

**LA GRANDEE INTERNATIONAL COLLEGE**

**Simalchaur-8, Pokhara**

A Project Report on

**Snake Game**

**Submitted to**

LA GRANDEE International College

Bachelor of Computer Application (BCA) Program

*In partial fulfillment of the requirements for the degree of BCA under*

Pokhara University

Submitted by

|  |  |  |
| --- | --- | --- |
| Name of the Student | PU Registration Number | Semester |
| Pratik Gautam | 2023-1-53-0337 | BCA-II |
| Ashish Basnet | 2023-1-53-0322 | BCA-II |
| Shital Dhakal | 2023-1-53-0345 | BCA-II |
| Kaushal Shrestha | 2023-1-53-0329 | BCA-II |

July 4, 2024

# **Acknowledgement**

We extend our heartfelt gratitude to everyone who supported and guided us throughout the development of our Snake game project. Firstly, we thank our **Supervisor, Ramesh** **Chalise,** for their invaluable guidance, constructive feedback, and continuous encouragement. Their expertise significantly enhanced the quality of our work.

We are deeply thankful to each team member for their dedication, hard work, and collaborative effort. This project’s success is a testament to our collective commitment and contributions.

Lastly, we appreciate our peers and classmates for their support and collaborative spirit, as well as friends for their unwavering encouragement and understanding during challenging times. Together, with the support of these individuals and the resources available, we successfully completed our Snake game project.

Sincerely,

**Pratik Gautam**

**Ashish Basnet**

**Shital Dhakal**

**Kaushal Shrestha**

# **Student Declaration**

We, the undersigned members, declare that the work presented in this Snake game project report is original and has been carried out by us collectively during the 2nd Semester. All external sources used have been duly acknowledged and referenced in the report. This project has not been submitted for any other course or examination.

…………………………….

**Ashish Basnet**

PU Reg. No. 2023-1-53-0322

…………………………….

**Pratik Gautam**

PU Reg. No. 2023-1-53-0337

…………………………….

**Kaushal Shrestha**

PU Reg. No. 2023-1-53-0329

…………………………….

**Shital Dhakal**

PU Reg. No. 2023-1-53-0345

Date: July 3, 2024

# **Supervisor’s Declaration**

I hereby recommend that this project entitled “**Snake Game**” is done under my supervision by **Kaushal shrestha, Shital Dhakal, Pratik Gautam and Ashish Basnet** during their 2nd Semester in partial fulfillment of the requirements for the degree of **BCA** under **Pokhara University** is completed to my satisfaction and be processed for final evaluation.

………………………

**Mr. Ramesh Chalise**

Date: July 4, 2024

****

Date: 2024-07-04

### Letter of Approval

We certify that we have examined this report entitled “**Snake Game**” and are satisfied with the project defense. It is satisfactory in the scope and qualify as project in partial fulfillment of the requirements for the degree of BCA under Pokhara University.

….……………………..

**Mr. Ramesh Chalise**

Supervisor

………………………..

**Er. Damodar Baral**

External Examiner

……………………...

**Er. Kiran K.C**

Principal

# **Abstract**

This project presents the development of a Snake game implemented in C programming language. The game includes three levels easy, medium, and hard each with progressively challenging gameplay mechanics. Key features include dynamic snake movement, food generation, collision detection, and score tracking. High scores are stored and displayed level-wise using file handling. The project also emphasizes effective teamwork, iterative development, and comprehensive documentation. Through thorough planning and collaboration, the game demonstrates efficient implementation of game mechanics and user interaction. Future enhancements could focus on additional features and optimizing gameplay experience based on user feedback and testing.

**Table of Contents**

[Acknowledgement ii](#_Toc170812408)

[Student Declaration iii](#_Toc170812409)

[Supervisor’s Declaration iv](#_Toc170812410)

[Letter of Approval v](file:///C:\Users\ACER\Documents\report%20final\snake%20game%20final%20report.docx#_Toc170812411)

[Abstract vi](#_Toc170812412)

[List of Tables viii](#_Toc170812413)

[List of Figures ix](#_Toc170812414)

[Abbreviations x](#_Toc170812415)

[1. Introduction 1](#_Toc170812416)

[2. Problem Statement 2](#_Toc170812417)

[3. Objectives 3](#_Toc170812418)

[4. Background Study 4](#_Toc170812419)

[5. Requirement Document 5](#_Toc170812420)

[6. System Design 6](#_Toc170812421)

[6.1 Algorithm 6](#_Toc170812422)

[6.2 Flowchart 7](#_Toc170812423)

[6.3 Data flow Diagram 10](#_Toc170812424)

[7. Development 12](#_Toc170812425)

[8. Testing 15](#_Toc170812426)

[9. Project Result 19](#_Toc170812427)

[10. Future Enhancements 21](#_Toc170812428)

[11. Conclusion 23](#_Toc170812429)

[12. Recommendation 24](#_Toc170812430)

[13. Annex 25](#_Toc170812431)

[14. Gantt Chart 32](#_Toc170812432)

[References](#_Toc170812433)

# **List of Tables**

[Table 7. 1 Work Division 14](#_Toc170801733)

[Table 8. 1 Sanke Movement Test Case 15](#_Toc170801746)

[Table 8. 2 Collision Detection Test Case 16](#_Toc170801747)

[Table 8. 3 Score Mechanism Test Case 17](#_Toc170801748)

[Table 8. 4 Level-Specific Testing Test Case 17](#_Toc170801749)

# **List of Figures**

[Figure 6.2. 1 Game Initialization Flowchart 7](#_Toc170801425)

[Figure 6.2. 2 Snake Movement and Collision Detection Flowchart 8](#_Toc170801426)

[Figure 6.2. 3 End Game Handling Flowchart 9](#_Toc170801427)

[Figure 6.3. 1 Level 0 Data Flow Diagram 10](#_Toc170801441)

[Figure 6.3. 2 Level 1 Data Flow Diagram 10](#_Toc170801442)

[Figure 6.3. 3 Level 2 Data Flow Diagram 11](#_Toc170801443)

[Figure 7. 1 Waterfall Model 12](#_Toc170801458)

[Figure 14. 1 Gantt Chart 32](#_Toc170812539)

# **Abbreviations**

* PU: Pokhara University
* BCA: Bachelor of Computer Application
* PC: Personal Computer
* ASCII: American Standard Code for Information Interchange
* DFD: Data Flow Diagram
* UI: User Interface
* ID: **Identification**
* TC: Test Case

# **Introduction**

The Snake Game is a timeless arcade classic that has fascinated players since the early days of video gaming. It’s simple yet engaging gameplay has made it a beloved title across generations. Our project seeks to bring this classic game back to life using the C programming language, demonstrating both its enduring appeal and the capabilities of modern programming techniques.

Our main goal is to develop a feature-rich version of the Snake game with three difficulty levels: easy, medium, and hard. Each level offers a unique challenge with faster snake speeds and more complex maze layouts. Players will navigate the snake through these mazes, avoiding obstacles and collecting food to score points. The game will feature dynamic movement controls, accurate collision detection, strategic food placement, and a clear scoring system to track player achievements.

This project also serves as an opportunity to explore fundamental programming concepts. We use data structures like linked lists to manage the snake’s body segments, which helps in efficient memory management and performance. We also design algorithms for collision detection and food spawning to ensure smooth gameplay. Additionally, we incorporate file handling techniques to save and display high scores, adding a competitive element to the game.

Through this project, we aim to deepen our understanding of software development practices and enhance our problem-solving and teamwork skills. By leveraging the C programming language, we showcase its power and versatility in creating interactive and engaging gaming experiences.

In summary, this project celebrates the classic appeal of the Snake game while demonstrating our commitment to innovative game development and our growth as software engineers.

# **Problem Statement**

The project involves developing a classic Snake Game using C programming. The game will feature core gameplay mechanics such as snake movement, food generation, and collision detection, along with additional enhancements like adjustable game speed and customizable settings. Through this project, we aim to provide a fun and interactive gaming experience while honing our skills in C programming and game development.

# **Objectives**

The objective of creating this program are as follows:

* Implement easy, medium, and hard difficulty levels.
* Implement effective user input handling.

# **Background Study**

The Snake game, born in the late 1970s, quickly became a favorite in arcades. Players guide a snake to eat food and avoid crashing into walls or its own tail. It’s simple but addictive gameplay has kept it popular for decades.

Recently, there's been a renewed interest in old games like Snake. Developers update them with better graphics and controls for today's devices. Our project aims to remake Snake using the C programming language, known for its efficiency and ability to handle complex tasks.

C lets us control hardware directly, making games run smoothly and respond quickly. By using C, we want to bring back the fun of Snake while adding improvements for a better experience. This project celebrates a classic game and shows how it can still entertain in modern times.

# **Requirement Document**

**Functional Requirements:**

* Gameplay Features: Implement snake movement, food spawning, and collision detection.
* Difficulty Levels: Include easy, medium, and hard modes with distinct challenges.
* Scoring System: Track and display scores based on gameplay performance.
* User Interface: Design intuitive menus and controls for easy navigation.
* File Handling: Store and retrieve high scores and game settings.

**Non-Functional Requirements:**

* Performance: Ensure smooth gameplay with minimal lag or delays.
* Usability: Provide clear instructions and responsive controls for an enjoyable user experience.
* Security: Safeguard user data and prevent unauthorized access.
* Maintainability: Document code and design decisions for future updates and modifications.

**Constraints:**

* Technology: Use C programming language with standard libraries.
* Platform: Ensure compatibility with desktop environments (PCs and laptops).

# **System Design**

Our Snake game system design integrates a robust game engine for handling movement and collision detection, a user-friendly interface with intuitive controls, and a scoring system that tracks player achievements. Implemented in C, the design prioritizes efficiency and cross-platform compatibility to ensure smooth gameplay across various operating systems. By focusing on these elements, we aim to deliver a cohesive gaming experience that combines classic gameplay mechanics with modern design principles.

## **6.1 Algorithm**

The algorithm begins by initializing the game board, snake position, and food generation. It continuously moves the snake based on user input, checks for collisions with walls, itself, or food, updates the score accordingly, and ends the game if necessary, providing a seamless gameplay experience.

* + - Step 1: Start
    - Step 2: Initialize Game:
      * Initialize game board with specified size
      * Set initial position and direction of the snake
      * Generate initial position of the food
      * Set score to zero
    - Step 3: Get User input
    - Step 4: Generate Food:
      * Generate random position for food on game board
      * Ensure food does not overlap with snake's body
    - Step 5: Move Snake:
      * Move snake's head in current direction
      * Update positions of snake's body segments
    - Step 6: Check Collision:
      * If snake's head collides with walls or itself:
        + End Game
      * If snake's head collides with food:
        + Consume food and generate the food
        + Increase snake length and update score
    - Step 7: End Game
      * Display game over message and final score
      * Prompt user to play again or exit

## **6.2 Flowchart**

The flowchart illustrates the sequential steps of initializing the game, including setting up the game board, positioning the snake, and generating food. It then depicts the continuous loop of user input for directing the snake's movement, checking for collisions with walls, itself, or food, updating the score, and ending the game if necessary, ensuring clear visualization of the game's logic flow.

#### **Game initialization**

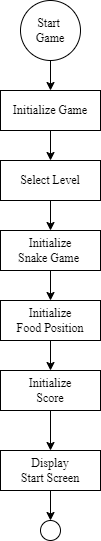


Figure 6.2. 1 Game Initialization Flowchart

#### **Snake Movement and Collision Detection**

#### 

Figure 6.2. 2 Snake Movement and Collision Detection Flowchart

**End Game Handling**

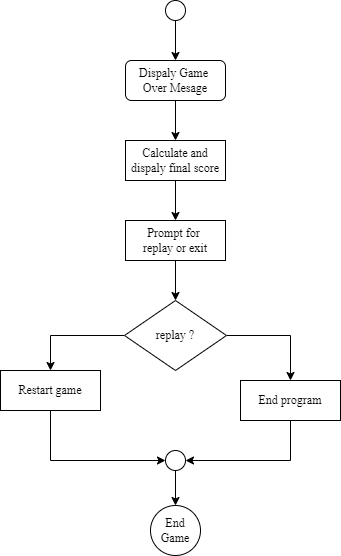


Figure 6.2. 3 End Game Handling Flowchart

## **6.3 Data flow Diagram**

The DFD visually maps how data moves between components like the game engine, user interface, and scoring module in the Snake game. It demonstrates how player actions, such as input for snake movement and interactions with food, affect game state changes and scoring updates. This diagram provides a clear representation of data flow and interaction within the game system, ensuring efficient gameplay mechanics.

**Level 0 DFD**

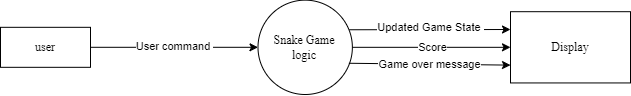
****

Figure 6.3. 1 Level 0 Data Flow Diagram

**Level 1 DFD**

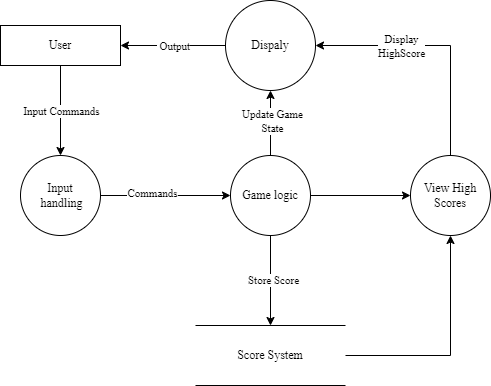
****

Figure 6.3. 2 Level 1 Data Flow Diagram

**Level 2 DFD**

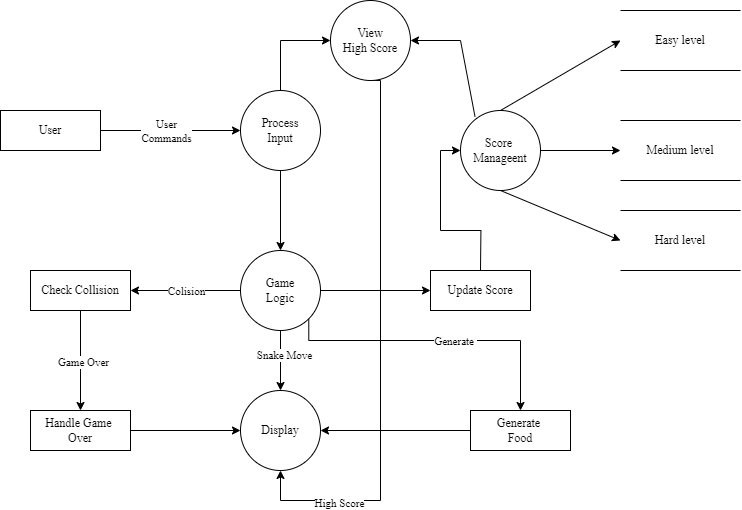


Figure 6.3. 3 Level 2 Data Flow Diagram

# **Development**

The development of the Snake Game in C followed the waterfall model, a sequential software development process. This approach divided the project into distinct phases: requirements gathering, design, implementation, testing, and deployment. Each phase had specific goals and deliverables, ensuring a systematic and structured approach to developing the game.

**Waterfall Model**

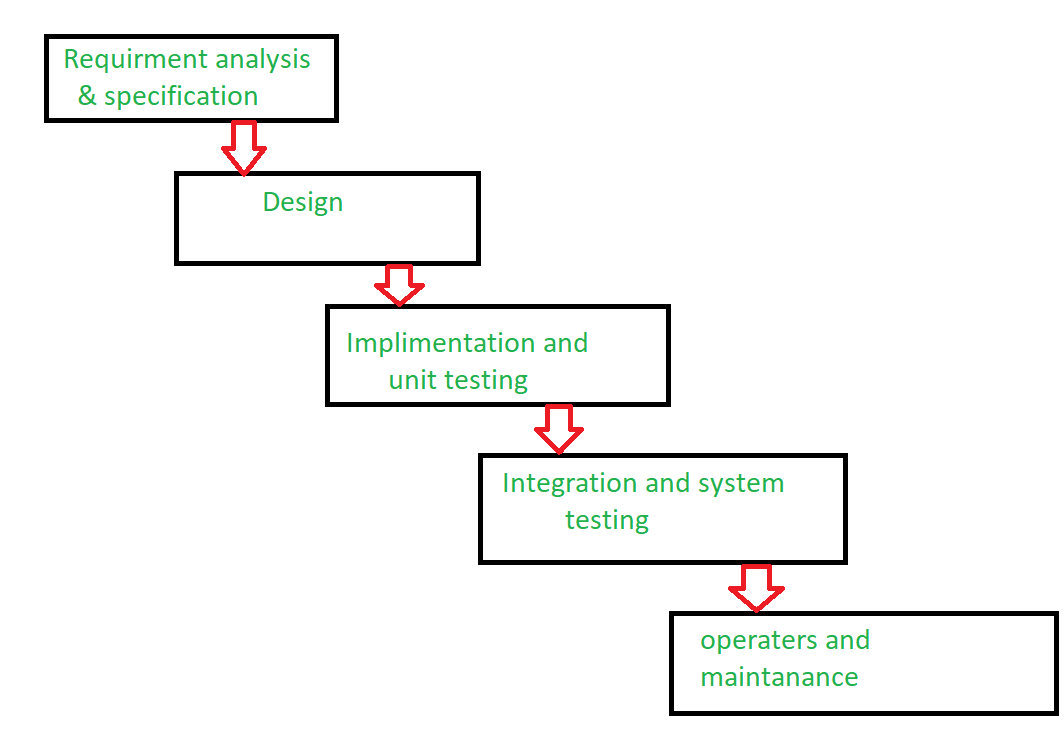
****

Figure 7. 1 Waterfall Model

**I.** **Requirements Gathering**: During this initial phase, the team identified and documented the functional and non-functional requirements for the Snake Game. This included defining gameplay mechanics such as snake movement, food spawning, and game levels (easy, medium, hard). Requirements were gathered through discussions with stakeholders and potential users to ensure clarity and completeness.

**II.** **Design**: Following the requirements gathering phase, the design phase focused on creating detailed specifications for the game architecture and user interface. This included designing the overall structure of the game, defining modules like the main menu, game engine, and high score management.

**III.** **Implementation**: With the design finalized, the implementation phase involved translating design specifications into actual code using the C programming language. Key features such as snake movement algorithms, scoring mechanisms, and file handling for high scores were implemented according to the design documentation.

**IV.** **Testing**: Once implementation was completed, the game underwent rigorous testing to identify and rectify any defects or issues. Unit testing verified the correctness of individual functions, ensuring they performed as expected. Integration testing focused on validating the interaction between different modules and game components, such as transitions between menu screens and gameplay states. User acceptance testing involved real users playing the game to gather feedback on gameplay mechanics, usability, and overall experience.

**V.** **Deployment**: The final phase of the waterfall model, deployment, involved preparing the Snake Game for release. This included packaging the game files, preparing installation instructions if applicable, and ensuring compatibility across different platforms and environments. Deployment also involved final checks to ensure all requirements were met and the game was ready for distribution to users.

The Snake Game project was developed step-by-step using the waterfall model. It included gathering requirements, designing game features and how players interact with the game, writing code in C, testing thoroughly, and preparing for release.

**Work Division**

Work division refers to the systematic allocation of tasks, responsibilities, and roles among team members within a project. It aims to distribute workload efficiently by assigning distinct areas of focus based on individual skills and expertise. This ensures that all aspects of the project are effectively managed and completed, contributing to overall project success and maximizing team productivity.

Table 7. 1 Work Division

|  |  |
| --- | --- |
| Team Member | Responsibilities |
| Ashish Basnet | Refining gameplay mechanics, contributing to UI design |
| Shital Dhakal | participating in requirement discussions, managing file handling for game data. |
| Pratik Gautam | Providing input on requirements, programming game state management. |
| Kaushal Shrestha | Project management, core game functionalities implementation, file handling for game data, project report. |

Each team member's focused contributions ensure comprehensive project management and successful implementation, aligning with their specialized roles and responsibilities for the Snake Game development.

# **Testing**

**Objective:** The testing phase aimed to ensure the functionality, reliability, and user experience of our Snake game across different game levels (easy, medium, hard). This section outlines the methodologies used, test cases implemented, and outcomes observed during the testing process.

**Methodologies:** We employed a combination of manual and automated testing methodologies to validate various aspects of the game:

* **Functional Testing:** Verified core gameplay mechanics such as snake movement, collision detection, and food generation across all levels.
* **User Interface Testing:** Ensured intuitive controls, responsive feedback, and visual clarity through systematic UI testing.
* **Performance Testing:** Evaluated the game's stability and responsiveness under varying load conditions, focusing on memory usage and frame rates.

**Test Cases:** A test case is a set of conditions and steps used to determine whether a specific functionality of an application works as intended. It includes preconditions, test steps, expected results, and actual results to ensure thorough testing and validation of the application's features.

**Test Case: Snake Movement**

Table 8. 1 Sanke Movement Test Case

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | |  | | --- | |  |   **TC001** |
| Description | Verify the snake movement in all directions (up, down, left, right) and interaction with walls. |
| Preconditions | |  | | --- | |  |  |  | | --- | | The game is running, the snake is visible on the screen, and walls are present. | |
| Test Steps | 1. Start the game. 2. Press the up-arrow key. 3. Observe the snake's movement and interaction with walls. 4. Press the down arrow key. 5. Observe the snake's movement and interaction with walls. 6. Press the left arrow key. 7. Observe the snake's movement and interaction with walls. 8. Press the right arrow key. 9. Observe the snake's movement and interaction with walls. 10. Move the snake towards a wall. 11. Observe the snake's behavior when it hits a wall. |
| Expected Result | 1. Snake moves upward when the up-arrow key is pressed and avoids walls. 2. Snake moves downward when the down arrow key is pressed and avoids walls. 3. Snake moves left when the left arrow key is pressed and avoids walls. 4. Snake moves right when the right arrow key is pressed and avoids walls. 5. Snake should trigger a game over when it hits a wall. |
| Actual Result | 1. Snake moved upward and avoided walls when the up-arrow key was pressed. 2. Snake moved downward and avoided walls when the down arrow key was pressed. 3. Snake moved left and avoided walls when the left arrow key was pressed. 4. Snake moved right and avoided walls when the right arrow key was pressed. 5. Snake triggered a game over when it hit a wall. |
| Status | Pass |

**Test Case: Collision Detection**

Table 8. 2 Collision Detection Test Case

|  |  |
| --- | --- |
| **Test Case ID** | **TC002** |
| Description | Verify the collision detection mechanism between the snake, walls, and itself. |
| Preconditions | The game is running, and collision detection for the snake's head with walls and its own body is implemented. |
| Test Steps | 1. Start the game with walls and obstacles configured. 2. Move the snake towards a wall. 3. Observe the game's reaction to the collision. 4. Move the snake towards its own body. 5. Observe the game's reaction to the collision. |
| Expected Result | 1. Game triggers a game over when the snake's head collides with a wall. 2. Game triggers a game over when the snake's head collides with its own body. |
| |  | | --- | | Actual Result |  |  | | --- | |  | | 1. Game over triggered correctly when the snake's head collided with a wall. 2. Game over triggered correctly when the snake's head collided with its own body. |
| Status | Pass |

**Test Case: Score Mechanism**

Table 8. 3 Score Mechanism Test Case

|  |  |
| --- | --- |
| **Test Case ID** | **TC003** |
| Description | Verify the scoring system accurately increments the score based on gameplay actions. |
| Preconditions | The game is running, and the score display is visible on the screen. |
| Test Steps | 1. Start the game. 2. Move the snake to eat a piece of food. 3. Observe the score increment. 4. Continue eating additional food pieces. 5. Observe the score update after each food consumption. |
| Expected Result | 1. Score increases by a predefined amount (e.g., 10 points) each time the snake eats a piece of food. 2. Score accurately reflects the total number of food pieces eaten by the snake. |
| Actual Result | 1. Score incremented by 10 points when the snake ate a piece of food. 2. Score updated correctly after each subsequent food consumption. |
| Status | |  | | --- | |  |  |  | | --- | | Pass | |

**Test Case: Level-Specific Testing**

Table 8. 4 Level-Specific Testing Test Case

|  |  |
| --- | --- |
| **Test Case ID** | **TC004** |
| Description | Verify the specific features and difficulty adjustments of each game level (easy, medium, hard) |
| Preconditions | The game is running, and the selected level (easy, medium, or hard) is active. |
| Test Steps | 1. Start the game in the selected level (easy/medium/hard). 2. Verify the speed of the snake's movement matches the level's difficulty. 3. Test the density and placement of food items based on the level. 4. Verify any additional features or challenges specific to the level (e.g., obstacles, special rules). |
| Expected Result | 1. Easy level: Slower snake movement speed, fewer obstacles, and simpler food placement. 2. Medium level: Moderate snake movement speed, moderate obstacles, and food placement. 3. Hard level: Faster snake movement speed, more obstacles, and challenging food placement. 4. Each level should provide appropriate difficulty and gameplay experience as per its description. |
| Actual Result | 1. Easy level: Snake moved at a slower speed, fewer obstacles encountered, food placement was straightforward. 2. Medium level: Snake moved at a moderate speed, encountered moderate obstacles, food placement was adequate. 3. Hard level: Snake moved at a faster speed, encountered numerous obstacles, challenging food placement. |
| Status | Pass |

**Conclusion:** The testing phase confirmed the robustness and usability of our Snake game, meeting project objectives for functionality, performance, and user satisfaction. Identified issues were promptly addressed, ensuring a polished final product ready for release.

# **Project Result**

The development of the Snake Game in C, a BCA 2nd semester project, has been successfully completed. The project met all predefined requirements and achieved its objectives, providing a functional and engaging game. The key outcomes of the project are as follows:

1. **Gameplay Functionality:**

* The Snake Game implemented classic gameplay mechanics, including smooth snake movement, food consumption, and score tracking.
* Players can control the snake using arrow keys, and the game ends upon collision with the snake itself or the game’s walls.

1. **Difficulty Levels:**

* Three distinct difficulty levels were implemented: easy, medium, and hard. Each level adjusts the snake's speed, food placement, and obstacle density accordingly.
* Testing confirmed that each level provides a suitable challenge, enhancing the game’s replay ability for players of varying skills.

1. **High Score System:**

* A high score management system was implemented to record and display the top scores for each difficulty level.
* Scores are saved in level-specific files and displayed in descending order, showing the top three scores along with the player's name.

1. **User Interface:**

* The game includes a main menu, game over screen, and high score display, providing an intuitive and user-friendly navigation experience.
* Players can easily start a new game, view high scores, and return to the main menu.

1. **Performance and Stability:**

* Rigorous testing ensured that the game runs reliably under various conditions and difficulty levels without crashes or significant bugs.
* Collision detection, scoring, and level transitions worked as intended throughout extensive playtesting.

1. **User Feedback:**

* User acceptance testing indicated positive feedback, with players appreciating the balance of difficulty and the nostalgic feel of the classic Snake game.
* The game successfully recreated the classic experience while adding new features to enhance gameplay.

The Snake Game project demonstrated the effective use of the waterfall model, guiding the development process from initial requirements gathering to final deployment. Collaboration among team members, thorough planning, and extensive testing were key factors in delivering a polished and fully functional game.

# **Future Enhancements**

Several enhancements can be considered for future development of the Snake Game in C to further enrich the gameplay experience:

1. **Improved Visual Feedback:**

* Enhancing console-based graphics with improved ASCII art or simple graphical elements.
* Adding animation effects for snake movement, food consumption, and game over events.

1. **Advanced Gameplay Features:**

* Introducing additional game modes, such as timed challenges where players must achieve a high score within a set time limit.
* Implementing difficulty modifiers that adjust game speed dynamically based on player performance or selected settings.

1. **Enhanced User Interface:**

* Refining the user interface (UI) for more intuitive menu navigation and clearer display of game settings and high scores.
* Adding options for players to customize game preferences, such as adjusting sound settings or choosing different color themes.

1. **Extended High Score System:**

* Expanding the high score system to include more detailed statistics and achievements that reward players for specific milestones or challenges.
* Providing a local leaderboard that displays the top scores for all difficulty levels on a single screen for easy comparison.

1. **Additional Challenges and Obstacles:**

* Introducing new types of obstacles or challenges that appear randomly during gameplay, requiring strategic planning and quick reflexes.
* Incorporating bonus items or power-ups that provide temporary advantages, such as increased movement speed or invulnerability.

These future enhancements aim to elevate the Snake Game experience by adding depth, replay ability, and visual appeal while retaining its core single-player offline nature. Each enhancement can be prioritized based on feasibility and potential impact on player enjoyment.

# **Conclusion**

Our Snake game project developed in C programming has been a fulfilling experience of teamwork and skill development. We successfully designed and implemented a game with intuitive controls, challenging levels, and a visually appealing interface. Throughout the project, we encountered and overcame various challenges, which strengthened our problem-solving abilities and collaboration skills. By incorporating feedback and refining our game mechanics, we ensured a fun and smooth gameplay experience across different platforms. Looking ahead, we aim to enhance the game further by adding new features and improving compatibility. We are grateful to our supervisor and teammates for their invaluable support and contributions to this project's success.

# **Recommendation**

Based on our group’s experience developing the Snake game project, we recommend future groups prioritize thorough planning with clear objectives and timelines. Effective collaboration through regular communication and meetings ensures tasks are completed efficiently. Adopting an iterative development approach allows for continuous testing and refinement of game features, enhancing functionality and user satisfaction. Detailed documentation, including code comments and user guides, facilitates understanding and future updates. Utilizing collaborative tools for communication and task management among team members improves efficiency and coordination. These strategies will support future groups in successfully executing and completing similar projects.

# **Annex**

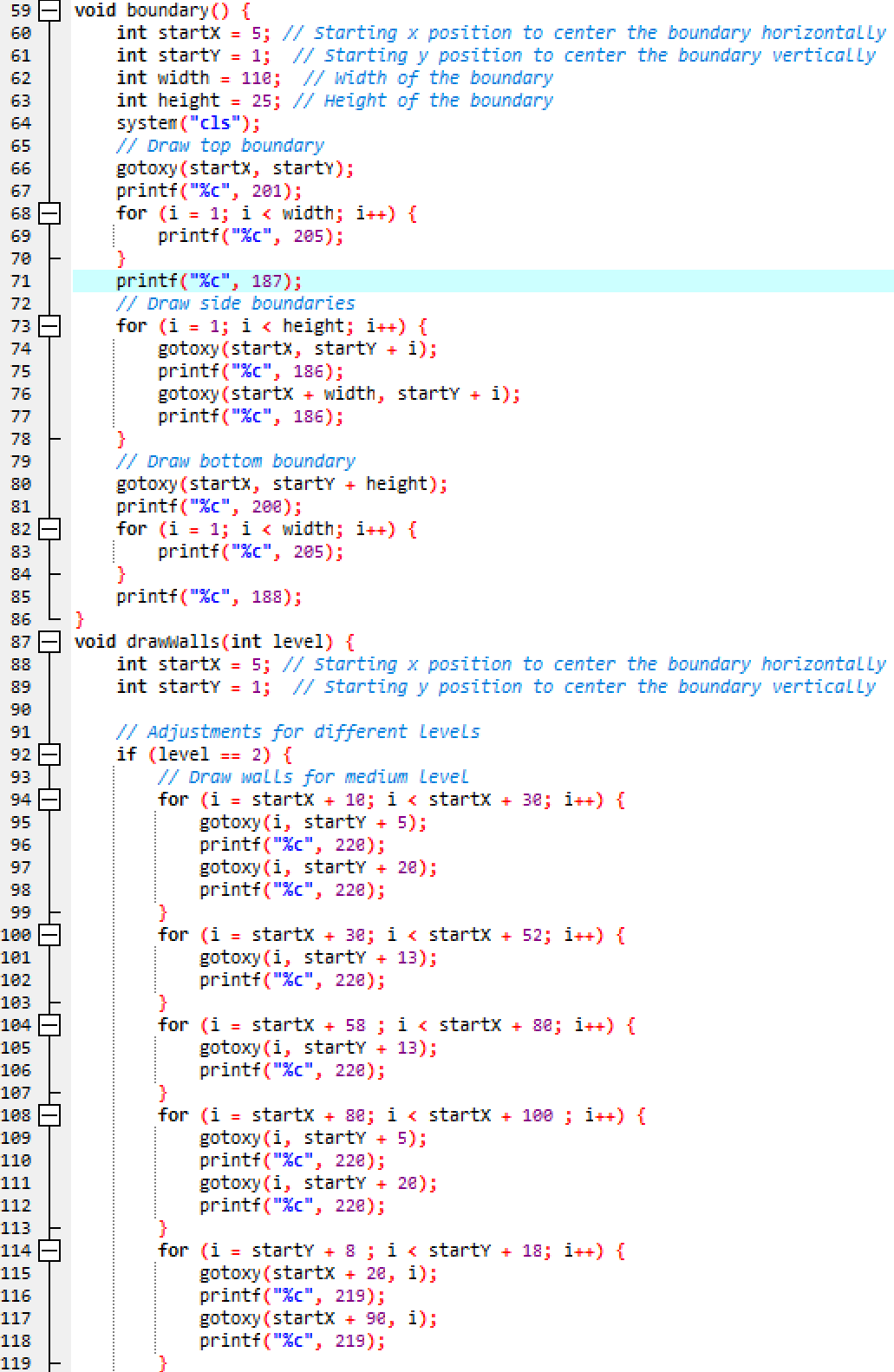
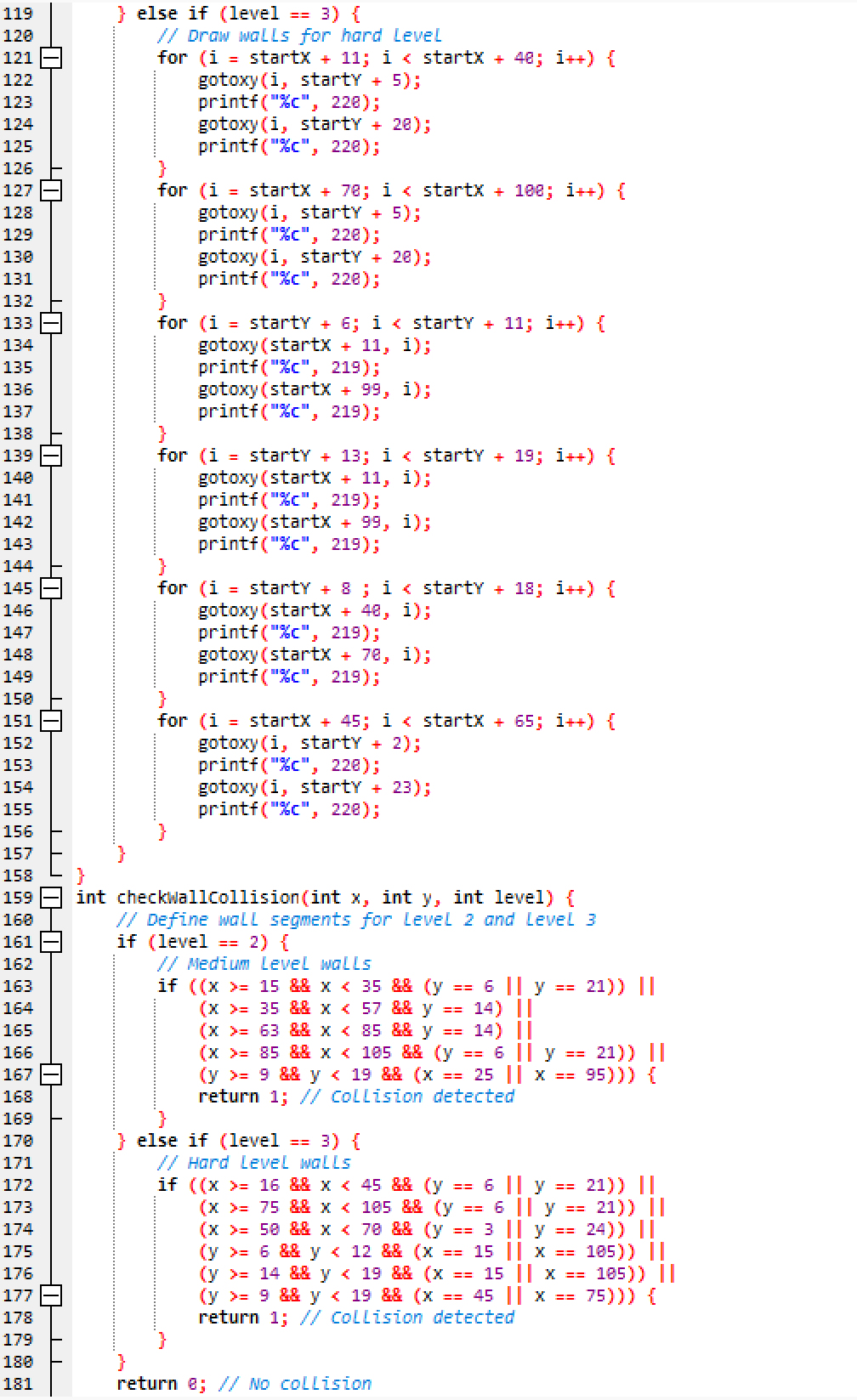
**Main function**

The main program initializes the Snake game, handles user input, and provides the primary menu interface. It prompts the player to enter their name, displays the main menu with options to play the game, view instructions, check high scores, or exit. Based on the player's choice, the corresponding function is called to execute the selected action.



**Level-specific Wall Drawing and Collision Detection Function Implementation**

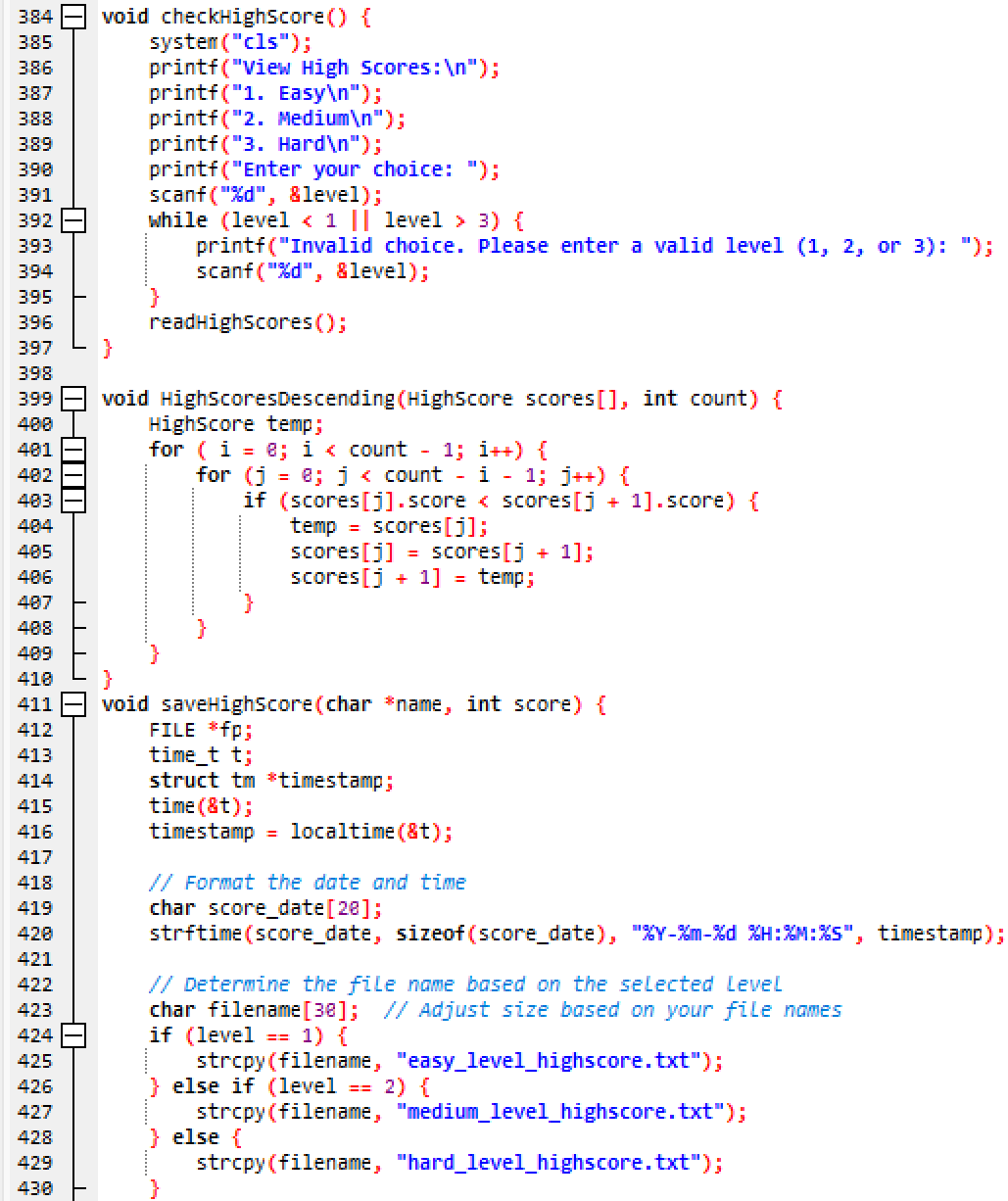
This section covers the implementation of functions responsible for drawing level-specific walls and boundaries and detecting collisions with these walls. These functions adjust the wall patterns based on the level to provide varying difficulty and ensure that the snake's collisions with the walls are accurately detected.

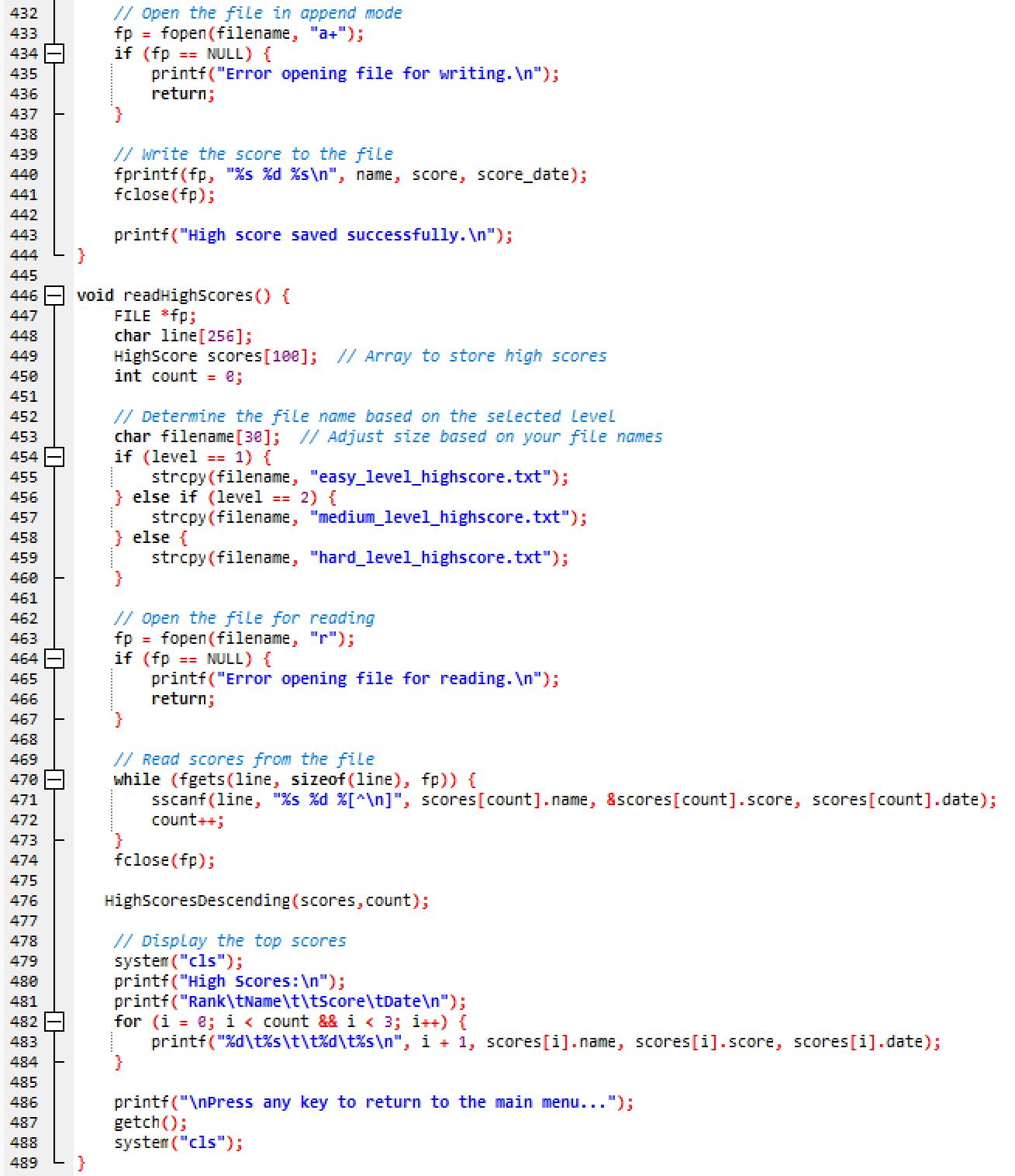
 

**High Score File Management Functions**

This section provides the implementation of functions responsible for managing high scores in the Snake Game. It includes:

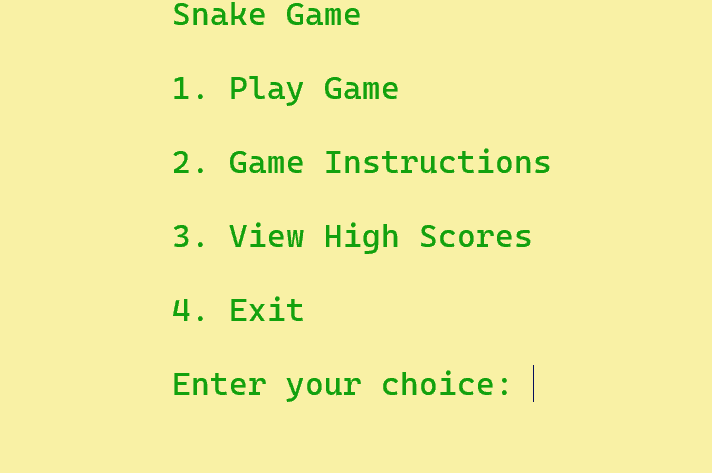
* **checkHighScore()**: This function prompts the player to select a difficulty level and then calls the function to read and display the high scores for the chosen level.
* **HighScoresDescending()**: This function sorts the high scores in descending order, ensuring that the highest scores are displayed at the top.
* **saveHighScore()**: This function saves the player's name, score, and timestamp to the appropriate high score file based on the selected difficulty level.
* **readHighScores()**: This function reads the high scores from the corresponding file, sorts them, and displays the top three scores along with the player's name and date of achievement.





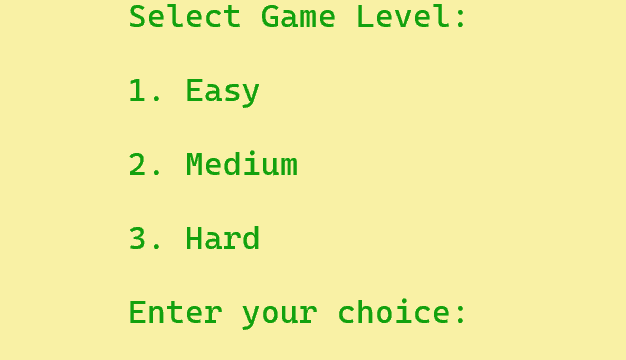
**Main Menu**

The main menu of the Snake Game project presents a user-friendly interface designed for intuitive navigation. It features options for selecting game levels, accessing high scores, and exiting the game. The menu design prioritizes simplicity and clarity to enhance user experience from the outset.

****

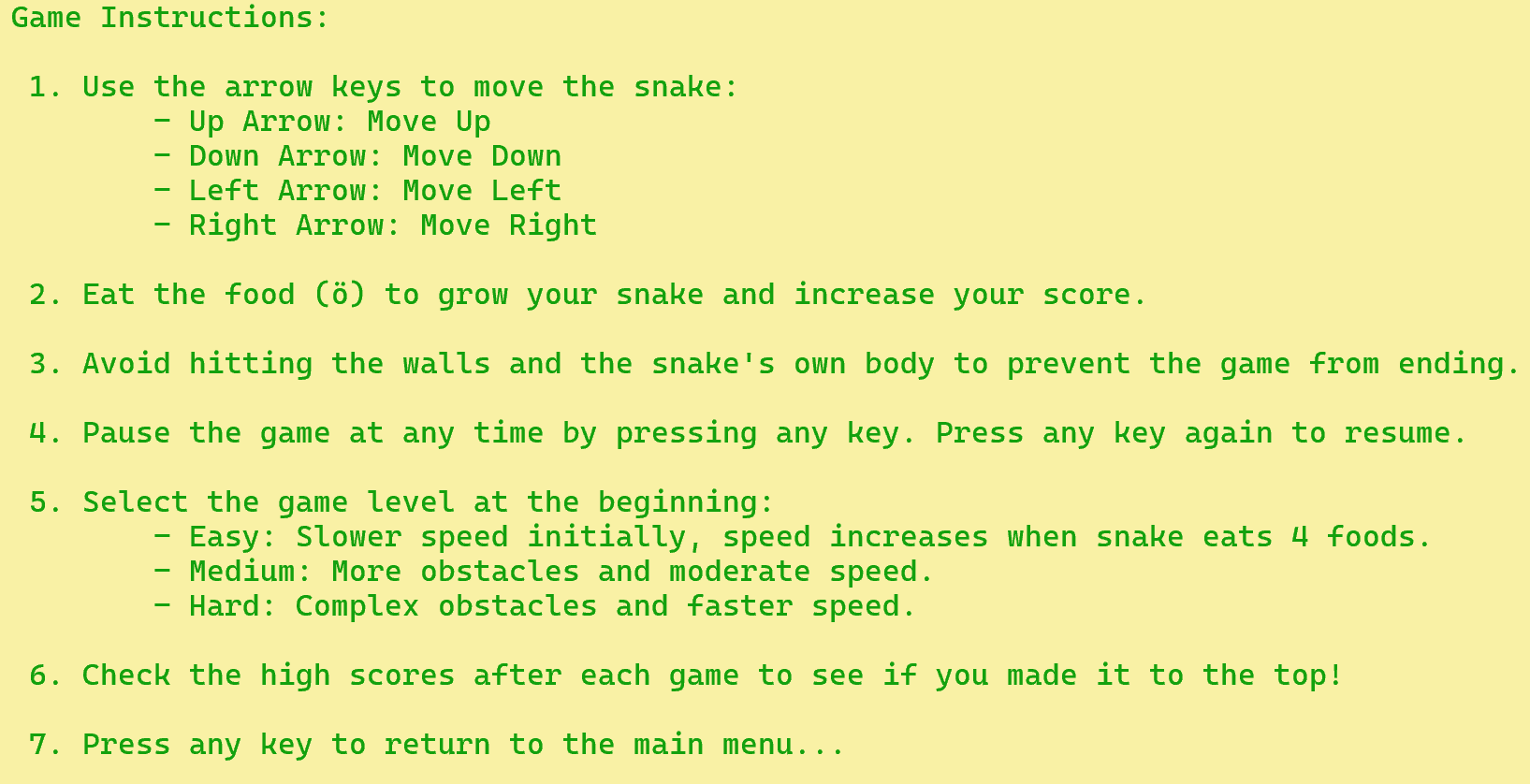
**Level Selection**

The level selection screen in the Snake Game project allows players to choose from three difficulty levels: easy, medium, and hard. Each level offers a unique challenge, catering to players of varying skill levels. This feature ensures an engaging experience by letting players select their preferred difficulty before starting the game.

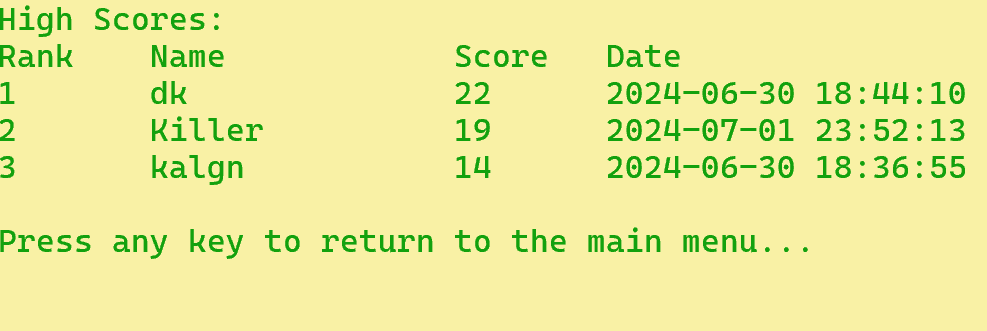


**Game Instruction**

The game instruction screen in the Snake Game project provides players with essential gameplay guidelines. It outlines how to control the snake using arrow keys, the objective of collecting food to grow longer, and avoiding collisions with the snake's own body or walls. Clear instructions ensure players understand the rules and mechanics, fostering a smooth and enjoyable gaming experience.



**View High Score**



# **Gantt Chart**

This Gantt chart outlines the project schedule for developing a Snake game in C programming, starting from project initiation on April 28, 2024, to its completion with a project presentation on July 4, 2024. Key phases include requirements gathering, design, development environment setup, core game mechanics implementation, level implementation, high score management, UI/UX design, testing, documentation, deployment, and final presentation. The chart ensures a structured timeline for each task, facilitating efficient project management and timely delivery.

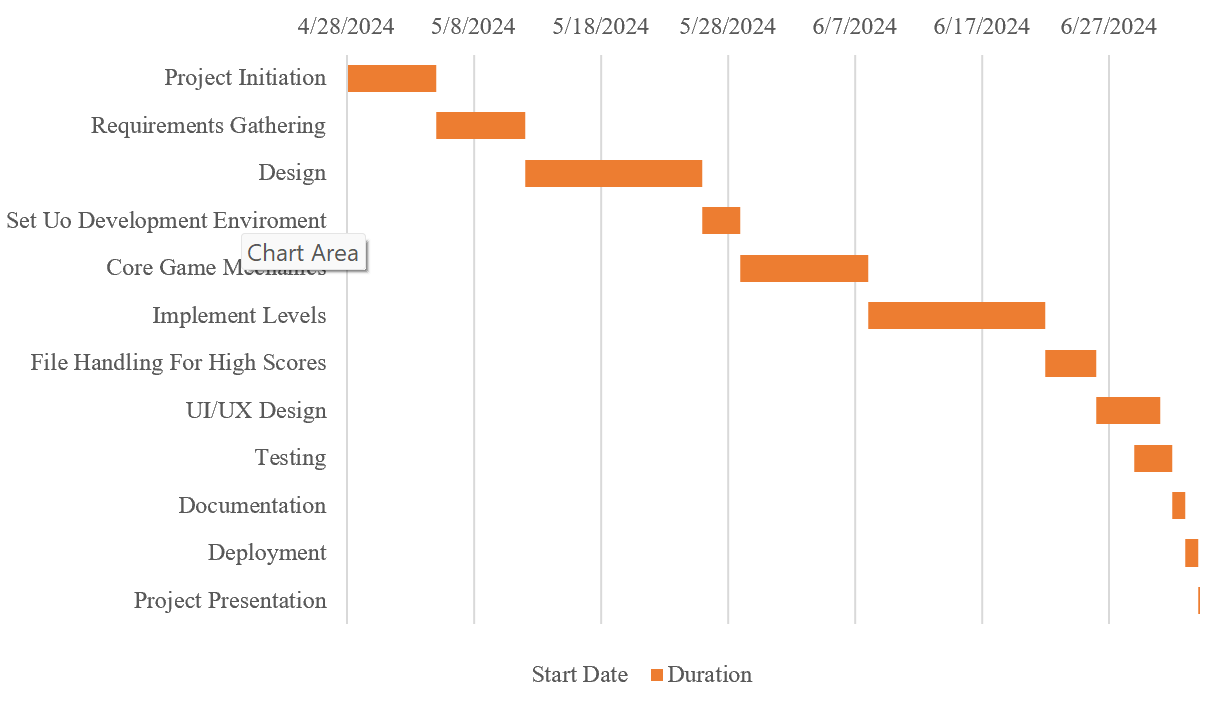


Figure 14. 1 Gantt Chart

# **References**

Ahad, M. A., & Imran, S. M. (January 29, 2024). *Learn Programming with C.* CRC Press.

BayasgalanGL. (2014, 04 14). *Snake Game Source Code in C*. Retrieved from SCRIBD: https://www.scribd.com/doc/218049705/Snake-Game-Source-Code-in-C

CodeWithC. (2022, 06 13). *Mini Project in C Snake Game*. Retrieved from CodeWithC: https://www.codewithc.com/mini-project-in-c-snake-game/?amp=1

Editorial, N. (2021, 08 19). *Developing a Snake Game in C*. Retrieved from NIIT: https://www.niit.com/india/knowledge-centre/Game-development-in-C

Kumar, A. (2017, 04 20). *project on snake game in c language*. Retrieved from https://www.slideshare.net/slideshow/project-on-snake-game-in-c-language/75217408

Publishing, C. (September 6, 2019). *The C Programming Language, 3rd Edition.* Independently Published.

Ravichandran, D. (1996). *D Ravichandran.* New Age International.