Analysis

Description of Problem

The goal of this project is to develop a simple text-based RPG game using Python with a focus on the core system and stage progression. The program has used some utilising fundamental programming concept such as 2D array, sorting algorithms binary search and database (SQL). The database will stores the game data that save from the player and also will store the item, buffs and stats of enemy.

The game will involve that the player advancing through t multiple stage, containing a series of battles that's randomly generated enemies. The main objective that the player should do in the game is to increase their stat that because of the players damage or survive abilities is rely on the base stats of the player. And also the difficulty of the enemy is increase through stages.

The player will also able to track their progress and return it by saving their game data into a database.

Scope

The scope of my project will include:

1. A complete design with a pseudocode, data dictionary, algorithm, and diagrams to demonstrate the core gameplay elements of the game, including the battle system, stage progression.
2. A functional game with a well made battle system that can let the player to fight enemies with random stats. For managing the enemy and player stat I will use 2D array to manage it.
3. The integration of a binary search and sorting algorithm will be used to manage the inventory and buff/debuffs that's on the player.
4. A simple text-based interface to display the battle or menu.
5. A full implementation of save/ load game feature using SQL to store the data of the player.

Constraints

There are numbers of technical, economic and time constraints.

1. I will use Thonny and Virtual Studio Code software to program Python to create this game because of having few years of experience on this language
2. The final version of the game will be run on window system
3. Microsoft Access will be used to store all the data in the game
4. There will be no cost needed in this project. It's all developed by myself, also Thonny, Access and Virtual Studio Code is free software to program with programming language

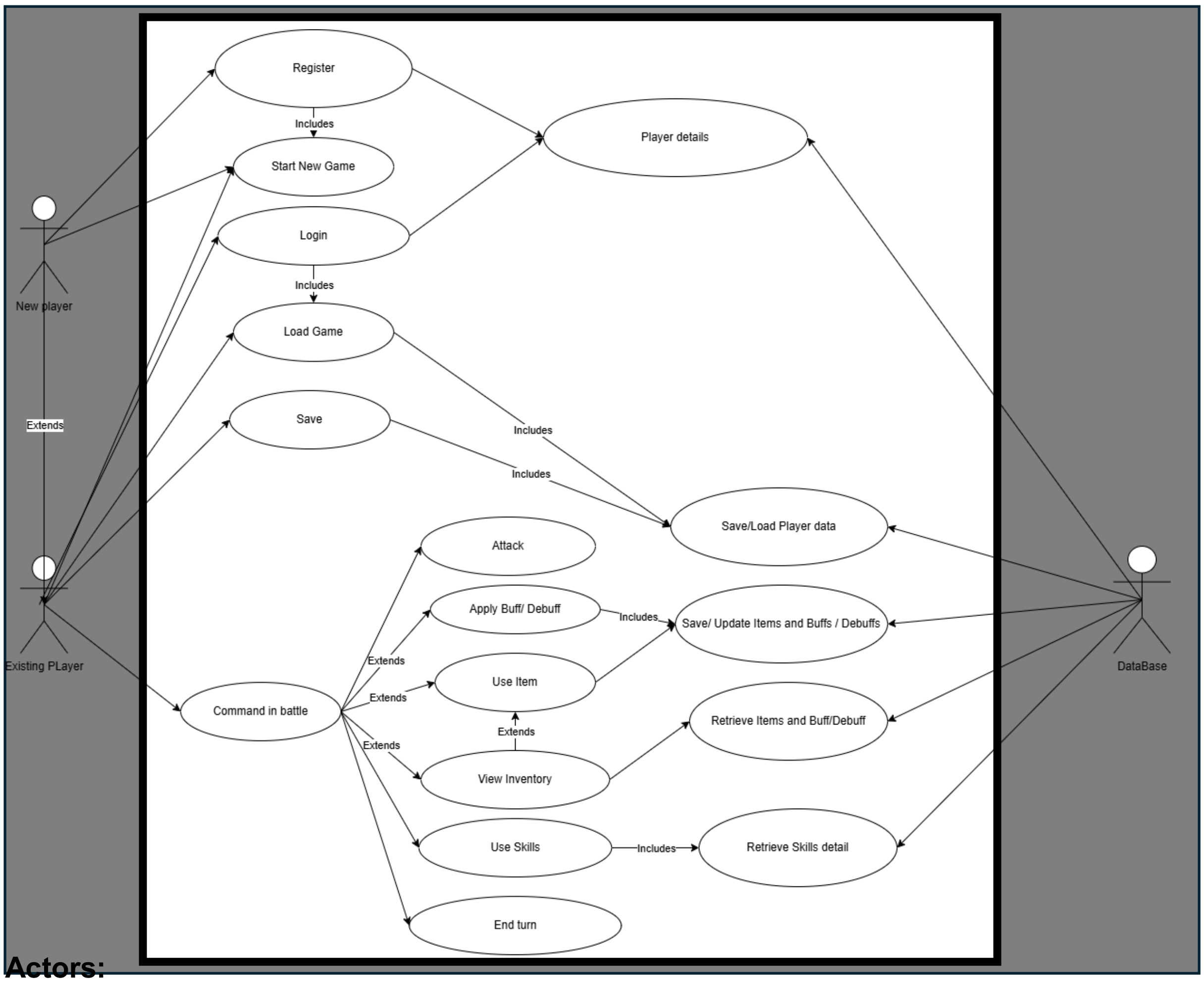
5. The time constraints of this project is to complete by the deadline of April as it needed to be submit to SQA marking.

Boundaries

What the project will be expected to work with :

1. Text-based Interface - The game will not feature any graphical interface or advanced visual components. All the interface will be showed through text.
2. Single-Player Only – The game will be designed strictly for a single player. No any multiplayer or online co-op function will be included.
3. Simple Databased usage – the SQL database will be used to saving and loading player's data. No complex queries or multiple tables will be involved beyond the basic store data and save/load function.
4. Sorting and Searching Constrains – The sorting algorithm will be only applied to inventory management and player stats. These algorithms wouldn't used in the battle system or other system that's not in the management function.
5. Limited save/load function – The game will just have the basic function, there would have any autosave, cloud save or other save function.

UML Use Case Diagram



Actors:

- Existing Player: Represents the user interacting with the game. The player can initiate actions such as starting a new game, continuing a saved game, or interacting with the game's combat and inventory systems.
- New player : A user who has not registered in the game before. Their primary actions include registering and starting a new game.
- Database : A system actor that stores and manages all game-related data, such as player profiles, game states, item details, buffs, debuffs, and skills.

Use Cases:

- Start New Game : Allows the player to start a new game session. It initializes player stats and game stage data.
- Register : The new player creates an account by providing necessary information (e.g., username, password).
- Login : The existing player logs into their account using their credentials (username and password).
- Save Game: Saves the current game state, including player stats and stage progress, to the database for future retrieval.
- Command in battle: The existing player issues commands during battles, such as attacking or using items.
- Load Game: Loads previously saved game data from the database.
- Attack : The existing player attacks an enemy during a battle.
- View Inventory: The existing player checks their inventory for items they possess.
- Use Item: The existing player uses an item from their inventory during battle.
- Use Skills : The existing player uses a skill during battle.
- End Turn : The existing player ends their turn during a battle, allowing the enemy to take their turn.
- Apply Buff/Debuff: The existing player applies buffs or debuffs to themselves or their enemies during battle.
- Save/Load Player Data : The system saves or loads player data as needed for new or existing players.
- Retrieve Items and Buff/Debuff : The database retrieves relevant item and buff/debuff data for the player.
- Save/Update Items and Buffs/Debuffs : The database saves or updates information about the items and buffs/debuffs applied to the player.
- Retrieve Skills Detail : The database provides details about the player's skills when they are used in battle.

Relationship

- **New Player → Register (Extends Existing Player)**
    - **Extends**: A new player must first register before they become an existing player. Once registered, they become part of the existing player group.
- **New Player → Start New Game**
    - **Includes**: After registration, the new player has the option to start a new game. It involves creating a new player profile (part of **Player Details**).
- **Existing Player → Login (Includes Player Details)**
    - **Includes**: The player must log in to access their profile and continue the game. Logging in includes accessing saved player details, like stats, items, and progress.
- **Existing Player → Load Game**
    - **Includes**: Once logged in, the existing player can load a previously saved game. This includes loading the **Save/Load Player Data** and **Player Details**.
- **Existing Player → Save**
    - **Includes**: During the game, the player can save their progress. Saving includes storing the player's data (stats, inventory, skills) in the database.
- **Existing Player → Command in Battle**
    - **Includes**: The player engages in a turn-based battle that includes different actions like attacking, using items, and applying buffs or debuffs.
- **Command in Battle → Attack**
    - **Extends**: During the battle, the player has the option to perform an attack.
- **Command in Battle → Apply Buff/Debuff (Includes Retrieve Items and Buff/Debuff)**
    - **Extends**: Players can apply buffs or debuffs in battle, which includes retrieving relevant data about items or buffs from the database.
- **Command in Battle → Use Item (Extends View Inventory)**
    - **Extends**: The player may choose to use an item during battle. Before using an item, the player can view their inventory.
- **Command in Battle → View Inventory**
    - **Extends**: Players can view their current inventory during battle.
- **Command in Battle → Use Skills (Includes Retrieve Skills Detail)**
    - **Extends**: Players can use their special skills during battle, which includes retrieving details about the skills from the database.
- **Command in Battle → End Turn**
    - **Extends**: After performing actions like attack, use items, or apply buffs, the player ends their turn.

- **Save → Save/Load Player Data (Includes Player Details)**
  - **Includes**: Saving the game includes saving the player's data (stats, inventory, skills) and updating the player details in the database.
- **Save → Save/Update Items and Buffs/Debuffs (Includes Retrieve Items and Buff/Debuff)**
  - **Includes**: Saving also includes updating the player's inventory and active buffs/debuffs.
- **Database → Retrieve Items and Buff/Debuff**
  - **Association**: The database stores items and buffs/debuffs, which are retrieved during battles when required by the player.
- **Database → Retrieve Skills Detail**
  - **Association**: The database stores details about the player's skills, which are retrieved when the player uses skills during battle.

Requirement Specification

Functional Requirements

1. User Interface

- Display a text-based menu system for navigating through the game options.
- Provide options for starting a new game, loading a saved game, viewing player stats, and exiting the game.

2. Player Management

- Maintain player stats including HP, attack, defense, speed, and experience points.
- Track player inventory and allow item usage during battles.

3. Stage and Enemy Management

- Implement a series of stages with increasing difficulty by the main stage.
- Each stage will have a predefined set of enemies with unique stats and abilities.
- Include boss fights at certain stages for increased challenge.

4. Combat System

- Implement a turn-based combat system where the player and enemy take turns attacking.
- Include basic attack options, special skills, and item usage (e.g., healing potions).
- Display the outcome of each battle and update player stats accordingly.

5. Database and Data Management

- Use a database to store player profiles, enemy information, and stage data.
- Retrieve and update data in real time during gameplay.

6. Sorting and Searching

- Implement sorting algorithms (e.g., bubble sort, insertion sort) for organizing player inventory.
- Use binary search for quick lookup of player items and enemy data during battles.

7. Saving and Loading

- Allow players to save their progress during the game.
- Load saved game data to continue from the last saved point.
- Handle saved files securely to prevent data loss.

8. Login system

- User Registration:
  - Players can create a new account by entering a unique username and password.
  - The system will check that the username does not already exist in the database and save the username and password securely.
- User Authentication:
  - Players will log in using their username and password.
  - The system will verify the entered credentials by checking the database, and if they match, it will load the player's saved game data.

End User Requirements

- Simple and Clean Interface Design:
  - A game screen that is not cluttered, making it easy to focus on playing.
- Easy Navigation:
  - Clear and simple menus to help players find what they need quickly.
- Straightforward Battle System:

    - A combat system that is easy to learn and play, with clear instructions.
- Clear Stage Progression:
    - Easy-to-follow paths between game stages, showing players where to go next.
- User-Friendly Controls:
    - Easy-to-use controls that can be customized if needed.

Inputs and Outputs

For Players Input:

- Character Name:
    - For the player's in-game character.
- Command:
    - To control the game

In-Game Output:

- Inventory:
    - Items and equipment collected during the game.
- Character Stats:
    - Health and abilities of the character that change as they progress.
- The battle process
    - The value (stats, hp etc.) change after each turn
    - The battle result after the battle
- Item, buff and debuff information
- The command the player can input

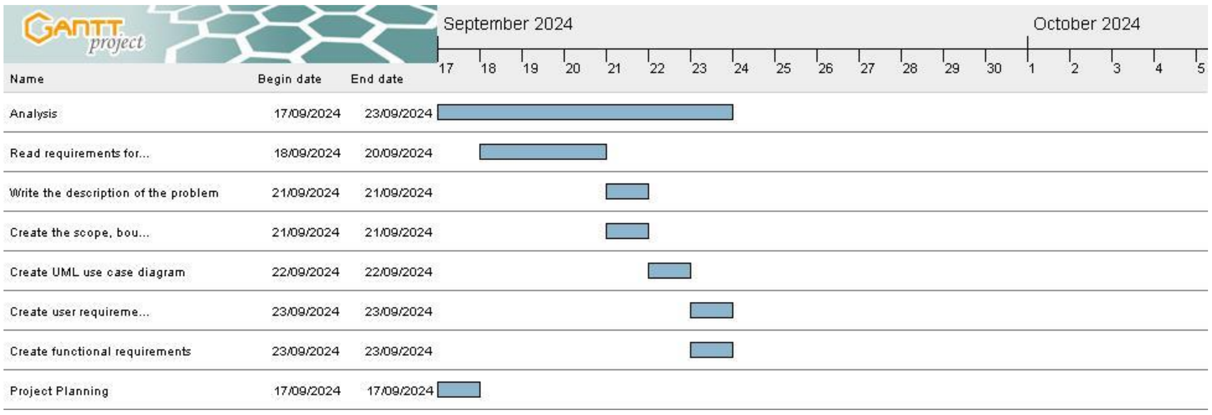
<u>Project Plan</u>

Analysis (Estimate 1 week to finish the Analysis)

- **Read requirements for the AH project and create a detail concept idea**
    - **0.5 days to read the requirement and mark notes on it**
    - **2 days to create the detail concept idea notes about the concept**
- **Write the description of the problem**
    - **0.5 days for the description**
- **Create the scope, boundaries and constraints**
    - **0.5 days for all the scope, boundaries and constraints**

- **Create UML use case diagram**
    - **0.5 days for the UML use case diagram**
- **Create user requirements description**
    - **0.5 days for the requirements description**
    - **0.5 days for the functional requirements description**
- **Project Planning**
    - **0.5 days for the Project Planning**

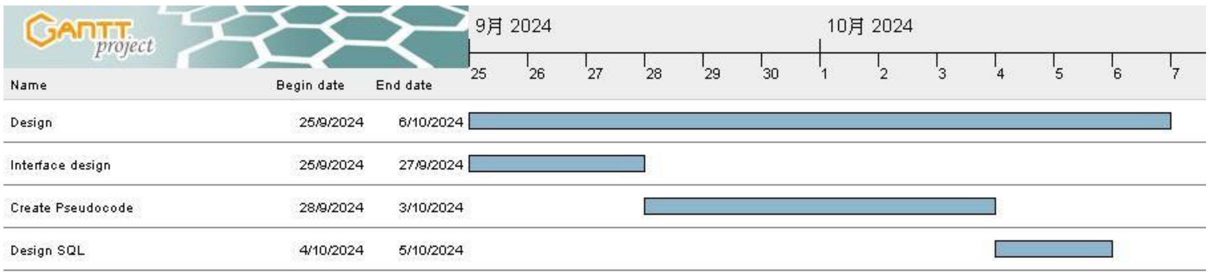
Gantt Chart :



Name	Begin date	End date
Analysis	17/09/2024	23/09/2024
Read requirements for...	18/09/2024	20/09/2024
Write the description of the problem	21/09/2024	21/09/2024
Create the scope, bou...	21/09/2024	21/09/2024
Create UML use case diagram	22/09/2024	22/09/2024
Create user requireme...	23/09/2024	23/09/2024
Create functional requirements	23/09/2024	23/09/2024
Project Planning	17/09/2024	17/09/2024

Design

- **Interface design – create the text interface of the game**
    - **3 days to create the text interface of a game**
- **for each part of the code**
    - **6 days to create the pseudocode**
- **Design SQL**
    - **2 days to design the SQL**
- **Design Structure Diagram**
    - **1 days to design structure diagram**



Name	Begin date	End date
Design	25/9/2024	6/10/2024
Interface design	25/9/2024	27/9/2024
Create Pseudocode	28/9/2024	3/10/2024
Design SQL	4/10/2024	5/10/2024

Implementation

Testing

- **Test plan**
    - **Test Plan will be create while implementing the game (program)**
- **Carrying out the testing result**
    - **Est. 2 days needed to carry out the the result**
- **Describe testing encountered problem**
    - **3 days to write the encountered problem**

Evaluations

- **Write Evaluation**
    - **5 days needed to write the evaluation of the program**

Resources Required

The resources is needed at each stage of the development are listed below.

Analysis	• Microsoft Office Word (version 2407) • Google Chrome () • Gantt Project 3.2.3240
Design	• Microsoft Office Word (version 2407) • Google Chrome • Freeform (Ipad)

Implementation	• Microsoft Office Word (version 2407) • Thonny 4.1.* • Virtual Studio Code (version 1.93) • Python 3.12 • NotePad 11.2302.16.0 • Microsoft Office Access 16.0
Final Testing	• Microsoft Office Word (version 2407) • Thonny 4.1.* • Virtual Studio Code (version 1.93) • Python 3.12 • Microsoft Office Access 16.0
Evaluation	• Microsoft Office Word (version 2407)