**CONTENTS**

1. Introduction
2. Aim
3. Objectives
4. Game Overview
5. Game Flow
6. Code
7. Output
8. Algorithm
9. Result
10. Analysis
11. **INTRODUCTION**

The Nokia inspired 2D Snake Game is a classic arcade-style game where players control a snake that grows longer as it consumes food items, navigating a confined space. This simple yet engaging game provides a platform to explore various programming concepts and game development principles. The provided code implements a classic Snake game in C++, utilizing fundamental programming concepts. The code uses a console-based approach and incorporates key features such as snake movement, fruit generation, scoring, and collision detection.

1. **AIM**

The primary aim of 2D Snake game is to deliver an enjoyable gaming experience that combines simplicity, strategy, and skill development. Players control a snake, guiding it to consume food while avoiding collisions with walls and its own tail while maximizing the length of the snake.

1. **OBJECTIVES**

* *Entertainment:* Provide a source of entertainment and recreation for players of all ages through a classic and timeless game.
* *Skill Development:* Encourage the development of skills such as hand-eye coordination, reflexes, and strategic thinking.
* *Scoring Challenge:* Motivate players to achieve higher scores by strategically navigating the snake to eat food and grow longer.

1. **GAME OVERVIEW**

The code consists of several functions, each responsible for a specific aspect of the game:

* Main Function (main): Initiates the game by calling the setup function and entering the main game loop.
* Setup Function (setup): The setup function initializes essential game parameters such as the snake's starting position, the position of the fruit, the score, and the direction of movement.
* Draw Function (draw): The draw function is responsible for rendering the game on the console. It includes the representation of the snake, the fruit, and the game boundaries.
* Input Function (input): The input function captures user input using the \_kbhit() and \_getch() functions, allowing players to control the direction of the snake.
* Logic Function (logic): The logic function governs the game's core logic. It handles the movement of the snake, checks for collisions with the tail and fruit, and updates the score accordingly.

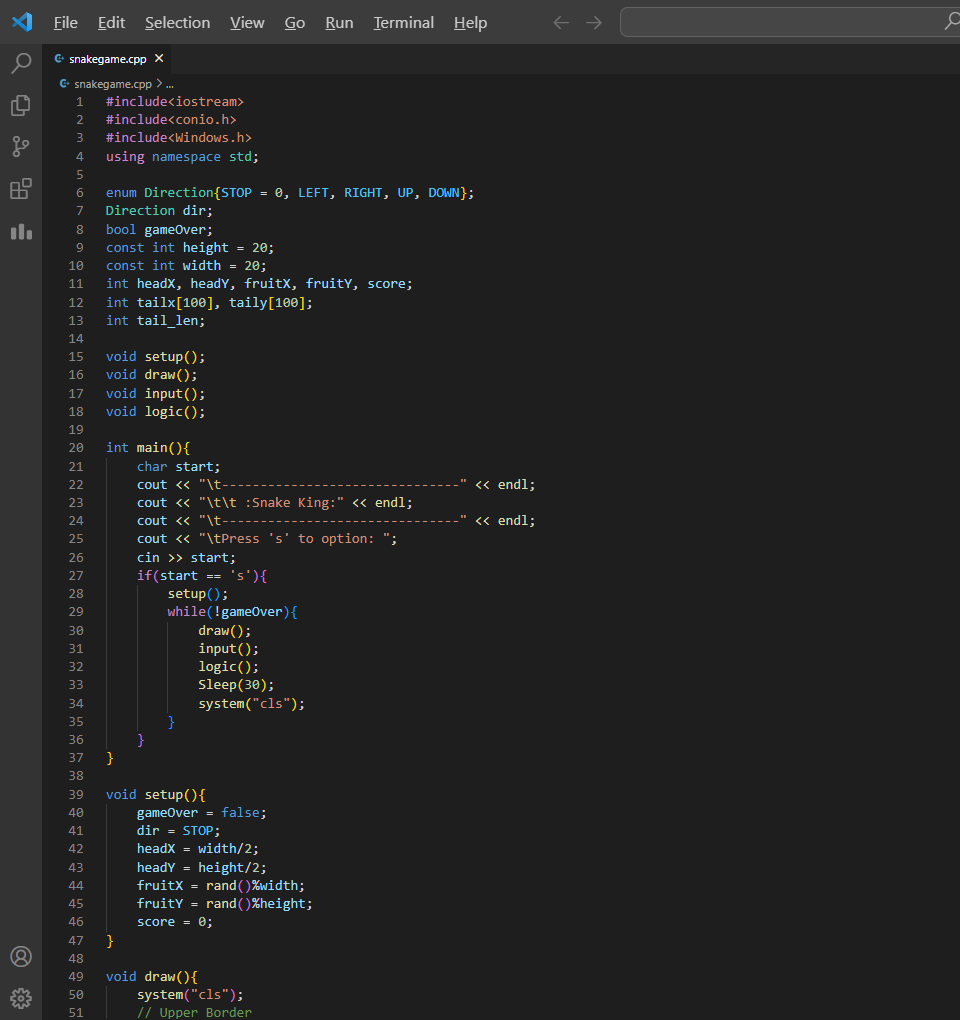
1. **GAME FLOW**

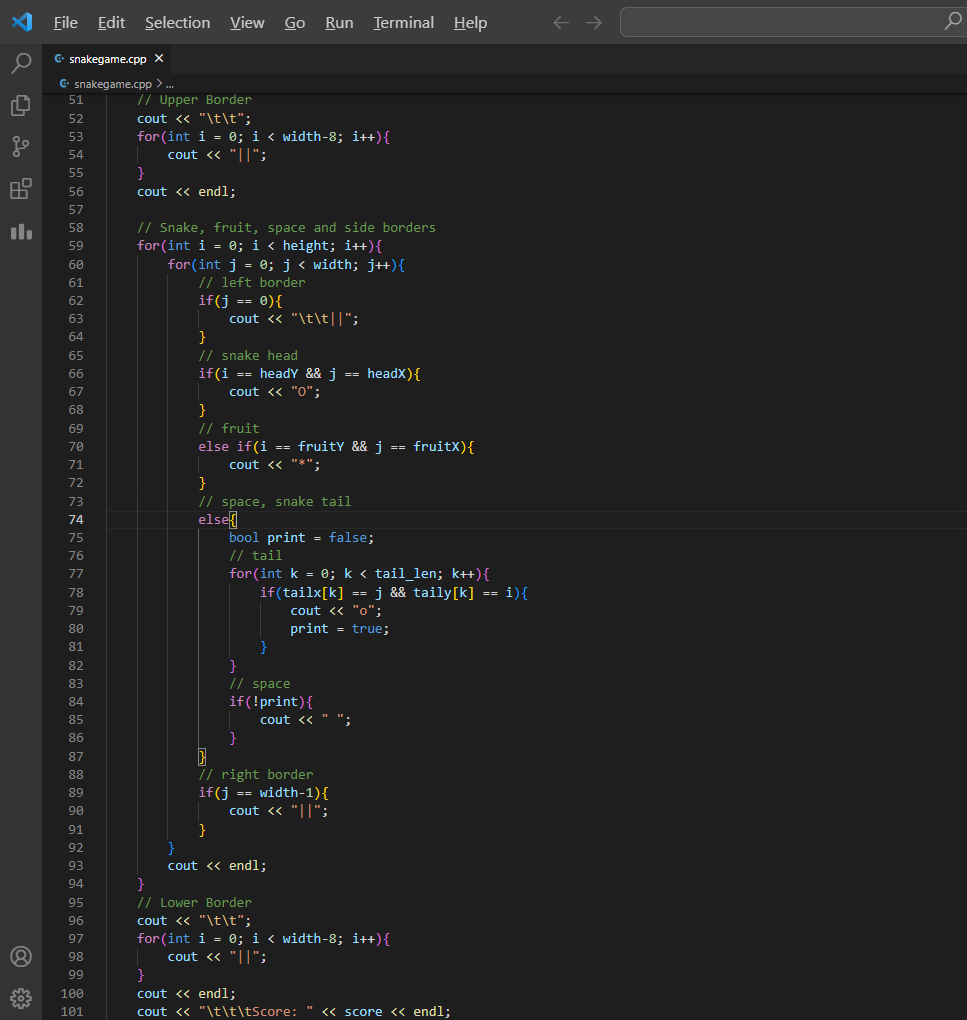
Players start the game by pressing 's'. The main function then initiates the game loop, repeatedly rendering the game, capturing user input, and updating the game state until the game concludes. The game screen includes the snake, fruit, and score, providing players with continuous feedback on their performance.

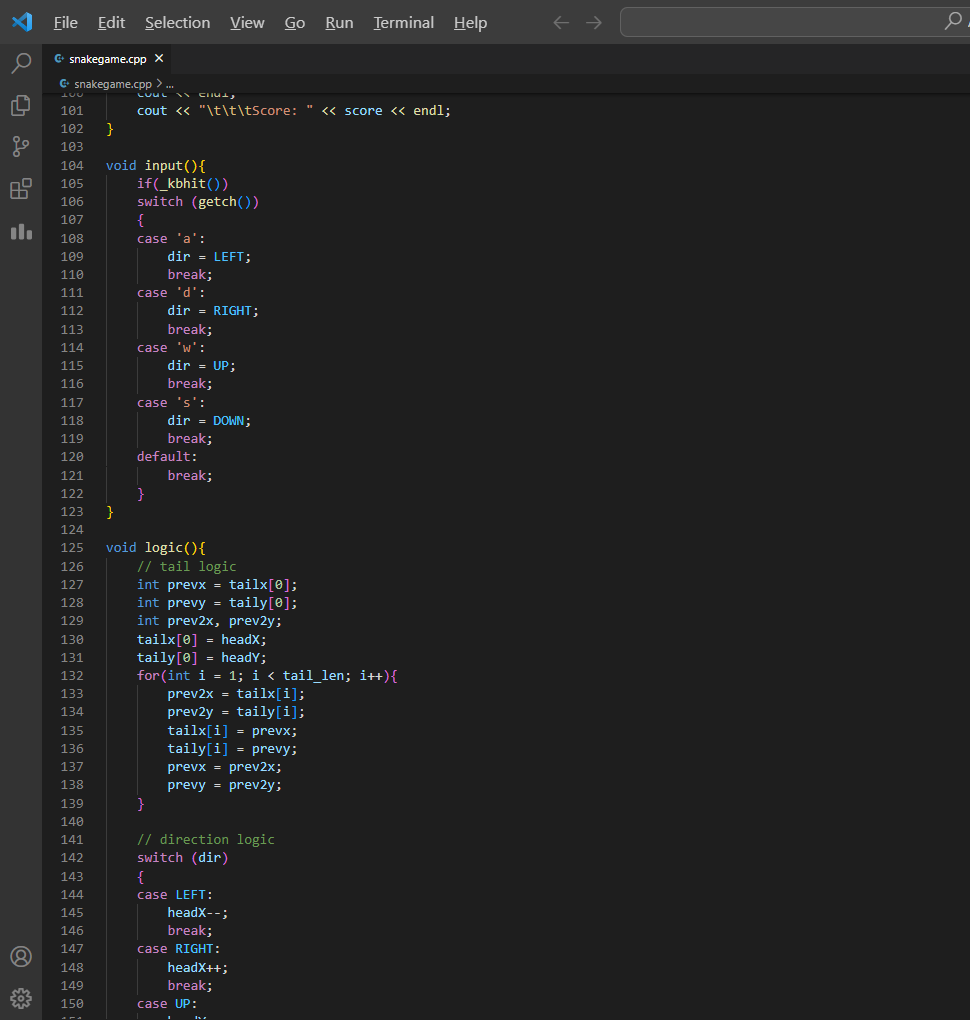
Key components of the C++ code include:

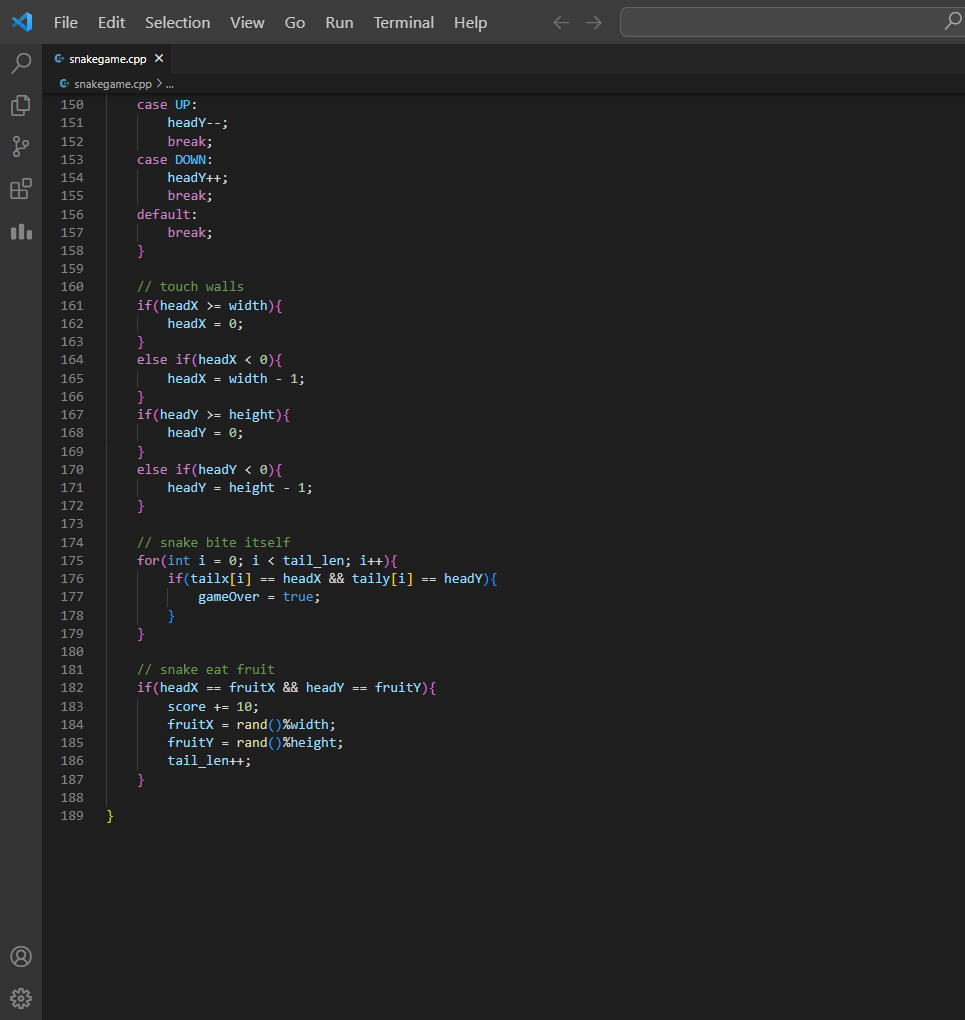
* Game Loop: Utilizes a loop to continually update and render the game until a termination condition is met. Handles user input for controlling the snake's direction.
* Snake Movement: Maintains the snake's position and direction, updating it based on user input and collision with food. Implements a linked list or array to represent the snake's body.
* Food Generation: Randomly places food on the game board. Handles collision detection between the snake and food.
* Collision Detection: Checks for collisions with the game boundaries, snake body, and food. Adjusts game state accordingly, updating score and snake length.
* Rendering: Prints the game board to the console, including the snake, food, and any relevant information (score, length).
* Score Tracking: Maintains a score variable that increments with each food item consumed.

1. **CODE**

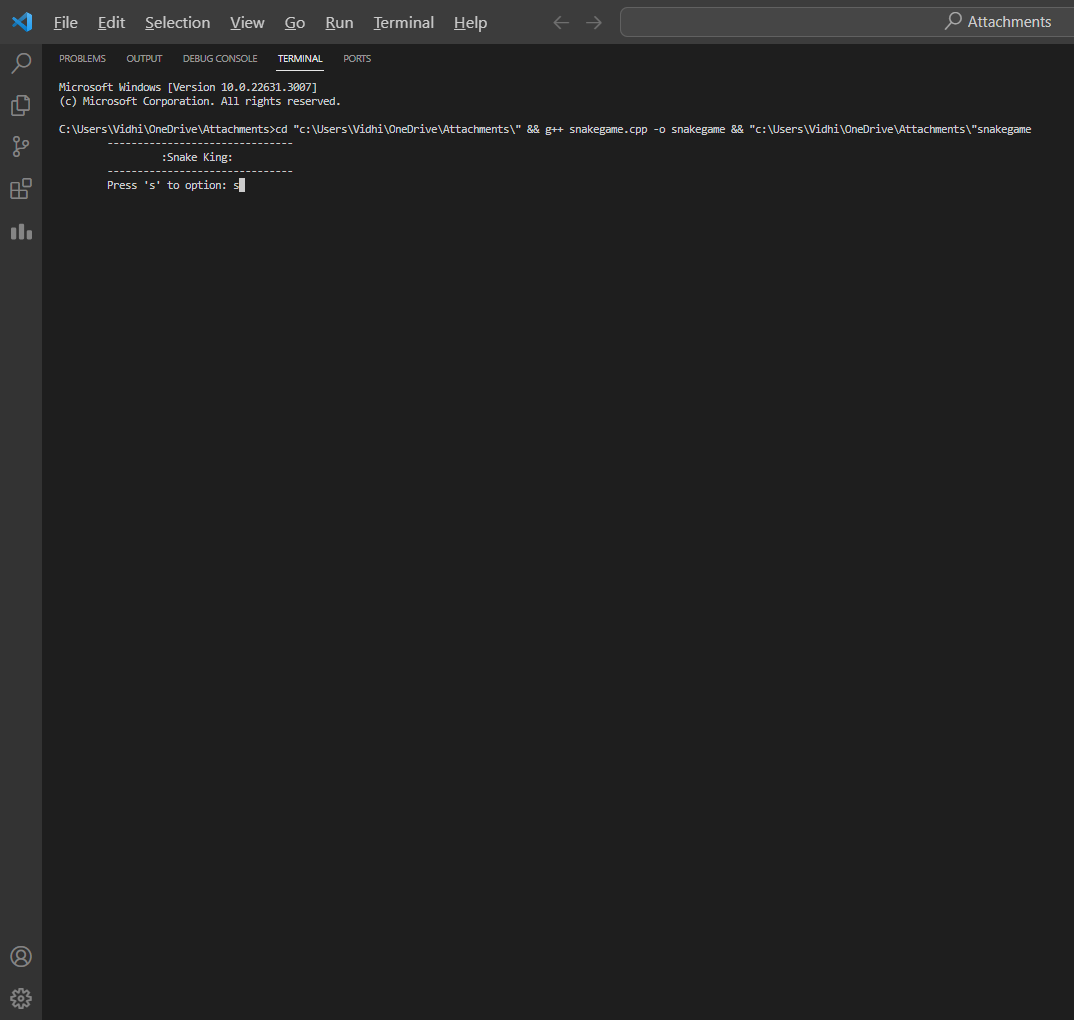
****

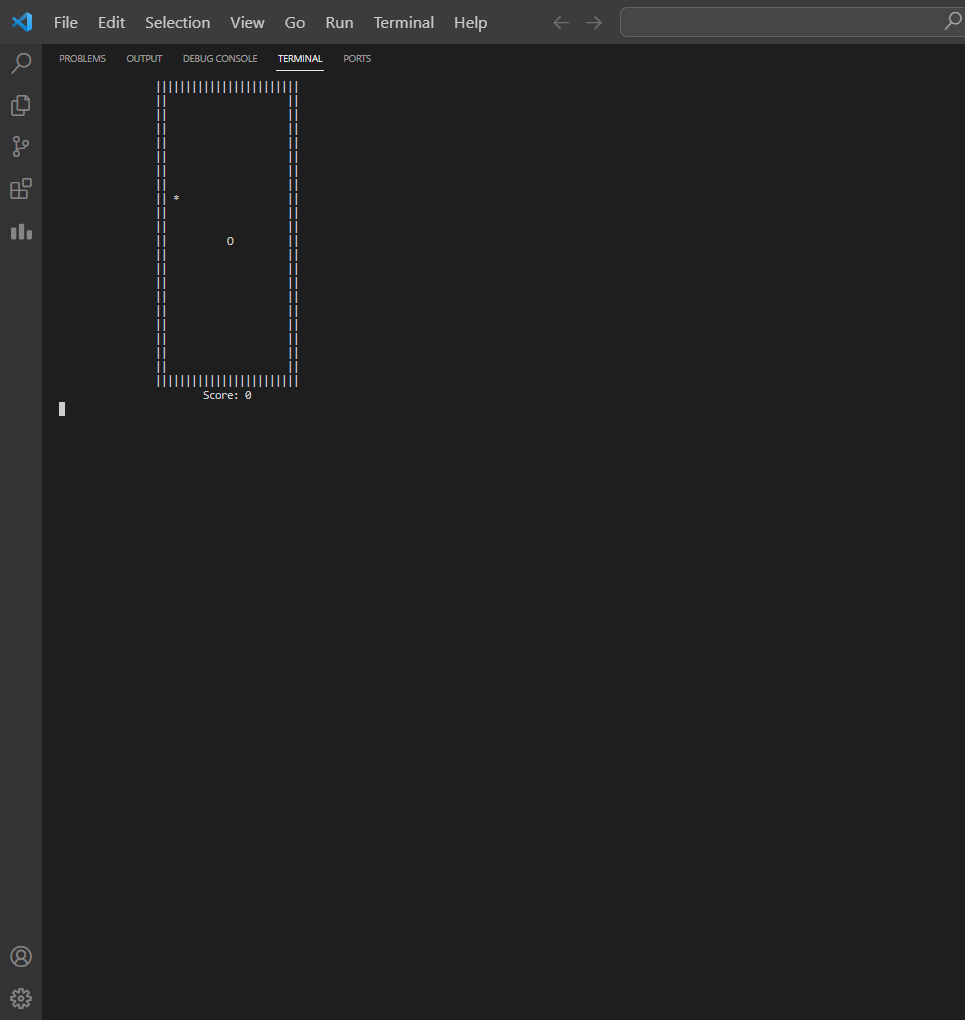






1. **OUTPUT**





1. **ALGORITHM**
2. Start of game
3. Initialization:

* Initialize game variables, such as gameOver, dir (snake direction), and other parameters.
* Display the game title and prompt the user to start by pressing 's'.
* If the user presses 's', call the setup() function to initialize the game state.

1. Game Loop:

* Enter a while loop until the game over condition is met.
* In each iteration of the loop:
* Clear the console screen using system("cls").
* Call the draw() function to display the game board, snake, fruit, and score.
* Call the input() function to handle user input for changing the snake direction.
* Call the logic() function to update the game logic, including snake movement, collisions, and scoring.
* Introduce a short delay using Sleep(30) to control the game speed.

1. Setup Function (`void setup()'):

* Set initial game state variables:
* gameOver to false.
* dir to STOP.
* Initialize snake head position (headX, headY).
* Set initial fruit position (fruitX, fruitY) randomly.
* Initialize score to 0.

1. Draw Function (`void draw()'):

* Display the game board and elements:
* Display upper and lower borders.
* Iterate over each cell of the game board:
* Display left and right borders.
* Display snake head, fruit, snake tail, or empty space based on the current state.
* Display the current score.

1. Input Function (`void input()'):

* Check if a key is pressed using \_kbhit().
* If a key is pressed, use getch() to get the key pressed and update the dir (snake direction) accordingly.

1. Logic Function (`void logic()'):

* Update the position of the snake:
* Update the position of the snake head based on the current direction.
* Update the positions of the snake tail based on the previous positions.

1. Handle collisions:

* Check if the snake hits the walls, and adjust the position accordingly.
* Check if the snake bites itself, set gameOver to true if true.
* Check if the snake eats a fruit:
* Increment the score.
* Generate a new random position for the fruit.
* Increase the length of the snake tail.

1. End of Game:

* When the game over condition is met, exit the game loop.
* The final score is displayed.

1. **RESULT**

The implementation of the 2D Snake Game yielded a functional and entertaining gaming experience. Players can control the snake's direction using specified keys, and the game mechanics successfully handle collisions with fruits and the snake's body. Additionally, the snake grows longer upon consuming food, challenging players to strategize and avoid self-obstruction.

1. **ANALYSIS**

The Snake game implemented in C++ is a well-constructed program that captures the essence of the classic game. The success of the 2D Snake Game lies in its simplicity and addictive nature. Its simplicity, combined with strategic depth, ensures that players of all skill levels can enjoy and appreciate the timeless appeal of the Snake game. The collision detection algorithms effectively prevent the snake from passing through itself, enhancing the game's realism. The gradual increase in difficulty, achieved by faster snake movement or a shorter time frame for food consumption, ensures a challenging experience for players of varying skill levels.