Green University of Bangladesh

Department of Software Engineering (SWE)
Semester: (Fall, Year: 2024), B.Sc. in SWE

The Fun side of Coding - Snake Game

Course Title: Structured Programming Lab
Course Code: SWE-104
Section: 242_D1

<u>Students Details</u>

Name	ID
MD. Mostafijul Islam	242034012

Submission Date: 19th December, 2024.
Course Teacher's Name: Ms. Aysha Banu

[For teachers use only: Don't write anything inside this box]

Lab Project Status

Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

The Snake Game is a simple yet iconic arcade game that has been popular since the late 1970s. The game-play revolves around controlling a snake that moves around the screen to eat food, causing it to grow longer with each bite. Despite its simplicity, the Snake Game is an excellent example of implementing fundamental programming concepts such as user input handling, and dynamic data structures. This project aims to recreate the classic game.

1.2 Motivation

The motivation behind creating the Snake Game project stems from several key factors:

- **Classic Appeal**
  The Snake Game is one of the most recognizable and timeless games, enjoyed by people of all ages. Its nostalgic value and simple mechanics make it an exciting project to recreate while preserving its essence.

- **Learning Opportunity**
  Developing the Snake Game provides an excellent opportunity to strengthen programming skills. It allows the implementation of key concepts such as handling user input, managing real-time updates, and detecting collisions, which are fundamental in game development.

- **Logical Thinking and Problem Solving**
  The project encourages logical thinking by requiring solutions to real-time challenges, such as updating the movement of the snake, growing the snake dynamically and preventing unintended collisions.

- **Fun and Interactive**
  Working on a game project adds an element of fun and interactivity to programming, making it more engaging compared to traditional coding tasks.

By combining learning, problem-solving, and creativity, the Snake Game project serves as an ideal platform for exploring both programming concepts and game development.

1.3 Goal

The primary goal of this project is to design and implement a functional version of the classic Snake Game while focusing on programming principles, user interactivity, and logical problem-solving. The specific objectives of this project include:

1. To recreate the Snake Game with simple snake game.
2. To strengthen fundamental programming skills like including input handling.
3. To gain hands-on experience in debugging, testing, and refining game logic.
4. To demonstrate creativity and problem-solving skills through the successful completion of the game.

By achieving these goals, this project aims to provide a solid understanding of basic C programming language concepts while offering a fun and interactive experience.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

Building upon the foundational concepts outlined in the introductory chapter, this section delves into the development and implementation of the Snake Game project. The project recreates the classic Snake Game, emphasizing the application of essential programming principles in C, such as handling user inputs, and efficient game state updates.

The implementation leverages the Ncurses library to create an interactive, terminal-based interface that mimics the game-play of the original Snake Game. This approach demonstrates the use of low-level system calls and graphical handling in C, providing a deeper understanding of how console-based applications can be developed.

This chapter outlines the technologies used, input-output mechanisms, algorithm design, and data structures used in the project, providing a comprehensive view of the development process. Each subsection provides specific details about the underlying logic and the practical challenges encountered during implementation.

2.2 Technologies Required

The development of the Snake Game required several key technologies to ensure smooth implementation and execution. These technologies include a suitable compiler and programming environment, the Ncurses library for terminal-based graphical output, and an Integrated Development Environment (IDE) for efficient coding and debugging. Each is detailed below:

- **Compiler and C Environment:** The Snake Game was implemented in the C programming language, necessitating a reliable C compiler. GCC (GNU Compiler Collection) was chosen for this project due to its wide availability, robustness, and compatibility with multiple platforms. The GCC compiler provides an efficient environment for compiling, debugging, and optimizing C programs, making it ideal for the development of this project.

- **Ncurses Library:** The Ncurses library is a powerful tool for creating terminal-based applications with graphical features. It provides functions for managing text-based windows, handling user inputs, and creating dynamic visual elements, which are critical for a game like Snake. Using Ncurses, the project achieves smooth rendering of the game grid, the snake's movement, and the placement of food, all while maintaining compatibility with various terminal environments.
- **Integrated Development Environment (IDE):** An Integrated Development Environment streamlines the development process by providing tools for writing, editing, and debugging code. For this project, `[insert IDE name]` was used. The chosen IDE offered features like syntax highlighting, code navigation, and integrated terminal access, which facilitated efficient coding and seamless integration with the compiler and Ncurses library.

In conclusion, these technologies provided the necessary foundation to develop a fully functional and interactive Snake Game. The combination of a C compiler, the Ncurses library, and a user-friendly IDE enabled smooth execution of the project while ensuring ease of development and debugging.

2.3 Data Structures Used

The Snake Game makes use of fundamental data structures in C, including arrays and structures, to efficiently manage the game's state and logic. These data structures are described below:

- **Array:** An array is used to represent the positions occupied by the snake on the game grid. It provides a simple and efficient way to store and access multiple coordinates that define the snake's body. Arrays in C are indexed collections of elements, which makes them suitable for tasks requiring sequential data storage.
- **Structure (`struct`):** The game defines a structure named `coord` to store the coordinates of the snake and food. The `coord` structure contains two integer members:

  ```
  struct coord {
      int x;
      int y;
  };
  ```

  - The `x` member represents the horizontal position on the grid. - The `y` member represents the vertical position on the grid.

  This structure provides a clear and organized way to group related data, simplifying the management of coordinates throughout the game.

By combining arrays and the `coord` structure, the game efficiently tracks the position and movement of the snake, as well as the placement of food items on the grid.

2.4 Pre-Steps for Execution

Before running the Snake Game, it is necessary to compile the C source file into an executable file and then execute it. This section outlines the steps to perform these tasks:

1. **Compiling the Source Code:** Ensure that the GCC compiler is installed on your system. Use the following command in the terminal to compile the source file and link the Ncurses library:

   ```
   gcc snake_game_project.c -lncurses -o game.out
   ```

   Here, the `-lncurses` flag links the Ncurses library, and the `-o game.out` option specifies the name of the output executable file.

2. **Running the Executable:** Once the compilation is successful, run the executable file using the following command:

   ```
   ./game.out
   ```

   This will launch the Snake Game in the terminal. Make sure your terminal supports Ncurses and that the library is properly installed on your system.

Following these steps ensures the successful compilation and execution of the Snake Game, providing a seamless gaming experience.

2.5 Input Details

The Snake Game is controlled entirely through the use of arrow keys on the keyboard. These keys allow the player to navigate the snake within the game grid. The input details are as follows:

- **Arrow Keys:** The four arrow keys (`Up`, `Down`, `Left`, `Right`) are used to control the direction of the snake's movement. The snake will move continuously in the last direction pressed until a new direction is input by the player.
- **Input Handling:** The game utilizes the Ncurses library to capture real-time user input without requiring the player to press the `Enter` key. This ensures smooth and responsive gameplay.
- **Invalid Inputs:** Only the arrow keys are considered valid inputs. If no input is provided, the snake continues moving in its current direction. Invalid or unrecognized inputs are ignored to prevent disruption of gameplay.

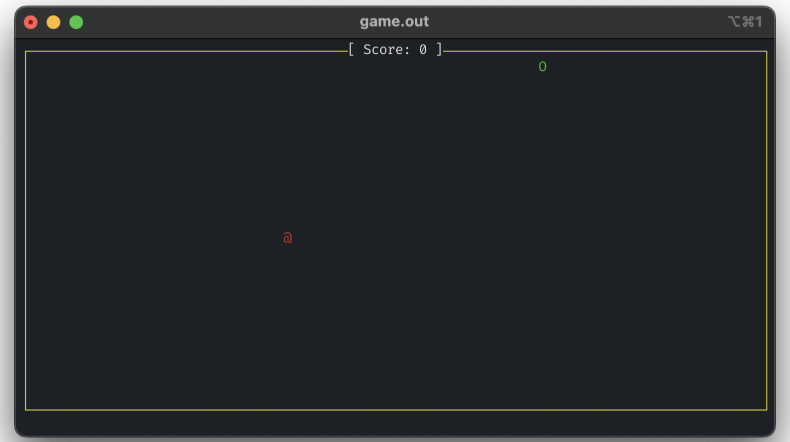
This simple input mechanism ensures intuitive and seamless control, allowing the player to focus entirely on the gameplay without unnecessary complexity.

2.6 Output Design

The Snake Game features a simple yet functional output design that utilizes terminal characters to create the game elements. The key aspects of the output design are as follows:

- **Snake Head:** The character `O` represents the snake's head, making it easy to identify the direction the snake is moving.
- **Snake Body:** The body of the snake is displayed using a diamond-shaped character (◇), visually distinguishing it from the head and other game elements.
- **Food Item:** The character `@` is used to represent the food (apple or berry) that the snake must eat to grow longer and increase the score.
- **Score Display:** The score is displayed at the top of the game screen, keeping the player informed about their progress.
- **Walls:** The game screen is surrounded by walls, which are depicted using terminal characters. Colliding with the walls results in the snake's death, adding a challenge to the game-play.

Below are two screenshots showcasing the game-play:

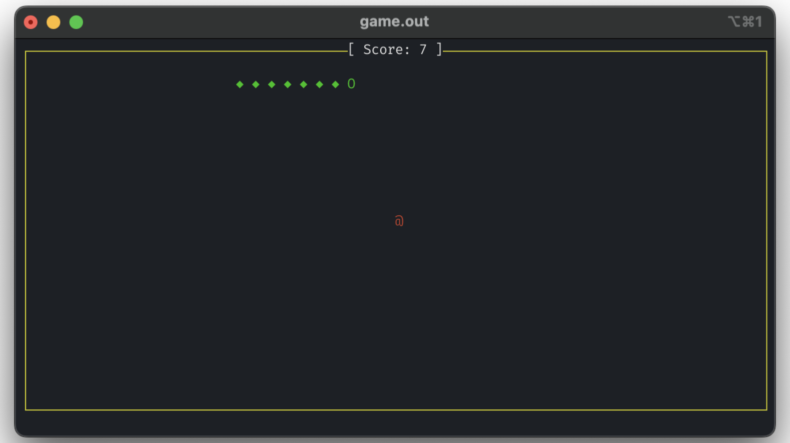


```
[ Score: 0 ]
                              O



@
```

Figure 2.1: Initial Game Screen: Snake and food placement at the start.



```
[ Score: 7 ]
◆◆◆◆◆◆◆0

@
```

Figure 2.2: In-Game Screen: Snake movement and Score display.

The simple graphical elements and layout ensure the game remains engaging and intuitive while running smoothly within the terminal environment.

2.7 Algorithm and Logic Design

The Snake Game follows a simple yet effective algorithm that controls the flow of the game. The logic is structured around a main game loop that continuously processes input, updates the game state, and draws the updated game screen. The following steps outline the core algorithm of the game.

1. **Initialization:** The game starts by calling the `init` function. This function sets up the terminal window size, configures the console to handle continuous input, and prepares the screen for drawing the game elements (snake, food, and walls).
2. **Main Game Loop:** Once the initialization is complete, the game enters a while loop that continues as long as the global variable `is_running` is true. The loop contains three primary functions that are executed sequentially:
   - **process_input:** The `process_input` function captures and processes user input (arrow keys) to control the snake's direction. It ensures that the snake can only change direction once per key press, preventing multiple directions at the same time.
   - **update:** The `update` function updates the game state. It moves the snake in the current direction, checks for collisions (with the wall or itself), and handles food consumption (growing the snake and increasing the score).
   - **draw:** The `draw` function renders the updated game screen. It draws the snake, food, score, and surrounding walls in the terminal window.

3. **Game Over / Continue Prompt:** If the snake collides with a wall or its own body, the game checks for the player's desire to continue or quit. A prompt is displayed asking the player whether they want to continue the game by pressing the space bar, or quit the game by pressing the skip key.

4. **Quit Game:** Once the game ends or the user chooses to quit, the `quit_game` function is called to clean up the game state, free up resources, and exit the program.

The game logic ensures smooth gameplay, with clear separation of concerns for input processing, updating the game state, and rendering the game output. Below is the formal algorithmic representation of the game logic:

Algorithm 1 Game Algorithm

```
1: Initialize the Game:
2:     Call init function to set up the window and console settings
3: Main Game Loop:
4: while is_running = true do
5:         Call process_input to capture user input
6:         Call update to update the game state
7:         Call draw to render the game screen
8:     if snake collides with wall or itself then
9:             Display game over prompt: Press space to continue or skip to quit
10:        if user presses space then
11:                Continue game
12:        else
13:                Set is_running = false
14:        end if
15:    end if
16: end while
17: End Game:
18:     Call quit_game to clean up and exit
```

This algorithm ensures that the game proceeds smoothly, with real-time user input being processed and the game state being updated continuously. The game offers a simple yet interactive experience by combining basic game mechanics with a clear control flow.

Chapter 3

Conclusion

3.1 Discussion

The Snake Game project demonstrates the application of fundamental programming concepts in C, such as dynamic memory handling, input processing, and game logic design. The project effectively combines these concepts to create a functional and interactive game.

Developing the game required careful attention to modularity and code readability, ensuring that each component, such as input handling, game state updating, and rendering, worked seamlessly. The use of the Ncurses library allowed for real-time terminal-based graphics, enhancing the overall user experience.

This project not only reinforces the importance of structuring code effectively but also highlights the creativity involved in developing even simple games. It was a valuable learning experience in understanding how low-level details contribute to a working system.

3.2 Limitations

Despite its functionality, the game has certain limitations:

- The game only supports basic controls using arrow keys, which limits the gameplay experience.
- The terminal-based graphical interface, while functional, lacks the visual appeal of modern games.
- There is a fixed limit on the snake's growth due to the use of a fixed-size array, which restricts the maximum achievable score.
- There is no save or pause feature, which reduces the convenience for players.
- The game logic does not include advanced features, such as increasing difficulty levels or multiple gameplay modes.

These limitations leave room for future enhancements, as described in the next section.

3.3 Future Work

The following improvements could be made to enhance the Snake Game project:

- Replacing the fixed-size array with a dynamically resizable data structure, such as a linked list, to allow unlimited snake growth and remove the maximum score restriction.
- Adding advanced features such as increasing difficulty levels, power-ups, or obstacles.
- Implementing a pause and resume functionality to allow more flexible gameplay.
- Enhancing the graphical interface by migrating to a more advanced library such as SDL or OpenGL.
- Extending compatibility to multiple platforms, including Windows, by replacing Ncurses with a cross-platform solution.
- Introducing a scoring leaderboard to make the game more competitive.
- Optimizing the code for better performance and maintainability.

These proposed improvements would not only make the game more engaging but also provide additional learning opportunities in implementing advanced programming techniques.