

CSCI441\_VC\_Software Engineering

# Penultimate Adventure

**Team: G**

## Team Members

<b>Fernando Duffoo</b>	<b>Management &amp; Organization</b>
<b>Isaac Howard</b>	<b>Programming</b>
<b>Johnathan Bennett</b>	<b>Design</b>

**Project URL:** <http://github.com/PenultimateAdventure>

## **Table of Contents**

Title page	Pg. 1
Customer Statement of Requirements	Pg. 3
Glossary of Terms	Pg. 4
System Requirements	Pg. 5
Gantt Chart	Pg. 6
Stake Holders / Actors & Goals / UC's	Pg. 7
Traceability Matrix / UC Fully Dressed Descr.	Pg. 8
System Sequence Diagram	Pg. 11
Activity Diagram	Pg. 12
Architecture Style / Global Control Flow	Pg. 13
Interaction Diagram	Pg. 14
Domain Model / Concept Definition	Pg. 15
Association & Attribute Definition	Pg. 16
System Operation Contracts	Pg. 17
Data Structure & Algorithms	Pg. 24
User Interface Implementation	Pg. 25
Design of Test	Pg. 26
Integration Test	Pg. 27
History of Work	Pg. 28
Accomplishments	Pg. 29
Effort Estimation using Use Case Points	Pg. 30
Object Constraint Language	Pg.32

## **Part 1 Statement of Work and Requirements**

### **Customer Statement of Requirements**

In the contemporary gaming environment, numerous video games place a high emphasis on cutting-edge graphics, frequently utilizing 2D or 3D visuals to craft captivating and visually impressive universes. While these games unquestionably provide a wide array of experiences, they also bring about a series of socio-technological hurdles that may hinder inclusiveness and accessibility within the gaming community.

This project attempts to move away from the high graphics model and provide story based and engaging content. This allows for easier use of assisting tools such as screen readers. This additionally permits more lenient hardware requirements.

The application aims to have a fun and creative gaming experience that appeals to a large audience. The application will be able to challenge decision making skills while navigating through a dangerous new world.

The user will be able to access the menu to start game, save game, load game, and quit the game.

The user interface will display the menu as well as the main functional portion of the application.

In summary, the central socio-technological issue addressed by this project pertains to the exclusionary aspects found in contemporary gaming, stemming from demanding system specifications and insufficient accessibility for those with disabilities. Our mission is to advocate for inclusivity, aiming to enhance accessibility and enjoyment in gaming for a wider audience, thereby tackling the critical socio-technological challenges within the gaming industry.

### **Glossary of Terms**

*User*- a person who intends to use the application for entertainment or educational purposes.

*User interface*- the means by which the user and a computer system interact, in particular the use of input devices and software.

## System Requirements

### Functional Requirements:

Identifier	PW	Requirement
REQ1	5	The system's interface shall allow the user to start game, save, load, and exit game.
REQ2	5	The systems interface shall allow for text commands to control their progress through the game.
REQ3	4	The system will allow for shortcuts for common commands.

### Non-Functional Requirements:

Identifier	PW	Requirement
REQ4	3	The system shall provide a non-cluttered, user friendly, easy to understand interface.
REQ5	3	The system shall have preferences for font and display styles.

### On-Screen Appearance Requirements:

Identifier	PW	Requirement
REQ6	5	The system shall use a display containing the menu and the functional portion of the game application.

**Gantt Chart**

	<b>Start Date</b>	<b>End Date</b>	<b>Status</b>
<b>Text Adventure Game</b>	08-28-2023	12-04-2023	Incomplete
<b>Design Discussion</b>	09-01-2023	09-04-2023	Complete
<b>Initial Design</b>	09-04-2023	09-11-2023	Complete
<b>Detailed Design</b>	09-11-2023	09-18-2023	Incomplete
<b>Analysis of Implementable Tools</b>	09-18-2023	10-08-2023	Incomplete
<b>Perform System Testing</b>	10-08-2023	10-09-2023	Incomplete
<b>Document Bugs Found</b>	10-09-2023	10-10-2023	Incomplete
<b>Debug</b>	10-10-2023	10-29-2023	Incomplete
<b>First Demo</b>	10-29-2023	10-30-2023	Incomplete
<b>Review Feedback</b>	10-31-2023	11-01-2023	Incomplete
<b>Redesign</b>	11-01-2023	11-13-2023	Incomplete
<b>Develop &amp; Integrate</b>	11-13-2023	11-15-2023	Incomplete
<b>System Testing</b>	11-15-2023	11-20-2023	Incomplete
<b>Document Bugs Found</b>	11-20-2023	11-23-2023	Incomplete
<b>Debug</b>	11-23-2023	11-25-2023	Incomplete
<b>Deployment</b>	11-25-2023	11-26-2023	Incomplete
<b>Second Demo</b>	11-26-2023	12-04-2023	Incomplete

## Stakeholders

Our primary stakeholders are users who will be directly using the system for their entertainment experience.

### Actors and Goals:

Actors	Goals	Use Cases
Player/User	Immerse themselves in the game world, explore and progress through the storyline.	UC1, UC2, UC3, UC4, UC5
NPC	Interact with the player and advance storyline through player choices.	UC3
Save/Load	To allow players to save or load their progress.	UC2
Enemies	To challenge the player through combat encounters.	UC4

### Use Cases:

**UC1: Start Game** – Player starts a new game.

**UC2: Save/Load** – Player saves their game progress or loads a previous saved game.

**UC3: NPC Interaction** – Player engages in conversation with an NPC.

**UC4: Combat Encounter** – Player engages in combat with one or more enemies.

**UC5: Game Over** – The game ends and the player must start over or quit.

### Traceability Matrix

Requirements	PW	UC1	UC2	UC3	UC4	UC5
REQ1	5	x	x			
REQ2	5		x	x	x	x
REQ3	4		x			x
REQ4	3	x	x	x	x	x
REQ5	3					
REQ6	5	x	x			
Max PW	5					
Total PW		13	22	8	8	12

### Fully Dressed Description

#### Use Case UC1:

**Related Requirements:** REQ1, REQ 4, REQ 6.

**Actor's Goal:** Immerse themselves in the game world, explore and progress through the storyline.

**Initiating Actor:** The Player.

**Participating Actor:** NPC, Enemies, and the Scoring System.

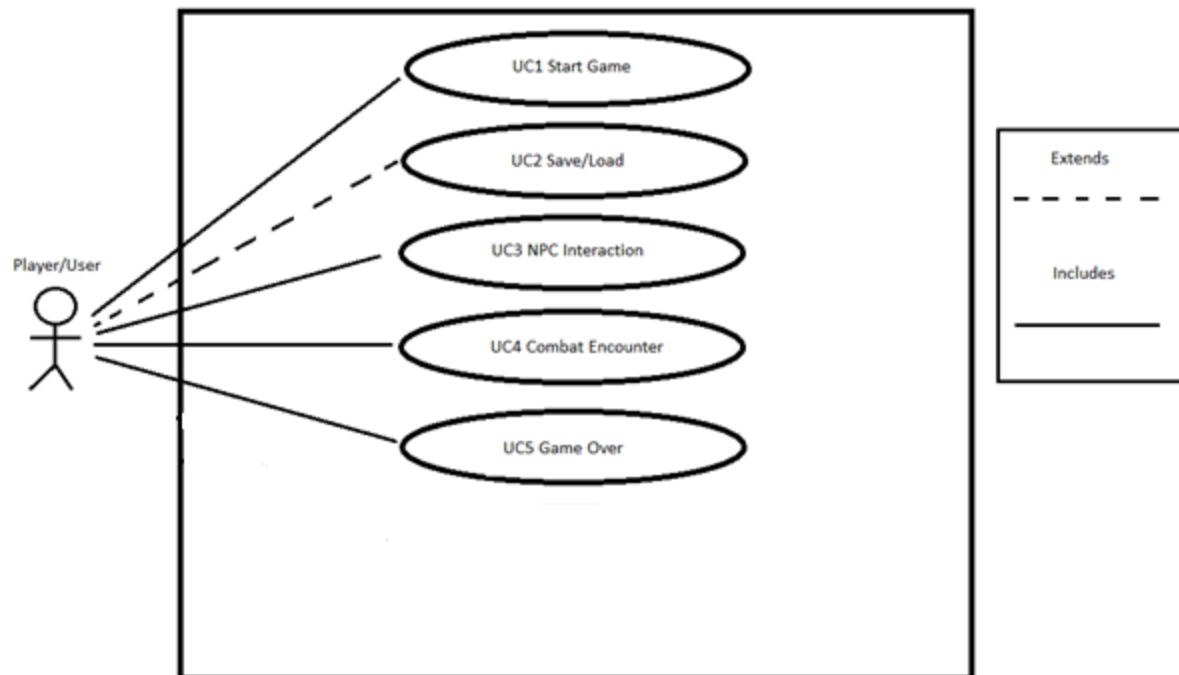
**Preconditions:** The player starts the application.

**Postconditions:** The player can save the game and quit.

#### **Flow of Events for Main Success Scenario:**

- ➔ 1. System displays the game's menu.
- ➔ 2. Player selects start or load game.
- ➔ 3. System starts the game and displays GUI.
- ➔ 4. Player progresses through game until they save, quit, or there is a game over.
- ➔ 5. System displays main menu.





### **Use Case UC3:**

**Related Requirements:** REQ2, REQ4.

**Actor's Goal:** Interact with the player and advance storyline through player choices.

**Initiating Actor:** The Player.

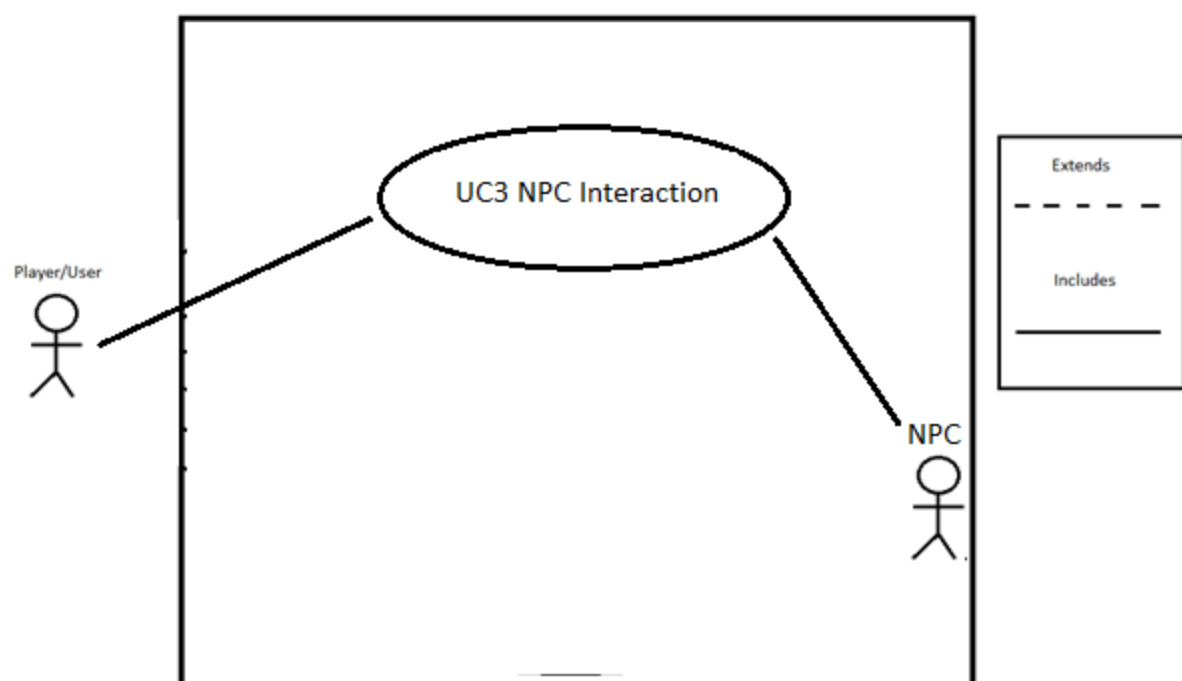
**Participating Actor:** NPC and Player.

**Preconditions:** The Player approaches NPC.

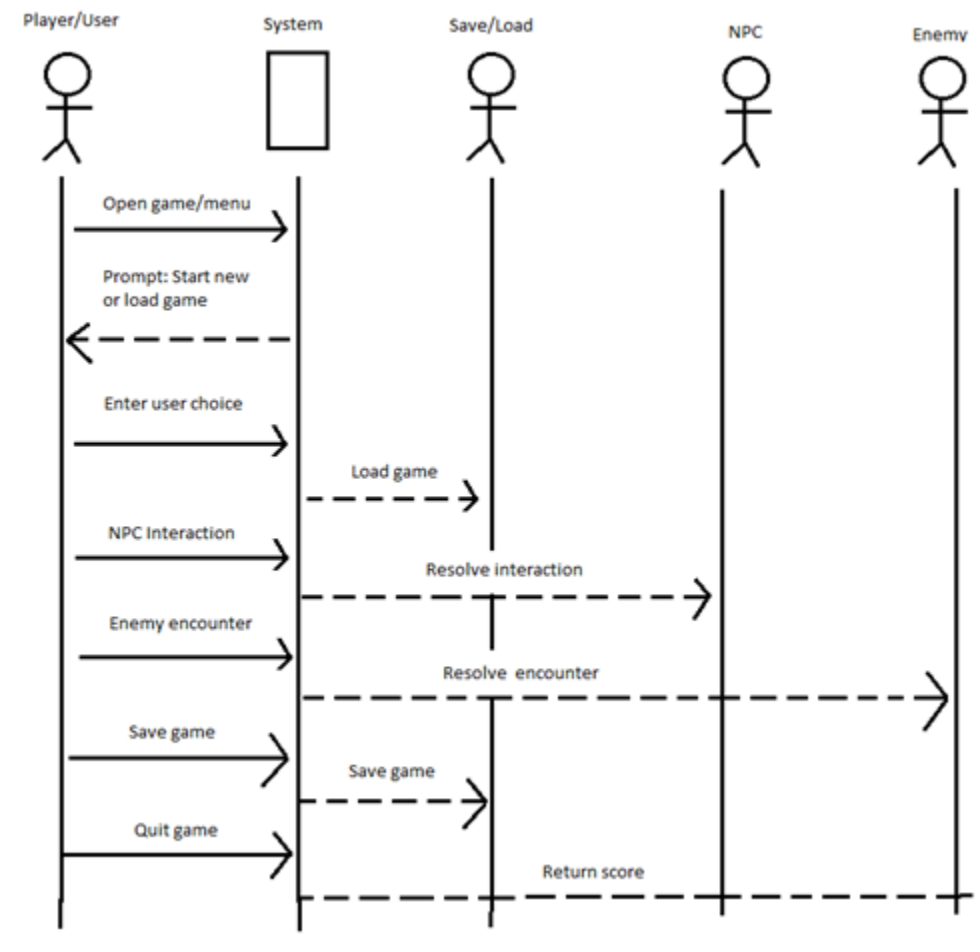
**Postconditions:** The Player gains information, an item, or completes a quest.

#### **Flow of Events for Main Success Scenario:**

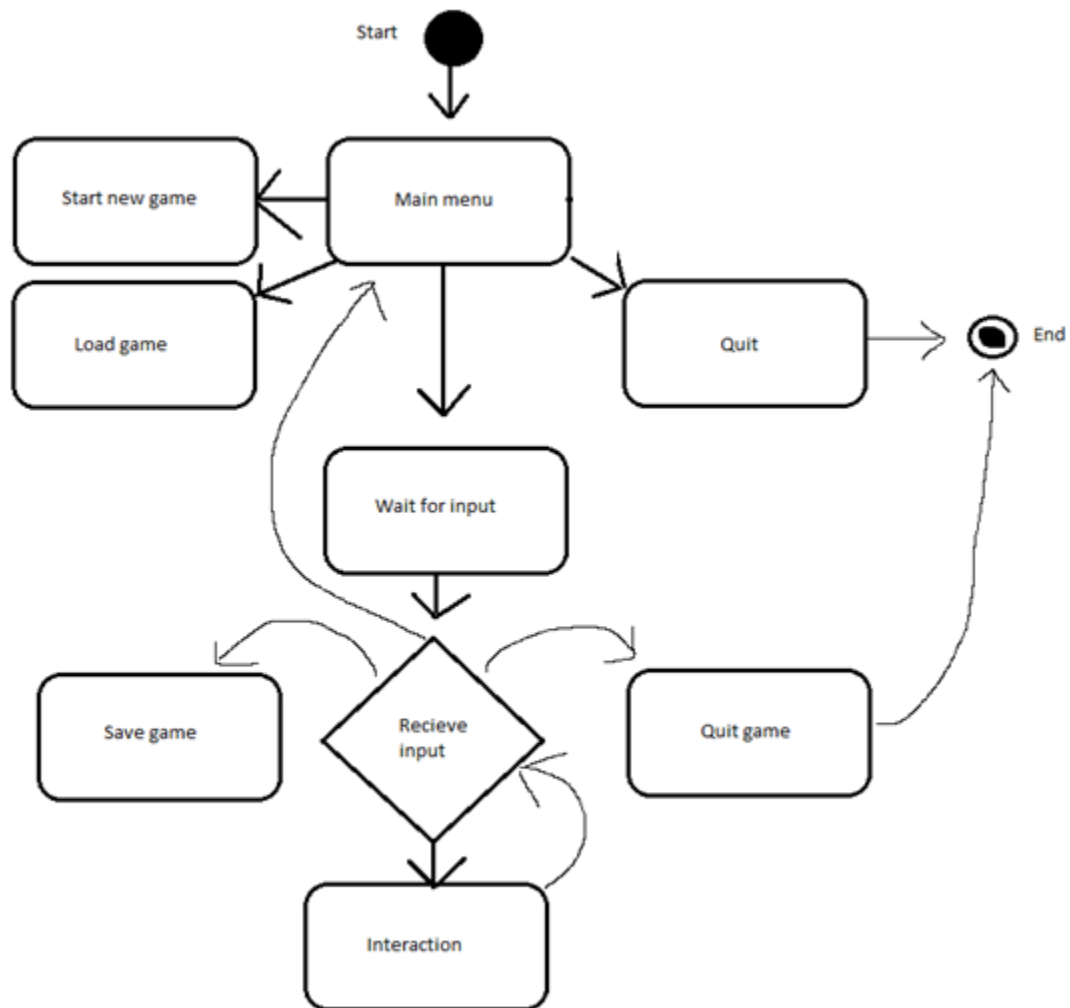
- ➔ 1. Player initiates interaction with NPC.
- ➔ 2. NPC provides varying options for the Player.
- ➔ 3. The Player chooses their option.
- ➔ 4. The NPC provides an item, quest, story completion, or information.
- ➔ 5. The player leaves the NPC.



### System Sequence Diagram



### Activity Diagram



### **Architecture Style**

The text-based adventure game follows a software architectural style known as Entity Component system (ECS). ECS is typically used in game development. This allows for efficient management of complex interactions and dynamic play. In this pattern there are entities, representing objects, and components representing things such as inventory, health, and location. The game system processes entities with specific components. This allows for modularity and scalability, allowing for expansion of the narrative.

### **Global Control Flow**

#### **Time Dependency:**

The game does not rely on real-time constraints. It is not a real time system. Instead, it operates in a turn-based system. Where time doesn't play a role. This allows players to make decisions and progress through the game at their own pace.

#### **Execution Order:**

The text-based adventure game operates as an event-driven system, reacting to actions initiated by the user, including selecting dialogue options, making in-game decisions, and initiating interactions with non-playable characters (NPCs). The game's underlying logic relies on these events, shaping the narrative in response to the player's choices. The game does not adhere to real-time constraints or follow periodic actions; instead, it dynamically adjusts to the player's decisions, advancing the storyline accordingly.

#### **Hardware Requirements:**

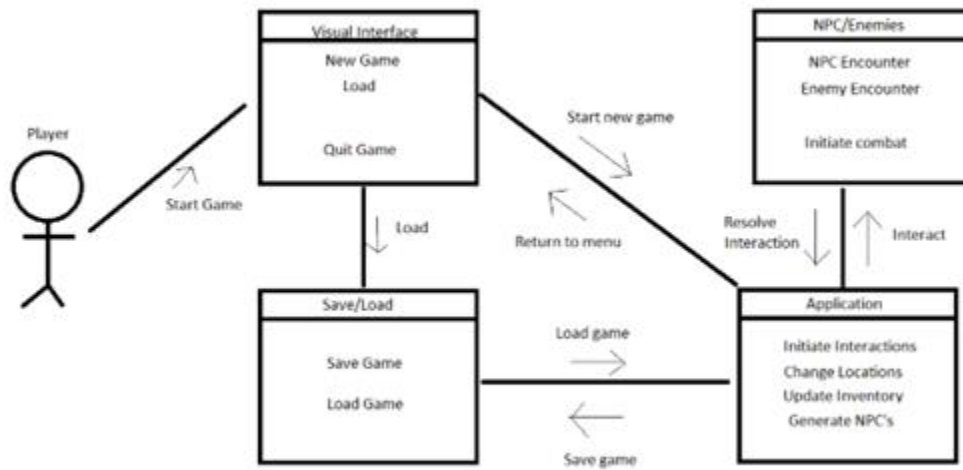
Memory: 8GB

Graphics card: minimum of GeForce 700 for Intel or Radeon 400 series for AMD

Monitor: basic LED monitor

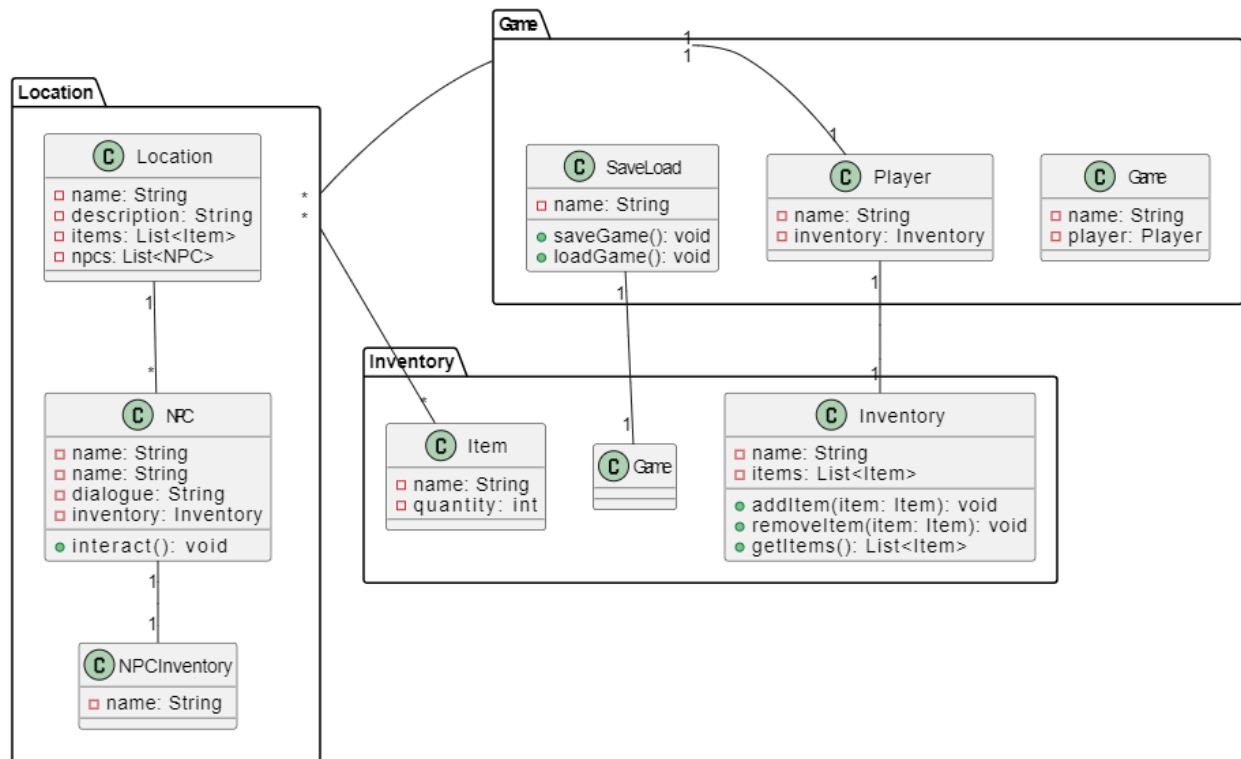
Hard drive: at least 1GB of space

## Interaction Diagram



By looking at the use cases and requirements of the application, we were able to create our domain model. Internal concepts include the NPC's and the application. When the user opens the application, they may start a new or load an existing game. The user has direct access to the application at any time.

### Domain Model



### Concept Definitions

Name	Definition	Category
Game	The central entity that represents the game itself. It contains information about the game state, progress, and overall management.	Knowing (K)
Player	The entity representing the game's player character. It stores information about the player's progress, inventory, and interactions.	Doing (D)
SaveLoad	Functionality responsible for saving	Doing (D)

	and loading the game's state, allowing players to resume their progress.	
Location	Represents different places or scenes within the game world where player actions and interactions occur.	Doing (D)
NPC	Non-Player Characters within the game world with whom players can interact.	Doing (D)

#### **Association Definitions**

<b>Concept Pair</b>	<b>Association Description</b>	<b>Association Name</b>
Game-Player	Tracks players progression and actions, provides environment	Tracking progression
Game-Save/Load	Save and loads game states	Saving and loading
Location-NPC	Stores NPC, ensures interactions happen at right locations and points in the game	Updating NPC location

#### **Attribute Definitions**

<b>Concept</b>	<b>Attribute</b>	<b>Attribute Description</b>
Game	Name	Name of the player
	Player	The user
Player	Name	Name of the player
	Inventory	Items in player possession
Save/Load	Name	Name of the player
NPC	Name	Name of the NPC
	Dialogue	Output of NPC interaction
Location	Name	Name of current location
	Description	Describes the location for player



### **System Operation Contracts**

**Operation Name:** StartNewGame

**Description:** The system allows the player to initiate a new game session, resetting game progress and initializing game resources.

**Preconditions:**

The game application is running.

Player chooses to start a new game session.

**Postconditions:**

Game progress is reset.

Game resources are initialized.

**Cross-Reference to Use Case:** UC1: Start Game

**System Operation Contract:** Save/Load Game

**Operation Name:** SaveGame

**Description:** The system allows the player to save their current game progress for future retrieval.

**Preconditions:**

The game application is running.

Player chooses to save their game progress.

**Postconditions:**

Current game progress is saved for future retrieval.

**Cross-Reference to Use Case:** UC2: Save/Load

**System Operation Contract:** Save/Load Game (Load)

**Operation Name:** LoadGame

**Description:** The system allows the player to load a previously saved game session.

**Preconditions:**

The game application is running.

Player chooses to load a saved game session.

**Postconditions:**

The game state is restored to the state of the previously saved game.

**Cross-Reference to Use Case:** UC2: Save/Load

**System Operation Contract:** NPC Interaction

**Operation Name:** InteractWithNPC

**Description:** The system allows the player to engage in conversation or interaction with a non-player character (NPC) within the game world.

**Preconditions:**

The game application is running.

Player initiates interaction with an NPC.

**Postconditions:**

The conversation or interaction with the NPC proceeds as per the game's logic.

**Cross-Reference to Use Case:** UC3: NPC Interaction

**Traceability Matrix**

	Visual Interface	NPC/Enemy	Save/Load	Application
UC1	x		x	x
UC2	x		x	x
UC3	x	x		x

Our data model has been thoughtfully designed with user convenience in mind. It takes the form of a JSON model, chosen for its simplicity, making it easy to read and edit. This intuitive design empowers users to seamlessly pick up their game right where they left off.

Within this model, we focus on persisting key elements that enhance the gaming experience. These include the user's current location, essential statistics, and their inventory. These vital aspects are effortlessly captured in our flat file format.

What sets our approach apart is the absence of any cumbersome databases, streamlining the user experience. In this project, we prioritize user-friendly functionality over complex mathematical models, ensuring a smooth and enjoyable gaming journey.

### **Interaction Diagram**

#### **Use Case UC1:**

**Related Requirements:** REQ1, REQ 4, REQ 6.

**Actor's Goal:** Immerse themselves in the game world, explore and progress through the storyline.

**Initiating Actor:** The Player.

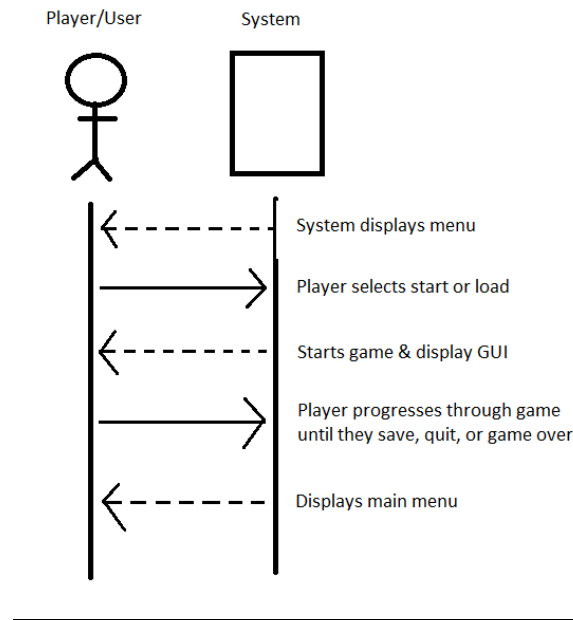
**Participating Actor:** NPC, Enemies, and the Scoring System.

**Preconditions:** The player starts the application.

**Postconditions:** The player can save the game and quit.

#### **Flow of Events for Main Success Scenario:**

- ➔ 1. System displays the game's menu.
- ➔ 2. Player selects start or load game.
- ➔ 3. System starts the game and displays GUI.
- ➔ 4. Player progresses through game until they save, quit, or there is a game over.
- ➔ 5. System displays main menu.



We have chosen to entrust the pivotal responsibility of orchestrating the presentation of the main menu and graphical user interface (GUI) to the capable hands of the system itself. Within this design framework, the system bears the pivotal role of facilitating the player's interaction with an array of essential options, encompassing actions such as save/load and the commencement of a new game. This strategic allocation of responsibilities ensures a seamless and user-friendly gaming experience, allowing the player to navigate effortlessly through the diverse facets of our gaming environment.

### **Use Case UC3:**

**Related Requirements:** REQ2, REQ4.

**Actor's Goal:** Interact with the player and advance storyline through player choices.

**Initiating Actor:** The Player.

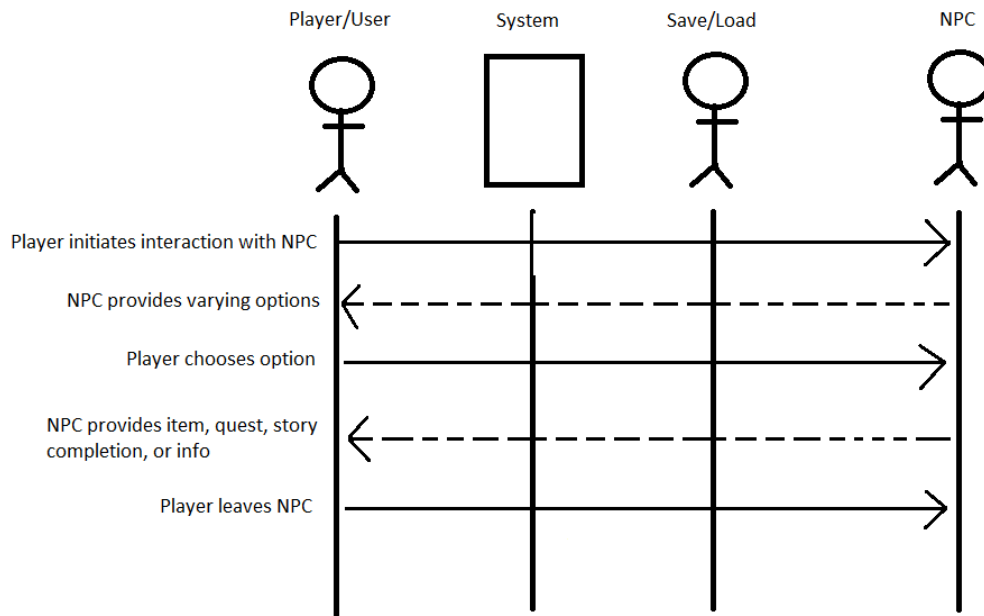
**Participating Actor:** NPC and Player.

**Preconditions:** The Player approaches NPC.

**Postconditions:** The Player gains information, an item, or completes a quest.

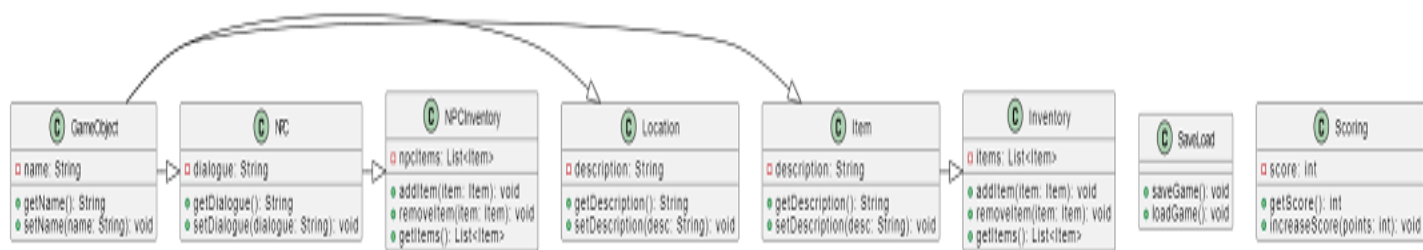
### **Flow of Events for Main Success Scenario:**

- ➔ 1. Player initiates interaction with NPC.
- ➔ 2. NPC provides varying options for the Player.
- ➔ 3. The Player chooses their option.
- ➔ 4. The NPC provides an item, quest, story completion, or information.
- ➔ 5. The player leaves the NPC.



The responsibility for providing the options to the player, including information, quests, items, and story completion is assigned to the NPC. The player will interact with the NPC and choose the available options.

### Class Diagram



### Data Types and Operation Signatures:

Class GameObject:

Purpose: This class represents the base object for all game objects.

Attributes:

name: String - The name of the game object.

Class Location:

Purpose: This class represents a location within the game.

Attributes:

description: String - A description of the location.

Class NPC:

Purpose: This class represents a Non-Player Character (NPC) within the game.

Attributes:

dialogue: String - The dialogue or text associated with the NPC.

Operations:

getDialogue(): String - Retrieves the dialogue of the NPC.

setDialogue(dialogue: String): void - Sets or updates the dialogue of the NPC.

**Traceability Matrix**

Requirements	GameObject	Location	NPC
Start game	X		
Save game	X		
Load game	X		
NPC Interaction	X	X	X
Combat Encounter	X	X	X
Game Over	X		

## **Data Structures and Algorithms**

### **Algorithms**

Our project will not require any intricate algorithms.

### **Concurrency**

Our system will not use multiple threads at this time.

### **Data Structures**

Our system we will employ various data structures, including arrays and lists, to manage player inventory and monster encounters. We will use lists to model the players' inventory to allow for the change in size as the users obtain more items. A list is dynamic and allows for storing and managing a collection of elements. The list can grow and shrink in size based upon need, which allows for management of variable sized data such as inventories. For monster encounters, we will use arrays to select the monster using a random number to associate with the index, since the number of monsters in the system will not be dynamic but fixed this will serve our purposes. Arrays allow storing of multiple elements of the same data type in a single block of memory that is contiguous. This provides an organized way to access data in a systematic way.



## User Interface and Implementation

Overall, our UI design has changed a small bit since the first design. We are no longer implementing a settings menu, as the changing of background color, and font size/color is not possible within the current system.

```
==== Text Adventure Main Menu ====  
1.New Game  
2.Load Game  
3.Quit Game  
Please enter your choice:
```

### Main Menu

```
==== Penultimate Adventure ====  
Current Location: Location 1  
Your current score: 0  
  
You have awakened in your bed. You remember the electricity going out last night, and it looked like it was all across the city. As you awaken a screen enters your view and asks you to select a class...
```

### Starting Location

## Design of Tests

For this demonstration, we want to test out multiple cases including new game, inventory management, location transition, and input validation. These are very important functions for our application. We want to complete these functions before working on other aspects as the system will not function without these use cases.

### Unit tests

#### **New Game Test:**

Description: To verify that starting a new game initialized the game with the correct state.

Test input: Start a new game.

Expected output: The game is initialized with the default character attributes, inventory and location. The score is reset to zero and the game begins at the starting location.

#### **Location Transition Test:**

Description: Test if the player can move between locations within the game world.

Test Input: Attempt to move the character from one location to another.

Expected Output: The character's location is updated.

## **Integration Testing**

We will be incorporating a combination of approaches, instead of a top down or bottom up. This will incorporate elements of both, and additional strategies to ensure comprehensive testing. Component interaction: This will involve testing how specific components interact with each other, such as inventory management and location transition as well as input validation.

Error Handling and input validation: This part tests the integration between the error handling and input validation. This will examine how these components interact and respond to valid inputs.

Performance and Stress Testing: Evaluate the game's performance as a whole. This includes testing how the game behaves when the player explores various locations, interacts with numerous characters, accumulates items, and plays for extended periods.

Game Logic and Item Interaction: Verify that the game's logic, such as combat or puzzle-solving, integrates smoothly with item interactions. For instance, using a specific item should affect game mechanics appropriately.

### History of Work

	Start Date	End Date	Status
<b>Text Adventure Game</b>	08-28-2023	12-04-2023	Incomplete
<b>Design Discussion</b>	09-01-2023	09-04-2023	Complete
<b>Initial Design</b>	09-04-2023	09-11-2023	Complete
<b>Detailed Design</b>	09-11-2023	09-18-2023	Complete
<b>Analysis of Implementable Tools</b>	09-18-2023	10-08-2023	Complete
<b>Perform System Testing</b>	10-08-2023	10-09-2023	Complete
<b>Document Bugs Found</b>	10-09-2023	10-10-2023	Complete
<b>Debug</b>	10-10-2023	10-29-2023	Complete
<b>First Demo</b>	10-29-2023	10-30-2023	Complete
<b>Review Feedback</b>	10-31-2023	11-01-2023	Complete
<b>Redesign</b>	11-01-2023	11-13-2023	Complete
<b>Develop &amp; Integrate</b>	11-13-2023	11-15-2023	Complete
<b>System Testing</b>	11-15-2023	11-20-2023	Complete
<b>Document Bugs Found</b>	11-20-2023	11-23-2023	Complete
<b>Debug</b>	11-23-2023	11-25-2023	Complete
<b>Deployment</b>	11-25-2023	11-26-2023	Complete
<b>Second Demo</b>	11-26-2023	12-04-2023	Incomplete

### Accomplishments

- **Conceptualization and Storyboarding:** Creating the game's concept, storyline, and initial storyboard.
- **Game Design and Mechanics:** Developing the game's mechanics, including the rules, choices, and how the story unfolds based on player decisions.
- **Writing and Dialogue:** Crafting engaging and immersive text-based narratives, dialogues, and descriptions for the game's world and characters.
- **Coding and Programming:** Implementing the game's functionality and interactivity, ensuring that player choices affect the storyline and create a dynamic experience.
- **User Interface Development:** Designing a user-friendly and intuitive interface for players to interact with the game.
- **Testing and Bug Fixing:** Running playtests, identifying bugs, and refining the game based on user feedback to improve the overall experience.

### Effort Estimation Using Use Case Points

UC1: Start Game

Complexity: Average

Number of Participating Actors: 1 (Player/User)

UCP = Complexity Weight 10 x Number of Participating Actors (1) = 10

UC2: Save/Load

Complexity: Average

Number of Participating Actors: 1 (Player/User)

UCP = Complexity Weight 10 for x Number of Participating Actors (1) = 10

UC3: NPC Interaction

Complexity: Average

Number of Participating Actors: 2 (Player/User and NPC)

UCP = Complexity Weight 10 ) x Number of Participating Actors (2) = 20

UC4: Combat Encounter

Complexity: Average

Number of Participating Actors: 2 (Player/User and Enemies)

UCP = Complexity Weight 10 x Number of Participating Actors (2) = 20

UC5: Game Over

Complexity: Average

Number of Participating Actors: 1 (Player/User)

UCP = Complexity Weight 10 x Number of Participating Actors (1) = 10

Total UCP = UCP for UC1 + UCP for UC2 + UCP for UC3 + UCP for UC4 + UCP for UC5

Total UCP = 10 + 10 + 20 + 20 + 10

Total UCP = 70

1. UAW:  $1(UC1) + 1(UC2) + 2(UC3) + 2(UC4) + 1(UC5) = 7$

2. UCW = 5

3. Distributed Systems (1.5)

Performance Requirements (1.5)

Complexity of Processing (1.5)

Reusability (1.5)

Ease of Installation (1.5)

$$\text{TCF} = 0.6 + (0.01 * (1.5 + 1.5 + 1.5 + 1.5 + 1.5)) = 0.6 + (0.01 * 7.5) = 0.6 + 0.075 = 0.675$$

4. E1. Familiarity with the Development Process (1.5)

E2. Object Oriented Experience (0.5)

E3. Motivation (1.0)

E4. IDE Complexity (-1.0)

$$\text{ECF} = 1.4 + (-0.03 * (1.5 + 0.5 + 1 + -1)) = 1 + (-0.03 * 2) = -0.06 + 1.4 = 1.34$$

5. UCP = 70

6. AUCP =  $(7 + 5) * (1.34 * .675) = 12 * .903 = 10.836$

## Object Constraint Language

### **Game class**

#### *Invariants*

The game must have a unique identifier.

The game must have a start time and an end time.

#### *Preconditions*

Creating a game requires specifying a unique identifier.

#### *Postconditions*

After the game is created, the start time and end time should be set.

### **GM class**

#### *Invariants*

A menu should have at least one option.

Each option on the menu must have a unique label.

#### *Preconditions*

When creating a menu, at least one option must be included.

#### *Postconditions*

After creating a menu, all options must have unique labels.

### **NPC class**

#### *Invariants*

An NPC must have a name and a valid room location.

#### *Preconditions*

Creating an NPC requires providing a name and a location.

#### *Postconditions*

After creating an NPC, the name and room should be assigned.

### **Monster class**

#### *Invariants*

A monster must have a name.

#### *Preconditions*

Creating a monster requires specifying a name.



*Postconditions*

After creating a monster, the name should be set.

**Room class***Invariants*

A room should have a unique identifier.

*Preconditions*

Creating a room requires setting a unique identifier.

*Postconditions*

After creating a room, the unique identifier should be set.

**-- Game class --**

Class Game

*Invariants*

self.uniqueIdentifier <> null  
self.startTime <> null and self.endTime <> null

*Preconditions*

Operation createGame(uniqueIdentifier: String)

*Body:*

self.uniqueIdentifier = uniqueIdentifier

*Postconditions*

Operation createGame

*Body:*

self.startTime <> null and self.endTime <> null

**-- GM class --**

Class GM

*Invariants*

self.menu <> null  
self.menu.options->notEmpty()  
self.menu.options->forAll(option | option.label <> null)

*Preconditions*

Operation createMenu(options: Set<Option>)

*Body:*

self.menu = new Menu(options)

Postconditions

Operation createMenu

Body:

self.menu.options->notEmpty()

self.menu.options->forAll(option | option.label <> null and self.menu.options->count(option.label) = 1)

**-- NPC class --**

Class NPC

Invariants

self.name <> null

self.location <> null

Preconditions

Operation createNPC(name: String, location: Room)

Body:

self.name = name

self.location = location

Postconditions

Operation createNPC

Body:

self.name <> null and self.location <> null

**-- Monster class --**

Class Monster

Invariants

self.name <> null

Preconditions

Operation createMonster(name: String)

Body:

self.name = name

Postconditions

Operation createMonster

Body:

self.name <> null

**-- Room class --**

Class Room

#### Invariants

self.uniqueIdentifier <> null

#### Preconditions

Operation createRoom(uniqueIdentifier: String)

Body:

self.uniqueIdentifier = uniqueIdentifier

#### Postconditions

Operation createRoom

Body:

self.uniqueIdentifier <> null