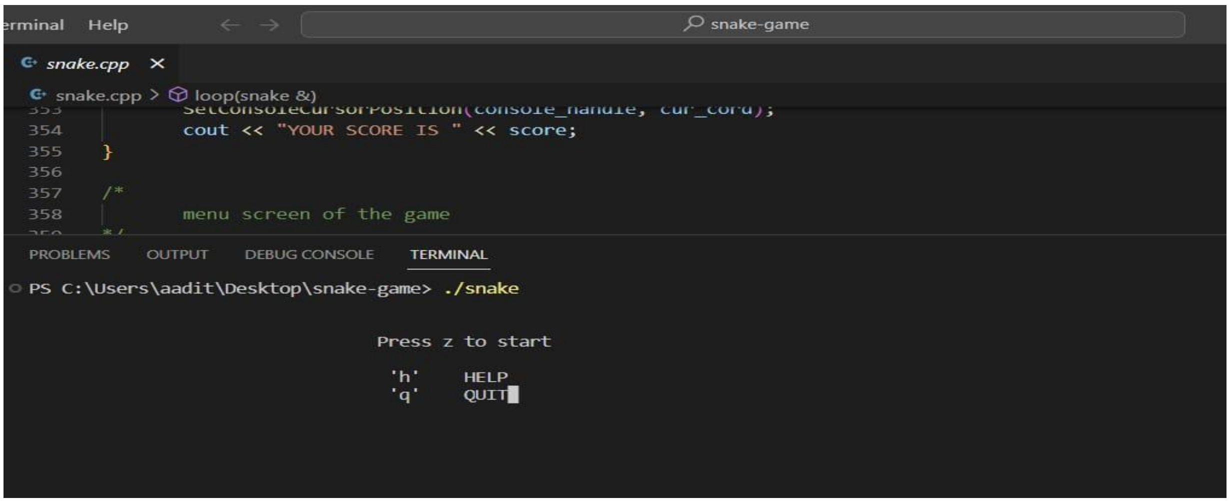
Terminal Help snake-game

snake.cpp > loop(snake &)

```
        SetConsoleCursorPosition(console_handle, cur_cord);
        cout << "YOUR SCORE IS " << score;
    }

    /*
        menu screen of the game
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
SCORE : 0

######################################################################
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                YOU ARE DEAD                        #
#                                YOUR SCORE IS 0                     #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
######################################################################
```



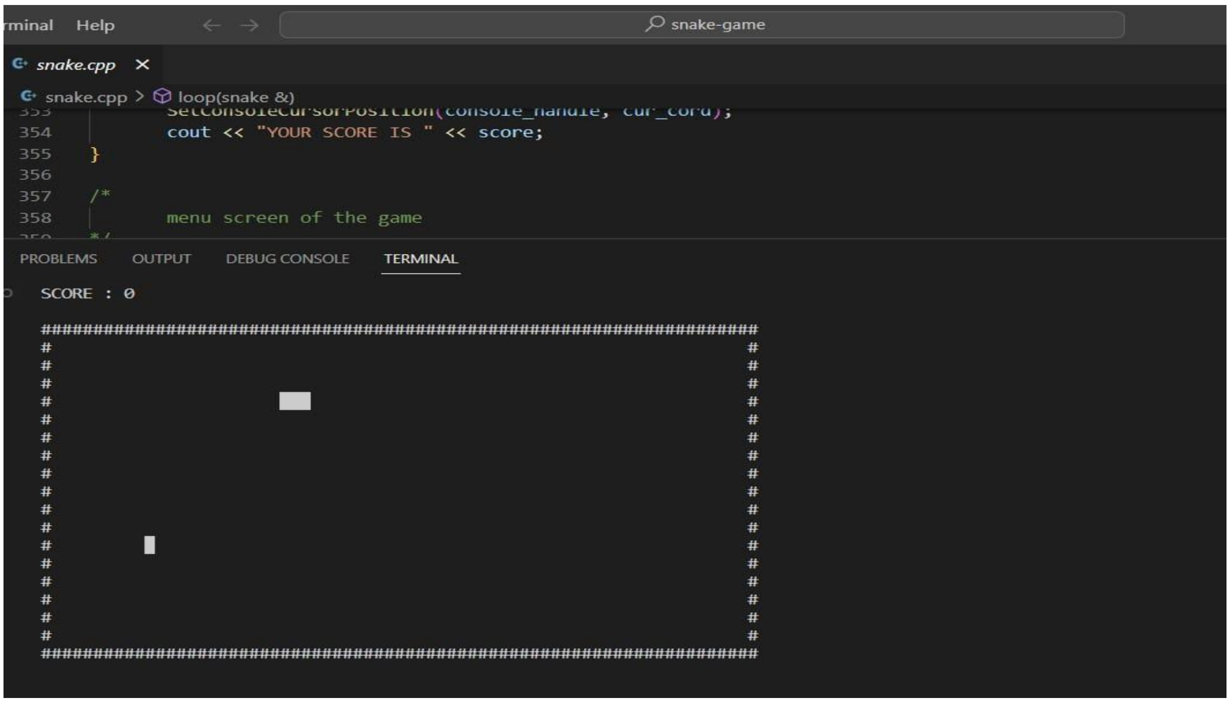
erminal Help snake-game

snake.cpp > loop(snake &)

```
        SetConsoleCursorPosition(console_handle, cur_cord);
        cout << "YOUR SCORE IS " << score;
    }

    /*
        menu screen of the game
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\aadit\Desktop\snake-game> ./snake


                              Press z to start

                               'h'    HELP
                               'q'    QUIT
```



rminal Help snake-game

snake.cpp > loop(snake &)

```
        SetConsoleCursorPosition(console_handle, cur_cord);
        cout << "YOUR SCORE IS " << score;
    }

    /*
        menu screen of the game
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
SCORE : 0

######################################################################
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
#                                                                    #
######################################################################
```

Conclusion and Result:

In conclusion, creating a snake game using C++ is a complex process that requires technical knowledge of programming, game design, and computer hardware. However, C++ provides developers with a high level of control over computer hardware, which can result in a fast and efficient game with smooth gameplay.

Once the game is created and thoroughly tested, it can be compiled into an .exe file that can be easily distributed and executed on other devices

References:

The C++ Programming Language (4th Edition) By Bjarne Stroustrup

C++ Pocket Reference 1st Edition Accelerated C++:

C++ All-in-One For Dummies 3rd Edition