Snake Game in C++ – Documentation

Crishtina KC

Student ID: 23085130

The British College, Kathmandu

Cyber Security and Digital Forensics

20th April 2025

Overview

This is a console-based Snake Game, a classic arcade-style game written in C++, with a few modern enhancements:

- Colored UI using Windows console app is.
- Moving Snake (Using W- for Up, A – for left, S – For Down, D – for right)
- Series of levels (levels changes after eating 5 *)
- Progressive levels with increasing speed
- Life system and scoring
- Tail growth mechanics (Tail increases after eating *)
- Tail becomes 0 and game restart after losing 3 life.
- Save game scores to a file (score.txt)

The game is played entirely in the terminal and offers a fun and interactive way to practice core programming skills like arrays, structs, loops, conditionals, and file I/O.

Libraries and Their Purpose

```
#include <iostream>     // For input and output
#include <conio.h>      // For _kbhit() and _getch() – real-time key press detection
#include <windows.h>    // For console coloring, cursor manipulation
#include <ctime>        // For time-related functions (e.g., seeding random)
#include <cstdlib>      // For rand() and srand()
#include <fstream>      // For file operations to save scores
#include <cmath>        // For math functions like pow()
```

This game will only compile and run on Windows due to conio.h and windows.

Global Constants

```
const int WIDTH = 30;
const int HEIGHT = 20;
```

Defines the playable game grid

```
const char WALL = 219;
const char PATH = ' ';
const char SNAKE_HEAD = 'O';
const char SNAKE_BODY = 219;
const char FRUIT = '*';
const char ENEMY = 'X';
```

Defines how each game element appears in the console.

Global Variables

```
int x, y, fruitX, fruitY;
```

- x, y: Position of the snake head
- fruitX, fruitY: Position of the fruit

```
int score, lives, speed, level;
```

- score: Current score

- Lives remaining
- speed: Time delay between moves (in milliseconds)
- level: Current level (increases with score)

```
int tailX[100], tailY[100];
int nTail;
```

- Arrays to track the coordinates of the snake's tail segments
- nTail: Current length of the tail

```
bool gameOver;
```

- Flag to determine if the game has ended

```
enum Direction { STOP = 0, LEFT, RIGHT, UP, DOWN };
Direction dir;
```

- Enum to keep track of movement direction

```
HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
COORD cursorPos;
```

- For cursor positioning and color manipulation in the console

Structs

Enemy

```
struct Enemy

{
   int x, y;
};
Enemy enemies[10];
```

Each enemy has x and y coordinates. A maximum of 10 enemies can be created as levels increase.

Utility Functions

SetCursorPosition(int x, int y)

Moves the console cursor to position (x, y) to create the illusion of a continuously updating screen (no flickering).

SetColor(int color)

Changes text color using Windows console attributes. Used to color snake, fruit, walls, score text, etc.

HideCursor()

Disables the blinking console cursor for cleaner display.

Game Lifecycle Functions

Setup()

Initializes game variables:

- Snake position is set to center
- Fruit is randomly placed
- Enemy positions are randomly generated and do not overlap with the snake or fruit
- Level and speed are initialized

Draw(char gameScreen[HEIGHT][WIDTH + 2])

Use a 2D buffer to build the visual scene:

- Draws top and bottom walls
- Side walls
- Places snake head and tail
- Places fruit
- Places enemies
- Displays score, lives, and level at the bottom

Optimized for performance by using buffer and SetCursorPosition(0, 0) to redraw over existing frame instead of clearing screen each time.

Input()

Non-blocking input using _kbhit() and _getch():

- W, A, S, D: control movement
- X: exit the game

Also prevents the snake from reversing direction into itself.

Logic()

Handles:

- Snake movement and tail following logic
- Fruit collision:
    - Increase score
    - Grows Tail
    - Adds enemies and increases level after certain thresholds
    - Recalculates speed with increasing difficulty
- Collision with:
    - Itself (resets tail and reduces lives)
    - Enemies (same behavior as above)
- Wall wrap-around mechanics (snake appears on opposite side)

SaveScore()

Writes the final score to a file (score.txt) for later reference.

RestartOrExit()

After the game is over, prompts the user to:

- Restart the game with a fresh setup
- Or exit gracefully

ShowMainMenu()

Displays a start menu with options:

1. Start Game
2. Exit

GameOver()

Called when lives run out. Shows:

- "Game Over"
- Final score
- Calls Restart Or
- Exit () to give player another chance

Main Loop

```
int main()
{
   srand(time(0));     // Seed randomness
   HideCursor();       // Clean visual
   ShowMainMenu();     // Display main menu
   if (gameOver) {
      GameOver();     // Trigger game over screen if ended
   }
```

```
    return 0;
}
```

Game Progression

- Leveling Up: Every 40 points (4 * level) increase the level.
- Difficulty Scaling:
    - New enemy is added per level

```
speed = 100 - (int)(5 * pow(1.5, level - 1));
```

Key Concepts Demonstrated

- Real-time user input handling
- Snake tail tracking using arrays
- Procedural generation (fruit/enemy spawning)
- Game loop mechanics
- Console manipulation (cursor, color, flicker reduction)
- File I/O for score saving
- Object-oriented design (enemies as struct)

Potential Improvements

- Add high score system
- Implement pause/resume
- Save game state and load later

- Improve enemy AI movement
- Port to Linux (remove conio.h and windows.h dependencies)