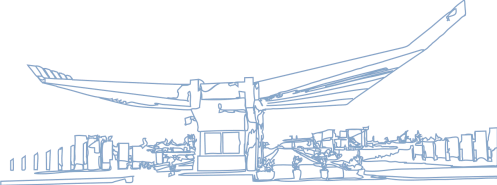
***[YourProject] Requirements Specification ***

**INTRODUCTION TO SOFTWARE ENGINEERING**

**[Name of the Project]**

****

**Team Members:**

**Kristi Samara**

**Name Surname [2]**

**Name Surname [3]**

Page 1 o f 11

***[YourProject] Requirements Specification ***

**[YourProject] Requirements Specification**

Use this Requirements Specification template to document the requirements for your product or service, including priority and approval. Tailor the specification to suit your project, organizing the applicable sections in a way that works best, and use the checklist to record the decisions about what is applicable and what isn't.

The format of the requirements depends on what works best for your project.

This document contains instructions and examples which are for the benefit of the person writing the document and should be removed before the document is finalized.

Page 2 o f 11

***[YourProject] Requirements Specification *Table of Contents**

**1. EXECUTIVE SUMMARY 3**

1.1 PROJECT OVERVIEW 3 1.2 PURPOSE AND SCOPE OF THIS SPECIFICATION 3

**2. PRODUCT/SERVICE DESCRIPTION 3**

2.1 PRODUCT CONTEXT 3 2.2 USER CHARACTERISTICS 3 2.3 ASSUMPTIONS 3 2.4 CONSTRAINTS 3 2.5 DEPENDENCIES 4

**3. REQUIREMENTS 4**

3.1 FUNCTIONAL REQUIREMENTS 5 3.2 NON-FUNCTIONAL REQUIREMENTS 5 *3.2.1 User Interface Requirements 5 3.2.2 Usability 5 3.2.3 Performance 6 3.2.4 Manageability/Maintainability 6 3.2.5 Security 8 3.2.6 Standards Compliance 8 3.2.7 Other Non-Functional Requirements 9* 3.3 DOMAIN REQUIREMENTS 9

4. **DESIGN THINKING METHODOLOGIES** 10 4.1 Negotiation 11 4.2 Empathy 11 4.3 Noticing 11 4.4 GUI 11

**5. SOFTWARE DESIGN** 12 5.1 Use Case 12 5.2 State Diagram 12 5.3 Class Diagram 12

**APPENDIX 1**3

APPENDIX A.DEFINITIONS, ACRONYMS, AND ABBREVIATIONS 13 APPENDIX B.REFERENCES 13 APPENDIX C. ORGANIZING THE REQUIREMENTS 13

Page 3 o f 11

***[YourProject] Requirements Specification ***1. Executive Summary

***1.1 Project Overview***

Describe this project and its intended audience.

***1.2 Purpose and Scope of this Specification***

Describe the purpose of this specification and its intended audience. Include a description of what is within the scope and what is outside of the scope of these specifications.

For example:

**In scope**

This document addresses requirements related to phase 2 of Project A:

● modification of Classification Processing to meet legislative mandate ABC. ● modification of Labor Relations Processing to meet legislative mandate ABC. **Out of Scope**

The following items in phase 3 of Project A are out of scope:

● modification of Classification Processing to meet legislative mandate XYZ. ● modification of Labor Relations Processing to meet legislative mandate XYZ. (Phase 3 will be considered in the development of the requirements for Phase 2, but the Phase 3 requirements will be documented separately.)

2. Product/Service Description

In this section, describe the general factors that affect the product/service and its requirements. This section should contain background information, not state specific requirements (provide the reasons why certain specific requirements are later specified).

***2.1 Product Context***

How does this product relate to other products? Is it independent and self-contained? Does it interface with a variety of related systems? Describe these relationships or use a diagram to show the major components of the larger system, interconnections, and external interfaces.

***2.2 User Characteristics***

Create general customer profiles for each type of user who will be using the product. Profiles should include:

● Student/faculty/staff/other

● experience

● technical expertise

● other general characteristics that may influence the product

***2.3 Assumptions***

List any assumptions that affect the requirements, for example, equipment availability, user expertise, etc. For example, a specific operating system is assumed to be available; if the operating system is not available, the Requirements Specification would then have to change accordingly.

Page 4 o f 11

***[YourProject] Requirements Specification 2.4 Constraints***

Describe any items that will constrain the design options, including

● parallel operation with an old system

● audit functions (audit trail, log files, etc.)

● access, management and security

● criticality of the application

● system resource constraints (e.g., limits on disk space or other hardware limitations) ● other design constraints (e.g., design or other standards, such as programming language or framework)

***2.5 Dependencies***

List dependencies that affect the requirements. Examples:

● This new product will require a daily download of data from X,

● Module X needs to be completed before this module can be built.

**3. 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 DAppendix 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)

Page 5 o f 11

***[YourProject] Requirements Specification ***● 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).

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

Page 6 o f 11

***[YourProject] Requirements Specification ***

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

***3.2 Non-Functional Requirements***

**In here try to provide the main categories of:**

**Product Requirements**

**Organizational Requirements**

**External Requirements**

And their subcategories, try to arrange them in a proper visible way.

**3.2.1 User Interface Requirements**

In addition to functions required, describe the characteristics of each interface between the product and its users (e.g., required screen formats/organization, report layouts, menu structures, error and other messages, or function keys).

**3.2.2 Usability**

Include any specific usability requirements, for example,

Learnability

● The user documentation and help should be complete

● The help should be context sensitive and explain how to achieve common tasks ● The system should be easy to learn

(See http://www.usabilitynet.org/)

**3.2.3 Performance**

Specify static and dynamic numerical requirements placed on the system or on human interaction with the system:

● Static numerical requirements may include the number of terminals to be supported, the number of simultaneous users to be supported, and the amount and type of information to be handled. ● Dynamic numerical requirements may include the number of transactions and tasks and the amount of data to be processed within a certain time period for both normal and peak workload conditions. All of these requirements should be stated in measurable form. For example, "95% of the transactions shall be processed in less than 1 second" rather than “an operator shall not have to wait for the transaction to complete”.

Page 7 o f 11

****3.2.4 Manageability/Maintainability**

3.2.1.1 Monitoring

Include any requirements for product or service health monitoring, failure conditions, error detection, logging, and correction.

3.2.1.2 Maintenance

Specify attributes of the system that relate to ease of maintenance. These requirements may relate to modularity, complexity, or interface design. Requirements should not be placed here simply because they are thought to be good design practices.

3.2.1.3 Operations

Specify any normal and special operations required by the user, including:

● periods of interactive operations and periods of unattended operations

● data processing support functions

● backup and recovery operations

● safety considerations and requirements

● disaster recovery and business resumption

**3.2.5 Security**

3.2.5.1 Protection:

To protect the Snake game system from malicious or accidental access, modification, disclosure, destruction, or misuse, the following factors could be considered:

●Access controls: Implement access controls to restrict access to sensitive data and system resources to authorized personnel only.

●Encryption: Encrypt sensitive data such as passwords, user data, and game settings to prevent unauthorized access or disclosure.

●Error handling and exception handling: Proper error handling and exception handling should be implemented to avoid system crashes or data corruption due to invalid input or unexpected errors.

●Data backups: Regular backups of game data should be taken to avoid data loss due to system failure or other unexpected events.

●Antivirus and malware protection: Install an antivirus and malware protection software to protect the system from viruses and other malicious software.

3.2.5.2 Authorization and Authentication:

To ensure proper Authorization and Authentication, the following factors could be considered:

●User authentication: Implement a secure user authentication mechanism such as a username and password combination or multi-factor authentication to prevent unauthorized access.

●Role-based access control: Implement a role-based access control mechanism to restrict access to sensitive data and system resources based on user roles and privileges.

●Session management: Implement proper session management to prevent session hijacking and ensure that user sessions expire after a certain amount of time.

.

**3.2.6 Standards Compliance:**

To ensure compliance with existing standards, policies, regulations, or laws, the following requirements could be considered:

● Data privacy: Ensure compliance with data privacy regulations such as the General Data Protection Regulation (GDPR) and ensure that user data is collected and processed in accordance with these regulations.

●Accessibility: Ensure that the game is accessible to users with disabilities and complies with accessibility standards such as the Web Content Accessibility Guidelines (WCAG).

● Intellectual property rights: Ensure that the game does not infringe on any intellectual property rights such as copyrights or trademarks. Licensing: Ensure that the game complies with any licensing requirements for third-party software or libraries used in the development of the game.

**3.2.7 Other Non-Functional Requirements:**

Other non-functional requirements that could be considered include:

●Performance: The game should perform well and not have any significant lag or delays, even when the player's score is high.

●Usability: The game should be easy to use and understand, with clear instructions and intuitive controls.

● Scalability: The game should be able to handle an increasing number of users without any significant decrease in performance.

● Reliability: The game should be reliable and not crash or freeze during gameplay. Maintainability: The game should be designed and developed in a way that makes it easy to maintain and update in the future.

Page 8 o f 11

******

***3.3 Domain Requirements***

Game mechanics: The game should follow the basic rules of the classic snake game, including the movement of the snake, the appearance of food, and the game over conditions.

Graphics and user interface: The game should have simple graphics that are easy to understand and use, and a user interface that allows the player to start and restart the game, control the snake, and see their score.

Input and output: The game should accept input from the player's keyboard to control the snake's movement and display output to show the game board, score, and game over messages. Performance: The game should run smoothly and without lag, even as the snake grows longer and the game board becomes more crowded.

Compatibility: The game should be compatible with different operating systems, screen sizes, and Python versions.

Code structure and organization: The game code should be well-organized and easy to read and modify, with clear and concise comments and documentation.

Testing and debugging: The game should be thoroughly tested to ensure that it works as expected, and any bugs or errors should be identified and fixed in a timely manner

4. **Design thinking methodologies**

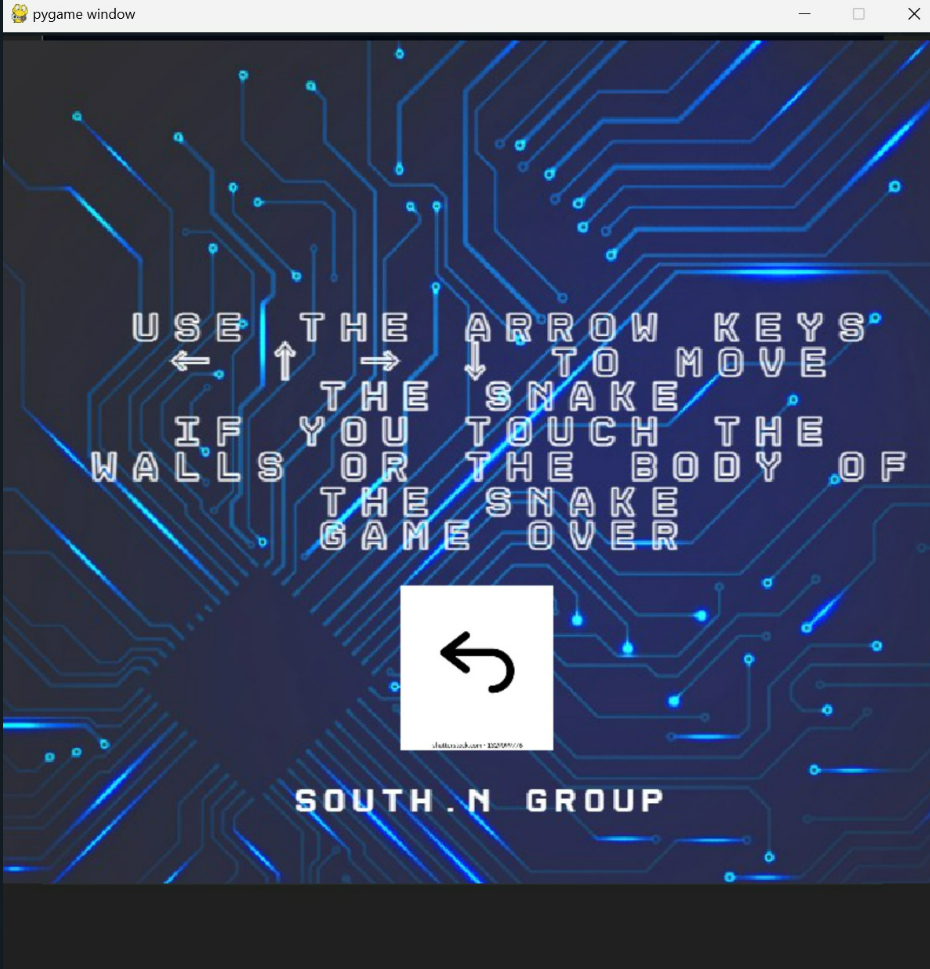
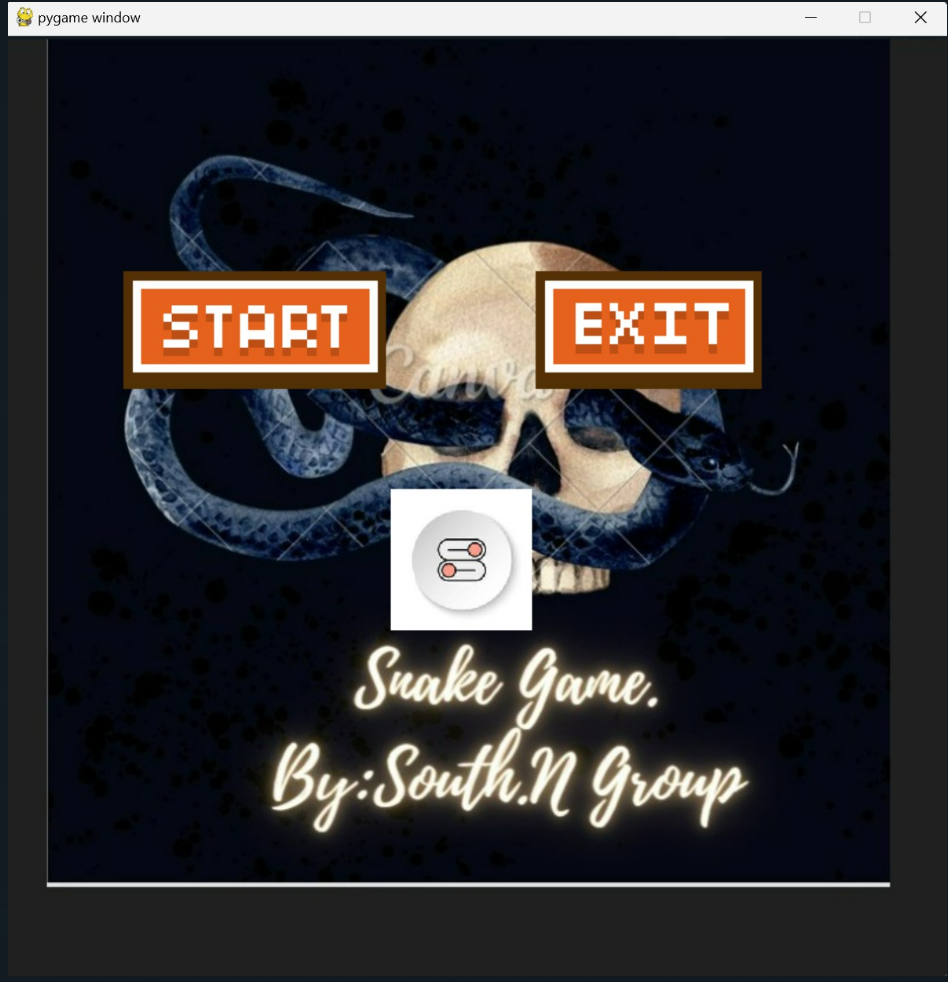
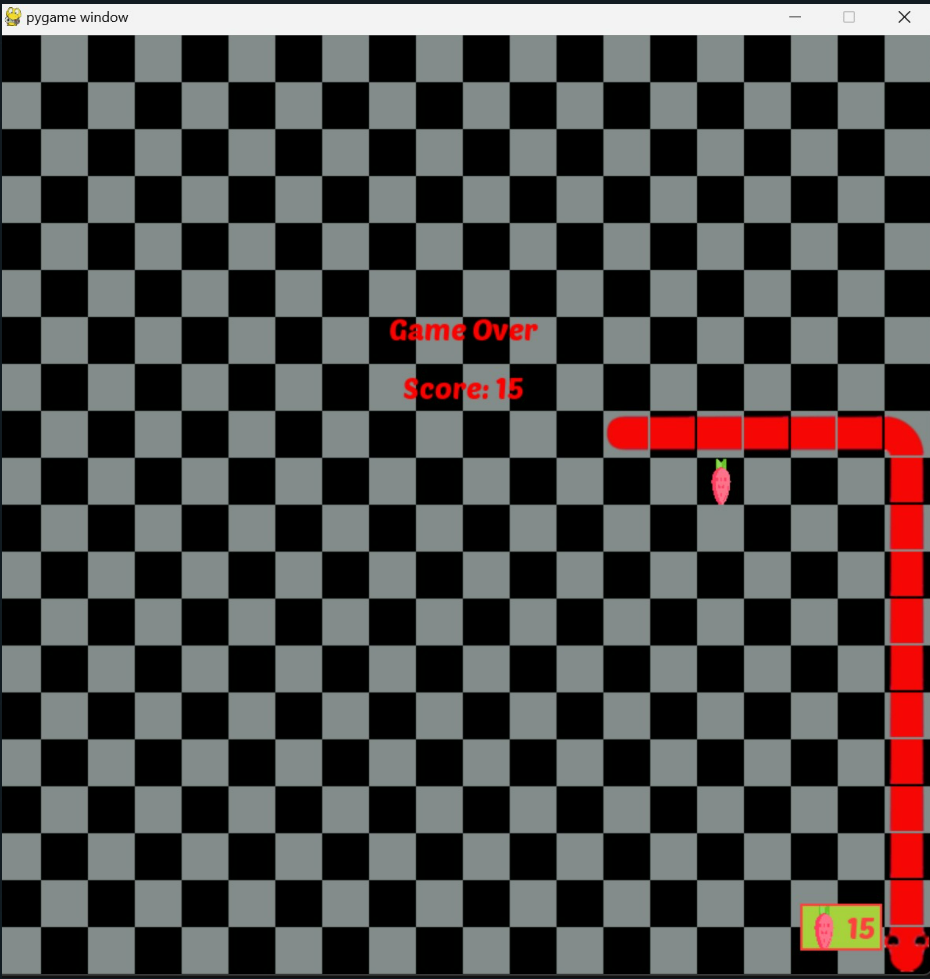
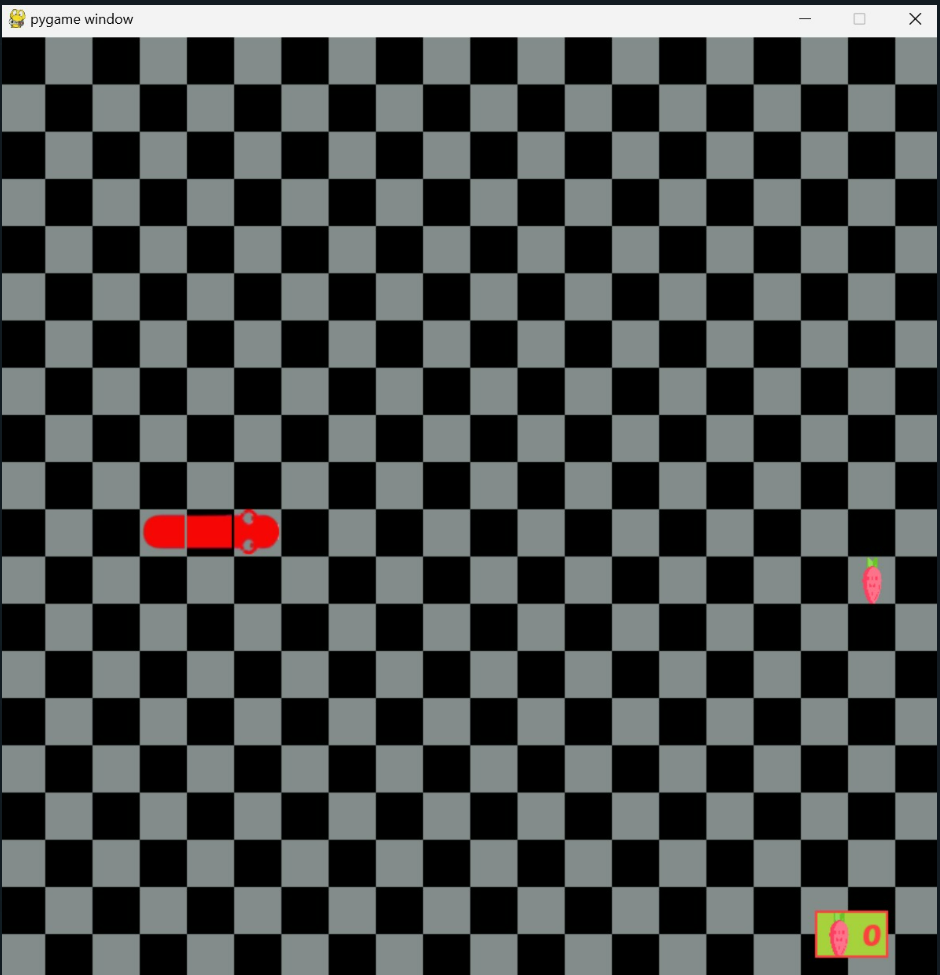
Provide all the Used Design Muscles in Software product

4.1 Negotiation involves working collaboratively with stakeholders to balance competing interests and achieve a common goal. In the context of a snake game, negotiation can involve working with developers, testers, and other team members to ensure that the game is both functional and enjoyable to play. This may involve making trade-offs between different features or design elements to achieve the best possible user experience

4.2 Empathy involves putting oneself in the user's shoes to understand their needs, goals, and challenges. In the context of a snake game, empathy can involve understanding the user's preferences in terms of the game's difficulty level, controls, and visual design. By empathizing with the user, the designer can develop a game that is both engaging and satisfying to play.

4.3 Noticing involves paying close attention to details and patterns in the user's behavior, environment, and interactions with the game. In a snake game, noticing can involve observing the user's gameplay to identify areas where the game can be improved. For example, if the user is struggling with a particular level or feature, the designer can identify the problem and make changes to improve the user experience.

4.4 GUI (Screenshots)



**5. Software Design**

5.1 Use Case

Provide a summary of the major functions that the product will perform. Organize the functions to be understandable to the customer or a first time reader. Include use cases and business scenarios, or provide a link to a separate document (or documents). A business scenario:

● Describes a significant business need

● Identifies, documents, and ranks the problem that is driving the scenario

● Describes the business and technical environment that will resolve the problem ● States the desired objectives

● Shows the “Actors” and where they fit in the business model

● Is specific, and measurable, and uses clear metrics for success

**5.2 State Diagram**

State diagram shows the behavior of classes in response to external stimuli. Specifically a state diagram describes the behavior of a single object in response to a series of events in a system.

**5.3 Class Diagram**

Please follow the following steps to create the class diagram

1. Step 1: Identify the **class** names. The first step is to identify the primary objects of the system. 1. Step 2: Distinguish relationships. Next step is to determine how each of the **classes** or objects are related to one another.

2. Step 3: Create the Structure.

Page 9 o f 11

***[YourProject] Requirements Specification ***

APPENDIX

The appendixes are not always considered part of the actual Requirements Specification and are not always necessary. They may include

● Sample input/output formats, descriptions of cost analysis studies, or results of user surveys; ● Supporting or background information that can help the readers of the Requirements Specification; ● A description of the problems to be solved by the system;

● Special packaging instructions for the code and the media to meet security, export, initial loading, or other requirements.

When appendixes are included, the Requirements Specification should explicitly state whether or not the appendixes are to be considered part of the requirements.

**Appendix A. Definitions, Acronyms, and Abbreviations**

Define all terms, acronyms, and abbreviations used in this document.

**Appendix B. References**

List all the documents and other materials referenced in this document.

**Appendix C. Organizing the Requirements**

This section is for information only as an aid in preparing the requirements document.

Detailed requirements tend to be extensive. Give careful consideration to your organization scheme. Some examples of organization schemes are described below:

**By System Mode**

Some systems behave quite differently depending on the mode of operation. For example, a control system may have different sets of functions depending on its mode: training, normal, or emergency.

**By User Class**

Some systems provide different sets of functions to different classes of users. For example, an elevator control system presents different capabilities to passengers, maintenance workers, and fire fighters.

**By Objects**

Objects are real-world entities that have a counterpart within the system. For example, in a patient monitoring system, objects include patients, sensors, nurses, rooms, physicians, medicines, etc. Associated with each object is a set of attributes (of that object) and functions (performed by that object). These functions are also called services, methods, or processes. Note that sets of objects may share attributes and services. These are grouped together as classes.

**By Feature**

A feature is an externally desired service by the system that may require a sequence of inputs to affect the desired result. For example, in a telephone system, features include local call, call forwarding, and conference call. Each feature is generally described in a sequence of stimulus-response pairs, and may include validity checks on inputs, exact sequencing of operations, responses to abnormal situations,

including error handling and recovery, effects of parameters, relationships of inputs to outputs, including input/output sequences and formulas for input to output.

**By Stimulus**

Page 10 o f 11

***[YourProject] Requirements Specification ***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.

Page 11 o f 11