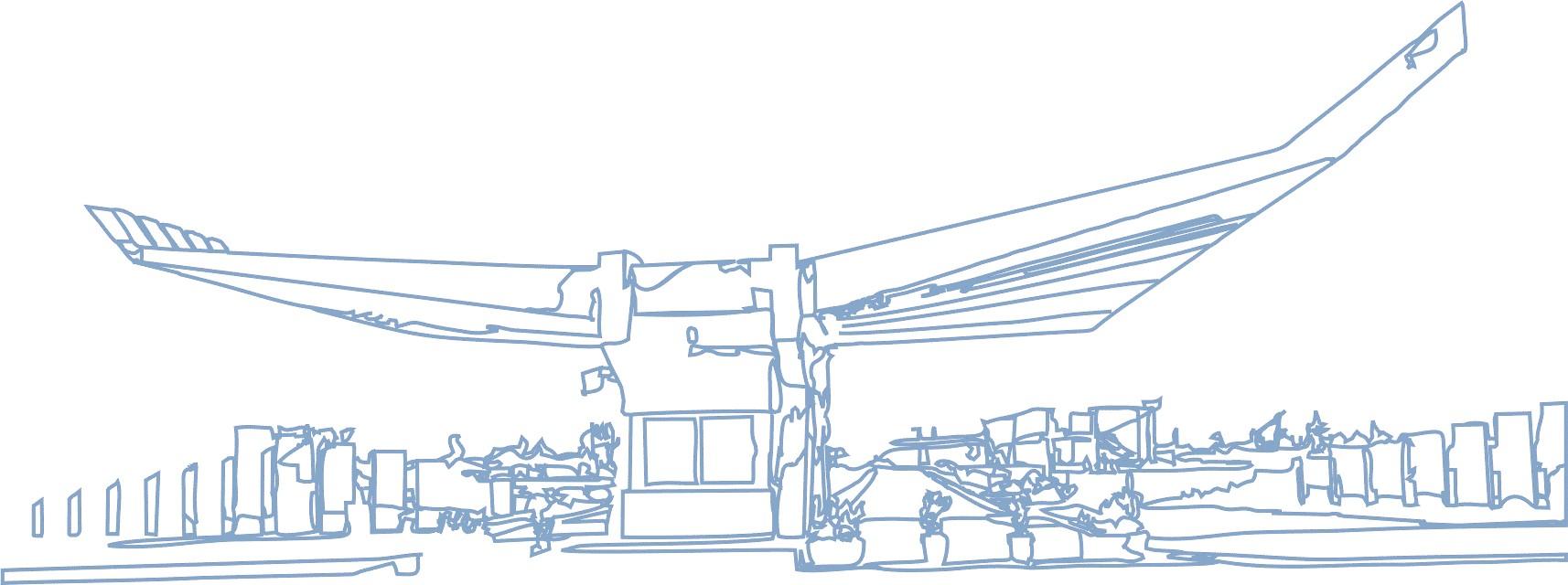
# INTRODUCTION TO SOFTWARE ENGINEERING

**SNAKE GAME**



**ERMIN LILAJ**

# Snake Game Requirements Specification

# Table of Contents

1. **EXECUTIVE SUMMARY 3**
   1. [PROJECT OVERVIEW 3](#_heading=h.gjdgxs)
   2. [PURPOSE AND SCOPE OF THIS SPECIFICATION 3](#_heading=h.30j0zll)
2. **PRODUCT/SERVICE DESCRIPTION 3**
   1. [PRODUCT CONTEXT 3](#_heading=h.1fob9te)
   2. [USER CHARACTERISTICS 3](#_heading=h.3znysh7)
   3. [ASSUMPTIONS 3](#_heading=h.2et92p0)
   4. [CONSTRAINTS 3](#_heading=h.tyjcwt)
   5. [DEPENDENCIES 4](#_heading=h.3dy6vkm)
3. [**REQUIREMENTS 4**](#_heading=h.1t3h5sf)
   1. [FUNCTIONAL REQUIREMENTS 5](#_heading=h.4d34og8)
   2. [NON-FUNCTIONAL REQUIREMENTS 5](#_heading=h.2s8eyo1)
      1. [*User Interface Requirements 5*](#_heading=h.17dp8vu)
      2. [*Usability 5*](#_heading=h.3rdcrjn)
      3. [*Performance 6*](#_heading=h.26in1rg)
      4. [*Manageability/Maintainability 6*](#_heading=h.lnxbz9)
      5. [*Security 8*](#_heading=h.35nkun2)
      6. [*Standards Compliance 8*](#_heading=h.1ksv4uv)
      7. [*Other Non-Functional Requirements 9*](#_heading=h.44sinio)
   3. [DOMAIN REQUIREMENTS 9](#_heading=h.2jxsxqh)
4. [**DESIGN THINKING METHODOLOGIES** 10](#_heading=h.z337ya)

|  |  |  |
| --- | --- | --- |
| 4.1 | Negotiation | 11 |
| 4.2 | Empathy | 11 |
| 4.3 | Noticing | 11 |
| 4.4 | GUI | 11 |

1. **SOFTWARE DESIGN** 12

|  |  |  |
| --- | --- | --- |
| 5.1 | Use Case | 12 |
| 5.2 | State Diagram | 12 |
| 5.3 | Class Diagram | 12 |

# APPENDIX 13

APPENDIX A. DEFINITIONS, ACRONYMS, AND ABBREVIATIONS 13

APPENDIX B. REFERENCES 13

APPENDIX C. ORGANIZING THE REQUIREMENTS 13

1. Executive Summary

## Project Overview

The Snake Game is a classic arcade-style game where players control a snake to eat food and grow in length while avoiding collisions with walls and the snake's own body. The game aims to provide an entertaining and challenging experience for players of all ages.

Intended Audience:

The Snake Game is designed for casual gamers and enthusiasts who enjoy retro-style games. It can be enjoyed by players of varying skill levels, from beginners looking for a simple yet engaging gaming experience to experienced gamers seeking a nostalgic throwback to the classic Snake game. The game's intuitive controls and straightforward gameplay make it accessible to a wide audience.

## Purpose and Scope of this Specification

This game has entertaing purpose.

Intended Audience:

The Snake Game is designed for casual gamers and enthusiasts who enjoy retro-style games. It can be enjoyed by players of varying skill levels, from beginners looking for a simple yet engaging gaming experience to experienced gamers seeking a nostalgic throwback to the classic Snake game. The game's intuitive controls and straightforward gameplay make it accessible to a wide audience.

1. Product/Service Description

## Product Context

The Snake Game is an independent and self-contained product that does not rely on or interface with any external systems or products. It is designed to be a standalone game that can be played on various platforms such as computers, smartphones, or gaming consoles.

The game does not have any major components or interconnections with larger systems. It operates within its own environment, providing a self-contained gaming experience. Players can launch the game and play it directly without any dependencies on external systems or interfaces.

## User Characteristics

Student:

Role: Students of all ages and educational levels.

Experience: Varied levels of gaming experience.

Technical Expertise: Familiar with basic computer operations.

General Characteristics: Interested in casual gaming, seeking entertainment and relaxation.

Faculty:

Role: Academic staff and teachers.

Experience: Varied levels of gaming experience.

Technical Expertise: Familiar with basic computer operations.

General Characteristics: May use the game as a leisure activity or as a brief break from work, seeking a simple and enjoyable gaming experience.

Staff:

Role: Administrative and support staff.

Experience: Varied levels of gaming experience.

Technical Expertise: Familiar with basic computer operations.

General Characteristics: May use the game during breaks or downtime, seeking a fun and engaging activity to relieve stress.

## Assumptions

Assumptions:

1.Availability of compatible hardware: It is assumed that users have access to a computer or mobile device that meets the minimum system requirements to run the Snake Game.

2.Availability of necessary software: It is assumed that users have access to a compatible operating system (e.g., Windows, macOS, Linux, Android, iOS) that can run the Snake Game.

3.Basic computer literacy: It is assumed that users have a basic understanding of computer operations, such as using a keyboard and mouse or touchscreen input.

4.Adequate display resolution: It is assumed that users have a display with a minimum resolution that allows them to view the game screen and interact with the game elements effectively.

5.Valid input devices: It is assumed that users have functional input devices, such as a keyboard to control the snake's movements in the game.

## Constraints

Compatibility with older systems: The Snake Game should be designed to operate parallelly with older systems, ensuring backward compatibility and allowing users on older hardware or software configurations to run the game smoothly.

Audit functions: The Snake Game should incorporate audit functions, such as maintaining an audit trail or log files, to track important events, actions, and errors for troubleshooting and analysis purposes.

Access, management, and security: The design of the Snake Game should consider access control, user management, and security measures to protect user data and prevent unauthorized access or malicious activities.

Application criticality: Although the Snake Game may not be considered critical in terms of potential risks or impacts, it should still be designed to provide a reliable and stable gaming experience without frequent crashes or failures.

System resource constraints: The Snake Game should be mindful of system resource constraints, such as limits on disk space, memory usage, and processing power. It should optimize resource utilization to ensure smooth gameplay even on devices with limited resources.

Design standards and frameworks: The Snake Game should adhere to relevant design standards and programming language or framework requirements. It may need to comply with specific design guidelines or follow industry best practices to ensure code quality, maintainability, and compatibility with development tools and platforms.

These constraints provide guidance for the design and development of the Snake Game, ensuring that it meets the necessary compatibility, security, resource utilization, and design standards to deliver a robust and user-friendly gaming experience.

## Dependencies

User input: The Snake Game is dependent on user input to control the movements of the snake and interact with the game elements.

Graphics rendering library: The Snake Game may depend on a specific graphics rendering library or framework to display the game graphics and animations effectively.

Input handling library: The Snake Game may depend on an input handling library or framework to efficiently capture and process user input from devices such as keyboards or controllers.

System platform: The Snake Game may have dependencies on specific system platforms, such as operating systems or hardware configurations, to ensure compatibility and proper functioning.

# Requirements

* Describe all system requirements in enough detail for designers to design a system satisfying the requirements and testers to verify that the system satisfies requirements.
* Organize these requirements in a way that works best for your project. See Appendix Appendix D, Organizing the Requirements for different ways to organize these requirements.
* Describe every input into the system, every output from the system, and every function performed by the system in response to an input or in support of an output. (Specify what functions are to be performed on what data to produce what results at what location for whom.)
* Each requirement should be numbered (or uniquely identifiable) and prioritized.

See the sample requirements in Functional Requirements, and System Interface/Integration, as well as these example priority definitions:

# Priority Definitions (This is not a section, but a guideline on how you should organize the requirements)

The following definitions are intended as a guideline to prioritize requirements.

* + Priority 1 – The requirement is a “must have” as outlined by policy/law
  + Priority 2 – The requirement is needed for improved processing, and the fulfillment of the requirement will create immediate benefits
  + Priority 3 – The requirement is a “nice to have” which may include new functionality

It may be helpful to phrase the requirement in terms of its priority, e.g., "The value of the employee status sent to DIS **must be** either A or I" or "It **would be nice** if the application warned the user that the expiration date was 3 business days away". Another approach would be to group requirements by priority category.

* A good requirement is:
  + Correct
  + Unambiguous (all statements have exactly one interpretation)
  + Complete (where TBDs are absolutely necessary, document why the information is unknown, who is responsible for resolution, and the deadline)
  + Consistent
  + Ranked for importance and/or stability
  + Verifiable (avoid soft descriptions like “works well”, “is user friendly”; use concrete terms and specify measurable quantities)
  + Modifiable (evolve the Requirements Specification only via a formal change process, preserving a complete audit trail of changes)
  + Does not specify any particular design
  + Traceable (cross-reference with source documents and spawned documents).

## Functional Requirements

In the example below, the requirement numbering has a scheme - BR\_LR\_0## (BR for Business Requirement, LR for Labor Relations). For small projects simply BR-## would suffice. Keep in mind that if no prefix is used, the traceability matrix may be difficult to create (e.g., no differentiation between '02' as a business requirement vs. a test case)

The following table is an example format for requirements. Choose whatever format works best for your project.

For Example:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Req#** | **Requirement** | **Comments** | **Priority** | **Date Rvwd** | **SME**  **Reviewed / Approved** |
| BR\_LR  \_05 | The system should associate a supervisor indicator with each job class. | Business Process = “Maintenance | 3 | 7/13/04 | Bob Dylan, Mick Jagger |
| BR\_LR  \_08 | The system should handle any number of fees (existing and new) associated with unions. | Business Process = “Changing Dues in the System”  An example of a new fee is an initiation fee. | 2 | 7/13/04 | Bob Dylan, Mick Jagger |
| BR\_LR  \_10 | The system should capture and maintain job class status (i.e., active or inactive) | Business Process = “Maintenance”  Some job classes are old and are no longer used. However, they still need to be maintained for legal, contract and historical purposes. | 2 | 7/13/04 | Bob Dylan, Mick Jagger |
| BR\_LR  \_16 | The system should assign the Supervisor Code based on the value in the Job Class table and additional | April 2005 – New requirement. It is one of | 2 |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | criteria as specified by the clients. | three new requirements from BR\_LR\_03. |  |  |  |
| BR\_LR  \_18 | The system should provide the Labor Relations office with the ability to override the system-derived Bargaining Unit code and the Union Code for  to-be-determined employee types, including hourly appointments. | April 2005 – New requirement. It is one of three new requirements from BR\_LR\_04.  5/11/2005 – Priority changed from 2 to 3. | ~~2~~  3 |  |  |

## Non-Functional Requirements

# Product Requirements:

# Performance: The game should run smoothly and respond quickly to user input.

# Graphics: The game should have visually appealing graphics and animations.

# Compatibility: The game should be compatible with different operating systems and hardware configurations.

# Stability: The game should be stable and free from crashes or unexpected errors.

# Scalability: The game should be designed to handle increasing levels of difficulty and gameplay complexity.

# Organizational Requirements:

# Documentation: Comprehensive documentation should be provided for the game, including user manuals and source code documentation.

# Collaboration: The development team should collaborate effectively to ensure timely delivery of the game and resolve any issues or conflicts.

# External Requirements:

# User Interface: The game should have a user-friendly interface that is easy to navigate.

# Accessibility: The game should be accessible to players with disabilities, considering factors such as visual impairments or motor disabilities.

# Security: The game should have measures in place to protect user data and prevent unauthorized access.

# Legal Compliance: The game should comply with relevant laws and regulations, such as copyright and intellectual property rights.

# User Interface Requirements

Main Menu:

The main menu should have a visually appealing layout with clear options for the players.

It should display the game title and provide options to start a new game, view high scores, access settings, and exit the game.

The menu should be easy to navigate using arrow keys or mouse input.

Game Screen:

The game screen should have a clear and uncluttered layout to provide a good gaming experience.

It should display the snake, food, score, and any other relevant game elements.

The snake's movements should be visually represented as it navigates through the game screen.

The score should be prominently displayed and updated as the player earns points.

If the snake collides with a wall or itself, an appropriate game over message should be displayed.

High Scores:

The high scores screen should display the top scores achieved by players.

It should show the player's name or initials and their corresponding score.

The scores should be listed in descending order, with the highest score at the top.

Error Messages:

The game should provide clear and informative error messages in case of any issues or invalid inputs.

Error messages should help the player understand the problem and provide guidance on how to resolve it.

Messages should be displayed in a visually distinct manner to draw attention.

Input Controls:

The game should support input from various sources, such as keyboard arrow keys or WASD keys for snake movement.

The player should be able to pause the game, restart the game, or navigate through menu options using intuitive input controls.

The controls should be responsive and provide a smooth gaming experience.

# Usability

**Learnability:**

The system should be easy to learn, allowing new players to quickly understand and play the game without extensive guidance.

The user interface should be intuitive, with clear instructions and prompts to guide the player.

The game should provide a brief tutorial or instructions at the beginning to familiarize the player with the controls and gameplay mechanics.

The system should have comprehensive user documentation and help resources that explain the game's features, controls, and any additional instructions or tips.

**Context-Sensitive Help:**

The help system should be context-sensitive, providing relevant information and guidance based on the player's current actions or menu options.

The help system should explain how to perform common tasks, such as starting a new game, navigating menus, and understanding game rules.

Help resources should be easily accessible within the game interface, allowing players to quickly access information without disrupting their gameplay experience.

**Clear Instructions and Prompts:**

The system should provide clear and concise instructions and prompts throughout the game to guide the player.

Instructions should be presented in a visually distinct manner to draw attention and ensure they are easily noticed by the player.

Prompts should be provided for important actions or events, such as eating food, reaching high scores, or encountering obstacles.

**Ease of Use:**

The system should be designed with a focus on simplicity and ease of use.

The user interface should have a clean and uncluttered layout, avoiding unnecessary complexity.

Controls and menu options should be logically organized and labeled, making it easy for the player to understand their purpose and functionality.

The game should provide feedback and visual cues to indicate the outcome of player actions, such as successful food consumption, score updates, and game over events.

# Performance

**Response Time:**

The system should respond to user inputs and actions with minimal delay, aiming for an average response time of X seconds or less.

The game screen and animations should render smoothly and without noticeable lag to provide a seamless gaming experience.

Actions such as snake movement, food generation, and score updates should be processed and displayed instantaneously.

**Game Fluidity:**

The system should ensure smooth and fluid gameplay, maintaining a consistent frame rate throughout the game session.

The snake's movements should be responsive and free from jerky or stuttering motion.

**Load Handling**:

The system should handle the processing requirements of the game efficiently, even during intense gameplay moments with high activity.

It should be able to handle the snake's movement and collision detection smoothly, without any noticeable slowdowns or interruptions.

**Game Startup Time:**

The system should have a fast startup time, allowing the player to begin playing the game quickly after launching it.

Any necessary initialization or loading processes should be optimized to minimize the waiting time for the player.

**Resource Usage:**

The system should utilize system resources, such as CPU and memory, efficiently to ensure optimal performance and avoid excessive resource consumption.

# Manageability/Maintainability

**3.2.4.1 Monitoring:**

The system should include built-in monitoring capabilities to detect and report any failures or errors.

It should provide error detection mechanisms that can identify and notify the user or system administrator of any issues.

The system should log relevant information about errors, failures, and system activities for troubleshooting and analysis purposes.

**3.2.4.2 Maintenance:**

The system should be designed in a modular and maintainable manner to facilitate ease of maintenance.

The codebase should follow best practices for clean and readable code, making it easier for developers to understand, modify, and fix issues.

The system should have clear documentation and comments that aid in understanding the code and its functionality.

Any external dependencies or libraries used should be well-documented and kept up-to-date to ensure maintainability.

**3.2.4.3 Operations:**

The system should support both interactive and unattended operations, allowing the user to actively interact with the game and also perform automated tasks.

Data processing support functions should be available to handle tasks such as saving and loading game state, managing player profiles, and storing high scores.

The system should have backup and recovery mechanisms in place to protect against data loss or system failures.

Safety considerations should be taken into account, ensuring that the game operates within acceptable limits and does not pose any risks to the user.

Disaster recovery and business resumption plans should be in place to handle unexpected events and ensure continuity of the game service.

# Security

**3.2.5.1 Protection:**

The system should incorporate encryption mechanisms to protect sensitive data from unauthorized access or modification.

Data integrity checks should be implemented to detect and prevent unauthorized modifications to data, ensuring the integrity and reliability of the system.

**3.2.5.2 Authorization and Authentication:**

The system should have user authentication in place to verify the identity of the sole user and ensure that only authorized access is granted.

# Standards Compliance

The system should comply with relevant industry standards, policies, regulations, and laws.

Compliance with data naming conventions should be ensured to maintain consistency and clarity.

The system should adhere to accounting procedures and standards, particularly in areas such as financial transactions and reporting.

If applicable, the system should provide audit tracing capabilities to record and track relevant activities for compliance purposes..

# Other Non-Functional Requirements

Reliability: The system should be reliable, with minimal downtime and a low probability of system failures.

Scalability: The system should be scalable to accommodate potential future growth in terms of data volume, user load, and functionality.

Availability: The system should be available for use during specified hours of operation to meet user needs.

Compatibility: The system should be compatible with relevant hardware, operating systems, and software components.

Data Security: The system should ensure the security and confidentiality of user data, employing measures such as encryption, access controls, and data backups.

Performance Efficiency: The system should perform efficiently, with fast response times and optimized resource utilization.

Error Handling: The system should handle errors gracefully, providing informative error messages and recovering from errors without data loss or system instability..

## Domain Requirements

Game Rules: The game should adhere to the standard rules of the classic Snake game, where the snake moves through the grid, consumes food to grow, and dies if it collides with walls or itself.

Grid Layout: The game should be played on a rectangular grid layout, with defined dimensions and boundaries.

Food Generation: The food should be randomly generated on the grid at regular intervals or when the previous food is consumed by the snake.

Score Calculation: The score should be calculated based on the length of the snake and the number of food items consumed.

Game Over Conditions: The game should end when the snake collides with the walls of the grid or its own body, displaying the final score and providing an option to restart the game.

Difficulty Levels: The game may include multiple difficulty levels, such as easy, medium, and hard, which affect factors such as the snake's speed and the frequency of food generation.

Sound Effects: The game may incorporate sound effects to enhance the gaming experience, such as playing sounds when the snake consumes food, collides with obstacles, or achieves certain milestones.

High Scores: The system should store and display the highest scores achieved by players, allowing users to compete for top rankings.

Error Handling: The game should handle errors gracefully, displaying informative error messages in case of any unexpected issues or failures.

User Feedback: The game should provide visual feedback to the player, such as displaying the current score, the length of the snake, and any relevant messages or notifications during gameplay.

Game Controls: The game should allow the player to control the snake's movements using arrow keys or other designated input controls.

User Interface: The game's user interface should be intuitive and user-friendly, providing clear instructions and visual cues to guide the player..

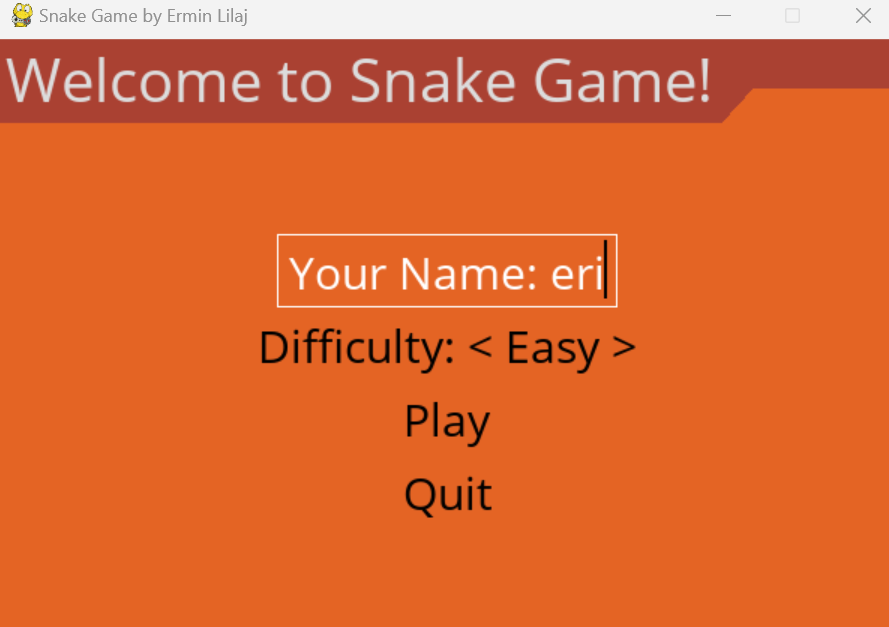
# Design thinking methodologies

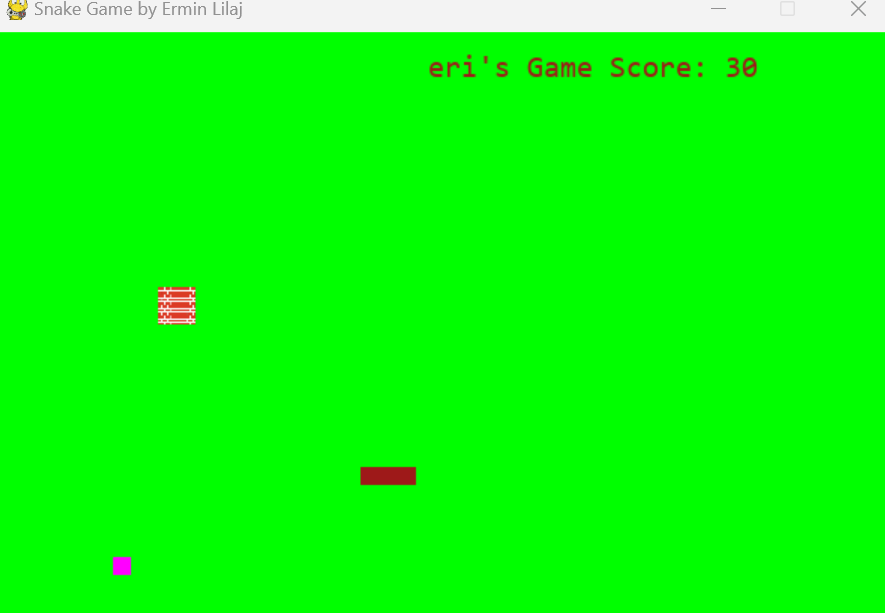
Empathy: Gaining a deep understanding of the players and their needs, ensuring that the game aligns with their preferences for gameplay mechanics, controls, and overall user experience.

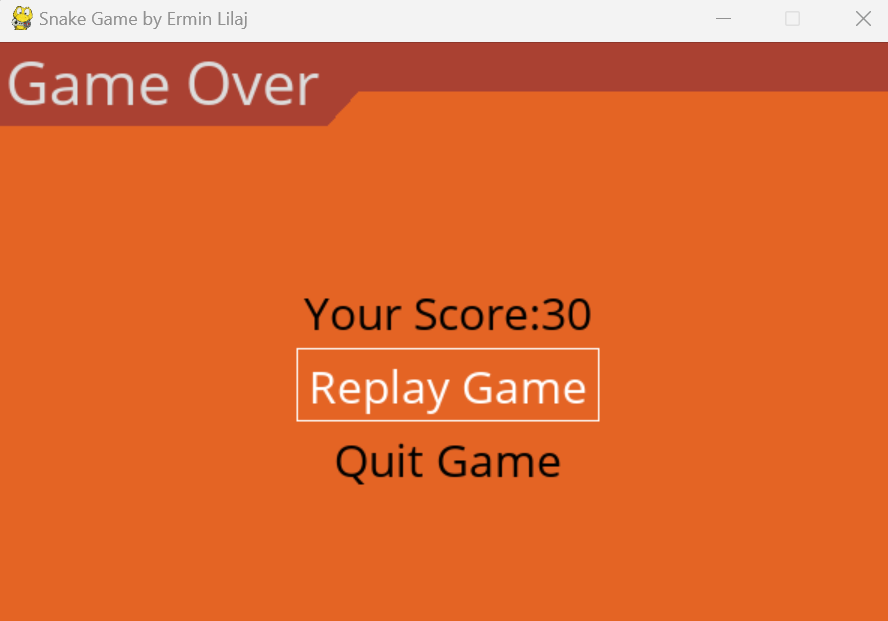
Define: Identifying the core problem or challenge in designing the snake game, such as creating an engaging and enjoyable gameplay experience while maintaining the simplicity and familiarity of the classic snake game concept.

Ideate: Generating various ideas and solutions to address the defined challenge, brainstorming innovative power-ups, different game modes, or visual enhancements that enhance the core gameplay mechanics.

Test: Conducting testing and gathering feedback from players to observe how they interact with the game, identifying areas for improvement, resolving issues, and refining the design to enhance the overall gameplay experience







# Software Design

* 1. Use Case

Start Game:

The player can start a new game session.

Control Snake:

The player can control the movement of the snake using arrow keys or designated controls.

The snake moves in the direction indicated by the player.

Collect Food:

The snake can collect food items (represented by dots or icons) to increase its length and score.

When the snake consumes food, it grows longer.

Avoid Obstacles:

The snake must avoid collisions with walls or its own body.If the snake collides with an obstacle, the game ends.

Score Tracking:

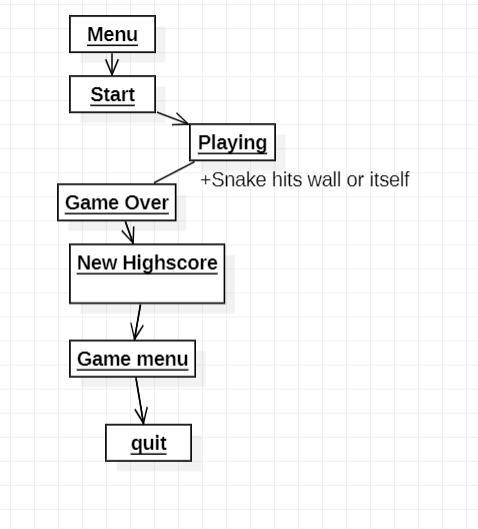
The game keeps track of the player's score based on the number of food items collected.

The score is displayed during gameplay and after the game ends.

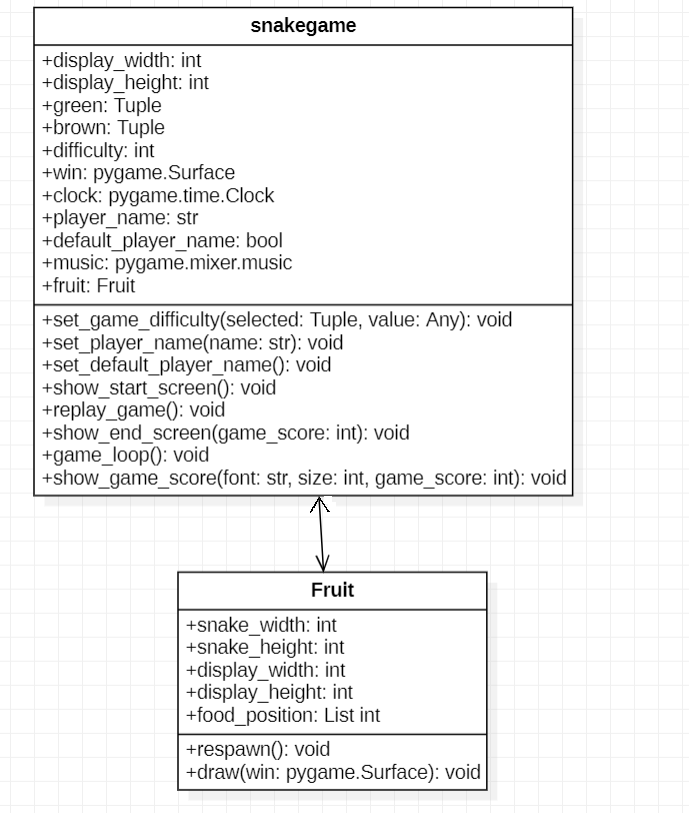
Game Over:

The game ends when the snake collides with an obstacle.The player can view their final score and an option to restart the game.

# State Diagram



# Class Diagram



Some systems can be best organized by describing their functions in terms of stimuli. For example, the functions of an automatic aircraft landing system may be organized into sections for loss of power, wind shear, sudden change in roll, vertical velocity excessive, etc.

# By Response

Some systems can be best organized by describing all the functions in support of the generation of a response. For example, the functions of a personnel system may be organized into sections corresponding to all functions associated with generating paychecks, all functions associated with generating a current list of employees, etc.

# By Functional Hierarchy

When none of the above organizational schemes prove helpful, the overall functionality can be organized into a hierarchy of functions organized by common inputs, common outputs, or common internal data access. Data flow diagrams and data dictionaries can be used to show the relationships between and among the functions and data.

# Additional Comments

Whenever a new Requirements Specification is contemplated, more than one of the organizational techniques given above may be appropriate. In such cases, organize the specific requirements for multiple hierarchies tailored to the specific needs of the system under specification.

There are many notations, methods, and automated support tools available to aid in the documentation of requirements. For the most part, their usefulness is a function of organization. For example, when organizing by mode, finite state machines or state charts may prove helpful; when organizing by object, object-oriented analysis may prove helpful; when organizing by feature, stimulus-response sequences may prove helpful; and when organizing by functional hierarchy, data flow diagrams and data dictionaries may prove helpful.