

CS 355-02 Spring 2025  
Alexis Baranauskas, and Edmond Cani  
Final Project : Snake Game in C

## Table of Contents

Abstract .....	pg 3
Acknowledgement .....	pg 3
Introduction .....	pg 3
Project Description .....	pg 3–4
○ Aim of the Project .....	pg 3
○ Challenges .....	pg 4
○ Solution Implementation .....	pg 4
Tutorial on Using the Program .....	pg 4-5
○ Step-by-Step Instructions .....	pg 4
○ Examples and Use Cases .....	pg 5
Code Details .....	pg 5-6
○ Explanation of Code Functionality .....	pg 5
○ Key Code Segments .....	pg 6-7
Conclusion .....	pg 7
○ Summary of Findings and Achievements .....	pg 7
Contribution of Team Members .....	pg 8
Course Reflection .....	pg 9
○ How the Course Facilitated Project Development .....	pg 9
○ Suggestions for Course Improvement .....	pg 9
Appendices .....	pg 9–10
○ Additional Material .....	pg 9
○ Source Code .....	pg 10

**Abstract:** This project is a recreation of the popular “Snake Game”, stripped down to the basics and built in C using ncurses library. Our project follows the same classic concept: you play as a snake and must eat food to grow larger, and avoid collisions with the border and yourself along the way. Key implementations include color codes graphics, timed trophies and collision detection utilizing position based array structure. With this project we were able to demonstrate game development concepts through this simple C terminal based game.

**Acknowledgement:** We would like to acknowledge the CS department at CCSU, and more specifically our Systems Programming Professor, Md Rafiul Hassan s for guidance and teaching us this spring semester.

**Introduction:** Our snake game is a perfect example for a simple interactive program, as it replicates gameplay like the classic snake game, just in a text-based terminal by utilizing the ncurses library. It was a great exercise to test not only our programming skills, but our teamwork and problem solving skills, and we were able to produce our version of the classic game with real time input handling, dynamic screen updating, and overall game logic in C programming.

## **Project Description**

**Aim of the project:** The goal of our project was to recreate the classic snake game at a much lower level, utilizing only terminal capabilities. We wanted to make it just as fun and challenging as the original, and make it look as similar to the snake game we all know and love. We implemented color coded graphics, and time limited trophies to achieve this. We also wanted a smooth game play experience, one that any user could enjoy.

**Challenges:** During our construction and testing of the snake game we ran into a couple problems along the way, one being ensuring that the snake and the apple or “trophy” did not overlap or spawn in the same exact spot. We also ran into an issue with the screen terminal appearing to flicker upon the redraws, along with handling user input without freezing or crashing.

**Solution Implementation:** To address these challenges we worked together as a team and were able to successfully fix these issues. To prevent the snake and apple from overlapping we ensured the apple was randomly generated at a set of coordinates that were checked against the snake's current position, which ensured the apple would not “spawn” on the snake. As for screen flicker we utilize the ncurses library, specifically clear() and refresh() to minimize flickering as best as we could. As for handling input we use the timeout() function along with some timing adjustments based on the snake, which has responsive controls (arrow keys) without freezing or crashing.

### **Tutorial on Using the Program Step-by-step instructions:**

To replicate our exact results please follow the following instructions

1. Open Google Cloud Shell
2. Create the source file - Copy our source code in and name it something like snake.c
3. Compile the code - Gcc snake.c o-snake -lncurses
4. Run the game - ./snake

### **Game Instructions**

- The game will begin immediately upon running
- Use the arrow keys on your keyboard to control where the snake moves

- When an apple ( a red @) appears on the screen guide the snake to “eat” it
- Each apple / trophy consumed adds onto the length of the snake
- Avoid colliding into walls, or yourself as this will end the game
- You will win when the snake grows to half the playing area

**Examples and use cases:** This code can be a great teaching tool or example to show how to use ncurses library for terminal bases graphics, and how to create basic game mechanics within C. It also covers user input handling, collision detection, along with loops and timing.

**Code details Explanation of Code Functionality:** Our snake game uses the ncurses library to recreate the classic snake game with the following key components:

- Data Structure: Snake body segments stored in a Position array
- Loop: Handles input, updates game state, and refreshes display
- Collision Detection: Checks for border and self-collisions
- Dynamic Updates: Game speed increases as snake grows

### **Key code segments:**

#### **Movement & Collision Detection**

```
void update_snake(int d_columns, int d_rows) {
    Position new_head = {snake[0].columns + d_columns, snake[0].rows +
d_rows};

    // Border & self collision checks
    // Move snake by shifting all segments
    for (int i = snake_size - 1; i > 0; i--) {
```

```

        snake[i] = snake[i - 1];
    }
    snake[0] = new_head;

    // Trophy consumption
    if (new_head.columns == trophy_columns && new_head.rows == trophy_rows)
    {
        snake[snake_size] = snake[snake_size - 1]; // extend the tail
        snake_size += 1;
        trophy_active = 0;
    }
}

```

### **Screen Refresh & Input Handling**

```

clear();
draw_borders();
draw_trophy();
update_snake(d_columns, d_rows);
draw_snake();
refresh();

timeout(130 - (snake_size * 1.3));
t
int ch = getch();
switch (ch) {
case KEY_UP: if (d_rows == 0) { d_rows = -1; d_columns = 0; } break;

```

## **Trophy Generation**

```
void generate_trophy() {  
    trophy_columns = rand() % (WIDTH - 2) + 1;  
    trophy_rows = rand() % (HEIGHT - 2) + 1;  
    trophy_active = 1;  
    time(&trophy_timer); // Start 10-second time  
}
```

**Conclusion:** Throughout working on this project it has shown us a fun responsive way to program, as we were successfully able to recreate the snake game in C programming. Throughout developing this we have grown our understanding and shown progress in utilizing the ncurses library, as we gained experience with screen manipulation, coloring graphics, and input handling which gives us a solid foundation to build upon should we choose to develop any other terminal based applications. We also were able to overcome technical challenges such as screen flickering during redraw, collision detection, and handling inputs without freezing in loops. We were able to develop our programming skills not only technically but logically as well, as this project showed us just how important teamwork and communication is.

## **Contribution of Team Members:**

### **Alexis Baranauskas:**

- Implemented apple “trophy” spawning, timing, and logic.
- Took lead on snake movement logic and border collision detection.
- Integrated trophy consumption into the snake growth system.
- Wrote Abstract, Acknowledgment, Introduction, and Conclusion.
- Helped assemble presentation slides.

### **Edmond Cani:**

- Managed game loop control and pacing (dynamic speed adjustment).
- Assisted in snake movement logic (tail shifting and self-collision checks).
- Implemented trophy generation validation (ensuring apples don’t spawn inside the snake).
- Wrote Project Description and Course Reflection (gathered teammates’ opinions).
- Helped assemble presentation slides.

## **Course Reflection**

**How the Course Facilitated Project Development:** Throughout the semester we learned a lot about C programming, giving us a strong foundation which was crucial for this project, in terms of logic, memory management and input handling. Learning more in depth about arrays, structs and the specific ncurses functions helped us organize the structure of our code.

**Suggestions for Course Improvement:** More Organized syllabus / lecture structure could benefit us more, guided homework assignments may be beneficial as well.

## **Appendices Additional Resources:**

Low Level Learning. Making snake with C and Ncurses. YouTube, 19 Dec. 2022, <https://youtu.be/IV-OPQhPvSM>.

Singh, Gurkirat. "Introduction to Ncurses (Part 1)." DEV Community, 2 May 2020, <https://dev.to/tbhaxor/introduction-to-ncurses-part-1-1bk5>.

The Coding Train. Coding Challenge #115: Snake Game. YouTube, 30 July 2018, [https://youtu.be/ufrPump5\\_aA](https://youtu.be/ufrPump5_aA)

## Source Code:

```
// Alexis Baranauskas, Edmond Cani
// CS 355
// Snake Game Project

#include <ncurses.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#define WIDTH 100
#define HEIGHT 25
#define INIT_SNAKE_SIZE 4
#define TROPHY_DURATION 10

typedef struct {
    int columns, rows;
} Position;
// Game variables
Position snake[WIDTH * HEIGHT];
int snake_size;
int game_over;
int trophy_columns, trophy_rows;
int trophy_active;
time_t trophy_timer;
int win_condition;
// Initialize game
void init_game() {
    initscr(); // Start ncurses
    cbreak();
    noecho();
    curs_set(0); // Hide cursor
    keypad(stdscr, TRUE); // Enable arrow keys
    timeout(300); // Initial delay
    // Set colors
    if (has_colors()) {
        start_color();
        use_default_colors();
        init_pair(1, COLOR_GREEN, -1); // Snake
        init_pair(2, COLOR_RED, -1); // Game Over message
```

```

init_pair(3, COLOR_RED, -1); // Apple trophy
}
// Center & initialize the snake
int init_columns = WIDTH / 2 - INIT_SNAKE_SIZE / 2;
int init_rows = HEIGHT / 2;
for (int i = 0; i < INIT_SNAKE_SIZE; i++) {

snake[i].columns = init_columns + i;
snake[i].rows = init_rows;
}
snake_size = INIT_SNAKE_SIZE;
trophy_active = 0;
int total_area = (WIDTH - 1) * (HEIGHT - 1);
win_condition = total_area / 2;
game_over = 0;
}
// Draw borders
void draw_borders() {
attron(A_BOLD);
for (int i = 0; i <= WIDTH; i++) {
mvprintw(0, i, "-");
mvprintw(HEIGHT, i, "-");
}
for (int i = 0; i <= HEIGHT; i++) {
mvprintw(i, 0, "|");
mvprintw(i, WIDTH, "|");
}
// Add corners
mvprintw(0, 0, "+");
mvprintw(0, WIDTH, "+");
mvprintw(HEIGHT, 0, "+");
mvprintw(HEIGHT, WIDTH, "+");
attroff(A_BOLD);
}
// Draw snake
void draw_snake() {
attron(COLOR_PAIR(1));
for (int i = 0; i < snake_size; i++) {

```

```

mvprintw(snake[i].rows, snake[i].columns, "o");
}
attroff(COLOR_PAIR(1));
}
// Draw apple (trophy)
void draw_trophy() {
attron(COLOR_PAIR(3) | A_BOLD);
mvprintw(trophy_rows, trophy_columns, "@");
attroff(COLOR_PAIR(3) | A_BOLD);
}

// Generate apple
void generate_trophy() {
trophy_columns = rand() % (WIDTH - 2) + 1;
trophy_rows = rand() % (HEIGHT - 2) + 1;
trophy_active = 1;
time(&trophy_timer);
}
// Move and update snake state
void update_snake(int d_columns, int d_rows) {
Position new_head = {snake[0].columns + d_columns, snake[0].rows + d_rows};
// Border collision
if (new_head.columns == 0 || new_head.columns == WIDTH ||
new_head.rows == 0 || new_head.rows == HEIGHT) {
game_over = 1;
return;
}
// Self collision
for (int i = 1; i < snake_size; i++) {
if (snake[i].columns == new_head.columns && snake[i].rows == new_head.rows) {
game_over = 1;
return;
}
}
// Move snake
for (int i = snake_size - 1; i > 0; i--) {
snake[i] = snake[i - 1];
}
}

```

```

snake[0] = new_head;
// Eat apple
if (new_head.columns == trophy_columns && new_head.rows == trophy_rows) {
snake_size += 1;
trophy_active = 0;
}
// Win
if (snake_size >= win_condition) {
game_over = 2;
}
}
// Display end game message
void display_game_over_message(const char* message) {
clear();
draw_borders();

int message_length = strlen(message);
int center_x = WIDTH / 2;
int center_y = HEIGHT / 2;
attron(COLOR_PAIR(2) | A_BOLD);
mvprintw(center_y, center_x - (message_length / 2), "%s", message);
attroff(COLOR_PAIR(2) | A_BOLD);
refresh();
napms(5000);
getch();
}
// Main loop
int main() {
srand(time(NULL));
int random_direction = rand() % 4;
int d_columns = 0, d_rows = 0;
switch (random_direction) {
case 0: d_columns = 1; break;
case 1: d_columns = -1; break;
case 2: d_rows = 1; break;
case 3: d_rows = -1; break;
}
init_game();

```

```

while (!game_over) {
    int ch = getch();
    // Changing snake direction
    switch (ch) {
        case KEY_UP: if (d_rows == 0) { d_rows = -1; d_columns = 0; } break;
        case KEY_DOWN: if (d_rows == 0) { d_rows = 1; d_columns = 0; } break;
        case KEY_LEFT: if (d_columns == 0) { d_columns = -1; d_rows = 0; } break;
        case KEY_RIGHT: if (d_columns == 0) { d_columns = 1; d_rows = 0; } break;
    }
    clear();
    draw_borders();
    timeout(130 - (snake_size * 1.3)); // dynamic speed
    if (!trophy_active) {
        generate_trophy();
    }
    draw_trophy();

    update_snake(d_columns, d_rows);
    draw_snake();
    mvprintw(HEIGHT + 1, 1, "Score: %d", snake_size - INIT_SNAKE_SIZE);
    refresh();
    // Game end
    if (game_over == 2) {
        display_game_over_message("WINNER!");
        break;
    }
    if (game_over == 1) {
        display_game_over_message("GAME OVER :(");
        break;
    }
    // Apple lasts for 10 seconds
    time_t current_time;
    time(&current_time);
    double diff = difftime(current_time, trophy_timer);
    if (trophy_active && diff >= TROPHY_DURATION) {
        trophy_active = 0;
    }
}

```

```
}  
endwin(); // cleanup ncurses  
return 0;  
}
```