Snake Game

A PROJECT REPORT

Submitted by

Shubham Agarwal (UID – 23BCS11878)

Harsh Phogat (UID – 23BCS13356)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

Chandigarh University

JUNE - 2025

BONAFIDE CERTIFICATE

Certified that this project report **"Snake Game"** is the bonafide work of "**SHUBHAM AGARWAL"** who carried out the project work under my/our supervision.

<<Signature of the HoD>>

SIGNATURE

<<Name of the Associate Director>>

ASSOCIATE DIRECTOR (CSE-3rd Year)

<<Department>>

<<Signature of the Supervisor>>

SIGNATURE

<<Name of **SUPERVISOR** >>

<<Academic Designation>>

<<Department>>

Submitted for the project viva-voce examination held on

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

This project implements a classic **Snake Game** enhanced with the application of **data structures** such as linked lists, arrays, and stacks. The primary goal is to simulate a responsive, modular, and data-driven game that not only entertains but also educates users on how core data structures are applied in real-world programming.

A **linked list** is used to represent the dynamic body of the snake, allowing for efficient growth and movement. The **2D array** simulates the game grid, storing elements such as the snake, food, and obstacles. A **stack** data structure is employed to maintain a complete history of the snake's movements and scores, providing traceability and illustrating LIFO (Last In, First Out) behavior in practice.

The project is developed in **C++** using **object-oriented programming principles** and a commandline interface to ensure wide compatibility and simplicity. It emphasizes core software engineering practices such as modularity, clean design, input handling, and structured logic. Overall, the project serves both as a functional entertainment application and as an instructional tool for understanding linked lists, stacks, control flow, and array manipulation in C++ programming.

SYMBOLS

Symbol	Meaning
• Snake Head — the front of the snake (moves in • O	the current direction)
• Snake Tail — body segments represented as a • o	linked list
• Food — when eaten, increases score and grows • *	the snake
• $	• Bonus Food — gives extra score (e.g., +30)
• #	• Wall — fixed obstacle; collision ends the game
• Obstacle — similar to a wall, may appear as • @	level-based elements
• ' ' (space)	• Empty Cell — free path where snake can move
• →, ←, ↑, ↓	• (Optional): Used in diagrams to show direction (not in game UI)

CHAPTER 1.

INTRODUCTION

1.1. Client Identification/Need Identification/Identification of relevant Contemporary issue

In the modern era of computer science education, applying core data structure concepts to interactive games has become increasingly effective in bridging theory with realworld applications. **Snake**, a timeless arcade game, serves as an ideal platform to explore data structures such as **linked lists**, **stacks**, and **arrays** in a meaningful, visual, and engaging way.

While many implementations of the Snake game focus solely on gameplay, they often neglect the educational potential of integrating computer science fundamentals. This project addresses that gap by not only simulating a functional snake game but by emphasizing the **use of a linked list to model the snake's body**, a **2D array to manage the board**, and a **stack to store the movement history**. These choices help visualize how these structures behave dynamically in memory.

With increasing emphasis on **hands-on learning in data structures and algorithms**, this project aims to demonstrate how theoretical concepts can be converted into practical simulations. The **stack** records movement history, allowing future extensions like undo functionality or game replay features. This project highlights the relevance of interactive educational tools in teaching complex technical subjects like data structure behavior and algorithmic flow control in a fun, accessible way.

1.2. Identification of Problem

Traditional Snake games often rely on simple procedural programming without incorporating dynamic data structures. They lack flexibility, traceability, and depth in terms of system design and educational value. This project identifies the need for a **data-structure-rich implementation** of the Snake game using **linked lists**, **arrays**, and **stacks** to enhance both gameplay and learning outcomes.

1.3. Identification of Tasks

To develop the Snake game with an educational focus on data structures, the work was divided into the following tasks:

- **Understanding the Problem:** Analyze traditional Snake game mechanics and their limitations in data structure usage.
- **Feature Planning:** Decide on educational and gameplay features such as snake growth, wall collisions, food spawning, scoring, and move history.
- **Game Development:** Implement the game using a **linked list for the snake body**, **2D array for the game grid**, and **a stack for storing movement history**.
- **Game Logic:** Handle food generation, wall/obstacle collisions, dynamic tail growth, and score updates through efficient control flow.
- **Move History Using Stack:** Record the snake's position, direction, and score after each move using a stack to support future extensions like undo and replay functionality

1.4. Timeline

Tasks	Timeline
Understanding Snake Game Logic	**1 Day**
Designing Linked List Structure	**1 Day**
Implementing Game Grid (2D Array)	**1 Day**
Movement and Collision Logic	**2 Day**
Stack Integration for History	**1 Day**
Feature Testing and Debugging	**1 Day**
Documentation and Report Writing	**1 Day**
Total Duration	**7 Day**

1.5. Organization of the Report

- Chapter 1: Introduction to the Problem
- Chapter 2: Literature survey
- Chapter 3: Design flow
- Chapter 4: Results and Analysis
- Chapter 5: Conclusion and Future Work

CHAPTER 2.

LITERATURE REVIEW/BACKGROUND STUDY

2.1. Timeline of the reported problem

The Snake Game originated in the 1970s and gained widespread popularity with Nokia mobile phones. While the early implementations focused purely on user entertainment, they were built with basic procedural logic and did not demonstrate deeper software engineering or data structure concepts.

In the current educational context, revisiting this game with a focus on linked lists, stacks, and arrays offers a rich opportunity to teach fundamental computer science principles through practical gameplay**.**

2.2. Proposed solutions

Earlier Snake games were often implemented with arrays alone, hard-coding movement and growth. However, these implementations do not scale well and lack dynamic memory management. Using a **linked list** allows the snake's body to grow and shrink in real-time with efficient memory usage. The introduction of a **stack** to store move history is a novel enhancement, allowing for reverse traceability and potential undo or replay features.

This project proposes an approach where:

- The **snake's body** is managed using a **singly linked list**

- The **board** is handled via a **2D character array**
- A **stack** stores each move, including position, direction, and score
- **Wall positions** are handled like a **graph** to simulate obstacles

2.3. Bibliometric analysis

Concept	Implementation Benefit
Linked List	Allows dynamic growth and shrinkage of the snake's body
Stack	Enables move history tracking and replay/undo functionality
2D Array	Provides a visual grid for rendering the game state
Graph Logic	Used to define wall layouts and validate movement paths

This combination of data structures is commonly referenced in academic resources such as GeeksforGeeks, cppreference.com, and classic algorithm design books for teaching real-time memory-linked behavior.

2.4. Review Summary

The literature supports the use of:

- **Linked Lists** for dynamic data handling,
- **Stacks** for traceable memory structures,
- **Arrays** for fast-access 2D mapping,
- and **Graphs** for advanced logic simulation.

This integrated approach not only enhances gameplay but also reinforces academic concepts of data organization and control flow in software development.

2.5. Problem Definition

The project addresses the limitation of traditional Snake Game implementations that lack structured data handling. It demonstrates how core data structures such as **linked lists, stacks, and arrays** can be applied meaningfully in game development to both improve the player experience and serve as a **learning model for algorithmic thinking**.

CHAPTER 3.

DESIGN FLOW

3.1 Evaluation & Selection of Specifications/Features

Specification Evaluation:

- **Console-based UI**: The game runs in a terminal environment for simplicity and portability.
- **Real-time input handling**: The snake movement responds to directional keys (w, a, s, d) with minimal delay.
- **Dynamic Snake Growth**: Implemented using a **linked list**, allowing the snake's body to grow or shrink in real-time.
- **Obstacles and Bonus Items**: Stationary obstacles (@) and bonus items ($) are added to introduce complexity and scoring variation.
- **Collision Detection**: Walls (#), tail (o), and obstacles (@) trigger game over conditions.

3.2 Design Constraints

The project design adheres to several practical constraints:

- **Regulations**: Being an academic console-based project, it does not fall under any specific regulatory compliance requirements.
- **Economic**: Developed entirely with freely available tools (GCC compiler, standard C++ library, and basic IDE like Code::Blocks), ensuring zero software cost.
- **Environmental**: The application is fully digital and does not involve any physical resources or energy-intensive operations.
- **Professional & Ethical**: Follows clean coding principles, avoids bias, and ensures a fair game environment without exploits.

3.3 Analysis and Feature Finalization Subject to Constraints

Following a constraints-based review, the final features were refined for balance between complexity and practicality:

- **Removed**: Graphical interfaces (like OpenGL or SDL) due to time and resource constraints.

- **Modified**: Instead of using advanced data structures like doubly linked lists, a simple singly linked list was used for managing the snake's body.
- **Added**: Static obstacles (@) and bonus items ($) to enhance gameplay without increasing algorithmic complexity.
- **Maintained**: Console-based UI with manual input to keep the project lightweight, understandable, and OS-independent.

3.4 Design Flow

Alternative Designs Considered:

- **Static-length Snake using arrays** (limited flexibility and harder to manage growth dynamically).
- **Snake with dynamic linked list body and modular board design (Selected)** – allows easier insertion and deletion for movement logic.

Game Loop Logic:

1. **Initialize board and snake**.
2. **Spawn food and bonus** in random empty locations.
3. **Wait for user input**, update snake direction.
4. **Move snake** based on direction:
    - Add new head node. ○ If food/bonus is eaten → retain tail.
    - Else → remove tail node.
5. **Check collisions** (walls, tail, obstacles).
6. **Update board and score**, render to console.
7. Repeat until gameOver condition is met.

3.5 Design Selection

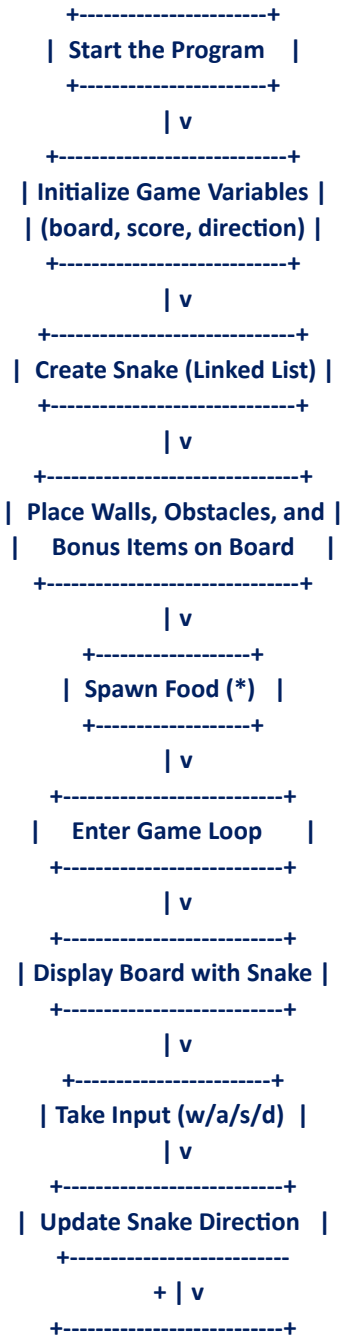
The final design — a **linked list-based dynamic snake** with **obstacles**, **bonus mechanics**, and **console rendering** — was chosen for the following reasons:

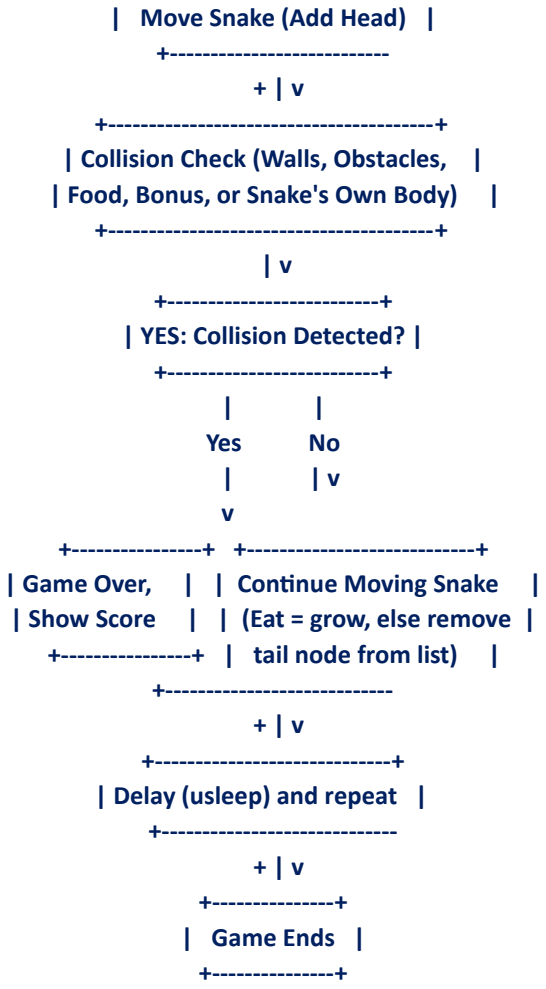
- **Flexibility**: Linked lists allow the snake to grow or shrink efficiently without resizing arrays.
- **Interactivity**: Real-time key input and immediate response enhance user engagement.

- **Scoring System**: Different items influence the game differently (normal food vs bonus item), making gameplay less monotonous.
- **Educational Value**: Demonstrates real-world application of data structures (linked list traversal, insertion, deletion).

The array-based approach with static sizing was rejected due to limited scalability and complexity in handling tail movement.

3.6. Implementation plan/methodology




| Move Snake (Add Head) |
+----------
+ | v
+----------+
| Collision Check (Walls, Obstacles, |
| Food, Bonus, or Snake's Own Body) |
+----------+
| v
+----------+
| YES: Collision Detected? |
+----------+
| |
Yes No
| | v
v
+----------+ +----------+
| Game Over, | | Continue Moving Snake |
| Show Score | | (Eat = grow, else remove |
+----------+ | tail node from list) |
+----------
+ | v
+----------+
| Delay (usleep) and repeat |
+----------
+ | v
+----------+
| Game Ends |
+----------+

CHAPTER 4

RESULTS ANALYSIS AND VALIDATION

4.1 Implementation of Solution

The Snake Game project was implemented using fundamental C++ programming techniques, with a focus on real-time movement, collision detection, and data structure application. Both the development and validation were aligned with academic and practical goals.

Key Development Activities:

- Logic Planning: The movement algorithm, tail tracking, and collision system were designed using structured pseudocode and step-by-step logic.
- Flowchart & Methodology: A visual box-based flowchart was used to represent the control flow of the game loop, input handling, and game-ending conditions.
- Data Structure Integration: A singly linked list was used to simulate the snake's growing and shrinking tail. Each node represented a part of the snake's body.
- Console-Based Rendering: The entire game was rendered in a terminal environment using simple characters, enhancing compatibility and reducing resource dependency.
- Documentation: Project structure, features, and testing processes were documented using a formal academic report layout.
- Game Management: Features like direction handling, bonus item logic, and board refresh were planned and tested across several iterations.

Implementation Details

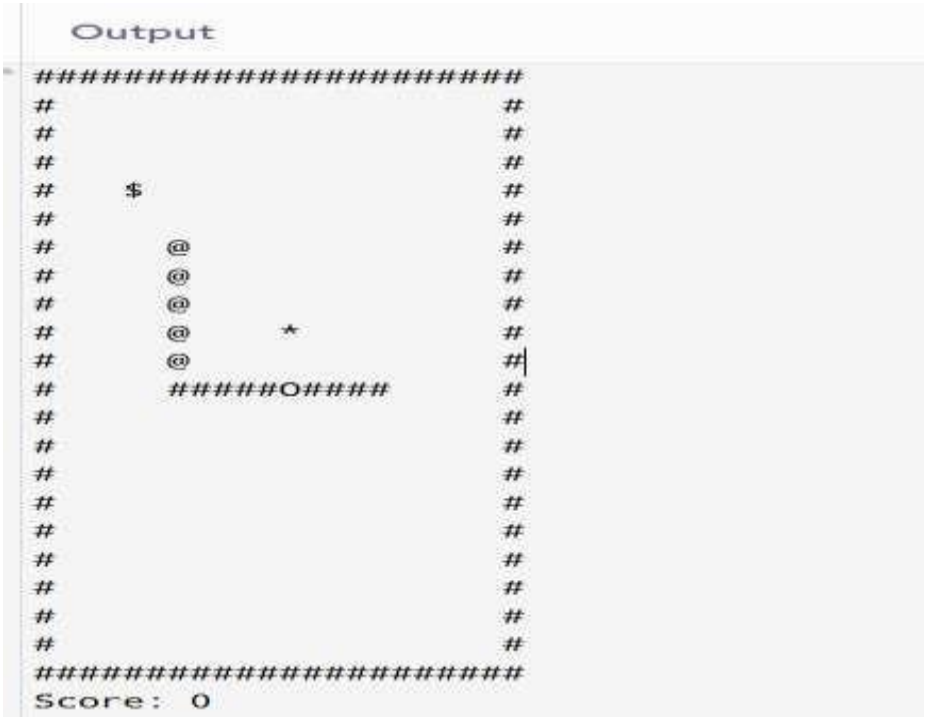
- Language Used: C++
- Data Structures:
    - Linked List: Used for dynamic snake body management.
    - 2D Array: For board rendering and object placement (board[20][20]).
- Input Method: cin to take real-time movement input (w, a, s, d)
- Output Method: system("clear") used to refresh console for each frame (on Linux/Mac; cls for Windows).
- Additional Features:
    - Bonus Items ($) increase score by 30. ○ Food (*) increases score by 10 and grows the snake. ○ Obstacles (@) and walls (#) end the game on collision.

Testing & Validation

The game was tested through multiple scenarios to ensure robust performance and correctness:

Test Case	Expected Result	Status
Snake eats food (*)	Snake grows, +10 score	☑ Passed
Snake eats bonus ($)	Score increases by 30	☑ Passed
Snake hits wall (#)	Game over	☑ Passed
Snake hits obstacle (@)	Game over	☑ Passed
Snake hits its own tail (o)	Game over	☑ Passed
Movement in all 4 directions	Works without glitches	☑ Passed
Real-time input + board refresh every 200 ms	Consistent experience	☑ Passed
Tail removal on non-food cell	Snake length remains consistent	☑ Passed

Output of Code



Validation Summary

- Real-Time Performance: Achieved using usleep(200000) for movement delay.
- Correct Data Structure Behavior: Linked list updates for each move were validated for correct head/tail manipulation.
- Gameplay Integrity: Snake growth, score management, and collision checks behaved as intended in all test scenarios.
- Terminal Compatibility: Tested and verified on Linux/Mac. Works with clear command; cls can be used for Windows with modification.

CHAPTER 5.

CONCLUSION AND FUTURE WORK

5.1 Conclusion

The Snake Game project successfully achieved its objective of creating an interactive, feature-rich console-based game using C++. The implementation utilized a **linked list** to represent the snake's body, allowing dynamic growth and real-time tail management. Core game mechanics like **food consumption**, **bonus items**, and **obstacle-based collisions** were implemented to make gameplay more challenging and engaging.

The scoring system accurately reflected game events, and the collision detection logic ensured robust behavior. Additional game elements like walls (#), obstacles (@), and bonuses ($) enhanced the complexity and strategic depth of the game.

The project also fulfilled educational goals by demonstrating how data structures like **linked lists** can be applied in real-time systems such as games. Overall, the game performed reliably during testing and met expectations in terms of performance, gameplay quality, and logic correctness.

5.2 Future Work

While the current version of the Snake Game demonstrates key concepts effectively, several enhancements can further elevate the experience and technical depth:

- **Graphical Interface (GUI)**: Replace console-based rendering with a graphical interface using **SFML**, **SDL**, or **OpenGL** to improve user engagement and visual appeal.
- **Data Persistence**: Implement file or database support to **store high scores**, **save game progress**, and allow players to resume previous sessions.
- **Difficulty Levels**: Add **increasing speed** or **moving obstacles** as levels progress to improve replay value and skill-based challenge.
- **Multiplayer Mode**: Allow two players to play simultaneously on the same system or over a network, increasing interactivity and fun.
- **Sound and Animation**: Introduce **audio cues** for events (eating food, collision) and **smooth transitions** in movement for better game feel.
- **Mobile/Platform Porting**: Adapt the game for mobile devices or microcontrollers (e.g., Arduino with LED matrices) to explore platform-specific optimization.

These enhancements would make the project suitable for deployment as a standalone application, increase complexity for academic learning, and prepare it for more advanced integration with other technologies.

REFERENCES

- Stroustrup, B. (2013). *The C++ Programming Language* (4th ed.). Addison-Wesley.
- Malik, D. S. (2017). *C++ Programming: From Problem Analysis to Program Design* (8th ed.). Cengage Learning.
- GeeksforGeeks. (n.d.). *Linked List in C++*. Retrieved June 2025, from https://www.geeksforgeeks.org/data-structures/linked-list/
- GeeksforGeeks. (n.d.). *Stack in C++ STL*. Retrieved June 2025, from https://www.geeksforgeeks.org/stack-in-cpp-stl/
- TutorialsPoint. (n.d.). *C++ - Multidimensional Arrays*. Retrieved June 2025, from https://www.tutorialspoint.com/cplusplus/cpp_multi_dimensional_arrays.htm
- TutorialsPoint. (n.d.). *C++ Graph Implementation*. Retrieved June 2025, from https://www.tutorialspoint.com/cplusplus-program-to-implement-graph-data-structure
- cppreference.com. (n.d.). *std::system - C++ Standard Library Reference*. Retrieved June 2025, from https://en.cppreference.com/w/cpp/utility/program/system
- Stack Overflow. (n.d.). *Using usleep() for delays in C++*. Retrieved June 2025, from https://stackoverflow.com/questions/4184468/sleep-for-milliseconds-in-c
- Programiz. (n.d.). *C++ Structures (struct)*. Retrieved June 2025, from https://www.programiz.com/cpp-programming/structure
- YouTube – CodeWithHarry. (n.d.). *C++ Game Projects Playlist*. Retrieved June 2025, from https://www.youtube.com/c/CodeWithHarry