CURRICULUM DESIGN OF DATA STRUCTURE

Design and Implementation of Dear Ex Snake Game

NAME：

STUDENT ID：

CLASS：

# CONTENTS

1. Statement and Analysis of the Task
   * 1.Functional Structure Diagram
2. **Data Structure Design**
   * 2.1 Logical Structure
   * 2.2 Storage Structure
3. **Algorithm Design**
   * 3.1 Input Handling
   * 3.2 Update Logic
   * 3.3 Rendering
   * 3.4 Game Over Handling
4. **System Implementation**
   * 4.1 Start Screen
   * 4.2 Gameplay Loop
   * 4.3 Game Over Screen
5. **System Description**
   * 5.1 Codin environment
   * 5.2 Runtime Environment
   * 5.3 System Performance
6. **Time and Space Complexity**
7. **Conclusion**
8. **Screenshots / Gameplay**
9. **References**

# Design and implementation of Dear Ex Snake Game

# 1. Statement and Analysis of the Task:

The **Dear Ex Snake Game** is a console-based interactive application written in C++, designed as a capstone project to demonstrate proficiency in **data structures**, **algorithm design**, and **real-time system programming**. The game is inspired by the classic "Snake" game but introduces a modernized interface, color-coded terminal interactions, and improved modular design.

The primary goal of the system is to simulate a dynamic and responsive environment where a snake (player-controlled) maneuvers within a bounded grid to consume randomly placed food. Each successful consumption increases the snake's length and the player's score, while also accelerating the gameplay pace. The game terminates when the snake collides with the wall or itself.

**Core Functional Objectives:**

* Accept **real-time keyboard input** using both arrow keys and WASD.
* Accurately maintain and update the **snake’s position** and **length** using an efficient data structure.
* Render the game board, snake, and food using **colored ASCII graphics** within a console window.
* Display **status information** such as current score and playtime (stamina).
* Handle game-over logic with animations and player feedback.

This system offers a clean separation of concerns between input handling, game logic processing, rendering, and system management, which are integrated to deliver a seamless gameplay experience.

* 1. Functional Structure Diagram

The system architecture is designed to reflect modular and reusable components. Each functional block contributes independently to the complete lifecycle of the game:

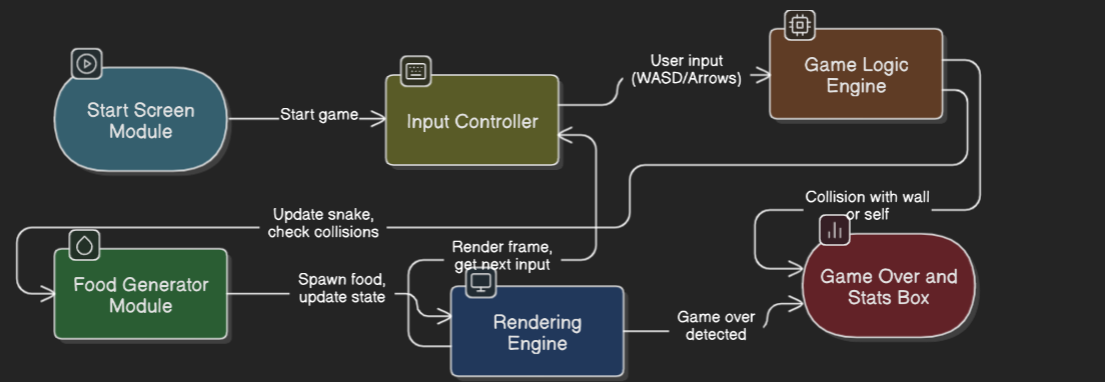


Figure 1.1: System Functional Structure Diagram

## 2. Data Structure Design

### 2.1 Logical Structure

The **Dear Ex Snake Game** leverages structured data modeling to represent in-game objects, their behaviors, and interactions. Below is an outline of the core logical elements and how they relate to each other.

**🔷 Snake Segment / Food – Point Structure**

The fundamental unit of position in the game is a 2D coordinate system represented using the Point structure:

struct Point {

int x, y;

bool operator==(const Point& o) const { return x == o.x && y == o.y; }

};=

* x, y – represent horizontal and vertical positions on the console grid.
* Overloaded equality operator allows easy comparison (e.g., to detect collision with food or body).

**🔷 Snake Body**

* Represented as a **linear sequence** of Point elements.
* The **head** is at the front of the list, and the **tail** is at the end.
* Movement is achieved by adding a new head and (usually) removing the tail.

**🔷 Food**

* A single Point placed randomly on the board.
* Logic ensures it never spawns inside the snake’s body.

**🔷 Game Board (Virtual Grid)**

* A **conceptual 2D grid** of size WIDTH × HEIGHT (e.g., 40×20).
* Boundaries are handled manually via conditions in the logic (e.g., checking wall collisions), not stored explicitly in memory.

**🔷 Status Elements**

* **Score**: Integer value incremented by 10 for every food item eaten.
* **Stamina**: Tracked using a chrono::steady\_clock timer that starts when the game begins and stops when it ends.

### 2.2 Storage Structure

To enable efficient game logic and real-time response, appropriate data structures have been used, notably:

**✅ Snake Body – std::deque<Point>**

* **Reason**: The snake must grow and shrink frequently from both ends.
* **Efficiency**: std::deque supports constant-time insertion/removal from both front and back.

deque<Point> body;

**Operations on deque<Point>**:

| **Operation** | **Reason** | **Time Complexity** |
| --- | --- | --- |
| push\_front() | Add new head (movement) | O(1) |
| pop\_back() | Remove tail (standard move) | O(1) |
| find() | Collision check | O(n) |

**✅ Food – Point**

* Stored as a single instance in memory.
* Generated using:

uniform\_int\_distribution<int> dx(1, WIDTH-2);

uniform\_int\_distribution<int> dy(1, HEIGHT-2);

* The coordinates are regenerated until they do not overlap with the snake’s body.

**❌ Explicit Grids or 2D Arrays**

* Not used for game board storage.
* Instead, logic and rendering are handled on-the-fly during traversal from (1,1) to (WIDTH-2, HEIGHT-2).

Summary of Storage Strategy:

| **Component** | **Data Type** | **Justification** |
| --- | --- | --- |
| Snake Body | deque<Point> | Efficient dynamic insertion/deletion |
| Food Position | Point | Lightweight and sufficient for single food element |
| Direction Enum | enum Direction | Clean and readable directional logic |
| Timer | chrono::steady\_clock | Accurate playtime tracking |

## 3. Algorithm Design

The **Dear Ex Snake Game** is architected around a structured **game loop**, following the core cycle:  
**Input → Update → Render → Sleep**, allowing it to operate in real-time while responding to user interactions.

The system consists of four algorithmically independent core components:

### 3.1 Input Handling

**Purpose:**  
To capture user input in real time using both WASD and arrow keys, enabling movement of the snake in four directions.

**Algorithm Details:**

* Use of \_kbhit() ensures **non-blocking** input detection.
* \_getch() captures character or extended key codes.
* Both **WASD keys** and **arrow keys** are supported.
* Prevents immediate reversal (e.g., from LEFT to RIGHT).

**Pseudocode:**

IF \_kbhit()

c ← \_getch()

IF c is extended key

arrow ← \_getch()

IF arrow == UP\_ARROW AND direction ≠ DOWN → direction ← UP

IF arrow == DOWN\_ARROW AND direction ≠ UP → direction ← DOWN

...

ELSE

IF c == 'w' AND direction ≠ DOWN → direction ← UP

IF c == 's' AND direction ≠ UP → direction ← DOWN

IF c == 'r' → reset game

IF c == 'x' → exit game

### 3.2 Update Logic

**Purpose:**  
To compute the snake's new position, check for collisions, grow the body if food is eaten, and manage game speed.

**Core Steps:**

* Calculate the next position of the snake’s head based on the current direction.
* Check if new head collides with wall or snake’s body.
* If collision: set alive = false.
* If new head equals food:
  + Increment score.
  + Reduce delay (speed up).
  + Do not remove tail → snake grows.
  + Generate new food.
* Else: remove tail → maintain size.

head ← front of body

Move head according to direction

IF head collides with wall OR itself → alive ← false

Insert head at front of deque

IF head == food

score += 10

delay -= 5

place new food

ELSE

remove last from deque

**Time Complexity:** O(n) — due to collision check (find in deque)

### 3.3 Rendering

### Purpose: To draw the entire game screen (border, snake, food, status) with colors using Windows API.

### Mechanism:

### Cursor position is reset using SetConsoleCursorPosition.

### Each grid cell is scanned from top-left to bottom-right.

### Snake head is drawn as 'O', body as 'o', and food as '\*'.

### Colors are applied based on the element:

### Border: Yellow

### Snake Head: Bright Red

### Snake Body: Red

### Food: Bright Green

### Status Bar:

### Score and stamina (elapsed time)

### Creator name

### Controls reminder

**Pseudocode:**

**SetConsoleCursorPosition(0, 0)**

**FOR each row y**

**FOR each col x**

**IF (x, y) == snake head → print 'O' (red)**

**ELSE IF (x, y) in body → print 'o' (red)**

**ELSE IF (x, y) == food → print '\*' (green)**

**ELSE → print ' ' (blank)**

**Print status bar:**

**Score, Time, Game Name, Author, Instructions**

### Time Complexity: O(WIDTH × HEIGHT)

### 3.4 Game Over Handling

**Purpose:**  
To gracefully terminate the game with a visual animation and summary stats.

**Mechanism:**

* Fade-out animation: fills screen with . for 10 frames.
* Clears screen and displays **Game Over Box** using color-coded borders and padding.
* Outputs:
  + Final Score
  + Total Stamina Time
  + Motivational message
* Awaits keypress with \_getch().

**Pseudocode:**

FOR i = 1 to 10

Clear screen with '.'

Sleep for 50ms

Clear screen

Print bordered box:

- GAME OVER!

- YOUR SCORE

- STAMINA: XX.XX sec

- THANKS FOR PLAYING

- Press any key to continue

Wait for keypress

### Time Complexity: O(WIDTH × HEIGHT)

### Space Complexity: O(1)

### 4. System Implementation

The **Dear Ex Snake Game** is implemented with attention to both logic and user experience. The game consists of three major stages, each designed with structured output and interactive elements. These stages are realized through modular components, ensuring clean code organization and maintainability.

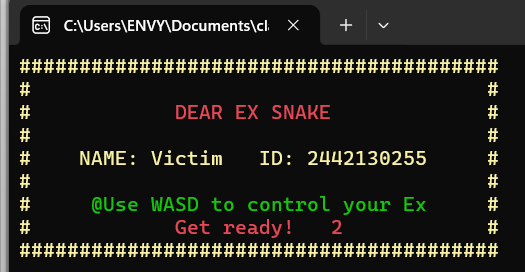
### 4.1 Start Screen

**Overview:**

The start screen acts as an introduction to the game, providing relevant player information and building engagement through an animated countdown.

**Key Features:**

* **Boxed Interface**: The welcome content is centrally aligned and enclosed within a # border for a clean visual layout.
* **Details Displayed**:
  + Game Title: DEAR EX SNAKE
  + Player Name and ID
  + Controls Guide: “Use WASD to control your Ex”
  + Countdown: “Get ready! 3”, “2”, “1” (displayed within the same line)



*Figure 4.1: Game start screen interface*

**Implementation Highlights:**

* Cursor positioning handled with SetConsoleCursorPosition.
* Color-coded text using SetConsoleTextAttribute.
* Countdown animation updates a specific line dynamically before starting the game.

### 4.2 Gameplay Loop

**Overview:**

This is the core of the application. The loop cycles through four essential operations:

1. **Input Handling**
2. **Game State Updating**
3. **Rendering**
4. **Frame Delay Control**

**What’s Rendered:**

* Snake head (O) in **bright red**
* Snake body (o) in **red**
* Food (\*) in **green**
* Borders (#) in **yellow**
* Score and Time (stamina) bar below the grid
* Developer signature and reminder of controls



*Figure 4.2: Gameplay interface*

**Performance Management:**

* Timing between frames is handled via std::chrono to ensure consistent speed.
* The game gets faster over time by decreasing the delay interval as the snake eats more food.

**Dynamic Behavior:**

* Snake grows upon eating food.
* Game ends when the snake hits a wall or itself.
* Player can restart (R) or quit (X) anytime.

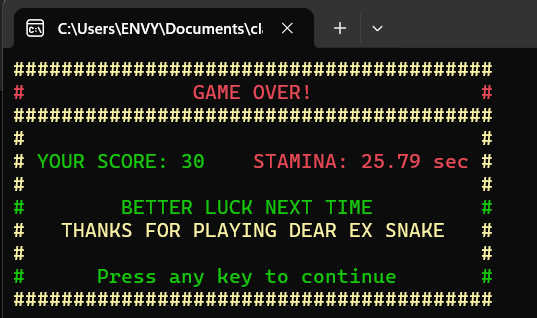
### 4.3 Game Over Screen

**Overview:**

A visually impactful ending is shown using console effects and formatted text, reflecting the user's performance and offering motivational closure.

**Animation:**

* Fade-out: fills the grid with . for 10 frames with 50ms delay.
* Transition to a formatted box screen with final results.



*Figure 4.3: Gameover interface*

**Highlights:**

* Utilizes ostringstream to align floating-point stamina values with fixed precision.
* Printed using printBoxLine() helper function to maintain symmetry and alignment.
* Use of color (red for stamina, green for success message) creates visual separation and hierarchy.

### 5. System System Descriptions

## 5.1 Coding Environment

The development of the *Dear Ex Snake Game* was performed in a Windows-based environment with the following configurations:

* **Programming Language**: C++ (C++11 and above standards for compatibility with chrono, thread, and STL containers)
* **Compiler**: MinGW (via **Code::Blocks** IDE) or Microsoft C++ Compiler (via Visual Studio)
* **Editor**:
  + Primary: **Code::Blocks**
  + Alternative: **Visual Studio 2019/2022** with Windows SDK and console subsystem enabled
* **Libraries Used**:
  + <windows.h> — for terminal graphics and cursor control
  + <conio.h> — for non-blocking keyboard input
  + <deque> — to model the snake body
  + <chrono>, <thread> — to control game timing and animation
  + <iomanip>, <iostream> — for formatted console output

This environment was chosen for its full support of Windows Console APIs and rapid terminal rendering capabilities.

## 5.2 Running Environment

The game is intended to be executed in a **Windows command-line interface** (CMD or PowerShell). Below are the supported and tested runtime specs:

* **Operating System**: Windows 10, Windows 11
* **Console Mode**: UTF-8 encoding (Set via SetConsoleCP(CP\_UTF8) and SetConsoleOutputCP(CP\_UTF8))
* **Minimum Hardware**:
  + 1.5GHz Dual-Core CPU
  + 2 GB RAM
  + No GPU required
* **Terminal Settings**:
  + Console window size is automatically configured
  + Cursor hidden for clean rendering
  + Supports color via ANSI and WinAPI color codes

No installation is required — it is a standalone executable that runs directly in the terminal.

## 5.3 System Performance

The game is highly optimized for real-time interaction and rendering in a lightweight command-line environment. Key performance characteristics:

| **Feature** | **Implementation Benefit** |
| --- | --- |
| std::deque<Point> | O(1) operations for head/tail in snake movement |
| Double-buffered logic | Prevents screen flickering |
| Direct cursor control | Redraws only the needed areas |
| Adaptive speed | Reduces delay dynamically for increasing difficulty |

* **Average Frame Rate**: Maintains consistent ~10 FPS, depending on system performance
* **Memory Usage**: Minimal — only a few KBs, since only coordinates and small state are stored
* **Latency**: Keyboard input is read non-blocking for instant response

This architecture guarantees a smooth user experience even on low-end systems.

## 6. Time and Space Complexity

Understanding the performance characteristics of each core component helps assess the system's efficiency:

| **Component** | **Time Complexity** | **Space Complexity** | **Details** |
| --- | --- | --- | --- |
| **Input Handling** | O(1) | O(1) | Captures key press using \_kbhit() and \_getch() instantly |
| **Snake Movement** | O(1) | O(n) | Adds new head and removes tail using deque operations |
| **Food Placement** | O(n) worst-case | O(1) | Generates new food until a free spot is found |
| **Collision Detection** | O(n) | O(n) | Linear scan to verify if head overlaps with the body |
| **Rendering** | O(WIDTH × HEIGHT) | O(1) | Each frame redraws the entire visible area using nested loops |
| **Game Over Animation** | O(WIDTH × HEIGHT × k) | O(1) | Pixel-style dot fade repeated k times across the whole screen |

## 7. Conclusion

The *Dear Ex Snake Game* is a well-structured and nostalgic reinterpretation of the classic snake game using C++. It successfully integrates core concepts from data structures and algorithms, such as:

* **Deque usage** for flexible and efficient snake modeling
* **Randomized logic** for food generation
* **Color-rendered terminal UI** using low-level Windows API
* **Real-time input and animation**, emphasizing responsiveness and timing

The project not only reinforces theoretical knowledge in computer science but also challenges students to think creatively about optimization and user experience in resource-limited environments like the terminal.

It stands as a demonstration of blending classical game mechanics with structured programming principles for educational and entertainment purposes.

## 8. Screenshots / Gameplay

| **Feature** | **Screenshot Reference** |
| --- | --- |
| Diagram | Figure 1 — System Functional Structure Diagram |
| Start Screen | Figure 4.1 — Game Welcome Screen |
| In-Game State | Figure 4.2 — Gameplay Interface |
| Game Over Summary | Figure 4.3 — Game Over Box |
| Gameplay Video | [Watch it](https://streamable.com/e/3z5pwm) — real-time input, food, border, color, and game over animation. |

9. **References**

[1] [GeeksforGeeks](https://www.geeksforgeeks.org/c/snake-game-in-c/) *Implementation of Snake Game in C++*.

[2] [Microsoft Docs](https://learn.microsoft.com/en-us/windows/console/console-functions) *Console Functions - Windows Console API*.

[3] [Cplusplus.com](https://cplusplus.com/reference/deque/deque/) *std::deque - C++ Reference*.

[4] [Github Repo](https://github.com/Tux-MacG1v/Dear-Ex-Snake-game) *Dear Ex Snake Game – GitHub Repository*.

[5] [Stack Overflow](https://stackoverflow.com/questions/5867544/how-to-read-a-single-character-from-console-input) *How to use \_kbhit() and \_getch() for Real-Time Keyboard Input in C++?*