Snake Game - Technical Architecture Document

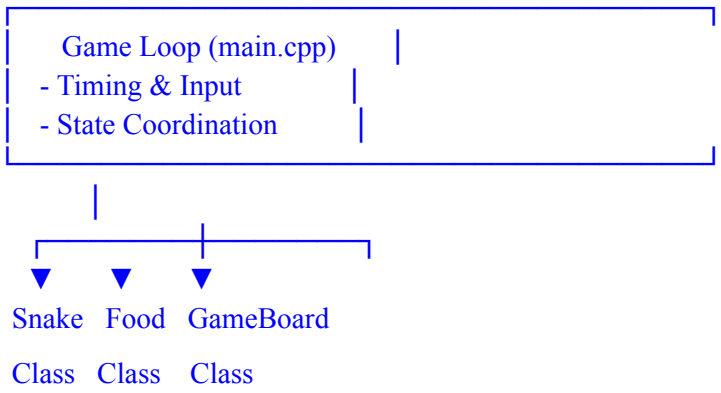
Course: Adv Prog of Business Sys in C++
Team Members: Jedidiah Anselm, Game Systems Programmer
MakaylaGreen, Interface Menu Programmer
Makai Hurst, Gameplay Support Programmer and GitHub Coordinator
Tanna James,
Miriangie Rondon Mota, Technical Project Manager & Programmer
Date: January 26, 2026
Project: Snake Game Implementation in C++

1. PROJECT OVERVIEW

Snake is a classic arcade game where the player controls a continuously moving snake on a grid. The objective is to consume food items to grow the snake and increase score. The game ends when the snake collides with walls or itself. This project will be implemented in C++ using object-oriented programming principles in a console environment.

2. SYSTEM ARCHITECTURE

2.1 High-Level Design



2.2 File Structure

```
src/
├── main.cpp          - Entry point and game loop
├── Snake.h/.cpp      - Snake management
├── Food.h/.cpp       - Food spawning
└── GameBoard.h/.cpp  - Rendering and display
```

3. DATA STRUCTURES

Point Structure

cpp

```
struct Point {
    int x, y;  // Grid coordinates
};
```

Direction Enumeration

cpp

```
enum Direction { UP, DOWN, LEFT, RIGHT };
```

```
```

Snake Body: `std::deque<Point>` - Efficient for adding head and removing tail

Game Board: `char gameBoard[20][20]` - 2D array representing the grid
- `' '` = Empty, `'#'` = Wall, `'O'` = Head, `'o'` = Body, `'*'` = Food

---

4. CLASS ARCHITECTURE

4.1 Snake Class
Responsibility: Manages snake behavior and state

Key Members:
- `std::deque<Point> body` - Snake segments
- `Direction direction` - Current movement direction
- `bool growing` - Growth flag

Key Methods:
- `Snake(int startX, int startY)` - Initialize at position
- `void move()` - Advance one step
- `void setDirection(Direction newDir)` - Change direction (validates against reversal)
- `void grow()` - Mark for growth
- `Point getHead() const` - Get head position
- `bool isOnSnake(int x, int y) const` - Collision check

4.2 Food Class
Responsibility: Manages food spawning and detection

Key Methods:
- `void spawn(const Snake& snake)` - Random valid position
- `Point getPosition() const` - Current location
- `bool isEaten(const Point& snakeHead) const` - Consumption check

4.3 GameBoard Class
Responsibility: Rendering and boundary collision

Key Methods:
- `void render(const Snake& snake, const Food& food, int score)` - Draw state
- `bool isWall(int x, int y) const` - Wall collision
- `void clearScreen()` - Clear console

---

5. GAME LOOP
```
Initialize Snake, Food, GameBoard

WHILE game running:
```

1. Render current state
2. Check keyboard input → update direction
3. Move snake
4. Check food collision → grow, update score, spawn new food
5. Check wall/self collision → end game
6. Delay 150ms (game speed)

Display final score

6. REQUIRED LIBRARIES

Standard:

- <iostream> - I/O
- <deque> - Snake body
- <cstdlib>, <ctime> - Random numbers

Platform-Specific:

- **Windows:** <windows.h>, <conio.h> - Sleep and keyboard input
- **Linux/Mac:** <unistd.h> - Sleep function

7. KEY ALGORITHMS

Movement (O(1))

- Calculate new head based on direction
- Add new head to front
- Remove tail (unless growing)

Collision Detection

- **Self:** O(n) - Check head against all body segments
- **Wall:** O(1) - Check if head at boundary
- **Food:** O(1) - Compare head with food position

Food Spawning (Average O(n))

1. Generate random coordinates
2. Verify not on snake or walls
3. Repeat until valid

8. GAME PARAMETERS

Parameter	Value
Grid Size	20×20
Starting Speed	150ms per move

Initial Length	3 segments
Food Points	10 per item

9. COLLISION SYSTEM

Wall Collision: Head at x=0, x=19, y=0, or y=19
Self Collision: Head coordinates match any body segment
Food Collision: Head coordinates match food position

10. COMPILATION

bash

```
g++ -std=c++11 -Wall main.cpp Snake.cpp Food.cpp GameBoard.cpp -o snake
```

Execution:

bash

```
./snake        # Linux/Mac
snake.exe      # Windows
```

11. DEVELOPMENT TIMELINE

- **Week 1-2:** Architecture and planning
- **Week 3:** Core class implementation
- **Week 4:** Game loop integration
- **Week 5:** Collision detection refinement
- **Week 6:** Testing and polish
- **Week 7:** Final documentation

12. OPTIONAL ENHANCEMENTS

- Difficulty levels
- High score persistence
- Pause functionality
- Speed progression
- Color support

Prepared By: Miriangie Rondon
Version: 1.0
Date: January 26, 2026