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 advances through multiple stages, containing a series of battles that are predesigned enemies. The main objective is 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. Also the difficulty of the enemy is increases 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 well designed stats. For managing the enemy and player stat I will use OOP & database to manage it.
3. The integration of a binary search and sorting algorithm will be used to manage the inventory and buff 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 a window system
3. Myphpadmin 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

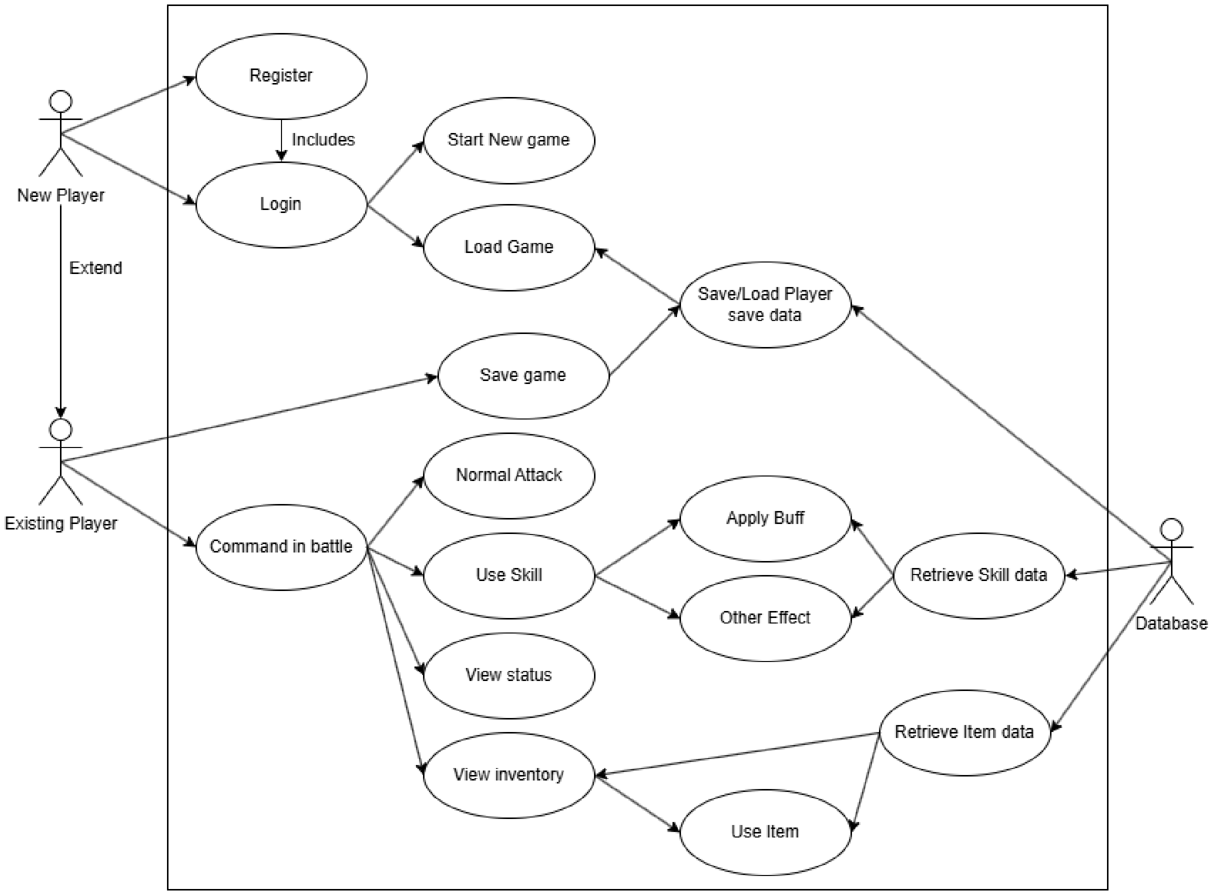
5. The time constraints of this project is to complete by the deadline of April as it needed to be submit to SQA marking.

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. Also it will be used to store data in it, for example items, enemy ect. 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. 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



Requirement Specification

Functional Requirements

1. User Interface

1.1 Display a text-based menu system for navigating through the game options.

1.2 Provide options for starting a new game, loading a saved game, viewing the instruction, and exiting the game.

2. Player Management

2.1 Maintain player stats, including HP, attack, defense, speed, and experience points.

2.2 Track player inventory and allow item usage during battles.

3. Stage and Enemy Management

3.1 Implement a series of stages with increasing difficulty by the main stage.

3.2 Each stage will have a predefined set of enemies with unique stats.

4. Combat System

4.1 Implement a turn-based combat system where the player and enemy take turns attacking based on their speed.

4.2 Include basic attack options, special skills, and item usage (e.g., healing potions).

4.3 Display the outcome of each battle and update player stats accordingly.

5. Database and Data Management

5.1 Use a database to store player profiles, enemy information, and stage data.

5.2 Retrieve data in real time during gameplay.

6. Sorting and Searching

6.1 Implement sorting algorithms (e.g., bubble sort, insertion sort) for organizing player inventory.

6.2 Use binary search for quick lookup of player items and enemy data during battles.

7. Saving and Loading

7.1 Allow players to save their progress after completing a stage.

7.2 Load saved game data to continue from the last saved point.

7.3 Handle saved files securely to prevent data loss**.**

8. Login System

8.1 User Registration:

8.1.1 Players can create a new account by entering a unique username and password.

8.1.2 The system will check that the username does not already exist in the file and save the username and password securely.

8.2 User Authentication:

8.2.1 Players will log in using their username and password.

8.2.2 The system will verify the entered credentials by checking the file, and if they match, it will load the player's saved game data.

End User Requirements

1 Simple and Clean Interface Design:

  1.1 A game screen that is not cluttered, making it easy to focus on playing.

2 Easy Navigation:

  2.1 Clear and simple menus to help players find what they need quickly.

3 Straightforward Battle System:

  3.1 A combat system that is easy to learn and play, with clear instructions.

4 Clear Stage Progression:

  4.1 Easy-to-follow paths between game stages, showing players where to go next.

5 User-Friendly Controls:

  5.1 Easy-to-use controls.

Inputs and Outputs

For Players Input:

- Character Name:
  - For the player's in-game character.
- Command:
  - To control the game by some simple one letter input.

In-Game Output:

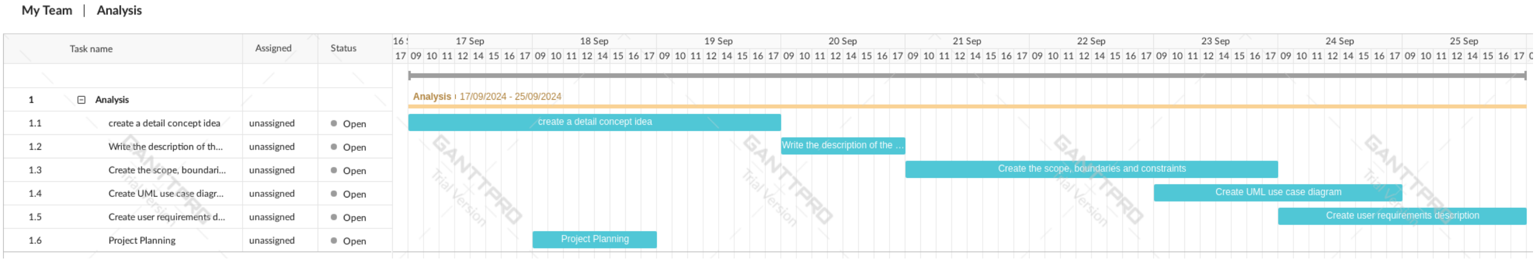
- Inventory:
  - Items and equipment collected during the game.
- Character Stats:
  - Health and abilities of the character that change as they progress.
- The battle process
  - The value (stats, hp etc.) change after each turn
  - The battle result after the battle
- Item, buff information
- The command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

Project Plan

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 :

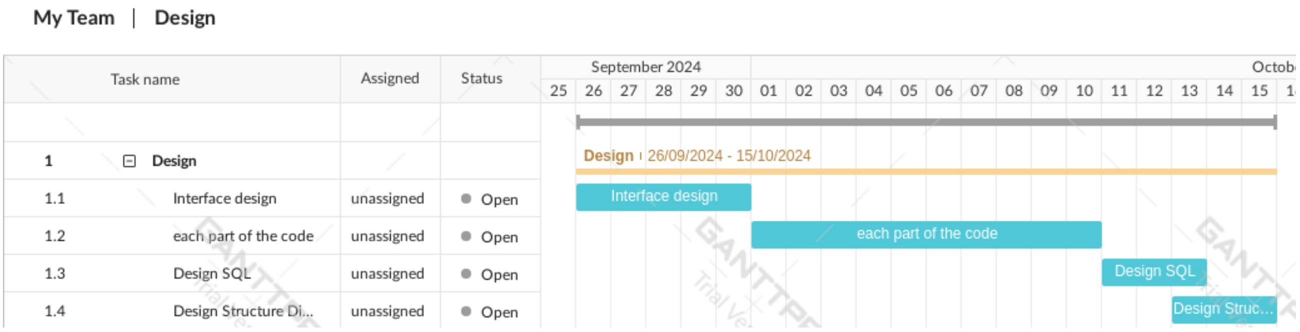


My Team | Analysis

	Task name	Assigned	Status
1	⊟ Analysis		
1.1	create a detail concept idea	unassigned	● Open
1.2	Write the description of th...	unassigned	● Open
1.3	Create the scope, boundari...	unassigned	● Open
1.4	Create UML use case diagr...	unassigned	● Open
1.5	Create user requirements d...	unassigned	● Open
1.6	Project Planning	unassigned	● Open

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**

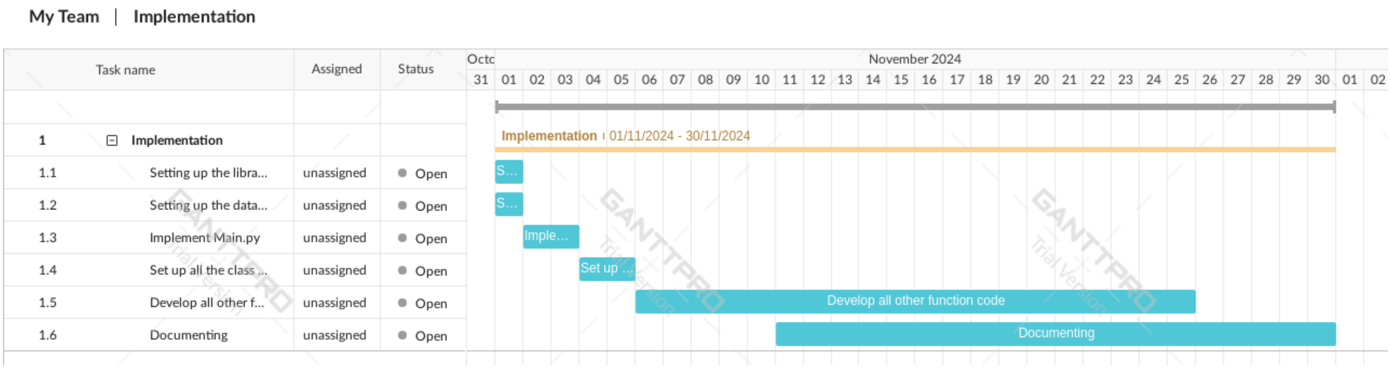


My Team | Design

	Task name	Assigned	Status
1	⊟ Design		
1.1	Interface design	unassigned	● Open
1.2	each part of the code	unassigned	● Open
1.3	Design SQL	unassigned	● Open
1.4	Design Structure Di...	unassigned	● Open

Implementation

- **Setting up the library and database that needed in the program**
  - Est. 1 days needed to set up the library and database
- **Implementing Main.py**
  - Est. 2 days needed to implement.
- **Set up all the classes for the game**
  - Est. 2 dates needed to implement
- **Develop all other function codes**
  - Est. 20 days needed to implement
- **Documenting**
  - Est. 20 days to finish



My Team | Implementation

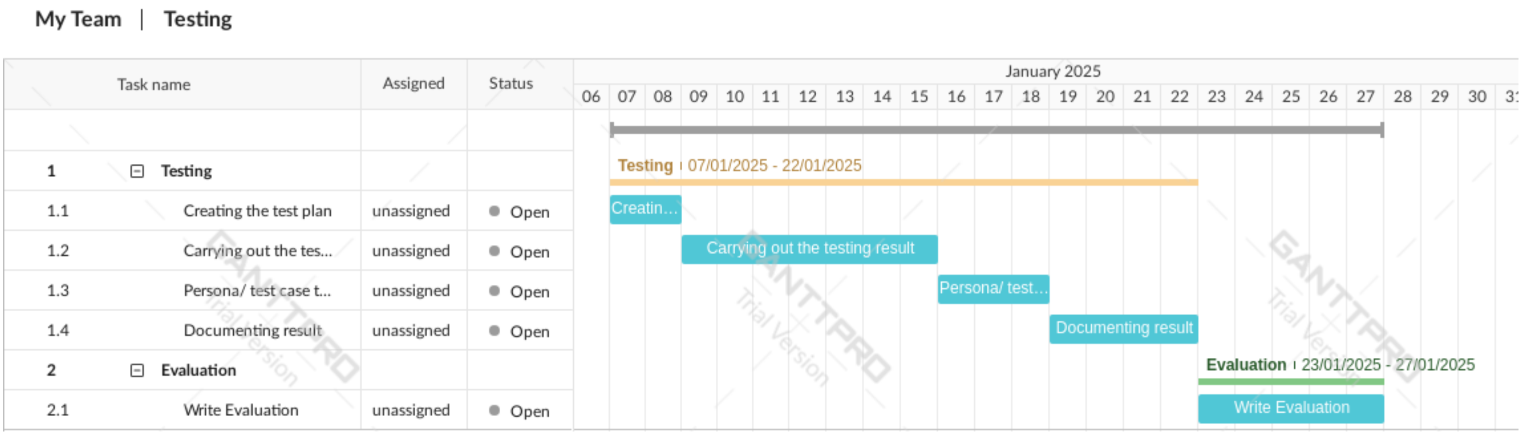
	Task name	Assigned	Status
1	⊟ Implementation		
1.1	Setting up the libra...	unassigned	● Open
1.2	Setting up the data...	unassigned	● Open
1.3	Implement Main.py	unassigned	● Open
1.4	Set up all the class ...	unassigned	● Open
1.5	Develop all other f...	unassigned	● Open
1.6	Documenting	unassigned	● Open

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

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



My Team | Testing

	Task name	Assigned	Status
1	⊟ Testing		
1.1	Creating the test plan	unassigned	● Open
1.2	Carrying out the tes...	unassigned	● Open
1.3	Persona/ test case t...	unassigned	● Open
1.4	Documenting result	unassigned	● Open
2	⊟ Evaluation		
2.1	Write Evaluation	unassigned	● Open

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)

Design

Software design

Structure Diagram and Pseudocode of each part of the program

File structure of the program :

```
├──── classes/
│   ├──── enemy.py
│   ├──── item.py
│   ├──── player.py
│   ├──── spell.py
│
├──── utils/
│   ├──── Assign.py
│   ├──── auth.py
│   ├──── battle.py
│   ├──── CalculateDamage.py
│   ├──── database_connection.py
│   ├──── enemyTurn.py
│   ├──── gameloop.py
│   ├──── Inventory.py
│   ├──── playerTurn.py
│   ├──── showStats.py
│   ├──── Sort.py
│   ├──── start.py
│
├──── main.py
├──── users.txt
```

This section serves as the game's startup point, displaying the main menu where players can register or log in to their accounts through the related functions. From the menu, players can access options to start a new game or load an existing saved game by invoking the appropriate functions.

Main.py

```
FUNCTION post_login_menu:
    WHILE True:
        DISPLAY "========================================"
        DISPLAY "Welcome to the RPG Game!"
        DISPLAY "1. Start New Game"
        DISPLAY "2. Load Saved Game”
        DISPLAY “3. Instructions”
        DISPLAY "4. Exit Game"
        DISPLAY "========================================"
```

Display the user interface

Get user’s choice

```
        INPUT choice
```

Execute the corresponding function based on the user’s choice

```
        IF choice IS '1':
            DISPLAY "Starting a new game..."
            CALL start_new_game()
        ELSE IF choice IS '2':
              CALL load_game(username)
        ELSE IF choice IS ‘3’:
              CALL print_instructions()
        ELSE IF choice IS '4':
            DISPLAY "Exiting game..."
            BREAK
        ELSE:
            DISPLAY "Invalid choice. Please try again."
          END IF
      END WHILE
END FUNCTION

FUNCTION main:
    WHILE True:
        DISPLAY "========================================"
        DISPLAY "Welcome to the RPG Game!"
        DISPLAY "1. Register"
        DISPLAY "2. Login"
        DISPLAY "3. Exit Game"
        DISPLAY "========================================"
```

Display the main menu

```
        INPUT choice

        IF choice IS '1':
            INPUT username AS "Enter a username: "
            INPUT password AS "Enter a password: "
            CALL register(username, password) -> success, message
            DISPLAY message
```

Get register information

Call related function and display message

Get login information

```
        ELSE IF choice IS '2':
            INPUT username AS "Enter your username: "
            INPUT password AS "Enter your password: "
            CALL login(username, password) -> success, message
            DISPLAY message
            IF success IS TRUE:
                CALL post_login_menu()
        ELSE IF choice IS '3':
            DISPLAY "Exiting game..."
            BREAK
        ELSE:
            DISPLAY "Invalid choice. Please try again."
          END IF
      END WHILE
END FUNCTION

IF __name__ IS "__main__":
    CALL main()
```

End the program by BREAK

This section is called by the main menu to initiate the game loop by calling the game_loop function. It initializes the player's class data and retrieves the starting items, applying them to the player accordingly.

Start.py

```
from utills.gameloop import apply_rewards
from classes.player import Player
from utills.gameloop import game_loop


FUNCTION start_new_game:
    DISPLAY "Starting a new game..."
```

Initialize the player data

```
    INITIALIZE player WITH:
        - name = "Hero"
        - level = 1
        - hp = 100
        - mp = 50
        - atk = 200
        - def_ = 1
        - spd = 10
        - exp = 0
```

Clear all items and spells from the list

```
    SET player.inventory, player.spells, Spells TO empty
```

Initialize the stage number

```
    SET stage_number = 0
```

Get the starting resources for the player

```
    CALL apply_rewards(player, fetch_rewards(stage_number))
```

Equip the items at the start

```
    IF player.inventory IS NOT EMPTY:
        FOR EACH item IN player.inventory:
            CALL player.equip_item(item)
        END LOOP
    END IF

    CALL game_loop(player, stage_number)
END FUNCTION
```

This section is called by main.py to handle the registration and login processes for the main menu. All account details are stored in a text file named “users.txt” .

Auth.py

```
CONSTANT USERS_FILE = 'users.txt'

FUNCTION load_users:
    INITIALIZE users AS EMPTY DICTIONARY
    TRY:
        OPEN USERS_FILE IN READ MODE AS file
        FOR EACH line IN file:
            SPLIT line INTO username, password BY ','
            ADD username:password TO users
    EXCEPT FileNotFoundError:
        PASS
    RETURN users
END FUNCTION
```

Open the users file and read each line

Split each line into username & password

If the file does not exist, return and empty dictionary

```
FUNCTION save_user(username, password):
    OPEN USERS_FILE IN APPEND MODE AS file
    WRITE "{username},{password}" TO file WITH NEWLINE
END FUNCTION
```

Write the username and password to the file

```
FUNCTION register(username, password):
    CALL load_users() -> users
    IF username EXISTS IN users:
        RETURN FALSE, "Username already exists."
    CALL save_user(username, password)
    RETURN TRUE, "Registration successful."
END FUNCTION
```

Load existing users into a list

Check if the username is already registered

```
FUNCTION login(username, password):
    CALL load_users() -> users
    IF username EXISTS IN users AND users[username] IS password:
        RETURN TRUE, "Login successful."
    RETURN FALSE, "Invalid username or password."
END FUNCTION
```

Load existing users into a list

Check if the username exists and the password matches

This section will handle applying rewards to the player, such as items or spells earned after completing a stage, or starter items provided at the beginning of the game.

apply_rewards.py

```
FUNCTION apply_rewards(player, rewards):
    FOR EACH reward IN rewards:
       IF reward['reward_type'] IS 'spell':
           FETCH spell_data FROM fetch_spells()
           FOR EACH Spells IN spell_data:
               IF Spells['spell_id'] IS reward['associated_id']:
                  CREATE new_spell USING:
                      - id = Spells['spell_id']
                      - name = Spells['spell_name']
                      - cost = Spells['cost']
                      - value = Spells['value']
                      - Type = Spells['Type']
                  ADD new_spell TO spells
                  DISPLAY "Received spell: {new_spell.name}"
             END IF
           END Loop
       ELSE IF reward['reward_type'] IS 'item':
           FETCH item_data FROM fetch_items()
           FOR EACH items IN item_data:
               IF items['item_id'] IS reward['associated_id']:
                  CREATE new_item USING:
                      - id = items['item_id']
                      - name = items['item_name']
                      - atk = items['atk']
                      - def_ = items['def']
                      - spd = items['spd']
                      - Type = items['Type']
                  FOR _ IN range(reward['quantity']):
                      CALL player.add_item_to_inventory(new_item)
                 END Loop
                  DISPLAY "Received item: {new_item.name} x{reward['quantity']}"
             END IF
           END Loop
       ELSE IF reward['reward_type'] IS 'exp':
           ADD reward['reward_value'] TO player.exp
           DISPLAY "Received {reward['reward_value']} EXP"
           WHILE player.exp >= 100:
               SUBTRACT 100 FROM player.exp
               INCREMENT player.level BY 1
               INCREMENT player.hp, player.mp, player.atk, player.def_, player.spd
BY 1
               DISPLAY "Leveled up! Now at level {player.level}"
           END Loop
       END IF
    END Loop
END FUNCTION
```

Iterate though the rewards received by the player after clear a stage

Fetch all available spells from the database

Fetch all available items from the database

Add exp reward to player

Handle level up

This section is mainly used keeping the game running as long as the player is alive. The things it needs to do include fetching enemy data from the database, calling the battle function to start a stage, checking the player's status, fetching and applying stage rewards, and providing the player with options to continue or save the game.

game_loop.py

Main game loop continues as long as the player is alive

```
FUNCTION game_loop(player, stage_number)
    WHILE player.hp > 0
        player.hp = 100
```

Fetch enemy data for the current stage

```
        enemy_data = fetch_enemy_by_stage(stage_number)
        IF enemy_data EXISTS THEN
            CREATE Enemy USING:
                - name = enemy_data['enemy_name']
                - hp = enemy_data['HP']
                - atk = enemy_data['ATK']
                - def_ = enemy_data['DEF']
                - spd = enemy_data['SPD']
                - level = enemy_data['level']

            DISPLAY "Encountered enemy: [Enemy name]"
        ELSE
            DISPLAY "No enemy found for stage [stage_number]"
        END IF

        DISPLAY "========================================"
        DISPLAY "Entering Stage [stage_number]"
        DISPLAY "========================================"
        DISPLAY "An enemy [Enemy name] appears!"
        DISPLAY "========================================"

        CALL battle(player, Enemy, stage_number)
```

Check if the player has been defeated

```
        IF player.hp <= 0 THEN
            DISPLAY "You have been defeated. Game Over!"
            BREAK
        END IF

        DISPLAY "Congratulations! You cleared Stage [stage_number]."
```

Fetch and apply reward for the cleared stage

```
        rewards = fetch_rewards(stage_number)
        CALL apply_rewards(player, rewards, spells)

        stage_number += 1
```

Check if the game has been completed

```
        IF stage_number > 10 THEN
            DISPLAY "Congratulations! You have defeated all enemies and won the game!"
            BREAK
        END IF
```

Allow the player to manage inventory or save the game

```
        DISPLAY "========================================"
        DISPLAY "What would you like to do?"
        DISPLAY "[C/Enter] Continue [I]Inventory [S] Save game [E] Exit game"
        choice = GET USER INPUT

        IF choice == "C" THEN
            DISPLAY "Continuing to the next stage..."
        ELSE IF choice == "I" THEN
            DISPLAY "Opening inventory..."
            CALL openInventory(player)
        ELSE IF choice == "E" THEN
            BREAK
        ELSE IF choice == "S" THEN
            CALL save(player, stage_number)
            BREAK
        ELSE
            DISPLAY "Invalid choice! Continuing to the next stage by default."
        END IF
    END WHILE
END FUNCTION
```

This section, called by game_loop.py to initiate a stage battle, primarily compares the player’s and enemy’s speed to determine who acts first. It also ensures that the player cannot win the battle while in a dead state (HP ≤ 0).

Battle.py

```
FUNCTION battle(player, enemy, stageNumber)
    SET turn_counter TO 1
    WHILE player.hp > 0 AND enemy.hp > 0 DO

        IF player.spd >= enemy.spd THEN
            DISPLAY "Your Turn"
            CALL PlayerTurn(player, enemy, stageNumber)
            ADD 1 TO turn_counter

            IF enemy.hp <= 0 THEN
                DISPLAY enemy.name + " has been defeated!"
                BREAK
            END IF

            DISPLAY "Enemy's Turn"
            CALL enemyTurn(enemy, player)
        ELSE
            DISPLAY "Enemy's Turn"
            CALL enemyTurn(enemy, player)

            IF player.hp <= 0 THEN
                DISPLAY "You have been defeated!"
                BREAK
            END IF

            DISPLAY "Your Turn"
            CALL PlayerTurn(player, enemy, stageNumber, turn_counter)
        END IF
    END WHILE

    IF player.hp > 0 THEN
        DISPLAY "You won the battle!"
    ELSE
        DISPLAY "Game Over!"
    END IF
END FUNCTION
```

Compare Speed to determine turn order

The player act first and call PlayerTurn

Check if the enemy is defeated

Check if the player is defeated

Show battle results

PlayerTurn.py

This section, called by battle.py, is used to displaying the battle interface, including player and enemy details such as HP and MP. It will also present the available spells and actions the player can choose from. After displaying the interface, it applies spell and skill effects based on their types. Additionally, it includes conditions to call functions for showing player stats and inventory. Lastly, it handles the expiration of buffs.

```
FUNCTION PlayerTurn(player, enemy, stageNumber, turn_counter)
```

Check if the player is charging a spell

```
    IF player IS charging a spell THEN
        DISPLAY message about remaining charge turns
        DECREASE charge turn counter by 1
        IF charge turn counter is 0 THEN
```

Apply the charged spell damage

```
            IF player HAS buff THEN
                APPLY (charged spell damage * 2) TO enemy
            ELSE
                APPLY charged spell damage TO enemy
            END IF
            DISPLAY message about spell discharge
        RETURN
    END IF

    DISPLAY stage and turn information
    DISPLAY enemy stats and health bar
```

Display buff and charge timer if applicable

```
    IF player HAS active buff THEN
        DISPLAY buff timer
    END IF

    IF player HAS active charge THEN
        DISPLAY charge timer
    END IF

    DISPLAY player stats and health bar
    DISPLAY spell options and special attacks

    PROMPT player for choice
```

Handle normal attack

```
    IF choice IS "N" THEN
        CALCULATE damage = player.atk - (enemy.def / 2)
        APPLY damage TO enemy
        INCREASE player.mp BY 5
        DISPLAY damage dealt
```

Handle spell or special attack by the choice

```
    ELSE IF choice IS "1" to "4" THEN
        DETERMINE selected spell
        IF player.mp >= spell.cost THEN
            DECREASE player.mp BY spell.cost
```

Apply effects based on spell type

```
            IF spell.Type IS "damage" THEN
                CALCULATE damage = spell.value + (player.mp / 5)
                APPLY damage TO enemy
                DISPLAY damage dealt
            ELSE IF spell.Type IS "heal" THEN
                INCREASE player.hp BY spell.value
                DISPLAY HP healed
            ELSE IF spell.Type IS "Atk" THEN
                INCREASE player.atk BY spell.value
                DISPLAY increased ATK
            ELSE IF spell.Type IS "Def" THEN
                INCREASE player.hp BY spell.value
                DISPLAY increased DEF
            ELSE IF spell.Type IS "area" THEN
                IF player HAS no active buff THEN
                    STORE player's original ATK
                END IF
                APPLY buff duration of 7 turns
                MULTIPLY player.atk BY spell.value
                DISPLAY area buff applied
            ELSE IF spell.Type IS "charge" THEN
                SET charge duration TO 3 turns
                STORE spell damage value
                DISPLAY charge initiation
            END IF
        ELSE
            DISPLAY insufficient MP message
        END IF
```

Open inventory

```
    ELSE IF choice IS "I" THEN
        CALL openInventory(player)
        RECURSIVELY CALL PlayerTurn
```

Show stats

```
    ELSE IF choice IS "S" THEN
        CALL show_stats(player)
        RECURSIVELY CALL PlayerTurn
    ELSE
        DISPLAY invalid choice message
    END IF
```

Handle buff expiration

```
    IF player HAS active buff THEN
        DECREASE buff turn counter BY 1
        IF buff expires THEN
            RESET player.atk TO original value
            DISPLAY buff expiration message
        END IF
    END IF

END FUNCTION
```

This section is designed to be simple, focusing on displaying the battle interface during the enemy's turn. It handles the enemy's attack and calculates the damage dealt to the player by calling CalculateDamage.py.

enemyTurn.py

```
FUNCTION enemyTurn(enemy, player):
    DISPLAY "<Enemy Name> is preparing to attack!"

    DISPLAY
"===================================================================
====================================================="
    DISPLAY "Enemy Name:"
    DISPLAY "HP: <Enemy HP>"
    DISPLAY "[> * (Enemy HP / 2) * Spaces (50 - Enemy HP / 2)]
(<Enemy HP>%) <- enemy (Health bar should be blue)"

    enemy_damage = CALL CalculateDamage with enemy, player, "normal"
    player.hp = player.hp - enemy_damage
    DISPLAY "<Enemy Name> dealt <Enemy Damage> damage to you"
```

Calculate and apply damage by enemy to player

```
    DISPLAY "<Player Name>:"
    DISPLAY "HP: <Player HP> ATK: <Player ATK> MP: <Player MP> SPD:
<Player SPD>"
    DISPLAY "[> * (Player HP / 2) * Spaces (50 - Player HP / 2)]
(<Player HP>%) <- You (Health bar should be red)"

    DISPLAY
"===================================================================
====================================================="

    DISPLAY "[N] Normal Attack"
    DISPLAY "[1-4] Spell/Special Attack"
    DISPLAY "[I] Inventory"
    DISPLAY "[S] Stats"
```

This section is a core function of the game, used during both the player's and the enemy's turns. It introduces a critical hit system, where the player has a 10% chance to land a critical attack that deals double damage.

CalculateDamage.py

```
FUNCTION CalculateDamage(attacker, defender, attackType):
    IF RANDOM number < 0.1:
        attackType = "critical"

    base_damage = attacker.attack - defender.defense

    IF base_damage < 0:
        base_damage = 0

    IF attackType == "normal":
        damage = base_damage  // Normal attack deals base damage
    ELSE IF attackType == "critical":
        damage = base_damage * 2  // Critical attack deals double
damage
    ELSE:
        damage = 0  // Unknown attack types deal no damage

    RETURN damage
```

This introduce 10% chance for a critical attack, and override the attack type to “critical”

Base formula for damage

Ensure the base damage isn’t negative

Calculate the damage based on the attack type

This section, called by openinventory.py, allows the player to assign skills for use in battle. However, the game limits the player to equipping only 4 skills at a time. Therefore, this section includes a check to see if the player already has 4 skills equipped. If so, the player can choose which skills to replace.

Assign.py

Function *assign_skill*

Display all spells that is available to the player

Display currently equipped spells

Allow the player to equip a new spell if they have less than 4 spells

Handle where the player already has 4 spells equipped

```
FUNCTION assign_skill(player)
    DISPLAY "Available spells:"
    FOR each spell in spells with index i
       DISPLAY spell details (index, name, type, value)
    END FOR

    DISPLAY "Current Equipped spells:"
    FOR each spell in player's equipped spells with index i
       DISPLAY spell details (index, name, type, value)
    END FOR

    IF the number of player's equipped spells is less than 4 THEN
       PROMPT user to choose a skill to equip
       IF user input is a valid number and corresponds to a spell in the list THEN
          SELECT the chosen spell
          ADD the spell to player's equipped spells
          DISPLAY "Equipped skill: " and spell name
       ELSE
          DISPLAY "Invalid choice! Please try again."
       END IF
    ELSE
       DISPLAY "You already have 4 spells equipped."
       PROMPT user if they want to replace a skill (yes/no)
       IF user chooses "yes" THEN
          DISPLAY the list of player's equipped spells
          PROMPT user to choose a skill to replace
          IF user input is a valid number and corresponds to a spell in the list THEN
             PROMPT user to choose a new spell to equip
             IF user input is a valid number and corresponds to a spell in the list THEN
                SELECT the new spell
                REPLACE the chosen equipped spell with the new spell
                DISPLAY "Replaced with skill: " and new spell name
             ELSE
                DISPLAY "Invalid choice! Please try again."
             END IF
          ELSE
             DISPLAY "Invalid choice! Please try again."
          END IF
       ELSE
          DISPLAY "No spells were replaced."
       END IF
    END IF
END FUNCTION
```

Function *equip_item*

This section is likely the same as assign_skill.py and will also be called by openinventory.py. It will allow the player to equip and change the item they are currently using. Additionally, if the same type of item is already equipped, the system will automatically equip the new one.

```
FUNCTION equip_item(player)
    DISPLAY "============================================================"
    DISPLAY "Inventory:"
    FOR each item in player's inventory with index y
        DISPLAY item details (index, name, ATK, DEF, SPD)
    END FOR
```

Display the player's inventory

```
    DISPLAY "============================================================"
    DISPLAY "Current Equipped weapon:"
    IF player has an equipped weapon THEN
        DISPLAY weapon details (name, ATK, DEF, SPD)
    ELSE
        DISPLAY "None"
    END IF

    DISPLAY "Current Equipped shield:"
    IF player has an equipped shield THEN
        DISPLAY shield details (name, ATK, DEF, SPD)
    ELSE
        DISPLAY "None"
    END IF

    DISPLAY "Current Equipped shoes:"
    IF player has equipped shoes THEN
        DISPLAY shoes details (name, ATK, DEF, SPD)
    ELSE
        DISPLAY "None"
    END IF

    DISPLAY "Current Equipped armor:"
    IF player has equipped armor THEN
        DISPLAY armor details (name, ATK, DEF, SPD)
    ELSE
        DISPLAY "None"
    END IF
```

Display the player equipped items

```
    PROMPT user to choose an item to equip
    IF user input is a valid number and corresponds to an item in the inventory THEN
        SELECT the chosen item
        CALL player's `equip_item` method with the chosen item
        DISPLAY "Equipped item: " and item name
    ELSE
        DISPLAY "Invalid choice! Please try again."
    END IF
END FUNCTION
```

Call player's "equip_item" method to equip and item

Allow the player to equip an item from their inventory

This section is another core function that keeps the game running. It establishes the connection to the database and includes different sections to fetch items, spells, and enemies.

Database_connection.py

```
FUNCTION connect_db:
    Establish a connection to the database
    Return the database connection
END FUNCTION
```

Set up the connection to the database from this program

```
FUNCTION fetch_items:
    Establish a connection to the database
    Create a cursor object for executing queries
    Execute a query to retrieve item data (id, name, attack,
defense, speed, type)
    Fetch all the result rows from the query
    Close the database connection
    Return the fetched items
END FUNCTION

FUNCTION fetch_enemy_by_stage(stage_number):
    Establish a connection to the database
    Create a cursor object for executing queries
    Execute a query to retrieve enemy data for the given
stage_number
    Fetch a single result row for the enemy
    Close the database connection
    Return the fetched enemy data
END FUNCTION

FUNCTION fetch_spells:
    Establish a connection to the database
    Create a cursor object for executing queries
    Execute a query to retrieve spell data (id, name, cost, value,
type)
    Fetch all the result rows from the query
    Close the database connection
    Return the fetched spells
END FUNCTION

FUNCTION fetch_rewards(stage_number):
    Establish a connection to the database
    Create a cursor object for executing queries
    Execute a query to retrieve reward data for the given
stage_number
    Fetch all the result rows from the query
    Close the database connection
    Return the fetched rewards
END FUNCTION
```

This section will be called by playerTurn.py. Its main purpose is to display all the items the player owns, along with sorting and search options that the player can use.

Inventory.py

```
FUNCTION openInventory(player)
    WHILE TRUE DO
        DISPLAY "===================================================="
        DISPLAY "INVENTORY"
        FOR y, Item IN ENUMERATE(player.inventory, START=1) DO
            DISPLAY y + ". " + Item.name + " [ATK: " + Item.atk + ", DEF: " + Item.def_ + "]"
        END FOR
        DISPLAY "===================================================="
        DISPLAY "[A] Sort by ATK"
        DISPLAY "[D] Sort by DEF"
        DISPLAY "[N] Sort by name"
        DISPLAY "[S] Search Item"
        DISPLAY "[AS] Assign Skill"
        DISPLAY "[EI] Equip Item"
        DISPLAY "[B] Back to game screen"
        choice ← INPUT("Choose an option: ").TO_UPPERCASE()
```

Set up the connection to the database from this program

```
        IF choice = "A" THEN
            CALL bubble_sort_by_atk(player.inventory)
        ELSE IF choice = "D" THEN
            CALL bubble_sort_by_def(player.inventory)
        ELSE IF choice = "N" THEN
            CALL bubble_sort_by_name(player.inventory)
        ELSE IF choice = "S" THEN
            search_term ← INPUT("Enter item name to search: ").TO_LOWERCASE()
            CALL binary_search(player.inventory, search_term)
            DISPLAY "Press Enter to return to inventory..."
            INPUT()
        ELSE IF choice = "AS" THEN
            CALL assign_skill(player)
        ELSE IF choice = "EI" THEN
            CALL equip_item(player)
        ELSE IF choice = "B" THEN
            BREAK
        END IF
    END WHILE
END FUNCTION
```

Process users choice by calling the related function

This section will be called by the openinventory.py. this has implement that the three bubble sort that has different search criteria and search feature inside.

Sort.py

```
FUNCTION bubble_sort_by_atk(items):
    Set n to the length of items
    Set swapped to True
    While swapped is True and n is greater than or equal to 0:
        Set swapped to False
        For i from 0 to n-2:
            If the attack value of item[i] is greater than the attack value of item[i+1]:
                Swap item[i] and item[i+1]
                Set swapped to True
               END IF
         END For
        Decrease n by 1
      END Loop
END FUNCTION
```

Get the number of items

Continue sorting until no swaps are made

Iterate through the list and swap if needed

```
FUNCTION bubble_sort_by_def(items):
    Set n to the length of items
    Set swapped to True
    While swapped is True and n is greater than or equal to 0:
        Set swapped to False
        For i from 0 to n-2:
            If the defense value of item[i] is greater than the defense value of item[i+1]:
                Swap item[i] and item[i+1]
                Set swapped to True
             END IF
         END For
        Decrease n by 1
      END Loop
END FUNCTION

FUNCTION bubble_sort_by_name(items):
    Set n to the length of items
    Set swapped to True
    While swapped is True and n is greater than or equal to 0:
        Set swapped to False
        For i from 0 to n-2:
            If the name of item[i] is greater than the name of item[i+1]:
                Swap item[i] and item[i+1]
                Set swapped to True
             END IF
         END For
        Decrease n by 1
      END Loop
END FUNCTION
```

Same process with the sort atk but just the atk change to def and name

```
FUNCTION binary_search(items, target):
    Set low to 0
    Set high to the length of items - 1
    Set found to False
    Convert target to lowercase

    While found is False and low is less than or equal to high:
        Set mid to the middle index between low and high
        If the name of item[mid] is equal to target:
            Print "Found at position mid + 1"
            Set found to True
        Else If the name of item[mid] is less than target:
            Set low to mid + 1
        Else:
            Set high to mid – 1
         END IF
     END Loop

    If found is False:
        Print "Target not found"
    END IF
END FUNCTION
```

Initialize the search boundaries

Continue searching until the target is found or the search space is exhausted

If the target is not found print a message

This section is just used to display the current status of the player.

showStats.py

```
FUNCTION show_stats(player):
    DISPLAY a separator line "=================================================="
    DISPLAY the player's name
    DISPLAY the player's level
    DISPLAY the player's HP
    DISPLAY the player's MP
    DISPLAY the player's attack (ATK) stat
    DISPLAY the player's defense (DEF) stat
    DISPLAY the player's speed (SPD) stat
    DISPLAY the player's experience points (EXP)

    DISPLAY "Equipped Items:"
    If the player has an equipped weapon:
        DISPLAY the weapon's name and its stats (ATK, DEF, SPD)
     END IF
    If the player has an equipped shield:
        DISPLAY the shield's name and its stats (ATK, DEF, SPD)
     END IF
    If the player has equipped shoes:
        DISPLAY the shoes' name and its stats (ATK, DEF, SPD)
     END IF
    If the player has an equipped armor:
        DISPLAY the armor's name and its stats (ATK, DEF, SPD)
     END IF

    DISPLAY "Equipped Spells:"
    For each spell in the player's list of spells:
        DISPLAY the spell's name, type, value, and cost
     END FOR

    DISPLAY a separator line "=================================================="
END FUNCTION
```

Save.py

```
FUNCTION save(player, stage_number):
    conn = CONNECT to the database
    cursor = conn.cursor(dictionary=True)

    Execute SQL query to get save IDs for player name
    saves = GET results of the query
```

Check the number of saves of the player

```
    IF length of saves >= 3:
        PRINT "You can only have 3 saves. Please choose a save to overwrite:"

        FOR i, save in saves:
            PRINT "Save ID: <save.id>"

        choice = GET user input for save number to overwrite

        IF choice not in [1, 2, 3]:
            PRINT "Invalid choice. Save operation cancelled."
            CLOSE database connection
            RETURN
```

Get the user’s choice for which save to overwrite

```
        selected_save_id = saves[choice - 1].id
        EXECUTE SQL query to delete the save with the selected save ID
```

Delete the chosen save

```
    player_data = {
        'stage_number': stage_number,
        'name': player.name,
        'level': player.level,
        'hp': player.hp,
        'mp': player.mp,
        'atk': player.atk,
        'def_': player.def_,
        'spd': player.spd,
        'exp': player.exp,
        'inventory': JSON serialize player.inventory,
        'spells': JSON serialize player.spells,
        'equipped_weapon': JSON serialize player.equipped_weapon if it exists else None,
        'equipped_shield': JSON serialize player.equipped_shield if it exists else None,
        'equipped_shoes': JSON serialize player.equipped_shoes if it exists else None,
        'equipped_armor': JSON serialize player.equipped_armor if it exists else None
    }
```

Prepare player data for saving

```
    EXECUTE SQL query to insert player data into savedgame table using player_data
```

Insert the save data into the database

```
    COMMIT changes to the database
    CLOSE database connection
    PRINT "Game saved successfully."
```

Load.py

```
FUNCTION load_game(username):
    conn = CONNECT to the database
    cursor = conn.cursor(dictionary=True)

    EXECUTE SQL query to fetch all saved games for the given username
    saves = GET results of the query
```

Fetch saved games for the player

```
    IF saves is empty:
        DISPLAY "No saved games found for this user."
        CLOSE database connection
        RETURN None, None
```

Check if there are no saves

```
    DISPLAY "Choose a save to load:"
    FOR i, save in saves:
        DISPLAY "Save ID: <save.id>, Stage: <save.stage_number>, Level: <save.level>"

    choice = GET user input for the save number to load

    IF choice is less than 1 or greater than the number of saves:
        DISPLAY "Invalid choice. Load operation cancelled."
        CLOSE database connection
        RETURN None, None
```

Prompt the user to select a save

```
    save_data = saves[choice - 1]

    player = CREATE Player object using save_data attributes (name, level, hp, mp, atk, def_,
spd, exp)
```

Create a player object from the saved data

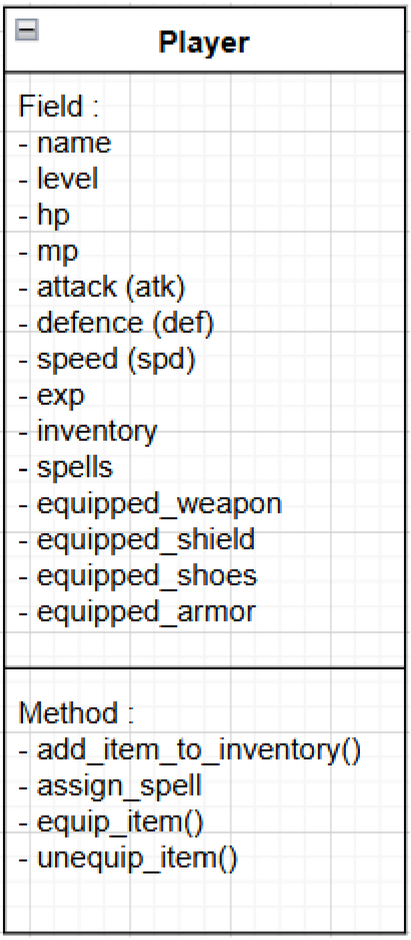
```
    player.inventory = CREATE list of Item objects from saved inventory data
    player.spells = CREATE list of spell objects from saved spells data

    IF equipped_weapon exists in saved data:
        player.equipped_weapon = CREATE Item object from saved equipped weapon data
    IF equipped_shield exists in saved data:
        player.equipped_shield = CREATE Item object from saved equipped shield data
    IF equipped_shoes exists in saved data:
        player.equipped_shoes = CREATE Item object from saved equipped shoes data
    IF equipped_armor exists in saved data:
        player.equipped_armor = CREATE Item object from saved equipped armor data

    stage_number = save_data.stage_number
```

Retrieve the stage number from the save data

```
    CLOSE database connection
    DISPLAY "Game loaded successfully."

    CALL game_loop with player and stage_number
```

Class Diagrams

Player class

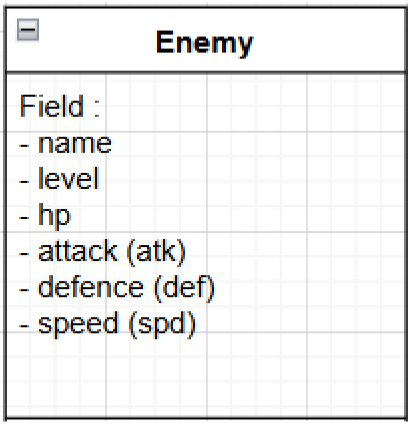


Player

Field :
- name
- level
- hp
- mp
- attack (atk)
- defence (def)
- speed (spd)
- exp
- inventory
- spells
- equipped_weapon
- equipped_shield
- equipped_shoes
- equipped_armor

Method :
- add_item_to_inventory()
- assign_spell
- equip_item()
- unequip_item()

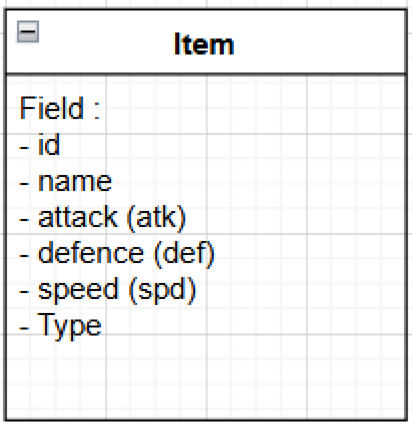
Enemy class



Enemy

Field :
- name
- level
- hp
- attack (atk)
- defence (def)
- speed (spd)

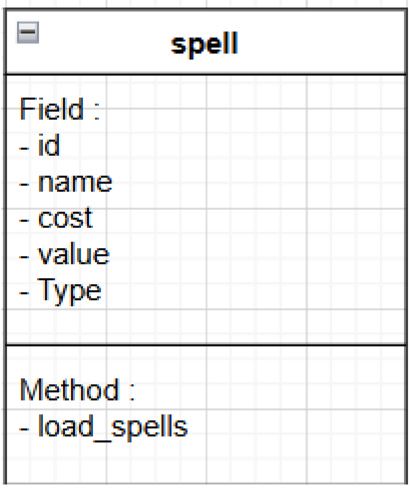
Item class



Item

Field :
- id
- name
- attack (atk)
- defence (def)
- speed (spd)
- Type

Spell class

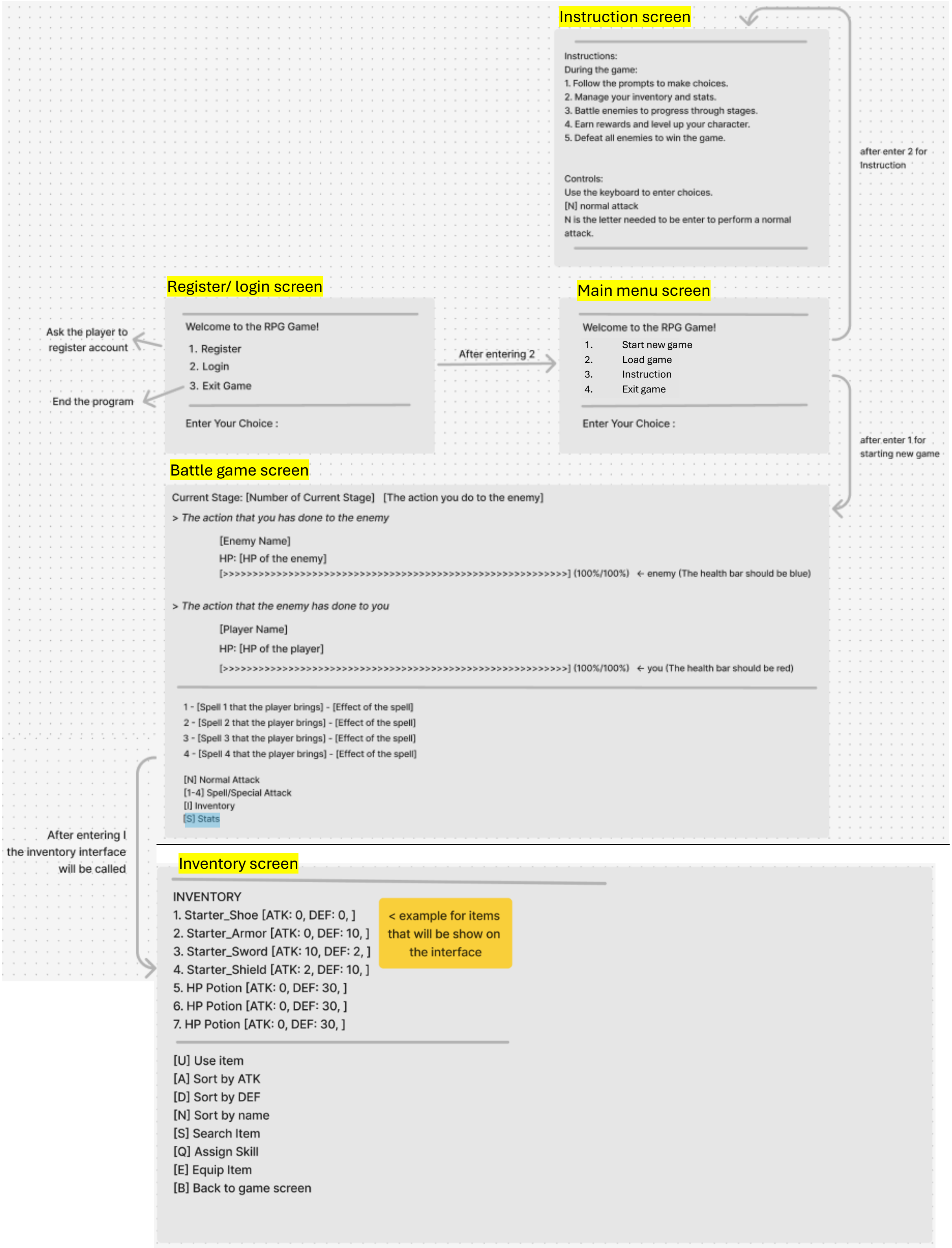


spell

Field :
- id
- name
- cost
- value
- Type

Method :
- load_spells

User-interface design



after enter 2 for Instruction
Ask the player to register account
End the program
After entering 2
after enter 1 for starting new game
After entering I the inventory interface will be called
< example for items that will be show on the interface

Instruction screen

```
Instructions:
During the game:
1. Follow the prompts to make choices.
2. Manage your inventory and stats.
3. Battle enemies to progress through stages.
4. Earn rewards and level up your character.
5. Defeat all enemies to win the game.


Controls:
Use the keyboard to enter choices.
[N] normal attack
N is the letter needed to be enter to perform a normal
attack.
```

Register/ login screen

```
Welcome to the RPG Game!
 1. Register
 2. Login
 3. Exit Game

Enter Your Choice :
```

Main menu screen

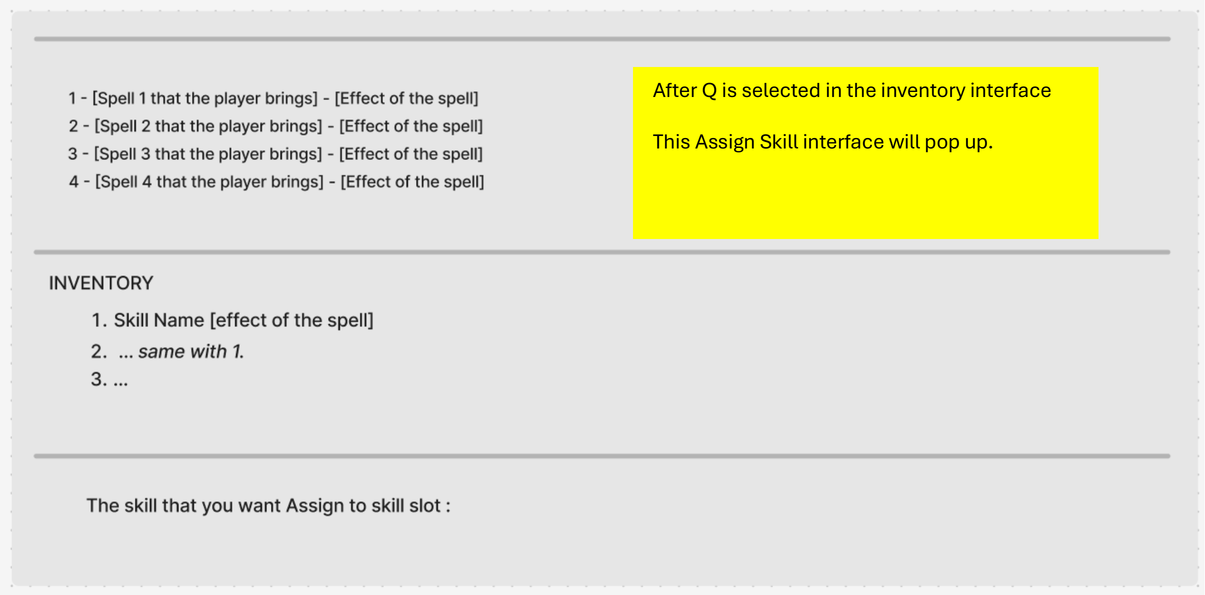
```
Welcome to the RPG Game!
1.      Start new game
2.      Load game
3.      Instruction
4.      Exit game

Enter Your Choice :
```

Battle game screen

```
Current Stage: [Number of Current Stage]   [The action you do to the enemy]
> The action that you has done to the enemy

        [Enemy Name]
        HP: [HP of the enemy]
        [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (100%/100%)  ← enemy (The health bar should be blue)

> The action that the enemy has done to you

        [Player Name]
        HP: [HP of the player]
        [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (100%/100%)  ← you (The health bar should be red)

  1 - [Spell 1 that the player brings] - [Effect of the spell]
  2 - [Spell 2 that the player brings] - [Effect of the spell]
  3 - [Spell 3 that the player brings] - [Effect of the spell]
  4 - [Spell 4 that the player brings] - [Effect of the spell]

  [N] Normal Attack
  [1-4] Spell/Special Attack
  [I] Inventory
  [S] Stats
```

Inventory screen

```
INVENTORY
1. Starter_Shoe [ATK: 0, DEF: 0, ]
2. Starter_Armor [ATK: 0, DEF: 10, ]
3. Starter_Sword [ATK: 10, DEF: 2, ]
4. Starter_Shield [ATK: 2, DEF: 10, ]
5. HP Potion [ATK: 0, DEF: 30, ]
6. HP Potion [ATK: 0, DEF: 30, ]
7. HP Potion [ATK: 0, DEF: 30, ]

[U] Use item
[A] Sort by ATK
[D] Sort by DEF
[N] Sort by name
[S] Search Item
[Q] Assign Skill
[E] Equip Item
[B] Back to game screen
```



1 - [Spell 1 that the player brings] - [Effect of the spell]
2 - [Spell 2 that the player brings] - [Effect of the spell]
3 - [Spell 3 that the player brings] - [Effect of the spell]
4 - [Spell 4 that the player brings] - [Effect of the spell]

After Q is selected in the inventory interface

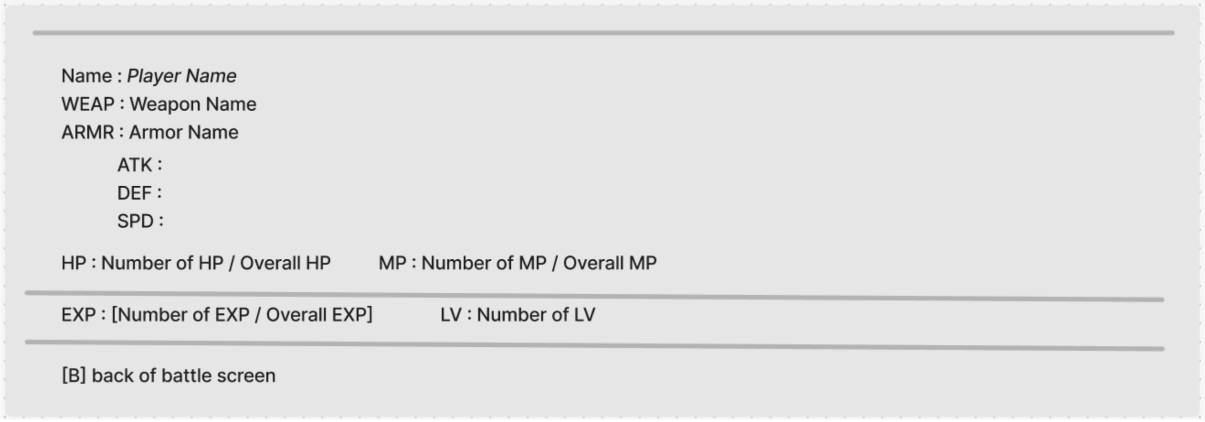
This Assign Skill interface will pop up.

INVENTORY

1. Skill Name [effect of the spell]
2. *... same with 1.*
3. ...

The skill that you want Assign to skill slot :

After selecting [S] in the battle screen the status screen will pop up



Name : *Player Name*
WEAP : Weapon Name
ARMR : Armor Name
ATK :
DEF :
SPD :

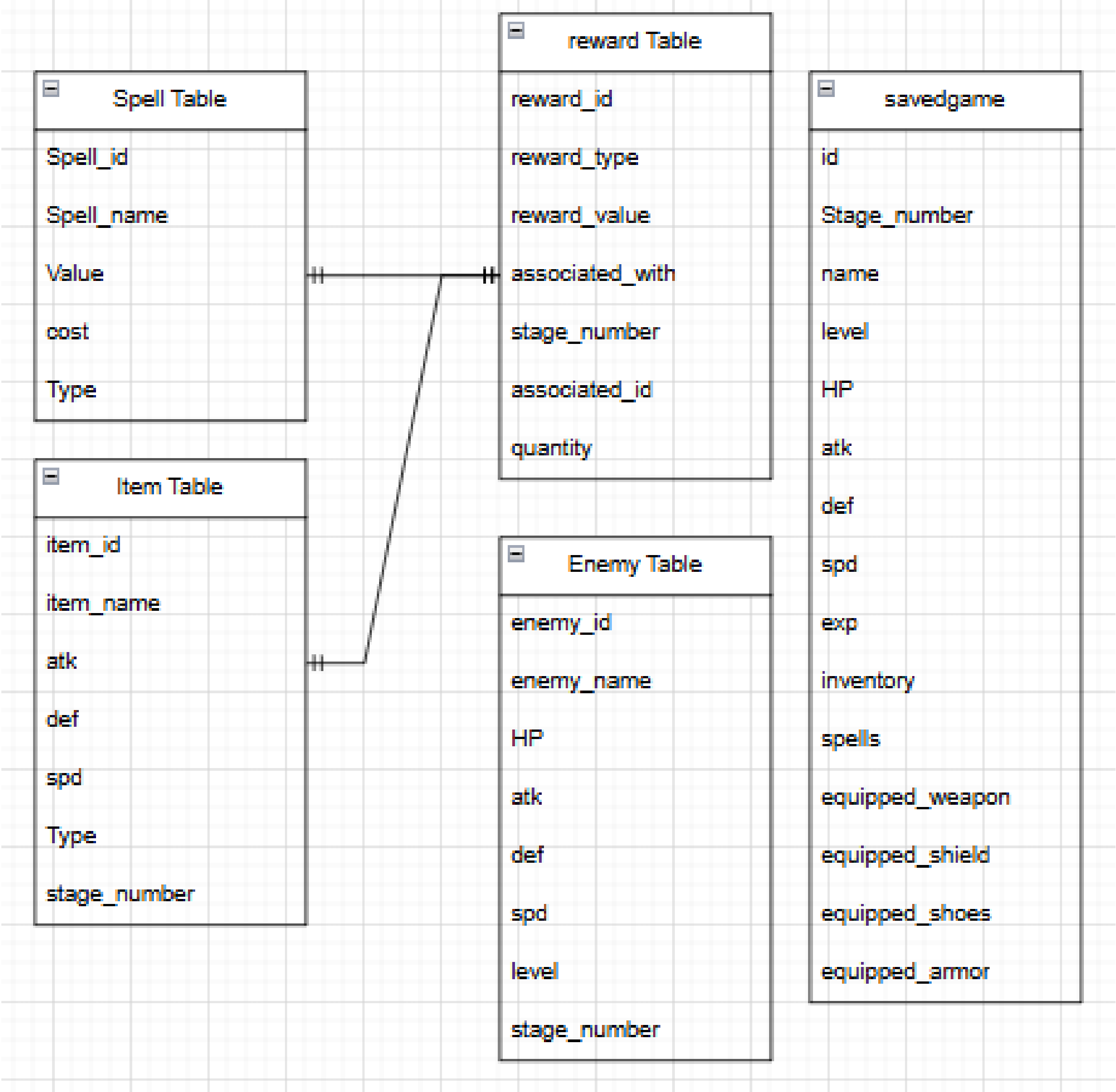
HP : Number of HP / Overall HP MP : Number of MP / Overall MP

EXP : [Number of EXP / Overall EXP] LV : Number of LV

[B] back of battle screen

Database design

Entity relationship diagram



Spell Table
Spell_id
Spell_name
Value
cost
Type

Item Table
item_id
item_name
atk
def
spd
Type
stage_number

reward Table
reward_id
reward_type
reward_value
associated_with
stage_number
associated_id
quantity

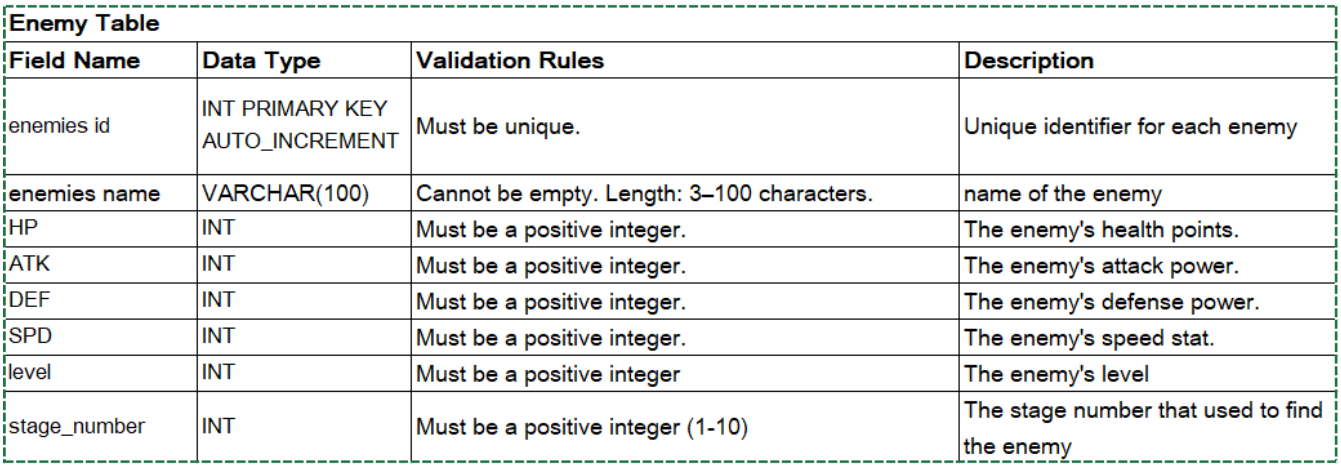
Enemy Table
enemy_id
enemy_name
HP
atk
def
spd
level
stage_number

savedgame
id
Stage_number
name
level
HP
atk
def
spd
exp
inventory
spells
equipped_weapon
equipped_shield
equipped_shoes
equipped_armor

All table relationships will be established using SQL code.

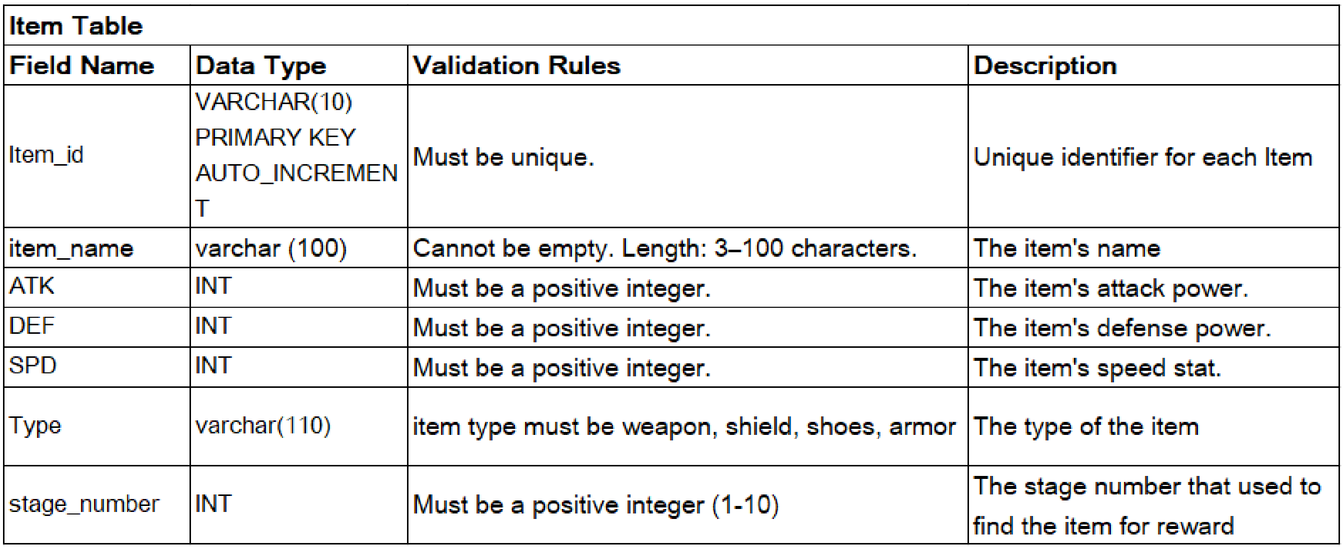
Data dictionary

Enemy Table



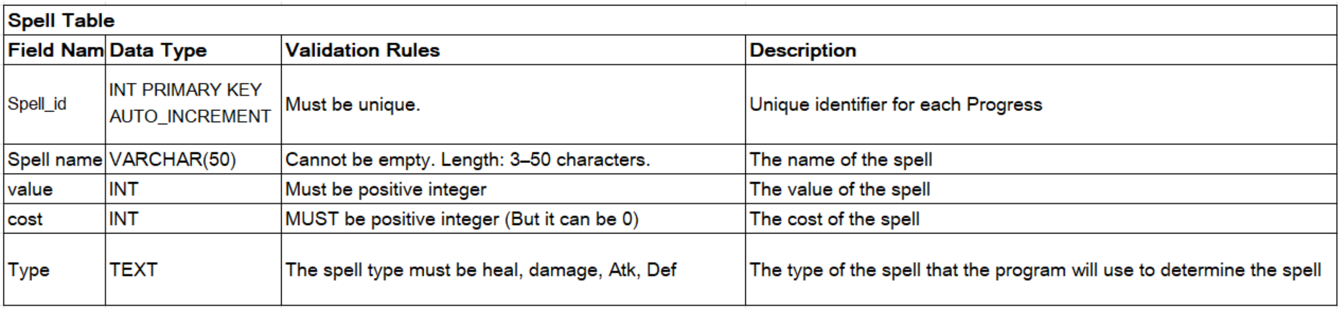
Enemy Table			
Field Name	**Data Type**	**Validation Rules**	**Description**
enemies id	INT PRIMARY KEY AUTO_INCREMENT	Must be unique.	Unique identifier for each enemy
enemies name	VARCHAR(100)	Cannot be empty. Length: 3–100 characters.	name of the enemy
HP	INT	Must be a positive integer.	The enemy's health points.
ATK	INT	Must be a positive integer.	The enemy's attack power.
DEF	INT	Must be a positive integer.	The enemy's defense power.
SPD	INT	Must be a positive integer.	The enemy's speed stat.
level	INT	Must be a positive integer	The enemy's level
stage_number	INT	Must be a positive integer (1-10)	The stage number that used to find the enemy

Item Table



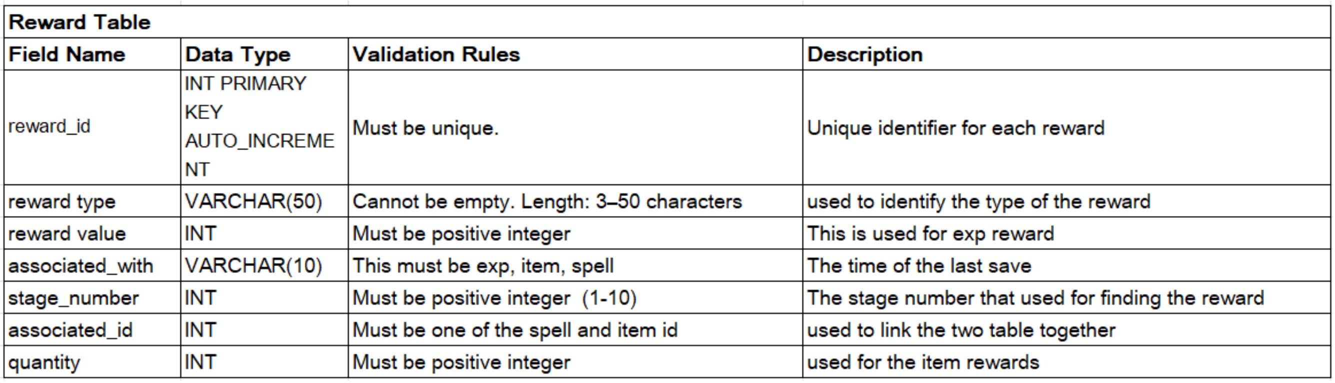
Item Table			
Field Name	**Data Type**	**Validation Rules**	**Description**
Item_id	VARCHAR(10) PRIMARY KEY AUTO_INCREMEN T	Must be unique.	Unique identifier for each Item
item_name	varchar (100)	Cannot be empty. Length: 3–100 characters.	The item's name
ATK	INT	Must be a positive integer.	The item's attack power.
DEF	INT	Must be a positive integer.	The item's defense power.
SPD	INT	Must be a positive integer.	The item's speed stat.
Type	varchar(110)	item type must be weapon, shield, shoes, armor	The type of the item
stage_number	INT	Must be a positive integer (1-10)	The stage number that used to find the item for reward

Spell Table



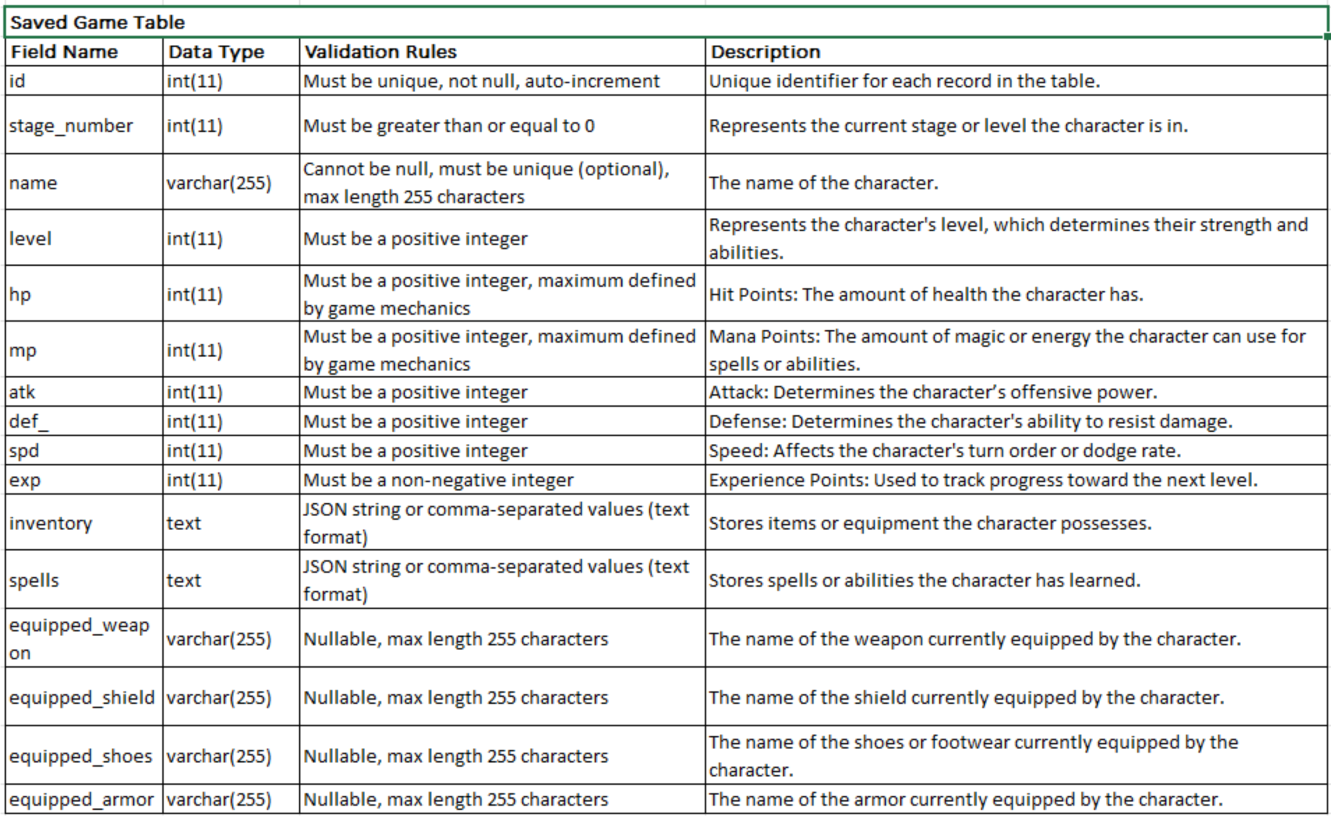
Spell Table			
Field Nam	**Data Type**	**Validation Rules**	**Description**
Spell_id	INT PRIMARY KEY AUTO_INCREMENT	Must be unique.	Unique identifier for each Progress
Spell name	VARCHAR(50)	Cannot be empty. Length: 3–50 characters.	The name of the spell
value	INT	Must be positive integer	The value of the spell
cost	INT	MUST be positive integer (But it can be 0)	The cost of the spell
Type	TEXT	The spell type must be heal, damage, Atk, Def	The type of the spell that the program will use to determine the spell

Reward Table



Reward Table			
Field Name	**Data Type**	**Validation Rules**	**Description**
reward_id	INT PRIMARY KEY AUTO_INCREMENT	Must be unique.	Unique identifier for each reward
reward type	VARCHAR(50)	Cannot be empty. Length: 3–50 characters	used to identify the type of the reward
reward value	INT	Must be positive integer	This is used for exp reward
associated_with	VARCHAR(10)	This must be exp, item, spell	The time of the last save
stage_number	INT	Must be positive integer (1-10)	The stage number that used for finding the reward
associated_id	INT	Must be one of the spell and item id	used to link the two table together
quantity	INT	Must be positive integer	used for the item rewards

SavedGame Table



Saved Game Table			
Field Name	**Data Type**	**Validation Rules**	**Description**
id	int(11)	Must be unique, not null, auto-increment	Unique identifier for each record in the table.
stage_number	int(11)	Must be greater than or equal to 0	Represents the current stage or level the character is in.
name	varchar(255)	Cannot be null, must be unique (optional), max length 255 characters	The name of the character.
level	int(11)	Must be a positive integer	Represents the character's level, which determines their strength and abilities.
hp	int(11)	Must be a positive integer, maximum defined by game mechanics	Hit Points: The amount of health the character has.
mp	int(11)	Must be a positive integer, maximum defined by game mechanics	Mana Points: The amount of magic or energy the character can use for spells or abilities.
atk	int(11)	Must be a positive integer	Attack: Determines the character's offensive power.
def_	int(11)	Must be a positive integer	Defense: Determines the character's ability to resist damage.
spd	int(11)	Must be a positive integer	Speed: Affects the character's turn order or dodge rate.
exp	int(11)	Must be a non-negative integer	Experience Points: Used to track progress toward the next level.
inventory	text	JSON string or comma-separated values (text format)	Stores items or equipment the character possesses.
spells	text	JSON string or comma-separated values (text format)	Stores spells or abilities the character has learned.
equipped_weapon	varchar(255)	Nullable, max length 255 characters	The name of the weapon currently equipped by the character.
equipped_shield	varchar(255)	Nullable, max length 255 characters	The name of the shield currently equipped by the character.
equipped_shoes	varchar(255)	Nullable, max length 255 characters	The name of the shoes or footwear currently equipped by the character.
equipped_armor	varchar(255)	Nullable, max length 255 characters	The name of the armor currently equipped by the character.

Query design

Fetching Item

Field(s) and/or calculations(s)	item_id, item_name, atk, def, spd, Type
Table(s) and/or query(-ies)	item_table
Search criteria	

Fetching enemy by stage

Field(s) and/or calculations(s)	enemy_name, HP, ATK, DEF, SPD, level
Table(s) and/or query(-ies)	enemy_table
Search criteria	stage_number

Fetching Spell

Field(s) and/or calculations(s)	spell_id, spell_name, cost, value, Type
Table(s) and/or query(-ies)	spell_table
Search criteria	

Fetching Reward

Field(s) and/or calculations(s)	reward_type, reward_value, associated_with, stage_number, associated_id, quantity
Table(s) and/or query(-ies)	reward_table
Search criteria	stage_number

Savedgame

Field(s) and/or calculations(s)	Id
Table(s) and/or query(-ies)	savedgame
Search criteria	Name = %s

Delete savedgame

Field(s) and/or calculations(s)	N/A
Table(s) and/or query(-ies)	savedgame
Search criteria	id = %s

Insert new savedata

Field(s) and/or calculations(s)	stage_number, name, level, hp, mp, atk, def_, spd, exp, inventory, spells, equipped_weapon, equipped_shield, equipped_shoes, equipped_armor
Table(s) and/or query(-ies)	savedgame
Search criteria	**N/A** (Inserts data, not a search)

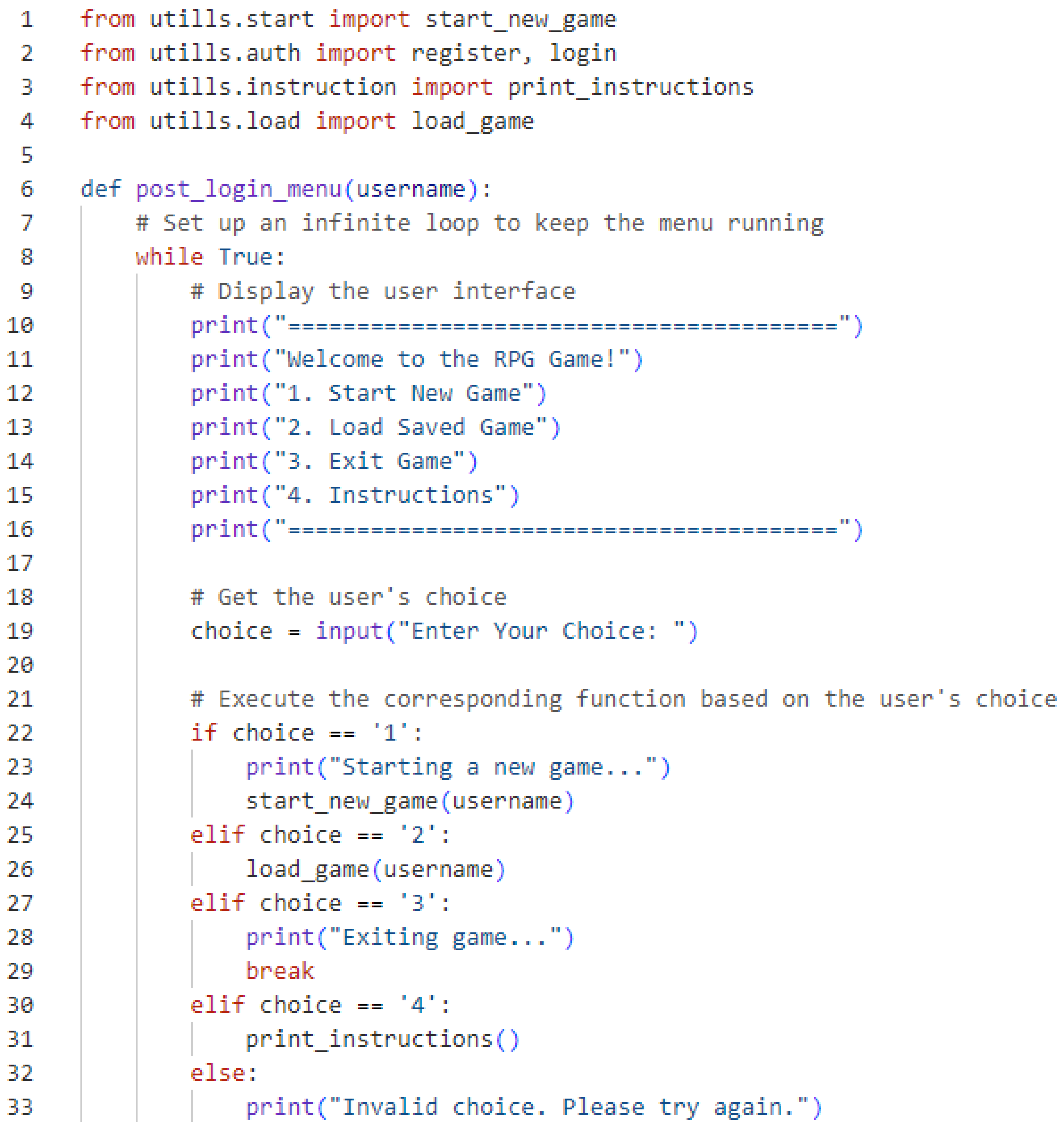
retrieve all saved game details

Field(s) and/or calculations(s)	* (all columns)
Table(s) and/or query(-ies)	savedgame
Search criteria	Name = %s

Implementation

SDD

Main.py (import section & post_login_menu)

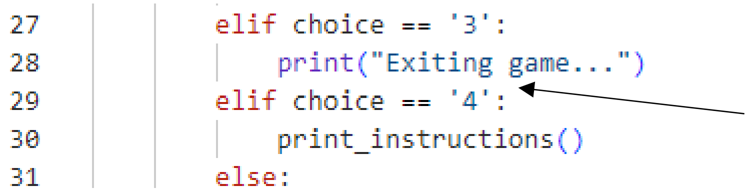


```
from utills.start import start_new_game
from utills.auth import register, login
from utills.instruction import print_instructions
from utills.load import load_game

def post_login_menu(username):
    # Set up an infinite loop to keep the menu running
    while True:
        # Display the user interface
        print("========================================")
        print("Welcome to the RPG Game!")
        print("1. Start New Game")
        print("2. Load Saved Game")
        print("3. Exit Game")
        print("4. Instructions")
        print("========================================")

        # Get the user's choice
        choice = input("Enter Your Choice: ")

        # Execute the corresponding function based on the user's choice
        if choice == '1':
            print("Starting a new game...")
            start_new_game(username)
        elif choice == '2':
            load_game(username)
        elif choice == '3':
            print("Exiting game...")
            break
        elif choice == '4':
            print_instructions()
        else:
            print("Invalid choice. Please try again.")
```

Log of ongoing testing

- **Testing : would the loop of the main menu works**
  - Problem : The while True loop has keep the game in hang and the user couldn't exit expect they just force close the game
  - Solve : Because of the error version didn't have break inside so the user couldn't stop the game by entering 3 to exit, after adding back break could solve this problem.

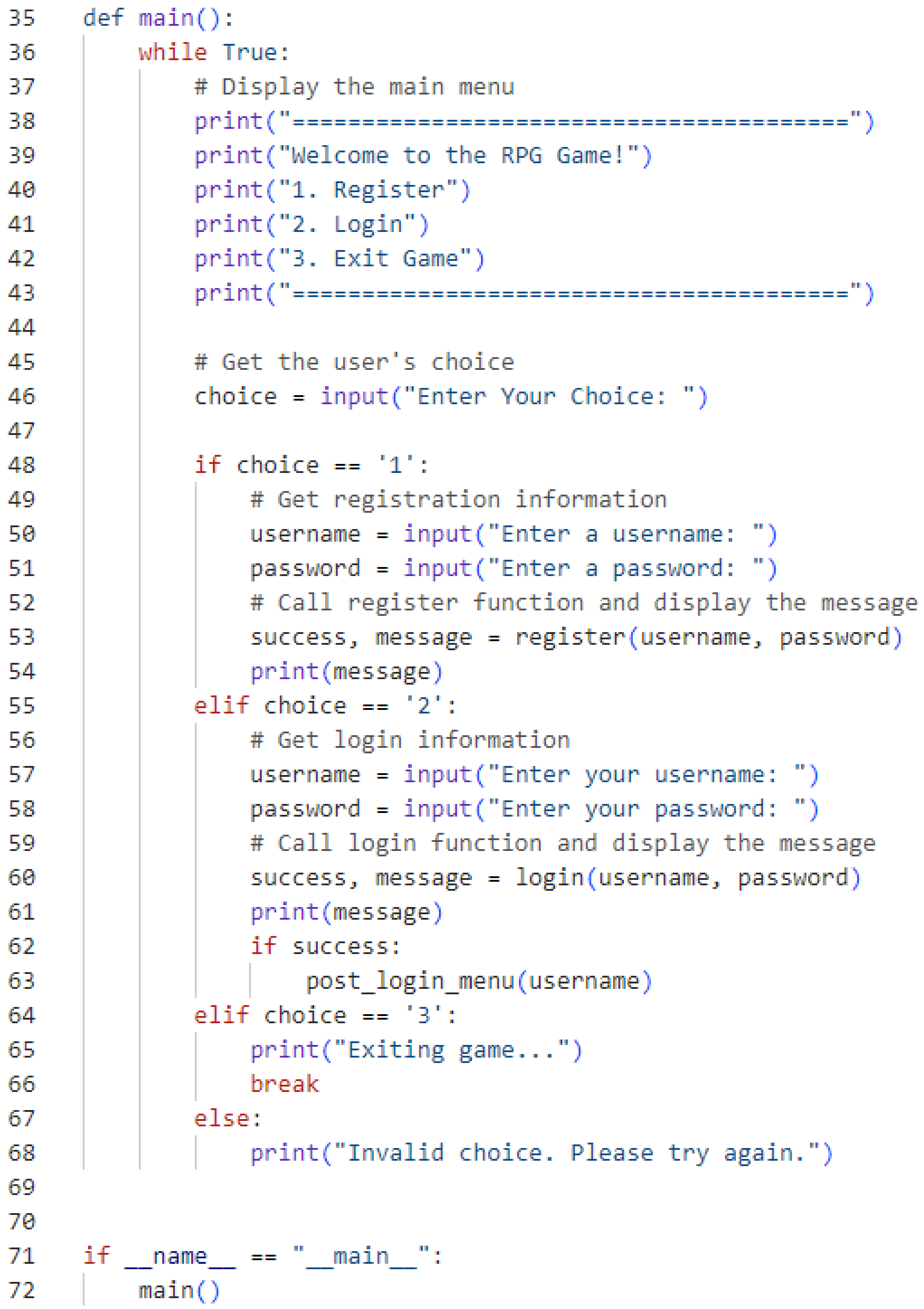
Error code:



```
        elif choice == '3':
            print("Exiting game...")
        elif choice == '4':
            print_instructions()
        else:
```

Missing "break" here

Main.py (main, activate section)

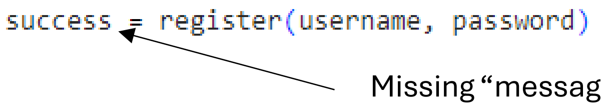


```python
def main():
    while True:
        # Display the main menu
        print("========================================")
        print("Welcome to the RPG Game!")
        print("1. Register")
        print("2. Login")
        print("3. Exit Game")
        print("========================================")

        # Get the user's choice
        choice = input("Enter Your Choice: ")

        if choice == '1':
            # Get registration information
            username = input("Enter a username: ")
            password = input("Enter a password: ")
            # Call register function and display the message
            success, message = register(username, password)
            print(message)
        elif choice == '2':
            # Get login information
            username = input("Enter your username: ")
            password = input("Enter your password: ")
            # Call login function and display the message
            success, message = login(username, password)
            print(message)
            if success:
                post_login_menu(username)
        elif choice == '3':
            print("Exiting game...")
            break
        else:
            print("Invalid choice. Please try again.")


if __name__ == "__main__":
    main()
```

Log of ongoing testing

- **Testing : would the message displayer normally**
    - **Problem :** the "Print(message)" isn't displaying any message like "Registration successfully "
    - **Error :** The part that calling the register & login function didn't have message as a return.
    - **Solve :** I have add back in the return for message

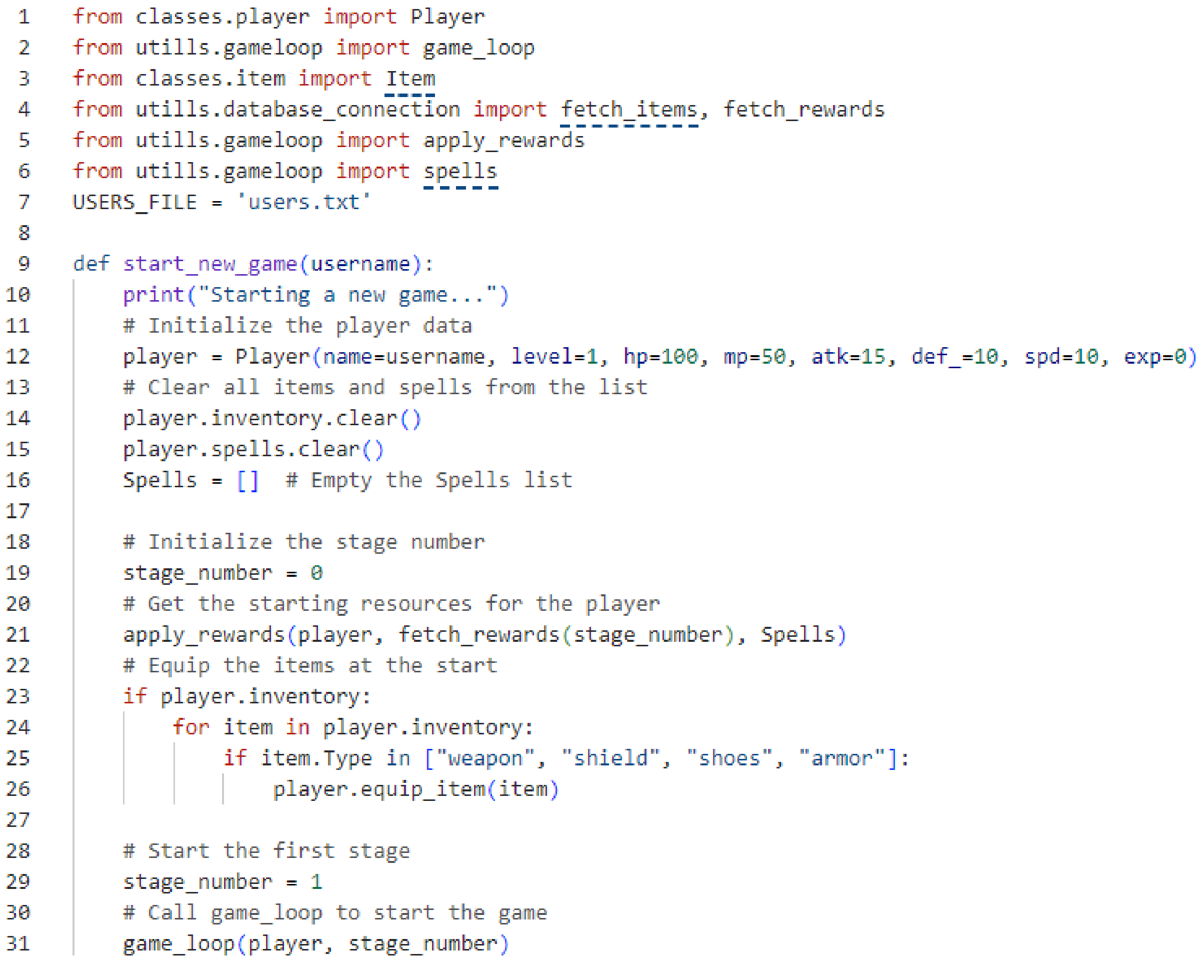
Error code:



```python
success = register(username, password)
```

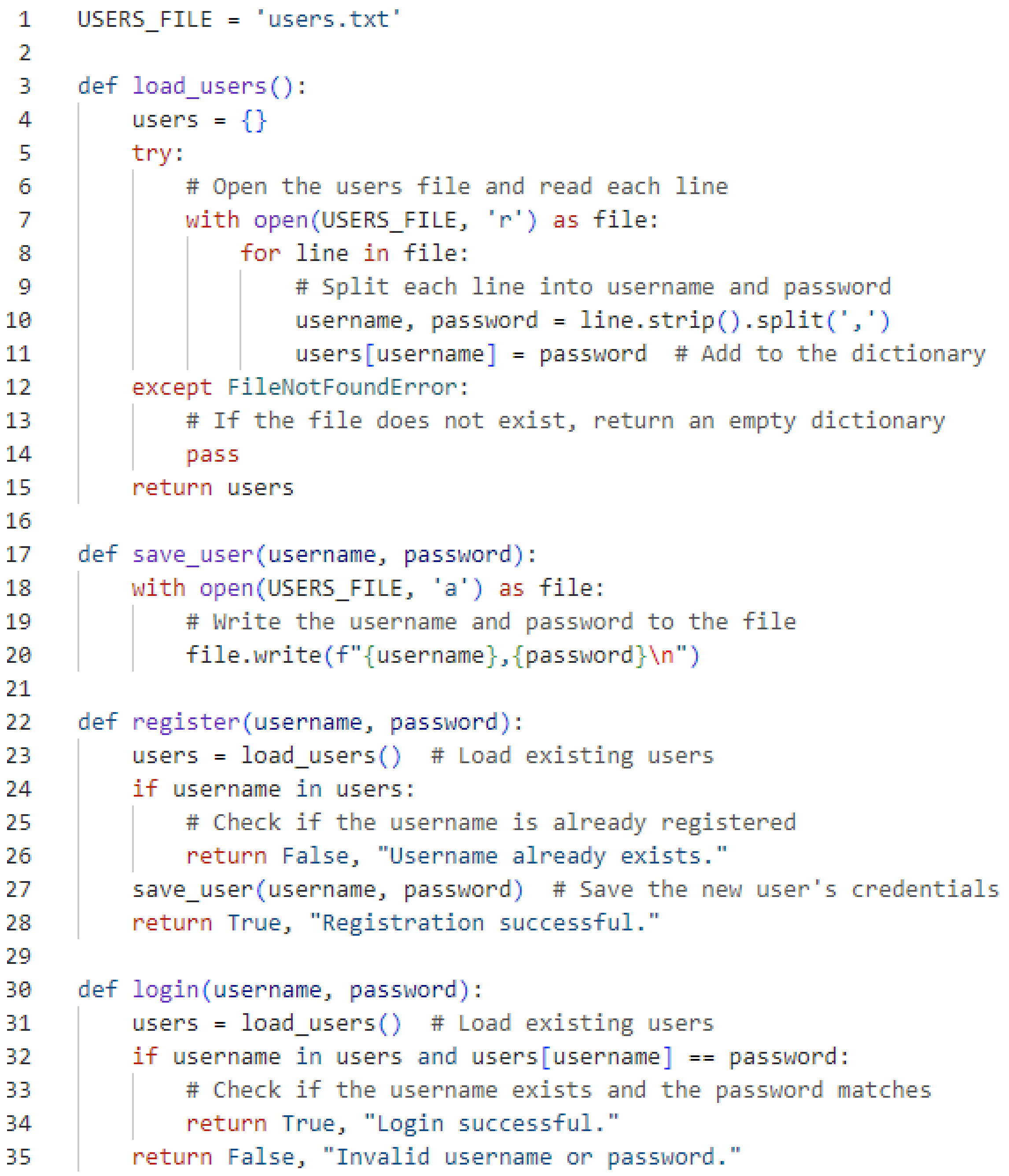
Missing "message" here for receiving the return message

Start.py



```python
from classes.player import Player
from utills.gameloop import game_loop
from classes.item import Item
from utills.database_connection import fetch_items, fetch_rewards
from utills.gameloop import apply_rewards
from utills.gameloop import spells
USERS_FILE = 'users.txt'

def start_new_game(username):
    print("Starting a new game...")
    # Initialize the player data
    player = Player(name=username, level=1, hp=100, mp=50, atk=15, def_=10, spd=10, exp=0)
    # Clear all items and spells from the list
    player.inventory.clear()
    player.spells.clear()
    Spells = []  # Empty the Spells list

    # Initialize the stage number
    stage_number = 0
    # Get the starting resources for the player
    apply_rewards(player, fetch_rewards(stage_number), Spells)
    # Equip the items at the start
    if player.inventory:
        for item in player.inventory:
            if item.Type in ["weapon", "shield", "shoes", "armor"]:
                player.equip_item(item)

    # Start the first stage
    stage_number = 1
    # Call game_loop to start the game
    game_loop(player, stage_number)
```

Auth.py

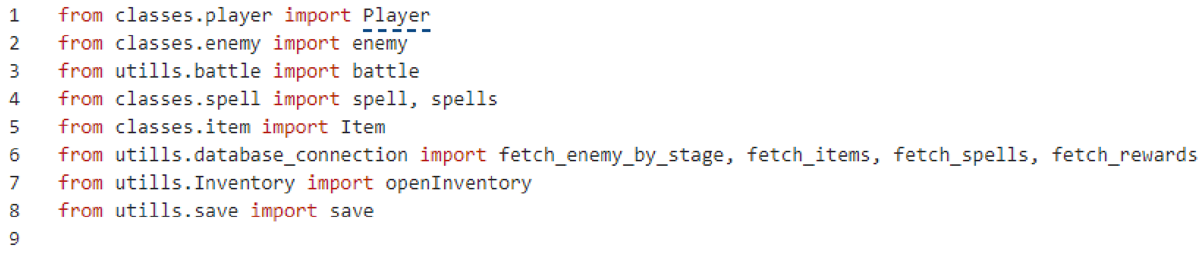


```python
USERS_FILE = 'users.txt'

def load_users():
    users = {}
    try:
        # Open the users file and read each line
        with open(USERS_FILE, 'r') as file:
            for line in file:
                # Split each line into username and password
                username, password = line.strip().split(',')
                users[username] = password  # Add to the dictionary
    except FileNotFoundError:
        # If the file does not exist, return an empty dictionary
        pass
    return users

def save_user(username, password):
    with open(USERS_FILE, 'a') as file:
        # Write the username and password to the file
        file.write(f"{username},{password}\n")

def register(username, password):
    users = load_users()  # Load existing users
    if username in users:
        # Check if the username is already registered
        return False, "Username already exists."
    save_user(username, password)  # Save the new user's credentials
    return True, "Registration successful."

def login(username, password):
    users = load_users()  # Load existing users
    if username in users and users[username] == password:
        # Check if the username exists and the password matches
        return True, "Login successful."
    return False, "Invalid username or password."
```

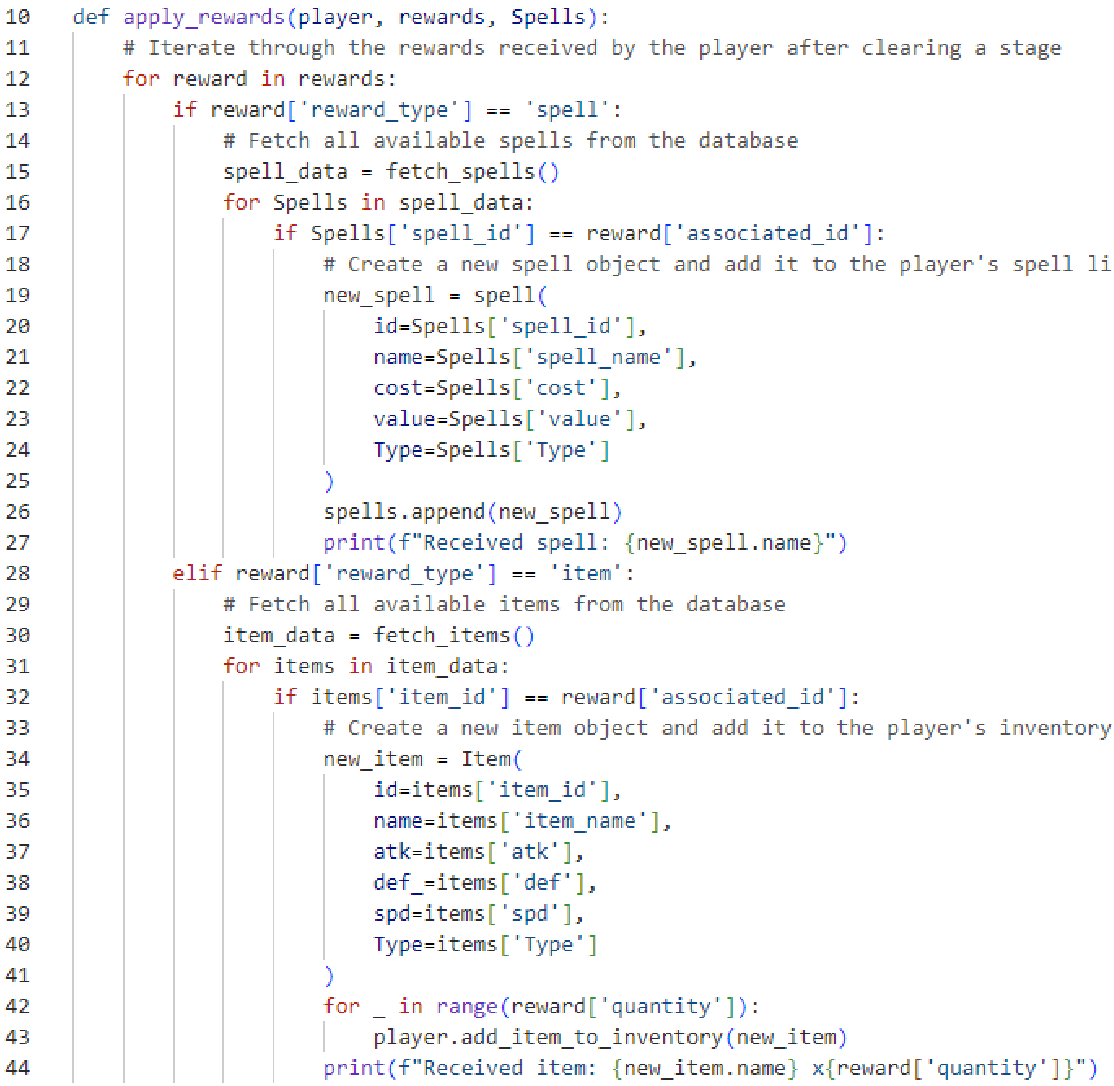
Log of ongoing testing

- **Testing : if the program will handle duplicate username and password**
    - **Problem :** The file accidentally stores or saves a duplicate username, this leads to there problem happening later
    - **Reason :** There isn't any code to prevent this problem happening
    - **Solve :** I have added a if statement to keep the username not duplicating in the file

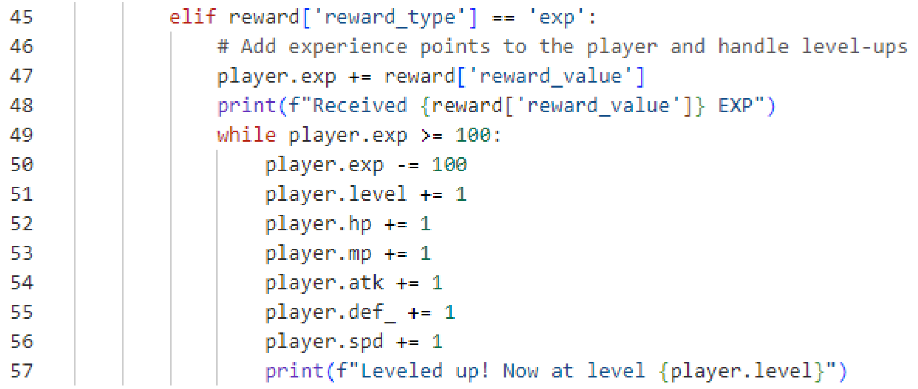
Gameloop.py (import & apply_rewards)



```python
from classes.player import Player
from classes.enemy import enemy
from utills.battle import battle
from classes.spell import spell, spells
from classes.item import Item
from utills.database_connection import fetch_enemy_by_stage, fetch_items, fetch_spells, fetch_rewards
from utills.Inventory import openInventory
from utills.save import save

```

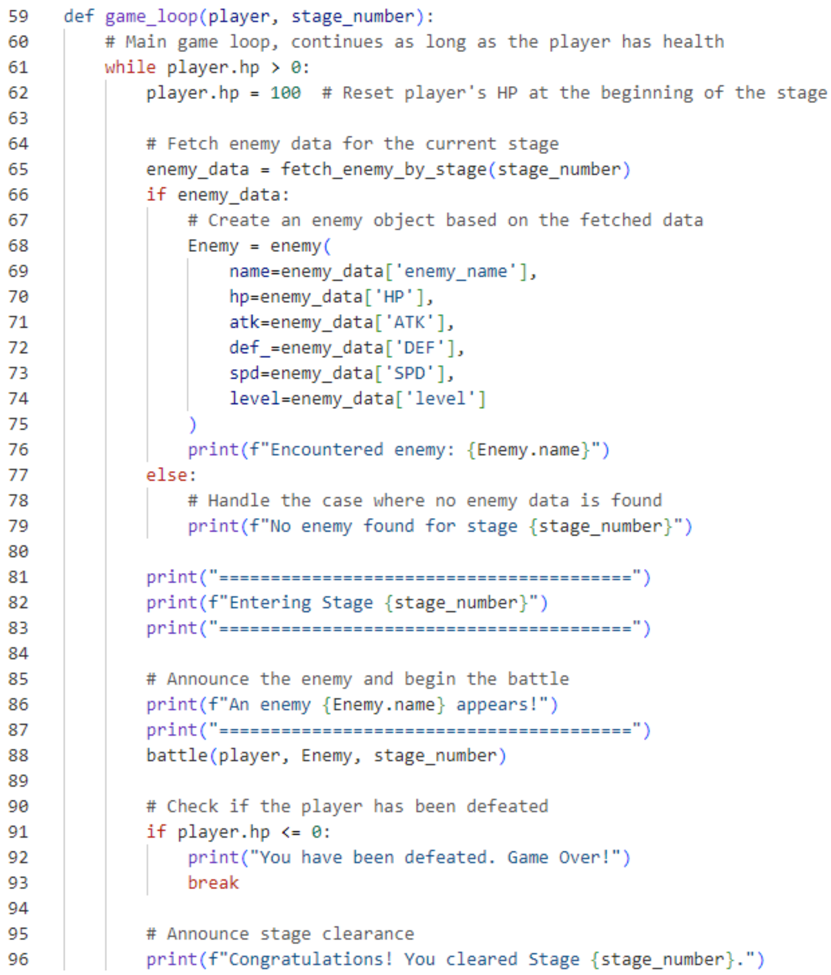


```python
def apply_rewards(player, rewards, Spells):
    # Iterate through the rewards received by the player after clearing a stage
    for reward in rewards:
        if reward['reward_type'] == 'spell':
            # Fetch all available spells from the database
            spell_data = fetch_spells()
            for Spells in spell_data:
                if Spells['spell_id'] == reward['associated_id']:
                    # Create a new spell object and add it to the player's spell list
                    new_spell = spell(
                        id=Spells['spell_id'],
                        name=Spells['spell_name'],
                        cost=Spells['cost'],
                        value=Spells['value'],
                        Type=Spells['Type']
                    )
                    spells.append(new_spell)
                    print(f"Received spell: {new_spell.name}")
        elif reward['reward_type'] == 'item':
            # Fetch all available items from the database
            item_data = fetch_items()
            for items in item_data:
                if items['item_id'] == reward['associated_id']:
                    # Create a new item object and add it to the player's inventory
                    new_item = Item(
                        id=items['item_id'],
                        name=items['item_name'],
                        atk=items['atk'],
                        def_=items['def'],
                        spd=items['spd'],
                        Type=items['Type']
                    )
                    for _ in range(reward['quantity']):
                        player.add_item_to_inventory(new_item)
                    print(f"Received item: {new_item.name} x{reward['quantity']}")
```

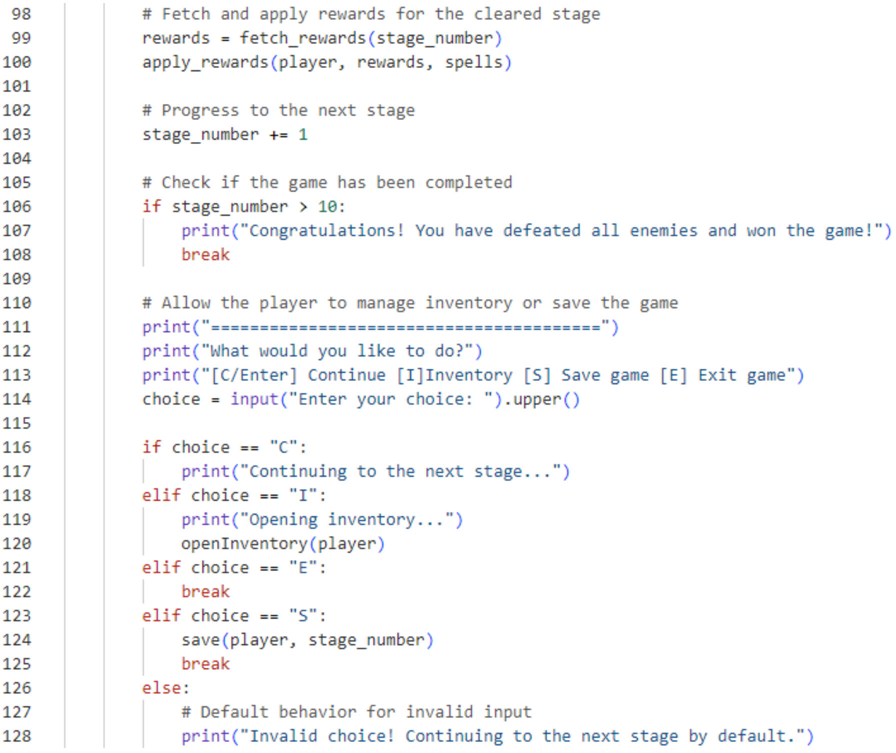


```python
        elif reward['reward_type'] == 'exp':
            # Add experience points to the player and handle level-ups
            player.exp += reward['reward_value']
            print(f"Received {reward['reward_value']} EXP")
            while player.exp >= 100:
                player.exp -= 100
                player.level += 1
                player.hp += 1
                player.mp += 1
                player.atk += 1
                player.def_ += 1
                player.spd += 1
                print(f"Leveled up! Now at level {player.level}")
```

Gameloop.py (game_loop)



```python
def game_loop(player, stage_number):
    # Main game loop, continues as long as the player has health
    while player.hp > 0:
        player.hp = 100  # Reset player's HP at the beginning of the stage

        # Fetch enemy data for the current stage
        enemy_data = fetch_enemy_by_stage(stage_number)
        if enemy_data:
            # Create an enemy object based on the fetched data
            Enemy = enemy(
                name=enemy_data['enemy_name'],
                hp=enemy_data['HP'],
                atk=enemy_data['ATK'],
                def_=enemy_data['DEF'],
                spd=enemy_data['SPD'],
                level=enemy_data['level']
            )
            print(f"Encountered enemy: {Enemy.name}")
        else:
            # Handle the case where no enemy data is found
            print(f"No enemy found for stage {stage_number}")

        print("========================================")
        print(f"Entering Stage {stage_number}")
        print("========================================")

        # Announce the enemy and begin the battle
        print(f"An enemy {Enemy.name} appears!")
        print("========================================")
        battle(player, Enemy, stage_number)

        # Check if the player has been defeated
        if player.hp <= 0:
            print("You have been defeated. Game Over!")
            break

        # Announce stage clearance
        print(f"Congratulations! You cleared Stage {stage_number}.")
```

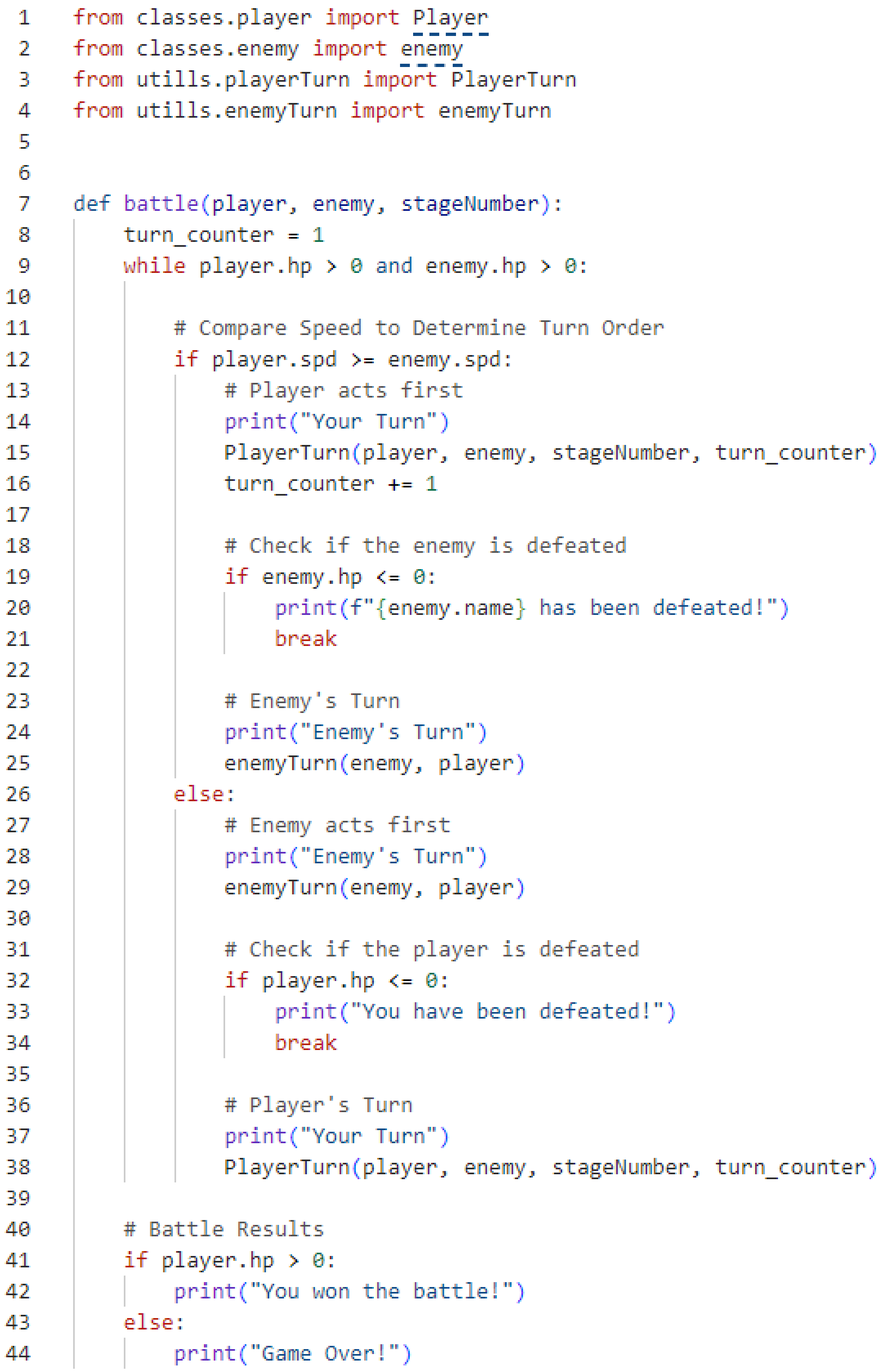


```python
        # Fetch and apply rewards for the cleared stage
        rewards = fetch_rewards(stage_number)
        apply_rewards(player, rewards, spells)

        # Progress to the next stage
        stage_number += 1

        # Check if the game has been completed
        if stage_number > 10:
            print("Congratulations! You have defeated all enemies and won the game!")
            break

        # Allow the player to manage inventory or save the game
        print("========================================")
        print("What would you like to do?")
        print("[C/Enter] Continue [I]Inventory [S] Save game [E] Exit game")
        choice = input("Enter your choice: ").upper()

        if choice == "C":
            print("Continuing to the next stage...")
        elif choice == "I":
            print("Opening inventory...")
            openInventory(player)
        elif choice == "E":
            break
        elif choice == "S":
            save(player, stage_number)
            break
        else:
            # Default behavior for invalid input
            print("Invalid choice! Continuing to the next stage by default.")
```

Log of ongoing testing

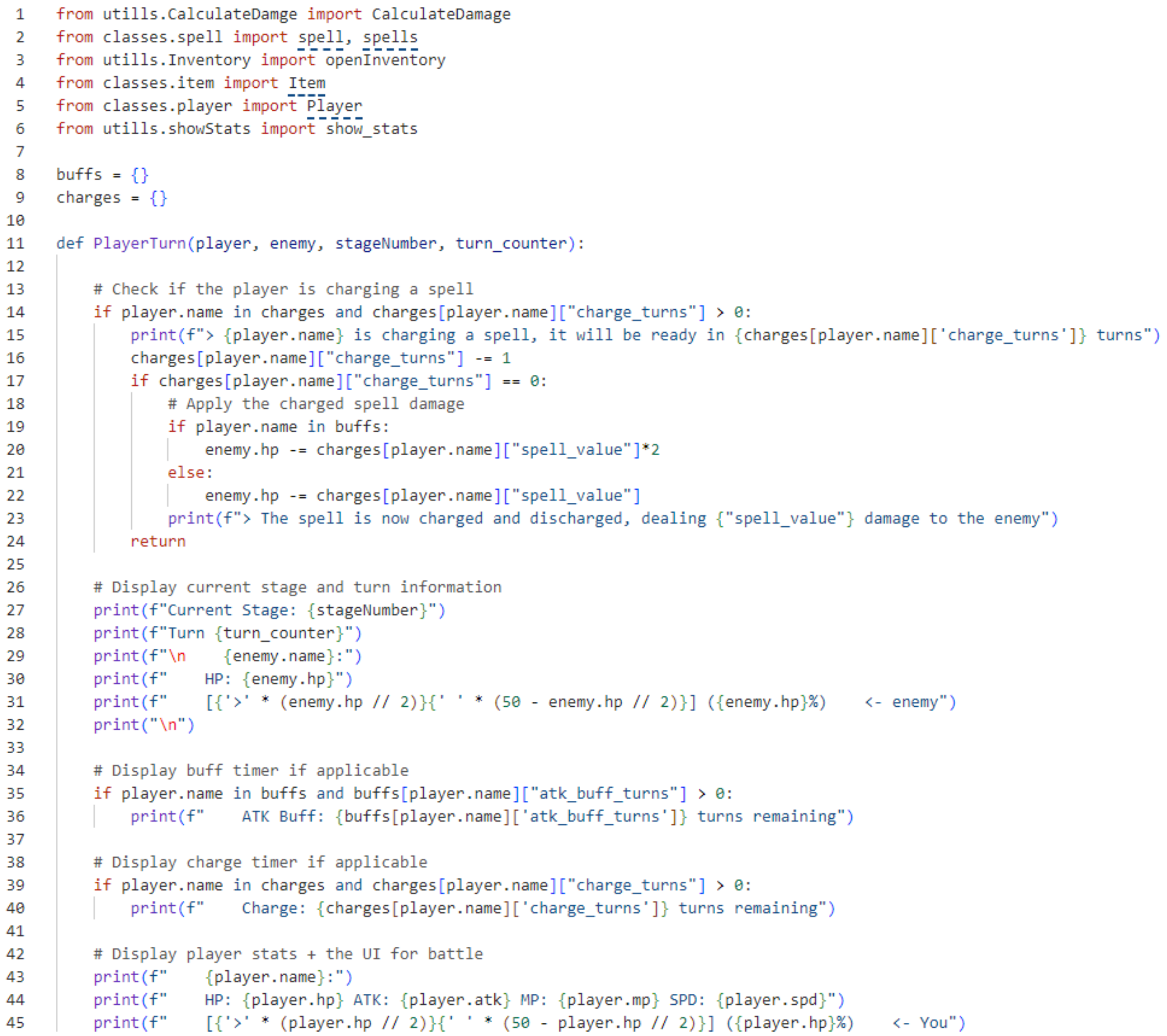
- **Testing : Would the player reset the hp?**
  - For my expectations is that if the player enter “C” then the next stage will load up and the player hp will be reset to 100hp.
  - **Problem :** The hp of the player isn’t reset after entering “C“ but the player’s hp didn’t reset
  - **Reason:** because of the “player.hp = 100” is after the code that continue the gameloop
  - **Solve :** I just sway the order of resetting player’s hp and the code that continue the loop

Battle.py

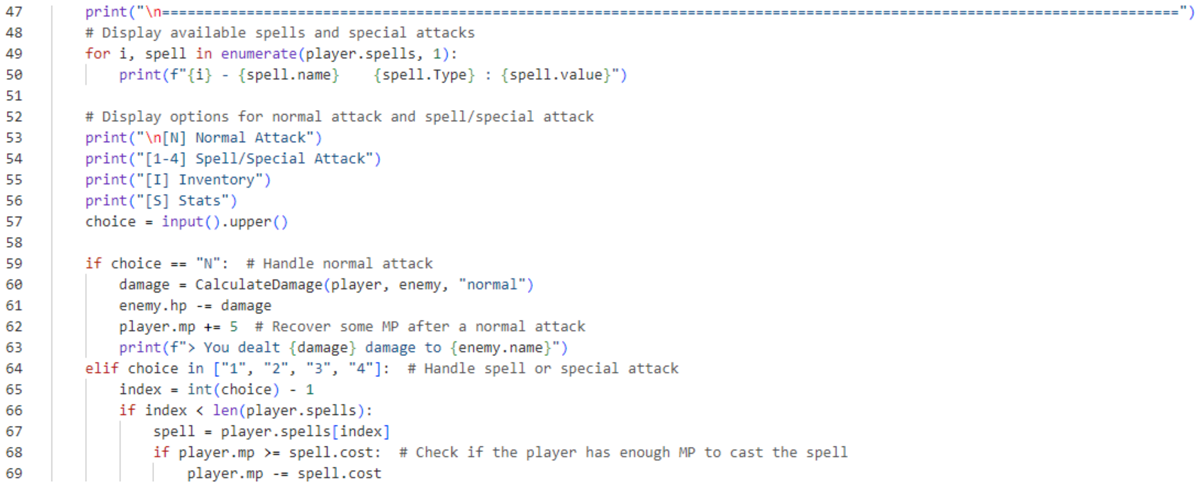


```python
from classes.player import Player
from classes.enemy import enemy
from utills.playerTurn import PlayerTurn
from utills.enemyTurn import enemyTurn


def battle(player, enemy, stageNumber):
    turn_counter = 1
    while player.hp > 0 and enemy.hp > 0:

        # Compare Speed to Determine Turn Order
        if player.spd >= enemy.spd:
            # Player acts first
            print("Your Turn")
            PlayerTurn(player, enemy, stageNumber, turn_counter)
            turn_counter += 1

            # Check if the enemy is defeated
            if enemy.hp <= 0:
                print(f"{enemy.name} has been defeated!")
                break

            # Enemy's Turn
            print("Enemy's Turn")
            enemyTurn(enemy, player)
        else:
            # Enemy acts first
            print("Enemy's Turn")
            enemyTurn(enemy, player)

            # Check if the player is defeated
            if player.hp <= 0:
                print("You have been defeated!")
                break

            # Player's Turn
            print("Your Turn")
            PlayerTurn(player, enemy, stageNumber, turn_counter)

    # Battle Results
    if player.hp > 0:
        print("You won the battle!")
    else:
        print("Game Over!")
```

PlayerTurn.py

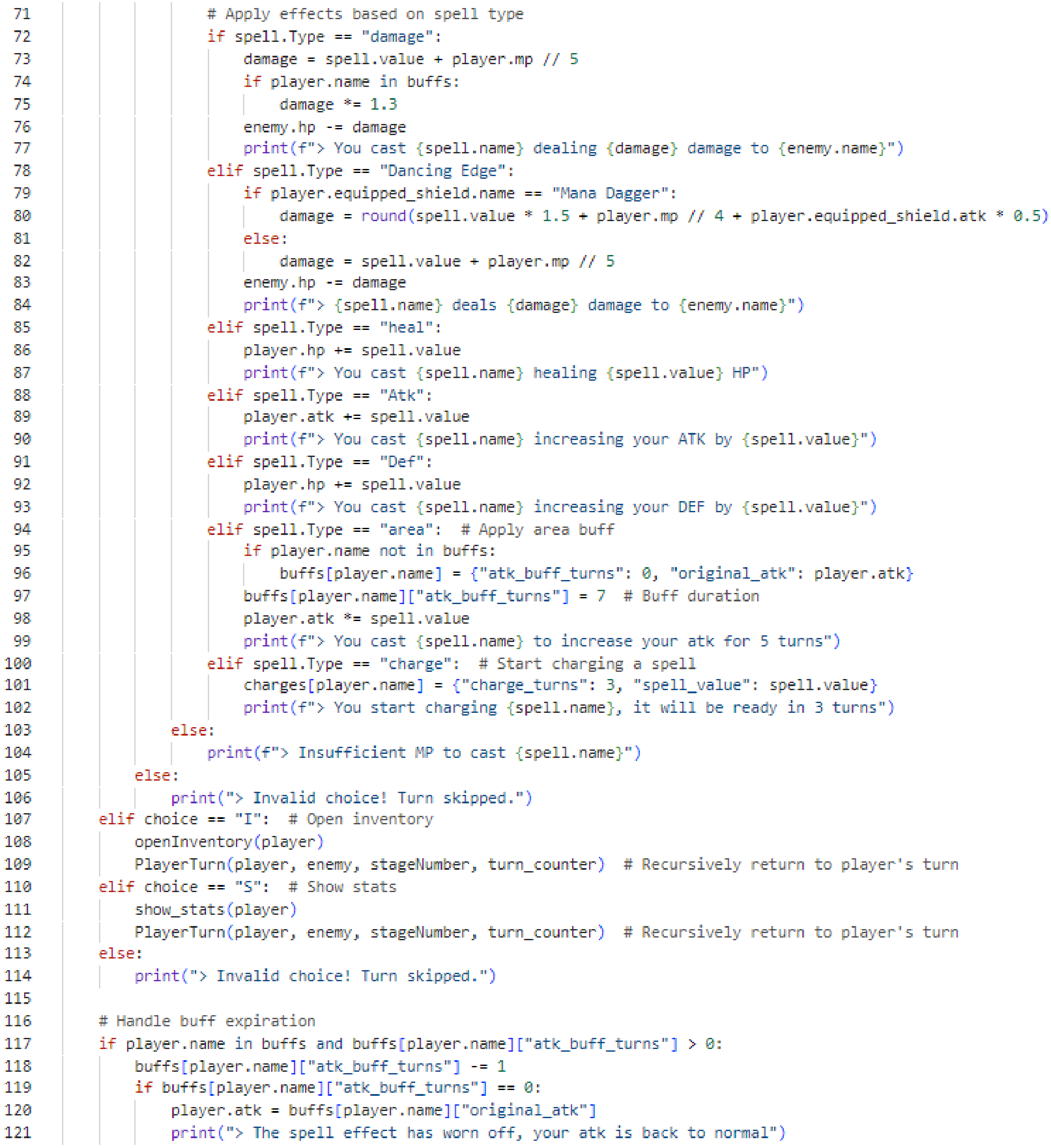


```python
from utills.CalculateDamge import CalculateDamage
from classes.spell import spell, spells
from utills.Inventory import openInventory
from classes.item import Item
from classes.player import Player
from utills.showStats import show_stats

buffs = {}
charges = {}

def PlayerTurn(player, enemy, stageNumber, turn_counter):

    # Check if the player is charging a spell
    if player.name in charges and charges[player.name]["charge_turns"] > 0:
        print(f"> {player.name} is charging a spell, it will be ready in {charges[player.name]['charge_turns']} turns")
        charges[player.name]["charge_turns"] -= 1
        if charges[player.name]["charge_turns"] == 0:
            # Apply the charged spell damage
            if player.name in buffs:
                enemy.hp -= charges[player.name]["spell_value"]*2
            else:
                enemy.hp -= charges[player.name]["spell_value"]
            print(f"> The spell is now charged and discharged, dealing {"spell_value"} damage to the enemy")
        return

    # Display current stage and turn information
    print(f"Current Stage: {stageNumber}")
    print(f"Turn {turn_counter}")
    print(f"\n    {enemy.name}:")
    print(f"    HP: {enemy.hp}")
    print(f"    [{'>' * (enemy.hp // 2)}{' ' * (50 - enemy.hp // 2)}] ({enemy.hp}%)    <- enemy")
    print("\n")

    # Display buff timer if applicable
    if player.name in buffs and buffs[player.name]["atk_buff_turns"] > 0:
        print(f"    ATK Buff: {buffs[player.name]['atk_buff_turns']} turns remaining")

    # Display charge timer if applicable
    if player.name in charges and charges[player.name]["charge_turns"] > 0:
        print(f"    Charge: {charges[player.name]['charge_turns']} turns remaining")

    # Display player stats + the UI for battle
    print(f"    {player.name}:")
    print(f"    HP: {player.hp} ATK: {player.atk} MP: {player.mp} SPD: {player.spd}")
    print(f"    [{'>' * (player.hp // 2)}{' ' * (50 - player.hp // 2)}] ({player.hp}%)    <- You")
```



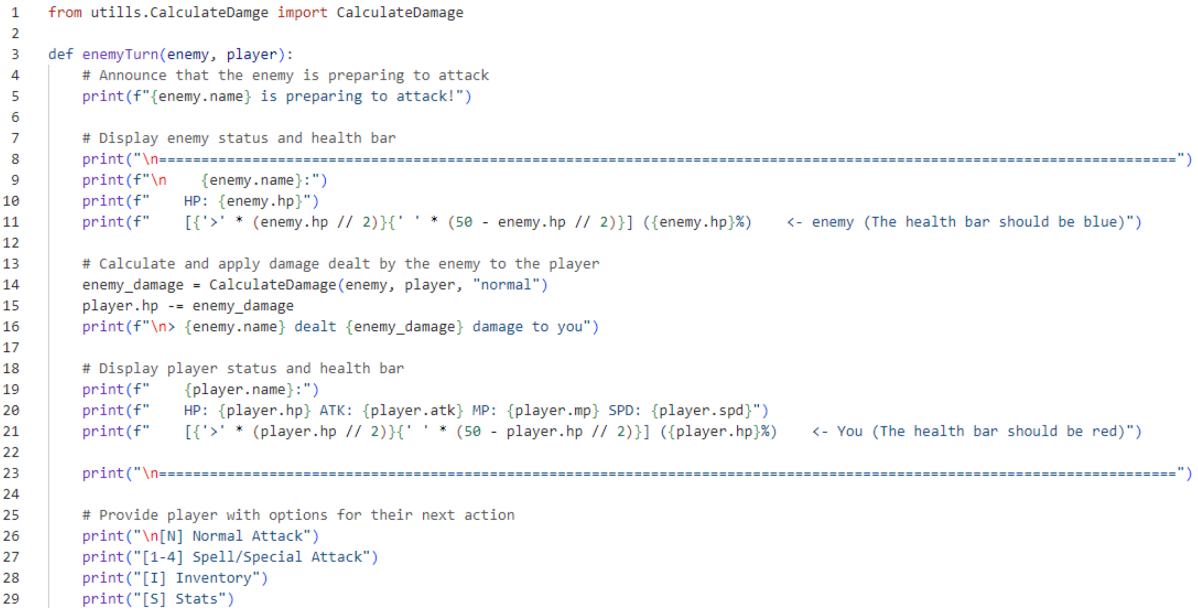
```python
    print("\n========================================================================================================================")
    # Display available spells and special attacks
    for i, spell in enumerate(player.spells, 1):
        print(f"{i} - {spell.name}    {spell.Type} : {spell.value}")

    # Display options for normal attack and spell/special attack
    print("\n[N] Normal Attack")
    print("[1-4] Spell/Special Attack")
    print("[I] Inventory")
    print("[S] Stats")
    choice = input().upper()

    if choice == "N":  # Handle normal attack
        damage = CalculateDamage(player, enemy, "normal")
        enemy.hp -= damage
        player.mp += 5  # Recover some MP after a normal attack
        print(f"> You dealt {damage} damage to {enemy.name}")
    elif choice in ["1", "2", "3", "4"]:  # Handle spell or special attack
        index = int(choice) - 1
        if index < len(player.spells):
            spell = player.spells[index]
            if player.mp >= spell.cost:  # Check if the player has enough MP to cast the spell
                player.mp -= spell.cost
```

PlayerTurn.py (continue)



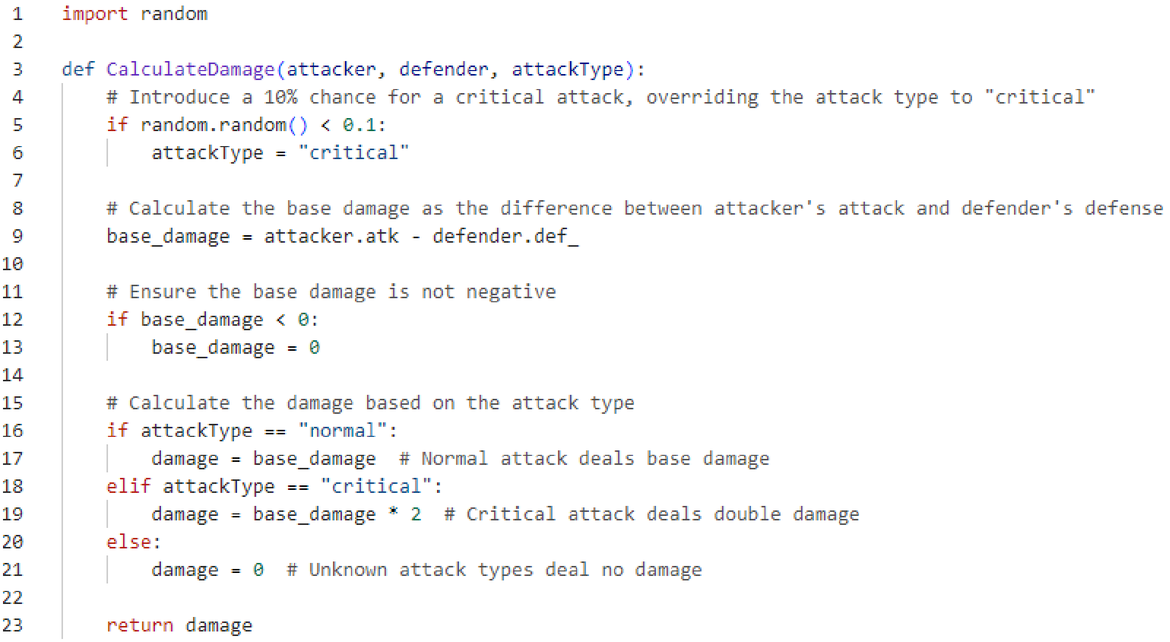
```python
                # Apply effects based on spell type
                if spell.Type == "damage":
                    damage = spell.value + player.mp // 5
                    if player.name in buffs:
                        damage *= 1.3
                    enemy.hp -= damage
                    print(f"> You cast {spell.name} dealing {damage} damage to {enemy.name}")
                elif spell.Type == "Dancing Edge":
                    if player.equipped_shield.name == "Mana Dagger":
                        damage = round(spell.value * 1.5 + player.mp // 4 + player.equipped_shield.atk * 0.5)
                    else:
                        damage = spell.value + player.mp // 5
                    enemy.hp -= damage
                    print(f"> {spell.name} deals {damage} damage to {enemy.name}")
                elif spell.Type == "heal":
                    player.hp += spell.value
                    print(f"> You cast {spell.name} healing {spell.value} HP")
                elif spell.Type == "Atk":
                    player.atk += spell.value
                    print(f"> You cast {spell.name} increasing your ATK by {spell.value}")
                elif spell.Type == "Def":
                    player.hp += spell.value
                    print(f"> You cast {spell.name} increasing your DEF by {spell.value}")
                elif spell.Type == "area":  # Apply area buff
                    if player.name not in buffs:
                        buffs[player.name] = {"atk_buff_turns": 0, "original_atk": player.atk}
                    buffs[player.name]["atk_buff_turns"] = 7  # Buff duration
                    player.atk *= spell.value
                    print(f"> You cast {spell.name} to increase your atk for 5 turns")
                elif spell.Type == "charge":  # Start charging a spell
                    charges[player.name] = {"charge_turns": 3, "spell_value": spell.value}
                    print(f"> You start charging {spell.name}, it will be ready in 3 turns")
            else:
                print(f"> Insufficient MP to cast {spell.name}")
        else:
            print("> Invalid choice! Turn skipped.")
    elif choice == "I":  # Open inventory
        openInventory(player)
        PlayerTurn(player, enemy, stageNumber, turn_counter)  # Recursively return to player's turn
    elif choice == "S":  # Show stats
        show_stats(player)
        PlayerTurn(player, enemy, stageNumber, turn_counter)  # Recursively return to player's turn
    else:
        print("> Invalid choice! Turn skipped.")

    # Handle buff expiration
    if player.name in buffs and buffs[player.name]["atk_buff_turns"] > 0:
        buffs[player.name]["atk_buff_turns"] -= 1
        if buffs[player.name]["atk_buff_turns"] == 0:
            player.atk = buffs[player.name]["original_atk"]
            print("> The spell effect has worn off, your atk is back to normal")
```

enemyTurn.py



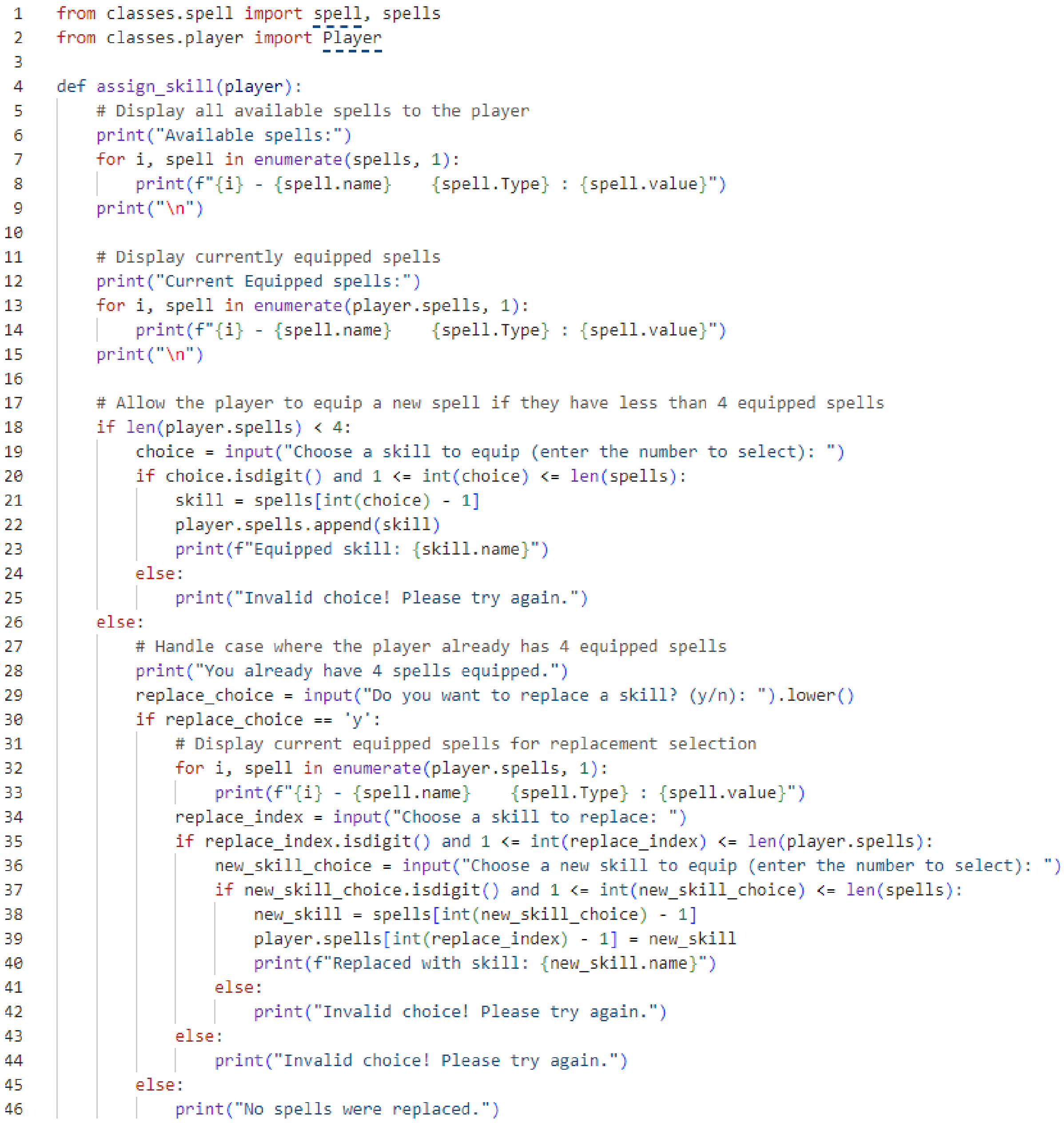
```python
from utills.CalculateDamge import CalculateDamage

def enemyTurn(enemy, player):
    # Announce that the enemy is preparing to attack
    print(f"{enemy.name} is preparing to attack!")

    # Display enemy status and health bar
    print("\n========================================================================================================================")
    print(f"\n    {enemy.name}:")
    print(f"    HP: {enemy.hp}")
    print(f"    [{'>' * (enemy.hp // 2)}{' ' * (50 - enemy.hp // 2)}] ({enemy.hp}%)    <- enemy (The health bar should be blue)")

    # Calculate and apply damage dealt by the enemy to the player
    enemy_damage = CalculateDamage(enemy, player, "normal")
    player.hp -= enemy_damage
    print(f"\n> {enemy.name} dealt {enemy_damage} damage to you")

    # Display player status and health bar
    print(f"    {player.name}:")
    print(f"    HP: {player.hp} ATK: {player.atk} MP: {player.mp} SPD: {player.spd}")
    print(f"    [{'>' * (player.hp // 2)}{' ' * (50 - player.hp // 2)}] ({player.hp}%)    <- You (The health bar should be red)")

    print("\n========================================================================================================================")

    # Provide player with options for their next action
    print("\n[N] Normal Attack")
    print("[1-4] Spell/Special Attack")
    print("[I] Inventory")
    print("[S] Stats")
```

CalculateDamage.py



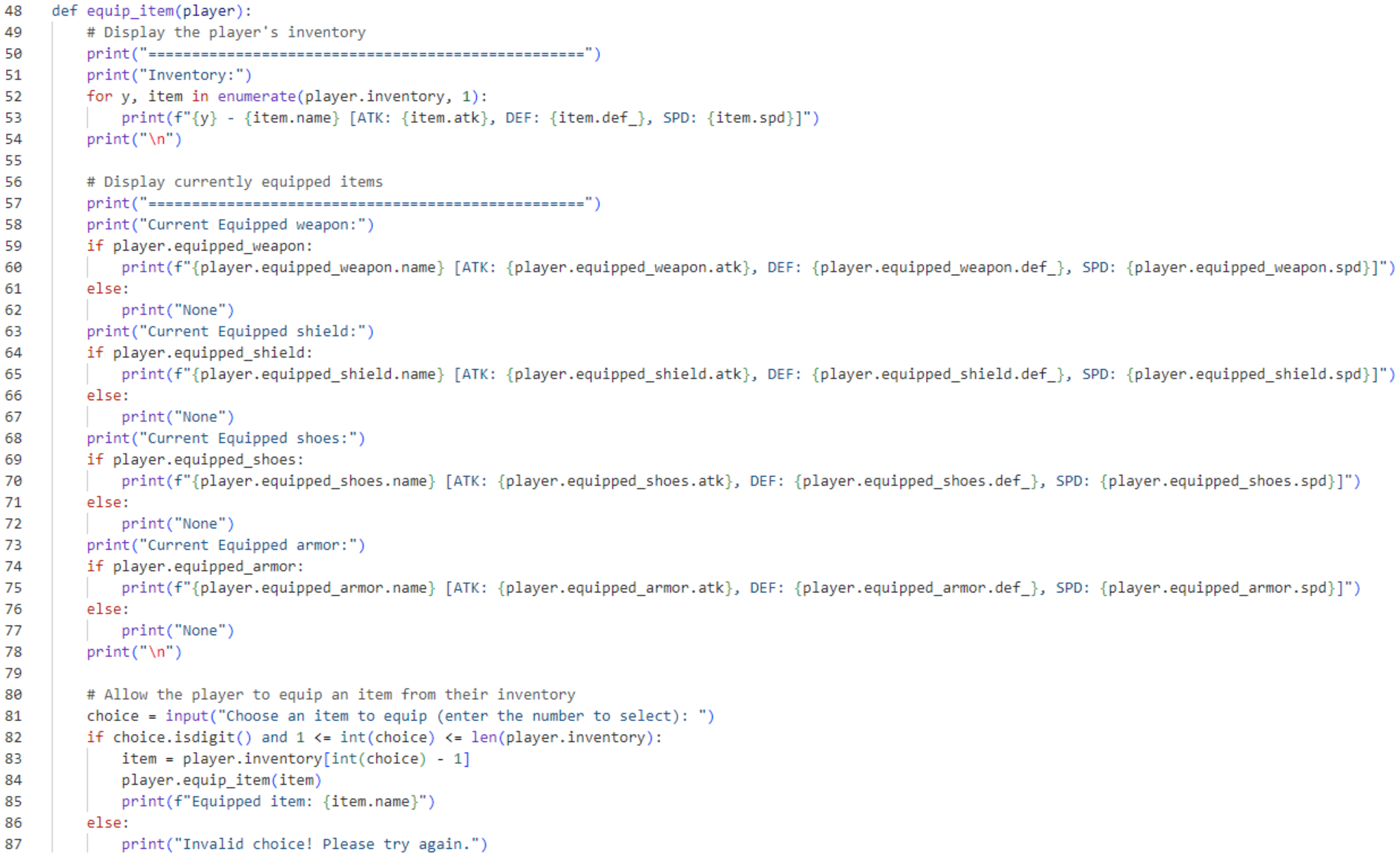
```python
import random

def CalculateDamage(attacker, defender, attackType):
    # Introduce a 10% chance for a critical attack, overriding the attack type to "critical"
    if random.random() < 0.1:
        attackType = "critical"

    # Calculate the base damage as the difference between attacker's attack and defender's defense
    base_damage = attacker.atk - defender.def_

    # Ensure the base damage is not negative
    if base_damage < 0:
        base_damage = 0

    # Calculate the damage based on the attack type
    if attackType == "normal":
        damage = base_damage  # Normal attack deals base damage
    elif attackType == "critical":
        damage = base_damage * 2  # Critical attack deals double damage
    else:
        damage = 0  # Unknown attack types deal no damage

    return damage
```

Asign.py (import & asign_skill)

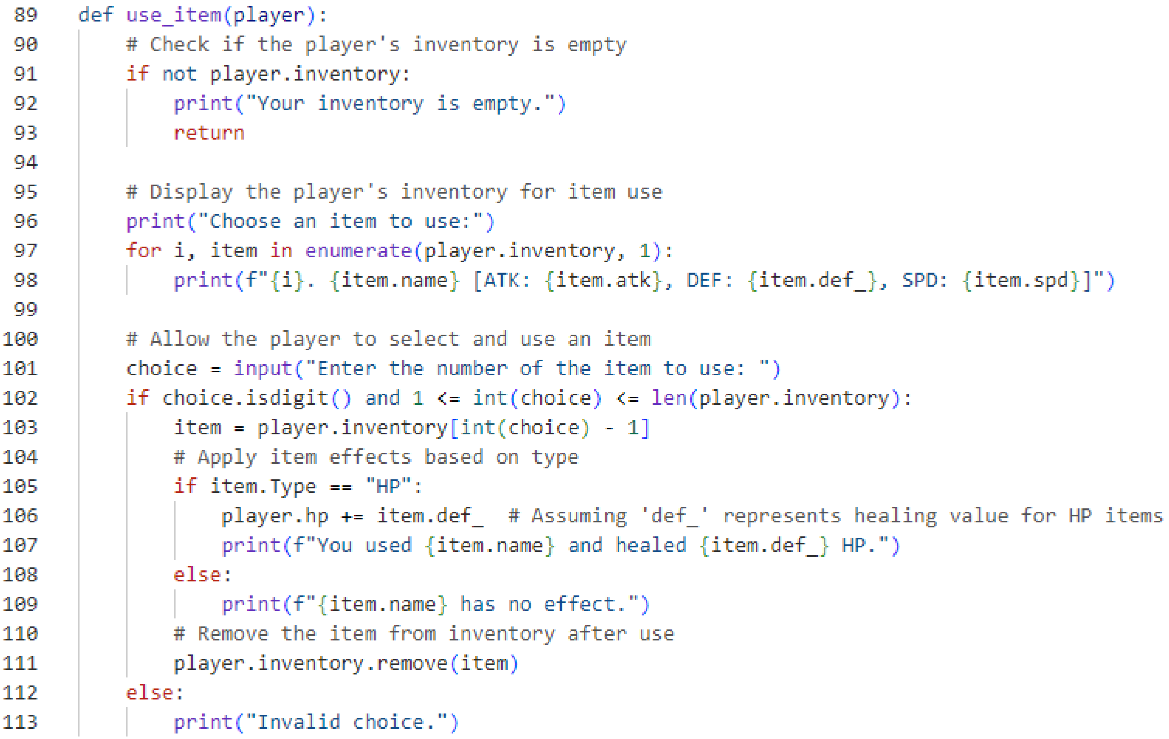


```python
from classes.spell import spell, spells
from classes.player import Player

def assign_skill(player):
    # Display all available spells to the player
    print("Available spells:")
    for i, spell in enumerate(spells, 1):
        print(f"{i} - {spell.name}    {spell.Type} : {spell.value}")
    print("\n")

    # Display currently equipped spells
    print("Current Equipped spells:")
    for i, spell in enumerate(player.spells, 1):
        print(f"{i} - {spell.name}    {spell.Type} : {spell.value}")
    print("\n")

    # Allow the player to equip a new spell if they have less than 4 equipped spells
    if len(player.spells) < 4:
        choice = input("Choose a skill to equip (enter the number to select): ")
        if choice.isdigit() and 1 <= int(choice) <= len(spells):
            skill = spells[int(choice) - 1]
            player.spells.append(skill)
            print(f"Equipped skill: {skill.name}")
        else:
            print("Invalid choice! Please try again.")
    else:
        # Handle case where the player already has 4 equipped spells
        print("You already have 4 spells equipped.")
        replace_choice = input("Do you want to replace a skill? (y/n): ").lower()
        if replace_choice == 'y':
            # Display current equipped spells for replacement selection
            for i, spell in enumerate(player.spells, 1):
                print(f"{i} - {spell.name}    {spell.Type} : {spell.value}")
            replace_index = input("Choose a skill to replace: ")
            if replace_index.isdigit() and 1 <= int(replace_index) <= len(player.spells):
                new_skill_choice = input("Choose a new skill to equip (enter the number to select): ")
                if new_skill_choice.isdigit() and 1 <= int(new_skill_choice) <= len(spells):
                    new_skill = spells[int(new_skill_choice) - 1]
                    player.spells[int(replace_index) - 1] = new_skill
                    print(f"Replaced with skill: {new_skill.name}")
                else:
                    print("Invalid choice! Please try again.")
            else:
                print("Invalid choice! Please try again.")
        else:
            print("No spells were replaced.")
```

Asign.py (equip_item & use_item)

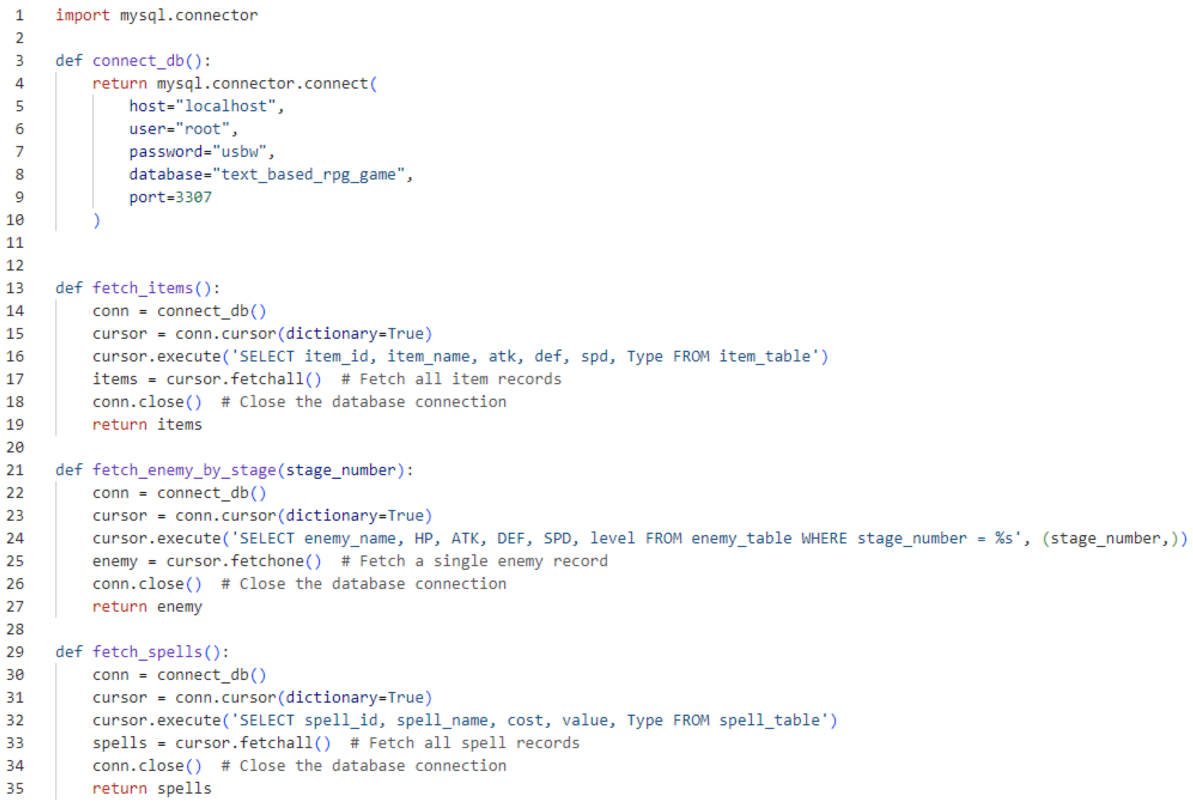


```python
def equip_item(player):
    # Display the player's inventory
    print("==================================================")
    print("Inventory:")
    for y, item in enumerate(player.inventory, 1):
        print(f"{y} - {item.name} [ATK: {item.atk}, DEF: {item.def_}, SPD: {item.spd}]")
    print("\n")

    # Display currently equipped items
    print("==================================================")
    print("Current Equipped weapon:")
    if player.equipped_weapon:
        print(f"{player.equipped_weapon.name} [ATK: {player.equipped_weapon.atk}, DEF: {player.equipped_weapon.def_}, SPD: {player.equipped_weapon.spd}]")
    else:
        print("None")
    print("Current Equipped shield:")
    if player.equipped_shield:
        print(f"{player.equipped_shield.name} [ATK: {player.equipped_shield.atk}, DEF: {player.equipped_shield.def_}, SPD: {player.equipped_shield.spd}]")
    else:
        print("None")
    print("Current Equipped shoes:")
    if player.equipped_shoes:
        print(f"{player.equipped_shoes.name} [ATK: {player.equipped_shoes.atk}, DEF: {player.equipped_shoes.def_}, SPD: {player.equipped_shoes.spd}]")
    else:
        print("None")
    print("Current Equipped armor:")
    if player.equipped_armor:
        print(f"{player.equipped_armor.name} [ATK: {player.equipped_armor.atk}, DEF: {player.equipped_armor.def_}, SPD: {player.equipped_armor.spd}]")
    else:
        print("None")
    print("\n")

    # Allow the player to equip an item from their inventory
    choice = input("Choose an item to equip (enter the number to select): ")
    if choice.isdigit() and 1 <= int(choice) <= len(player.inventory):
        item = player.inventory[int(choice) - 1]
        player.equip_item(item)
        print(f"Equipped item: {item.name}")
    else:
        print("Invalid choice! Please try again.")
```

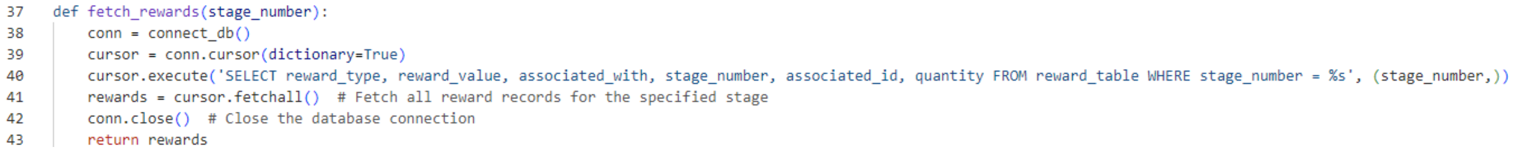


```python
def use_item(player):
    # Check if the player's inventory is empty
    if not player.inventory:
        print("Your inventory is empty.")
        return

    # Display the player's inventory for item use
    print("Choose an item to use:")
    for i, item in enumerate(player.inventory, 1):
        print(f"{i}. {item.name} [ATK: {item.atk}, DEF: {item.def_}, SPD: {item.spd}]")

    # Allow the player to select and use an item
    choice = input("Enter the number of the item to use: ")
    if choice.isdigit() and 1 <= int(choice) <= len(player.inventory):
        item = player.inventory[int(choice) - 1]
        # Apply item effects based on type
        if item.Type == "HP":
            player.hp += item.def_  # Assuming 'def_' represents healing value for HP items
            print(f"You used {item.name} and healed {item.def_} HP.")
        else:
            print(f"{item.name} has no effect.")
        # Remove the item from inventory after use
        player.inventory.remove(item)
    else:
        print("Invalid choice.")
```

Database.py



```python
import mysql.connector

def connect_db():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="usbw",
        database="text_based_rpg_game",
        port=3307
    )


def fetch_items():
    conn = connect_db()
    cursor = conn.cursor(dictionary=True)
    cursor.execute('SELECT item_id, item_name, atk, def, spd, Type FROM item_table')
    items = cursor.fetchall()  # Fetch all item records
    conn.close()  # Close the database connection
    return items

def fetch_enemy_by_stage(stage_number):
    conn = connect_db()
    cursor = conn.cursor(dictionary=True)
    cursor.execute('SELECT enemy_name, HP, ATK, DEF, SPD, level FROM enemy_table WHERE stage_number = %s', (stage_number,))
    enemy = cursor.fetchone()  # Fetch a single enemy record
    conn.close()  # Close the database connection
    return enemy

def fetch_spells():
    conn = connect_db()
    cursor = conn.cursor(dictionary=True)
    cursor.execute('SELECT spell_id, spell_name, cost, value, Type FROM spell_table')
    spells = cursor.fetchall()  # Fetch all spell records
    conn.close()  # Close the database connection
    return spells
```



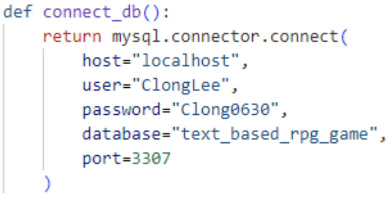
```python
def fetch_rewards(stage_number):
    conn = connect_db()
    cursor = conn.cursor(dictionary=True)
    cursor.execute('SELECT reward_type, reward_value, associated_with, stage_number, associated_id, quantity FROM reward_table WHERE stage_number = %s', (stage_number,))
    rewards = cursor.fetchall()  # Fetch all reward records for the specified stage
    conn.close()  # Close the database connection
    return rewards
```

Log of ongoing testing

- **Testing :** connect to the database
  - **Problem :** Can't connect to the database
  - **Situation :** For that time I have used "ClongLee" for the username of the database and use "Clong0630" for the password, and after activating the program error message has show up and the database isn't connected to the program.
  - **Solve :** I have used back the default Username and password to connect the database.
  - **References that help me :** https://codeytek.com/connect-python-to-mysql-database-with-pymysql-and-phpmyadmin/

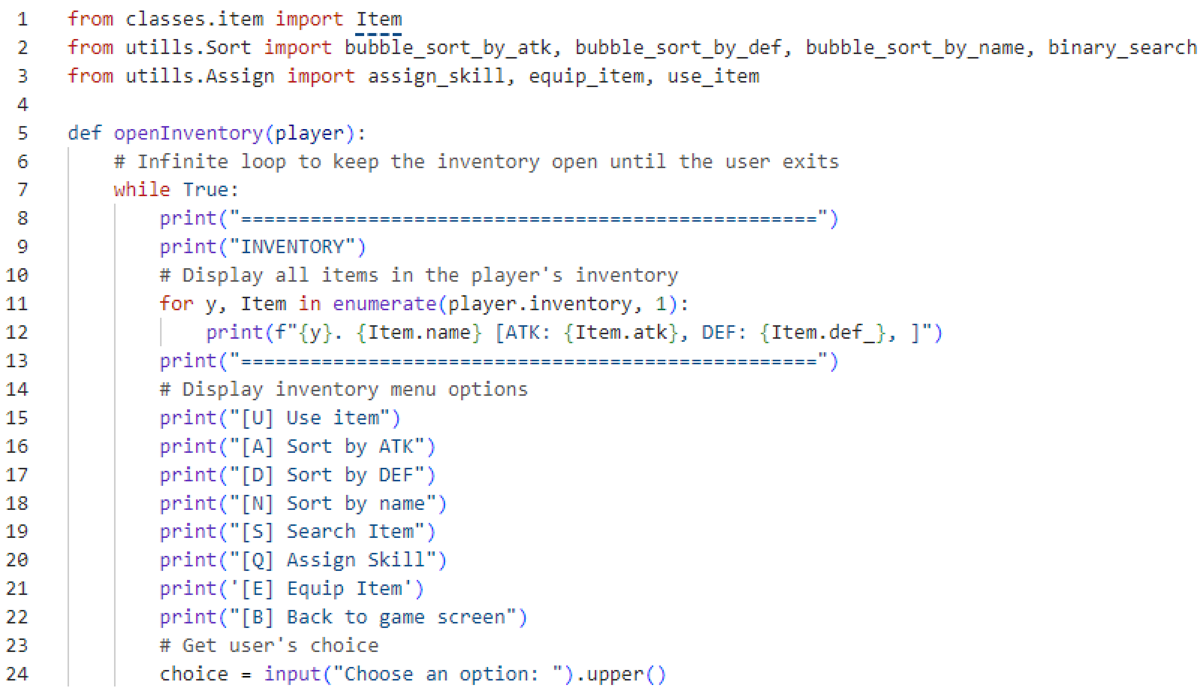
    When I look up for solve, this website reminds me that I could use the default user to connect the database

    Error code:

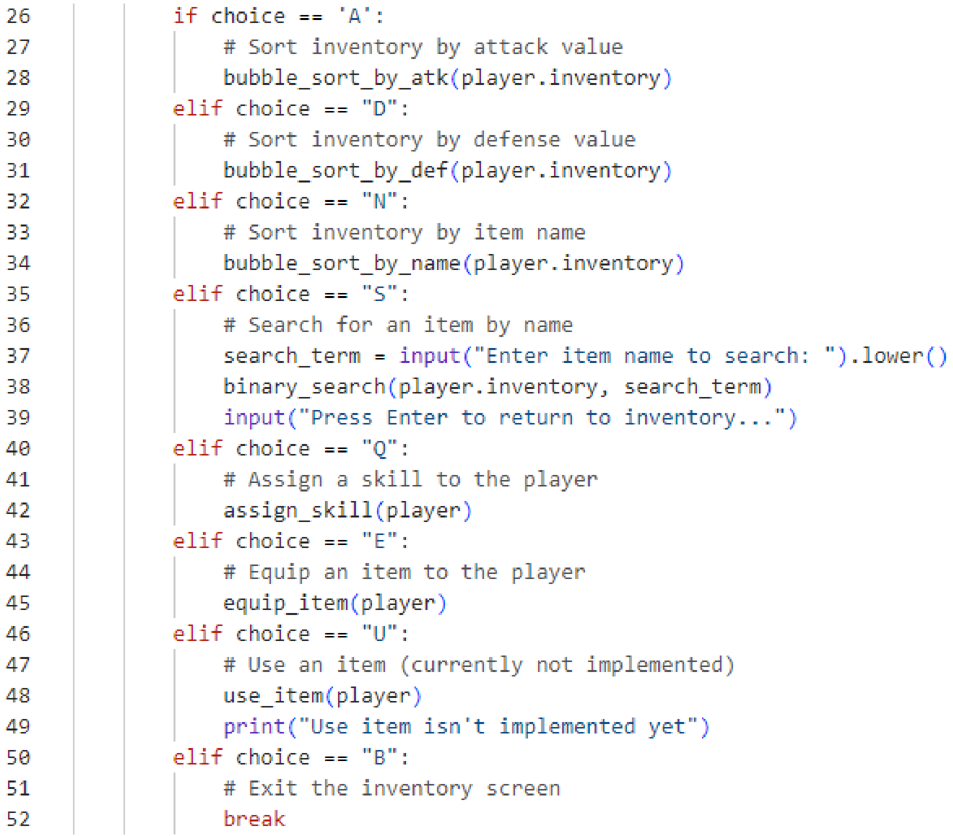


```python
def connect_db():
    return mysql.connector.connect(
        host="localhost",
        user="ClongLee",
        password="Clong0630",
        database="text_based_rpg_game",
        port=3307
    )
```

Inventory.py

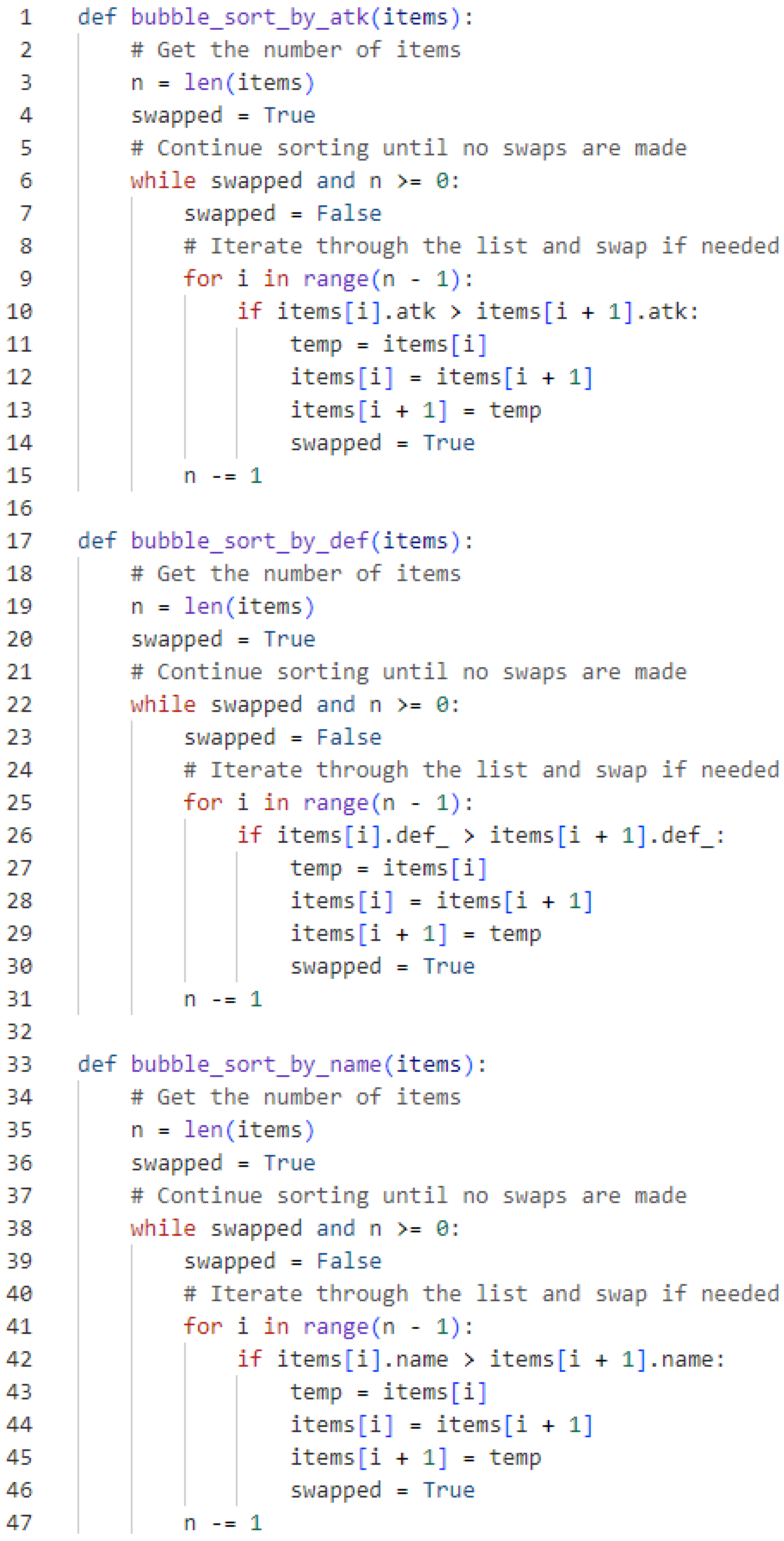


```python
from classes.item import Item
from utills.Sort import bubble_sort_by_atk, bubble_sort_by_def, bubble_sort_by_name, binary_search
from utills.Assign import assign_skill, equip_item, use_item

def openInventory(player):
    # Infinite loop to keep the inventory open until the user exits
    while True:
        print("==================================================")
        print("INVENTORY")
        # Display all items in the player's inventory
        for y, Item in enumerate(player.inventory, 1):
            print(f"{y}. {Item.name} [ATK: {Item.atk}, DEF: {Item.def_}, ]")
        print("==================================================")
        # Display inventory menu options
        print("[U] Use item")
        print("[A] Sort by ATK")
        print("[D] Sort by DEF")
        print("[N] Sort by name")
        print("[S] Search Item")
        print("[Q] Assign Skill")
        print('[E] Equip Item')
        print("[B] Back to game screen")
        # Get user's choice
        choice = input("Choose an option: ").upper()
```



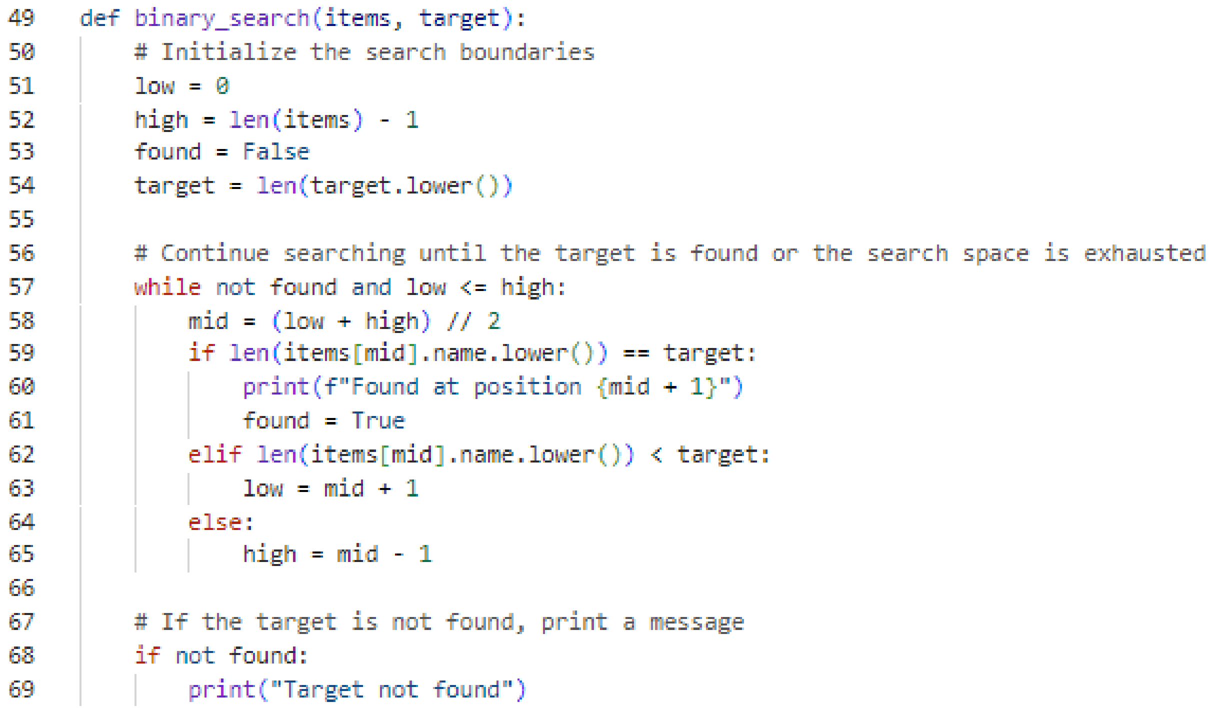
```python
        if choice == 'A':
            # Sort inventory by attack value
            bubble_sort_by_atk(player.inventory)
        elif choice == "D":
            # Sort inventory by defense value
            bubble_sort_by_def(player.inventory)
        elif choice == "N":
            # Sort inventory by item name
            bubble_sort_by_name(player.inventory)
        elif choice == "S":
            # Search for an item by name
            search_term = input("Enter item name to search: ").lower()
            binary_search(player.inventory, search_term)
            input("Press Enter to return to inventory...")
        elif choice == "Q":
            # Assign a skill to the player
            assign_skill(player)
        elif choice == "E":
            # Equip an item to the player
            equip_item(player)
        elif choice == "U":
            # Use an item (currently not implemented)
            use_item(player)
            print("Use item isn't implemented yet")
        elif choice == "B":
            # Exit the inventory screen
            break
```

Sort.py (bubble sort)



```python
def bubble_sort_by_atk(items):
    # Get the number of items
    n = len(items)
    swapped = True
    # Continue sorting until no swaps are made
    while swapped and n >= 0:
        swapped = False
        # Iterate through the list and swap if needed
        for i in range(n - 1):
            if items[i].atk > items[i + 1].atk:
                temp = items[i]
                items[i] = items[i + 1]
                items[i + 1] = temp
                swapped = True
        n -= 1

def bubble_sort_by_def(items):
    # Get the number of items
    n = len(items)
    swapped = True
    # Continue sorting until no swaps are made
    while swapped and n >= 0:
        swapped = False
        # Iterate through the list and swap if needed
        for i in range(n - 1):
            if items[i].def_ > items[i + 1].def_:
                temp = items[i]
                items[i] = items[i + 1]
                items[i + 1] = temp
                swapped = True
        n -= 1

def bubble_sort_by_name(items):
    # Get the number of items
    n = len(items)
    swapped = True
    # Continue sorting until no swaps are made
    while swapped and n >= 0:
        swapped = False
        # Iterate through the list and swap if needed
        for i in range(n - 1):
            if items[i].name > items[i + 1].name:
                temp = items[i]
                items[i] = items[i + 1]
                items[i + 1] = temp
                swapped = True
        n -= 1
```

Sort.py (binary search)

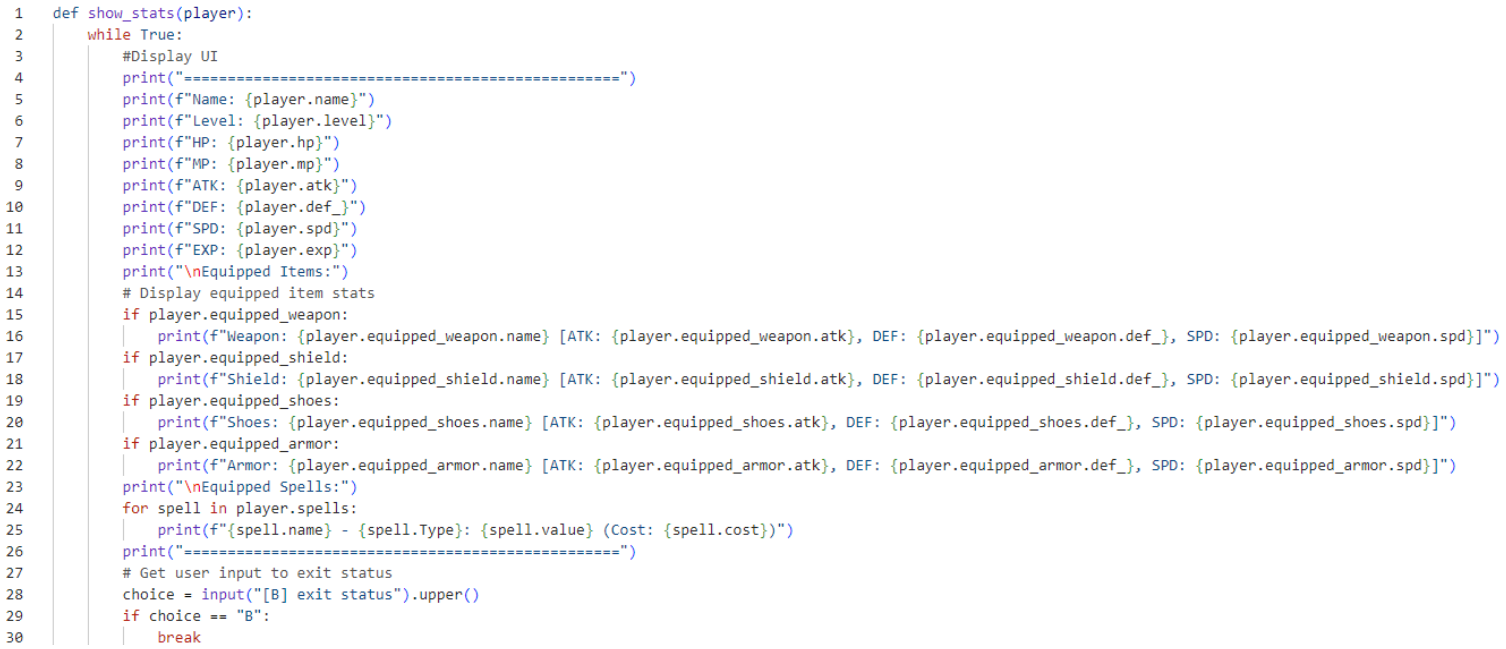


```python
def binary_search(items, target):
    # Initialize the search boundaries
    low = 0
    high = len(items) - 1
    found = False
    target = len(target.lower())

    # Continue searching until the target is found or the search space is exhausted
    while not found and low <= high:
        mid = (low + high) // 2
        if len(items[mid].name.lower()) == target:
            print(f"Found at position {mid + 1}")
            found = True
        elif len(items[mid].name.lower()) < target:
            low = mid + 1
        else:
            high = mid - 1

    # If the target is not found, print a message
    if not found:
        print("Target not found")
```

Log of ongoing testing

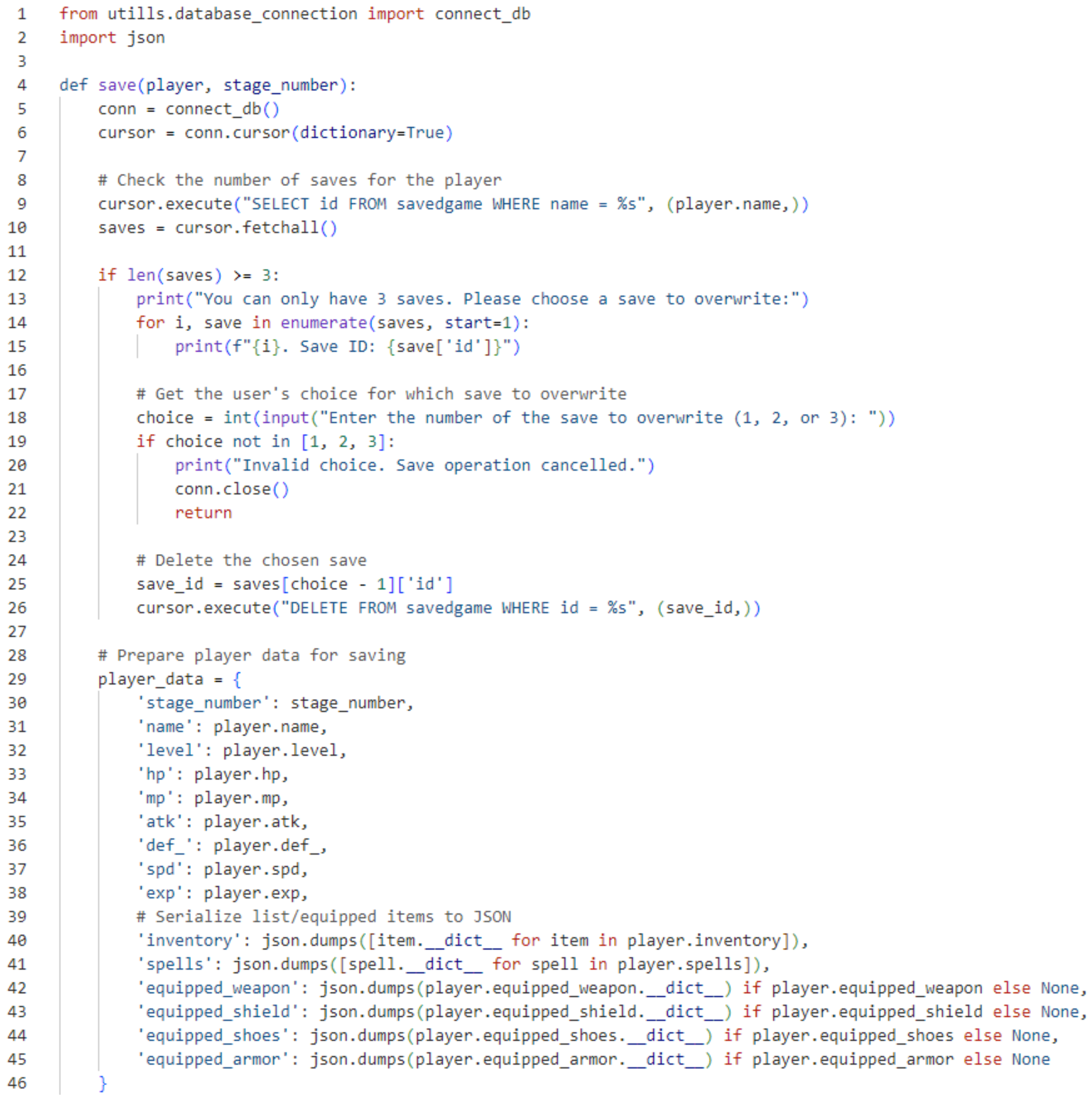
- **Testing :** Try to find the wanted item position
  - **Problem :** After entering in the item name the output in the terminal is “Target not found”
  - **Reason :** The first item that will be used to compare with the target is the middle item in the inventory, and the target and the item[mid] is both string, so this can really compare, so the outcome will be “Not found”
  - **Solve :** I have change the target and the item[mid] from a string to the item name’s length
  - **Reference that help me :** https://stackoverflow.com/questions/21714485/binary-search-for-name This forum post inspired me that I could compare the length of the name to get the correct output.

ShowStats.py

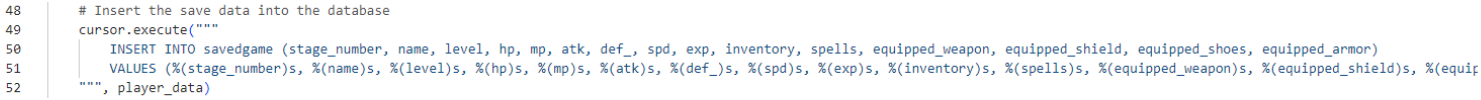


```python
def show_stats(player):
    while True:
        #Display UI
        print("================================================")
        print(f"Name: {player.name}")
        print(f"Level: {player.level}")
        print(f"HP: {player.hp}")
        print(f"MP: {player.mp}")
        print(f"ATK: {player.atk}")
        print(f"DEF: {player.def_}")
        print(f"SPD: {player.spd}")
        print(f"EXP: {player.exp}")
        print("\nEquipped Items:")
        # Display equipped item stats
        if player.equipped_weapon:
            print(f"Weapon: {player.equipped_weapon.name} [ATK: {player.equipped_weapon.atk}, DEF: {player.equipped_weapon.def_}, SPD: {player.equipped_weapon.spd}]")
        if player.equipped_shield:
            print(f"Shield: {player.equipped_shield.name} [ATK: {player.equipped_shield.atk}, DEF: {player.equipped_shield.def_}, SPD: {player.equipped_shield.spd}]")
        if player.equipped_shoes:
            print(f"Shoes: {player.equipped_shoes.name} [ATK: {player.equipped_shoes.atk}, DEF: {player.equipped_shoes.def_}, SPD: {player.equipped_shoes.spd}]")
        if player.equipped_armor:
            print(f"Armor: {player.equipped_armor.name} [ATK: {player.equipped_armor.atk}, DEF: {player.equipped_armor.def_}, SPD: {player.equipped_armor.spd}]")
        print("\nEquipped Spells:")
        for spell in player.spells:
            print(f"{spell.name} - {spell.Type}: {spell.value} (Cost: {spell.cost})")
        print("================================================")
        # Get user input to exit status
        choice = input("[B] exit status").upper()
        if choice == "B":
            break
```

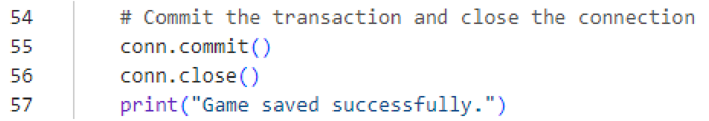
Save.py



```python
from utills.database_connection import connect_db
import json

def save(player, stage_number):
    conn = connect_db()
    cursor = conn.cursor(dictionary=True)

    # Check the number of saves for the player
    cursor.execute("SELECT id FROM savedgame WHERE name = %s", (player.name,))
    saves = cursor.fetchall()

    if len(saves) >= 3:
        print("You can only have 3 saves. Please choose a save to overwrite:")
        for i, save in enumerate(saves, start=1):
            print(f"{i}. Save ID: {save['id']}")

        # Get the user's choice for which save to overwrite
        choice = int(input("Enter the number of the save to overwrite (1, 2, or 3): "))
        if choice not in [1, 2, 3]:
            print("Invalid choice. Save operation cancelled.")
            conn.close()
            return

        # Delete the chosen save
        save_id = saves[choice - 1]['id']
        cursor.execute("DELETE FROM savedgame WHERE id = %s", (save_id,))

    # Prepare player data for saving
    player_data = {
        'stage_number': stage_number,
        'name': player.name,
        'level': player.level,
        'hp': player.hp,
        'mp': player.mp,
        'atk': player.atk,
        'def_': player.def_,
        'spd': player.spd,
        'exp': player.exp,
        # Serialize list/equipped items to JSON
        'inventory': json.dumps([item.__dict__ for item in player.inventory]),
        'spells': json.dumps([spell.__dict__ for spell in player.spells]),
        'equipped_weapon': json.dumps(player.equipped_weapon.__dict__) if player.equipped_weapon else None,
        'equipped_shield': json.dumps(player.equipped_shield.__dict__) if player.equipped_shield else None,
        'equipped_shoes': json.dumps(player.equipped_shoes.__dict__) if player.equipped_shoes else None,
        'equipped_armor': json.dumps(player.equipped_armor.__dict__) if player.equipped_armor else None
    }
```

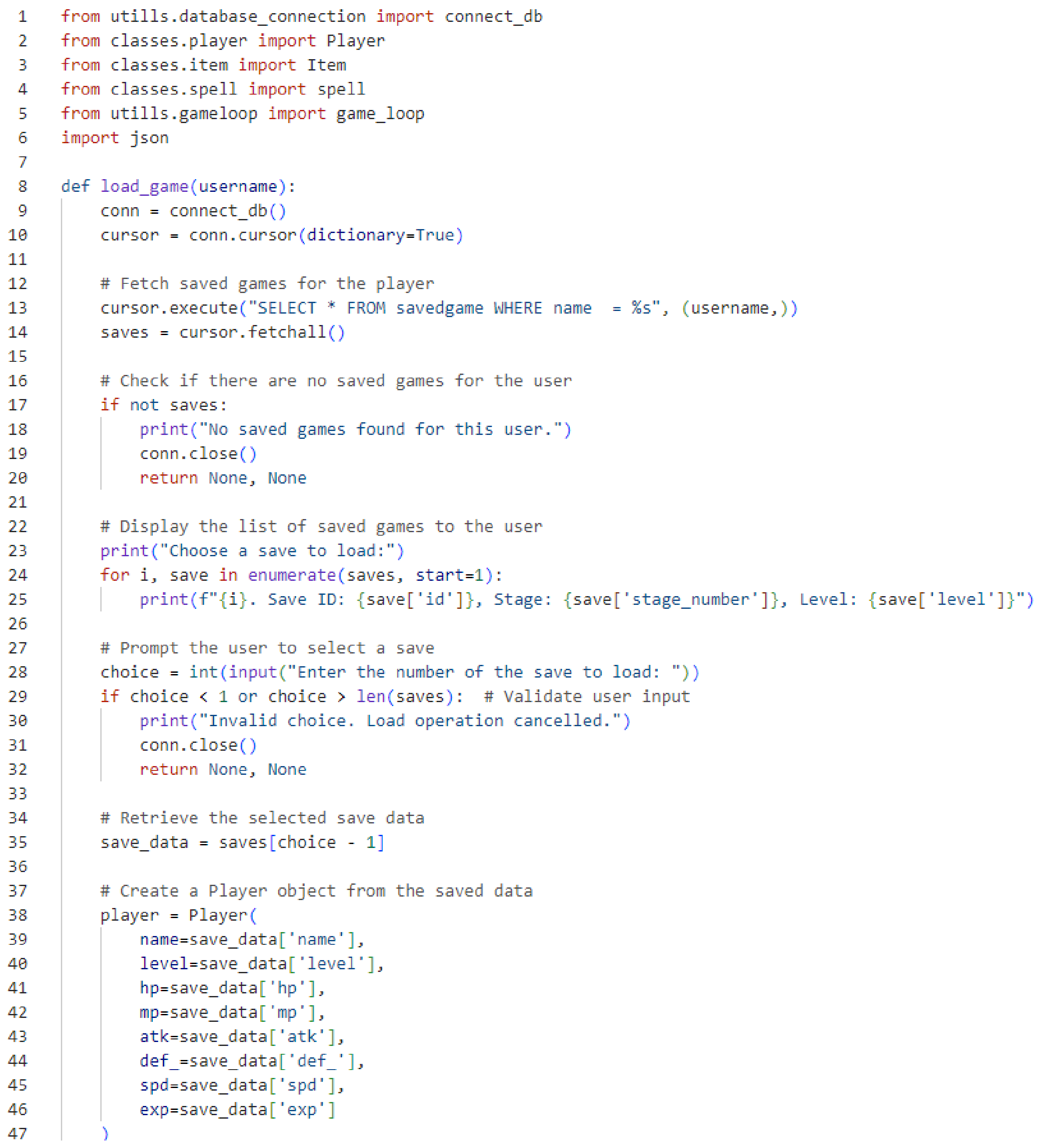


```python
    # Insert the save data into the database
    cursor.execute("""
        INSERT INTO savedgame (stage_number, name, level, hp, mp, atk, def_, spd, exp, inventory, spells, equipped_weapon, equipped_shield, equipped_shoes, equipped_armor)
        VALUES (%(stage_number)s, %(name)s, %(level)s, %(hp)s, %(mp)s, %(atk)s, %(def_)s, %(spd)s, %(exp)s, %(inventory)s, %(spells)s, %(equipped_weapon)s, %(equipped_shield)s, %(equip
    """, player_data)
```

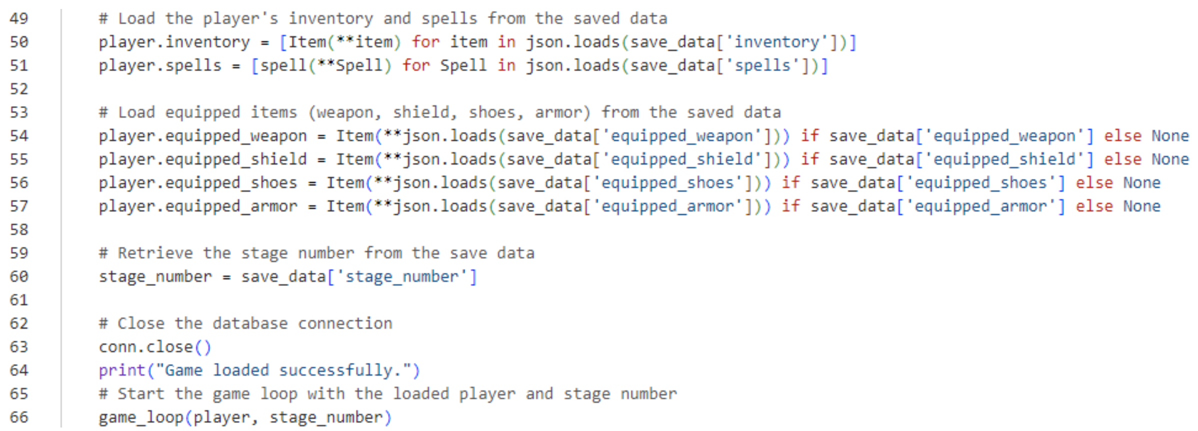


```python
    # Commit the transaction and close the connection
    conn.commit()
    conn.close()
    print("Game saved successfully.")
```

Load.py



```python
from utills.database_connection import connect_db
from classes.player import Player
from classes.item import Item
from classes.spell import spell
from utills.gameloop import game_loop
import json

def load_game(username):
    conn = connect_db()
    cursor = conn.cursor(dictionary=True)

    # Fetch saved games for the player
    cursor.execute("SELECT * FROM savedgame WHERE name  = %s", (username,))
    saves = cursor.fetchall()

    # Check if there are no saved games for the user
    if not saves:
        print("No saved games found for this user.")
        conn.close()
        return None, None

    # Display the list of saved games to the user
    print("Choose a save to load:")
    for i, save in enumerate(saves, start=1):
        print(f"{i}. Save ID: {save['id']}, Stage: {save['stage_number']}, Level: {save['level']}")

    # Prompt the user to select a save
    choice = int(input("Enter the number of the save to load: "))
    if choice < 1 or choice > len(saves):  # Validate user input
        print("Invalid choice. Load operation cancelled.")
        conn.close()
        return None, None

    # Retrieve the selected save data
    save_data = saves[choice - 1]

    # Create a Player object from the saved data
    player = Player(
        name=save_data['name'],
        level=save_data['level'],
        hp=save_data['hp'],
        mp=save_data['mp'],
        atk=save_data['atk'],
        def_=save_data['def_'],
        spd=save_data['spd'],
        exp=save_data['exp']
    )
```



```python
    # Load the player's inventory and spells from the saved data
    player.inventory = [Item(**item) for item in json.loads(save_data['inventory'])]
    player.spells = [spell(**Spell) for Spell in json.loads(save_data['spells'])]

    # Load equipped items (weapon, shield, shoes, armor) from the saved data
    player.equipped_weapon = Item(**json.loads(save_data['equipped_weapon'])) if save_data['equipped_weapon'] else None
    player.equipped_shield = Item(**json.loads(save_data['equipped_shield'])) if save_data['equipped_shield'] else None
    player.equipped_shoes = Item(**json.loads(save_data['equipped_shoes'])) if save_data['equipped_shoes'] else None
    player.equipped_armor = Item(**json.loads(save_data['equipped_armor'])) if save_data['equipped_armor'] else None

    # Retrieve the stage number from the save data
    stage_number = save_data['stage_number']

    # Close the database connection
    conn.close()
    print("Game loaded successfully.")
    # Start the game loop with the loaded player and stage number
    game_loop(player, stage_number)
```

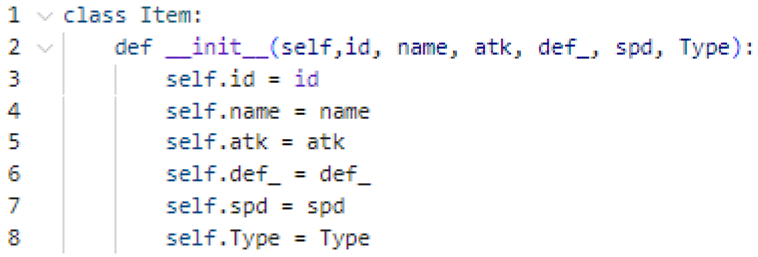
Log of ongoing testing

- **Testing :** Can the game save the process properly
  - **Problem :** while the game has got to the saving process, a error code “Python type list cannot be converted”
  - **Reason :** because of some fields in the player class are Python list objects, so cannot be directly inserted into a database.
  - **Solve :** I have done some research on how to solve this problem I find out the reason that I talk about, the way I have solve this problem is converting the list to json by using “json.dump()”.
  - **Reference that help me :** https://stackoverflow.com/questions/6222381/the-best-way-to-store-a-python-list-to-a-database

Because of the saving have a problem in saving the progress, so this means that my load also have problem, because of the python list has converted into json list, so here also needs to load back the json by use json.loads().

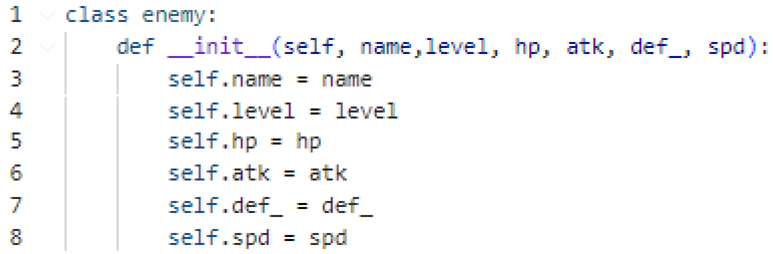
OOP part

Item class



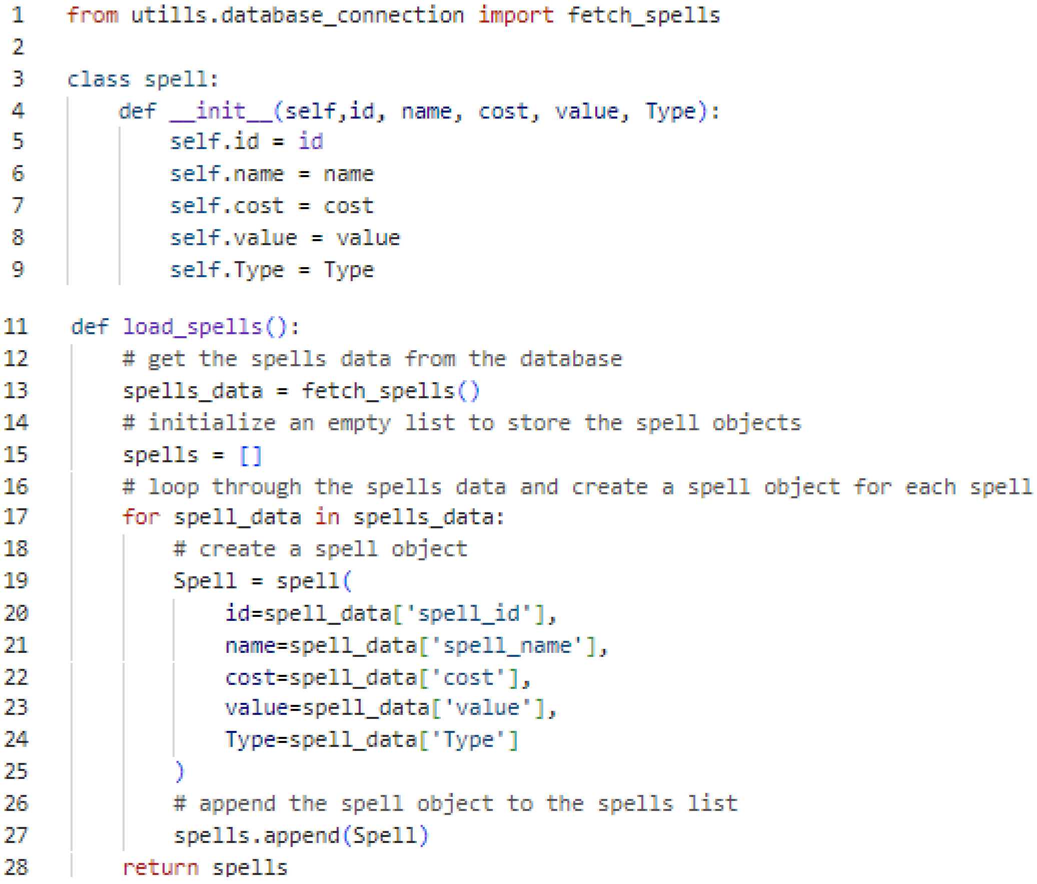
```python
class Item:
    def __init__(self,id, name, atk, def_, spd, Type):
        self.id = id
        self.name = name
        self.atk = atk
        self.def_ = def_
        self.spd = spd
        self.Type = Type
```

Enemy class



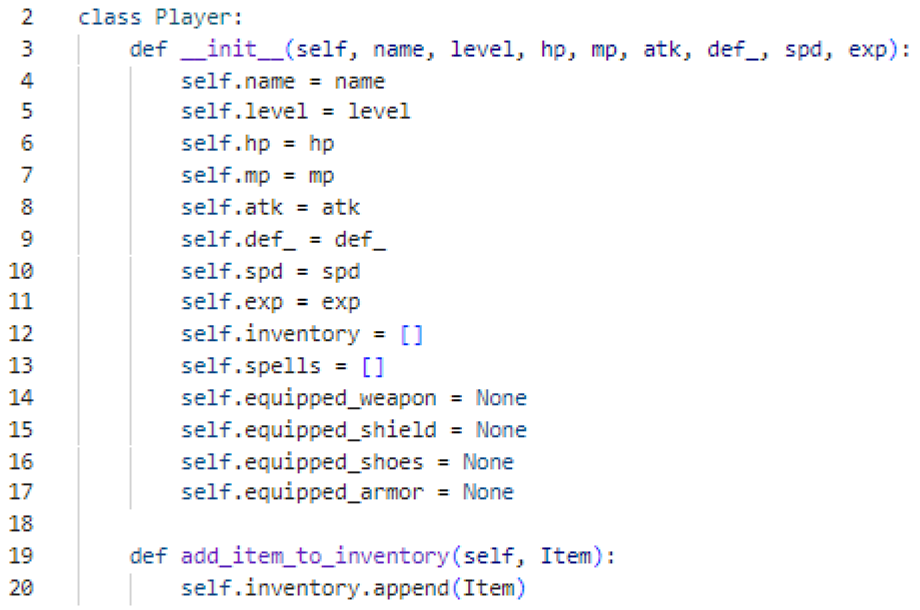
```python
class enemy:
    def __init__(self, name,level, hp, atk, def_, spd):
        self.name = name
        self.level = level
        self.hp = hp
        self.atk = atk
        self.def_ = def_
        self.spd = spd
```

Spell class

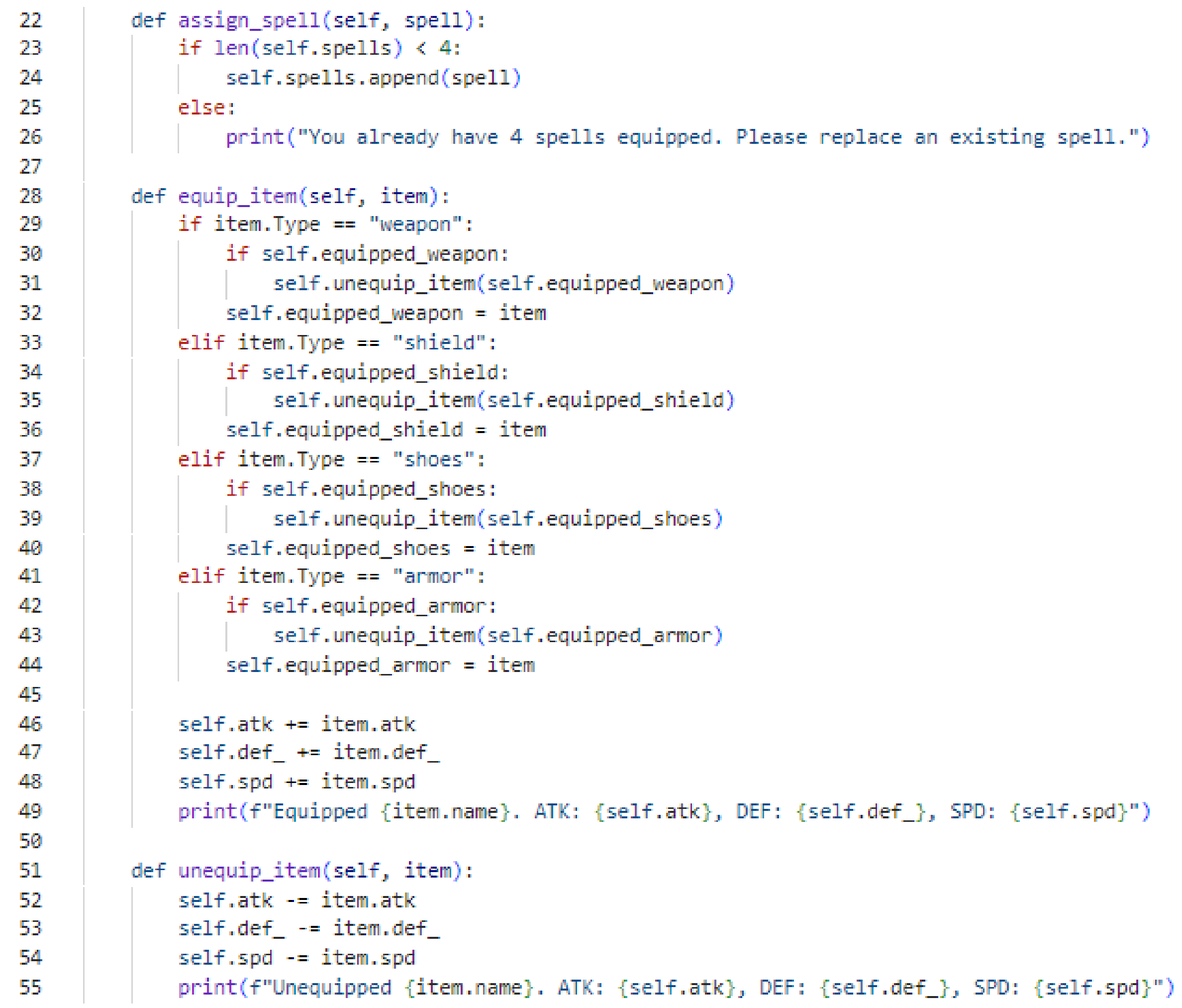


```python
from utills.database_connection import fetch_spells

class spell:
    def __init__(self,id, name, cost, value, Type):
        self.id = id
        self.name = name
        self.cost = cost
        self.value = value
        self.Type = Type

def load_spells():
    # get the spells data from the database
    spells_data = fetch_spells()
    # initialize an empty list to store the spell objects
    spells = []
    # loop through the spells data and create a spell object for each spell
    for spell_data in spells_data:
        # create a spell object
        Spell = spell(
            id=spell_data['spell_id'],
            name=spell_data['spell_name'],
            cost=spell_data['cost'],
            value=spell_data['value'],
            Type=spell_data['Type']
        )
        # append the spell object to the spells list
        spells.append(Spell)
    return spells
```

Player class



```python
class Player:
    def __init__(self, name, level, hp, mp, atk, def_, spd, exp):
        self.name = name
        self.level = level
        self.hp = hp
        self.mp = mp
        self.atk = atk
        self.def_ = def_
        self.spd = spd
        self.exp = exp
        self.inventory = []
        self.spells = []
        self.equipped_weapon = None
        self.equipped_shield = None
        self.equipped_shoes = None
        self.equipped_armor = None

    def add_item_to_inventory(self, Item):
        self.inventory.append(Item)
```



```python
    def assign_spell(self, spell):
        if len(self.spells) < 4:
            self.spells.append(spell)
        else:
            print("You already have 4 spells equipped. Please replace an existing spell.")

    def equip_item(self, item):
        if item.Type == "weapon":
            if self.equipped_weapon:
                self.unequip_item(self.equipped_weapon)
            self.equipped_weapon = item
        elif item.Type == "shield":
            if self.equipped_shield:
                self.unequip_item(self.equipped_shield)
            self.equipped_shield = item
        elif item.Type == "shoes":
            if self.equipped_shoes:
                self.unequip_item(self.equipped_shoes)
            self.equipped_shoes = item
        elif item.Type == "armor":
            if self.equipped_armor:
                self.unequip_item(self.equipped_armor)
            self.equipped_armor = item

        self.atk += item.atk
        self.def_ += item.def_
        self.spd += item.spd
        print(f"Equipped {item.name}. ATK: {self.atk}, DEF: {self.def_}, SPD: {self.spd}")

    def unequip_item(self, item):
        self.atk -= item.atk
        self.def_ -= item.def_
        self.spd -= item.spd
        print(f"Unequipped {item.name}. ATK: {self.atk}, DEF: {self.def_}, SPD: {self.spd}")
```

Research and development of new skills and/or knowledge (SDD)

Technical Skill

- **Isdigit()**
  - I have gone to learn this because of this could shorten the steps of checking if the input is number, I could solve this problem with a more simple by isdigit()
  - I have used this in the assign.py to check if the input that a number, but for save and load section I didn't use this because of that one player could just have 3 saves so I just check is the input equals to "1,2,3"
  - **Refences:**
    https://www.w3schools.com/python/ref_string_isdigit.asp
    https://www.geeksforgeeks.org/python-string-isdigit-method/

- **Enumerate()**
  - I have to learn this because of I want to keep the code looks as simple and clean as possible so I have use this to replace the normal loop method. Enumerate will provide me automatic numbering for lists.
  - I have use this in most of the cases that I need to display a list out to the terminal like the 3 save progress with number at front in the database (save.py) or list out the item that the player have in their inventory (inventory.py)
  - **Refences:**
    https://www.geeksforgeeks.org/enumerate-in-python/
    https://pythonbasics.org/enumerate/
    https://www.runoob.com/python/python-func-enumerate.html

- **Json (Json.dump(), Json.load() & some Json knowledge)**
  - I need to learn this because of the error code that turns up while I am implementing the save and load. Because of a database cannot accept python list while importing the data in the database, so after finding how to solve the error, I have found out that by converting the lists into json format could fix this issue, so I have learn some knowledge about json.
  - I have use this in save.py and load.py for converting python list into json and loading the json data form the database into the python list.
  - **Refences:**
    https://www.geeksforgeeks.org/json-dump-in-python/
    https://stackoverflow.com/questions/12309269/how-do-i-write-json-data-to-a-file
    https://www.geeksforgeeks.org/json-load-in-python/
    https://www.w3schools.com/python/python_json.asp
    https://www.w3schools.com/js/js_json_intro.asp

Concepts / ways to construct one of the system in the game

- **Gameloop**
  - I have researched some other people's projects game loops because they form the backbone of the game. By learning how game loops are structured, I can better understand that how a gameloop could be structured and be efficient.
  - I have use the knowledge I have learn from other and used on most of the codes that is related to the structure of the game.
  - **Refences:**
    https://github.com/rodmarkun/Python-Text-Turn-Based-RPG
    https://github.com/Reem19-15/python-rpg-game
    https://www.endyourif.com/building-a-text-based-rpg-game-in-python/

- **Text binary search**
  - I have researched this because of the binary search that is taught in scholar and textbook isn't working for text, so I have search up some forums for solve.
  - I have used this in the inventory.py to give the player a option to search up the item that they want.
  - I have got the inspiration from these two references:
    https://stackoverflow.com/questions/34327244/binary-search-through-strings
    https://stackoverflow.com/questions/21714485/binary-search-for-name

Database

SQL Code

Fetching Item

```
“SELECT item_id, item_name, atk, def, spd, Type FROM item_table”
```

Fetching enemy by stage

```
“SELECT enemy_name, HP, ATK, DEF, SPD, level FROM enemy_table WHERE stage_number = %s',
(stage_number,)”
```

Fetching Spell

```
“SELECT spell_id, spell_name, cost, value, Type FROM spell_table”
```

Fetching reward

```
“SELECT reward_type, reward_value, associated_with, stage_number, associated_id, quantity FROM
reward_table WHERE stage_number = %s', (stage_number,)”
```

Savedgame

```
“SELECT id FROM savedgame WHERE name = %s", (player.name,)”
```

Delete savedgame

```
“”DELETE FROM savedgame WHERE id = %s", (save_id,)”
```

Insert new savedata

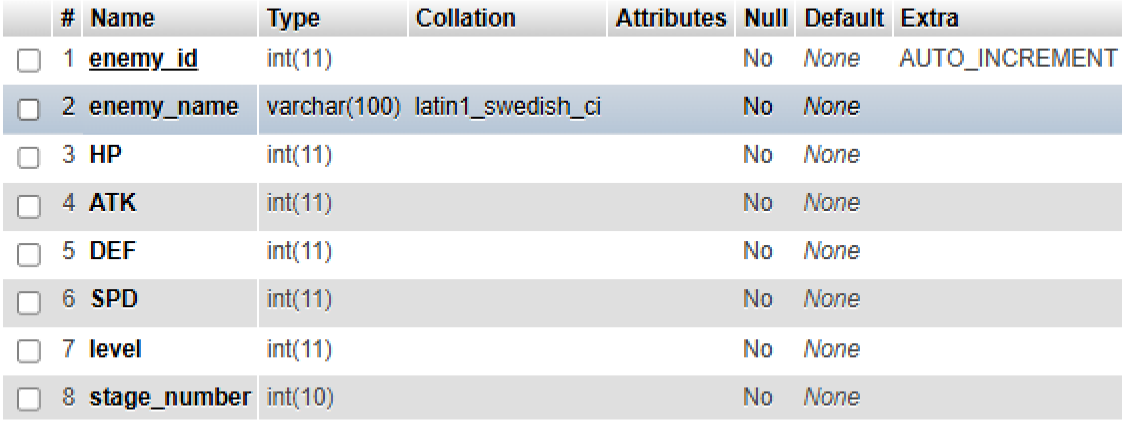
```
“ """INSERT INTO savedgame (stage_number, name, level, hp, mp, atk, def_, spd, exp,
inventory, spells, equipped_weapon, equipped_shield, equipped_shoes, equipped_armor)
VALUES (%(stage_number)s, %(name)s, %(level)s, %(hp)s, %(mp)s, %(atk)s, %(def_)s,
%(spd)s, %(exp)s, %(inventory)s, %(spells)s, %(equipped_weapon)s, %(equipped_shield)s,
%(equipped_shoes)s, %(equipped_armor)s)""", player_data) “
```

retrieve all saved game details

```
“"SELECT * FROM savedgame WHERE name  = %s", (username,)”
```

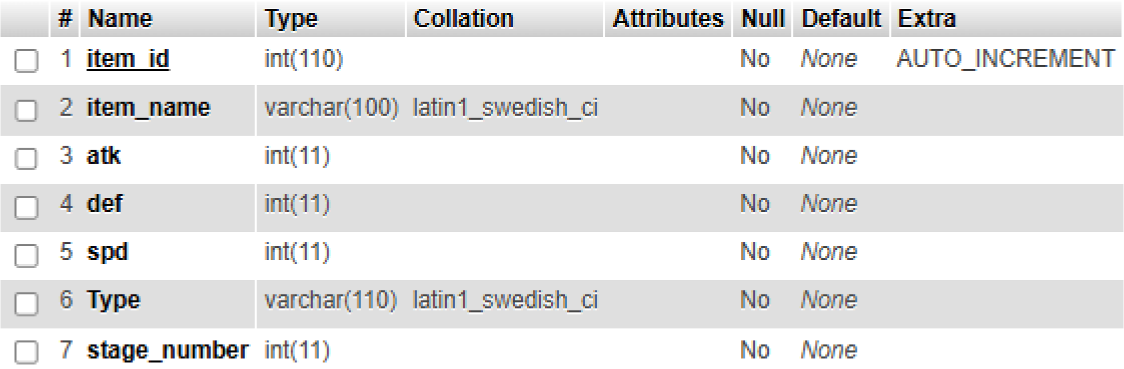
Structure of the database

Enemy table



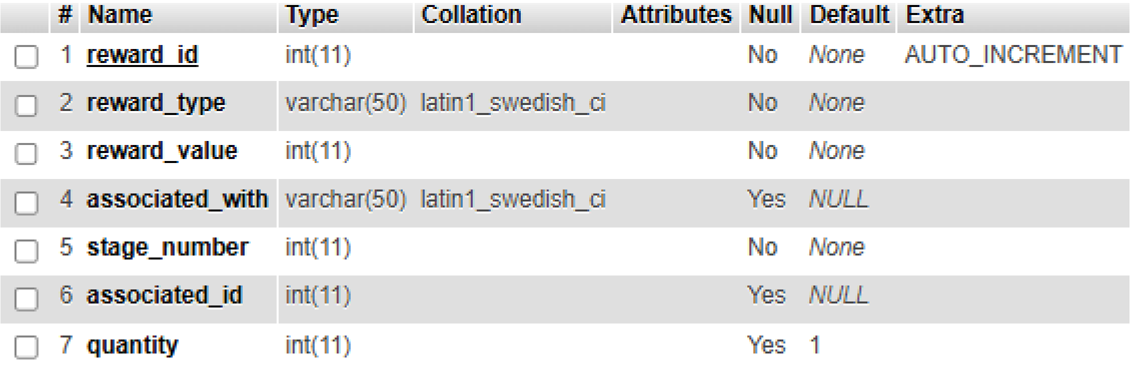
	#	Name	Type	Collation	Attributes	Null	Default	Extra
□	1	enemy_id	int(11)			No	*None*	AUTO_INCREMENT
□	2	enemy_name	varchar(100)	latin1_swedish_ci		No	*None*	
□	3	HP	int(11)			No	*None*	
□	4	ATK	int(11)			No	*None*	
□	5	DEF	int(11)			No	*None*	
□	6	SPD	int(11)			No	*None*	
□	7	level	int(11)			No	*None*	
□	8	stage_number	int(10)			No	*None*	

Item table



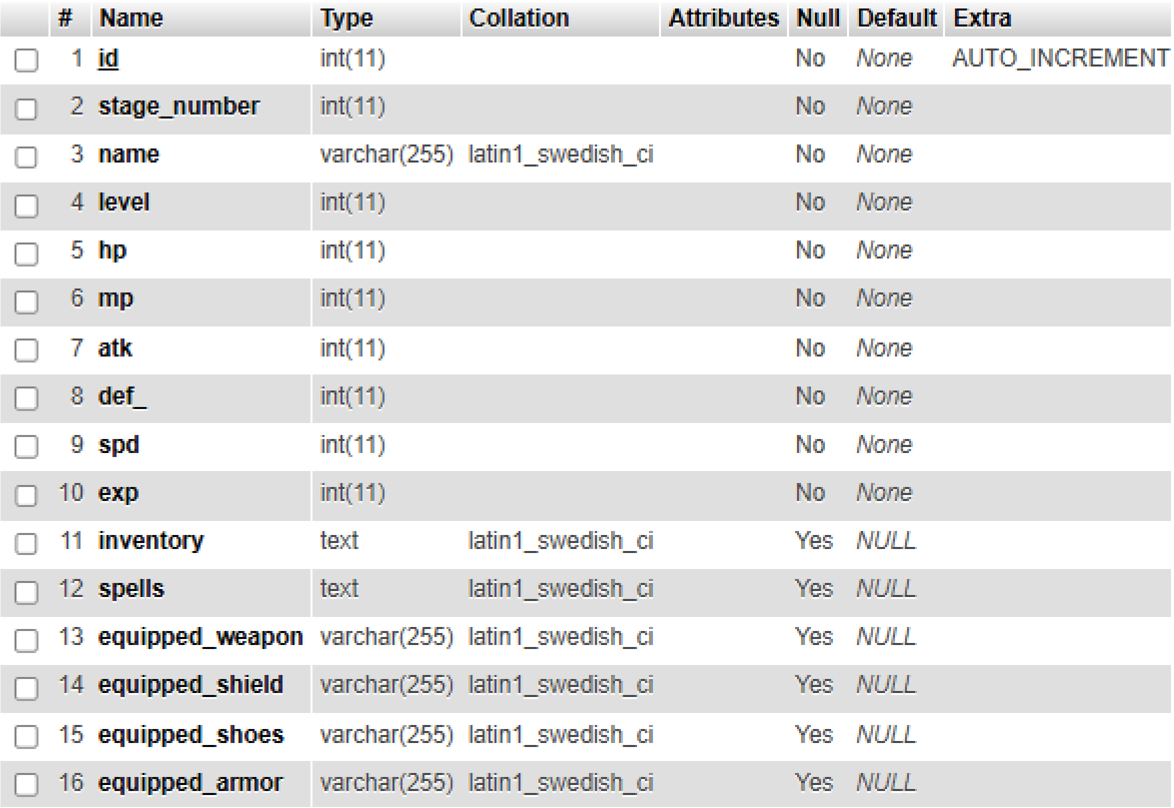
	#	Name	Type	Collation	Attributes	Null	Default	Extra
□	1	item_id	int(110)			No	*None*	AUTO_INCREMENT
□	2	item_name	varchar(100)	latin1_swedish_ci		No	*None*	
□	3	atk	int(11)			No	*None*	
□	4	def	int(11)			No	*None*	
□	5	spd	int(11)			No	*None*	
□	6	Type	varchar(110)	latin1_swedish_ci		No	*None*	
□	7	stage_number	int(11)			No	*None*	

Reward table



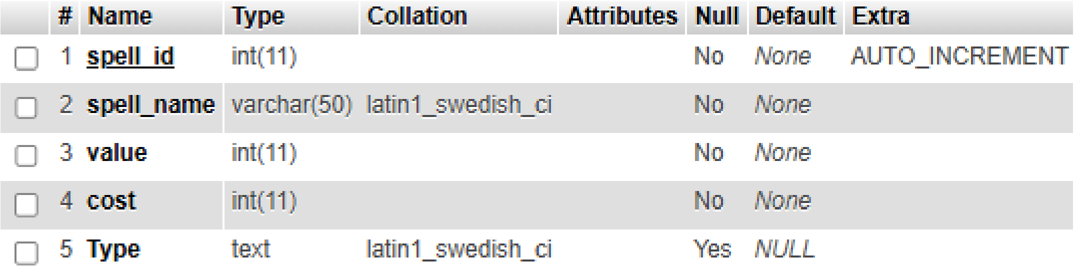
	#	Name	Type	Collation	Attributes	Null	Default	Extra
□	1	reward_id	int(11)			No	*None*	AUTO_INCREMENT
□	2	reward_type	varchar(50)	latin1_swedish_ci		No	*None*	
□	3	reward_value	int(11)			No	*None*	
□	4	associated_with	varchar(50)	latin1_swedish_ci		Yes	*NULL*	
□	5	stage_number	int(11)			No	*None*	
□	6	associated_id	int(11)			Yes	*NULL*	
□	7	quantity	int(11)			Yes	1	

Savedgame



	#	Name	Type	Collation	Attributes	Null	Default	Extra
□	1	id	int(11)			No	*None*	AUTO_INCREMENT
□	2	stage_number	int(11)			No	*None*	
□	3	name	varchar(255)	latin1_swedish_ci		No	*None*	
□	4	level	int(11)			No	*None*	
□	5	hp	int(11)			No	*None*	
□	6	mp	int(11)			No	*None*	
□	7	atk	int(11)			No	*None*	
□	8	def_	int(11)			No	*None*	
□	9	spd	int(11)			No	*None*	
□	10	exp	int(11)			No	*None*	
□	11	inventory	text	latin1_swedish_ci		Yes	*NULL*	
□	12	spells	text	latin1_swedish_ci		Yes	*NULL*	
□	13	equipped_weapon	varchar(255)	latin1_swedish_ci		Yes	*NULL*	
□	14	equipped_shield	varchar(255)	latin1_swedish_ci		Yes	*NULL*	
□	15	equipped_shoes	varchar(255)	latin1_swedish_ci		Yes	*NULL*	
□	16	equipped_armor	varchar(255)	latin1_swedish_ci		Yes	*NULL*	

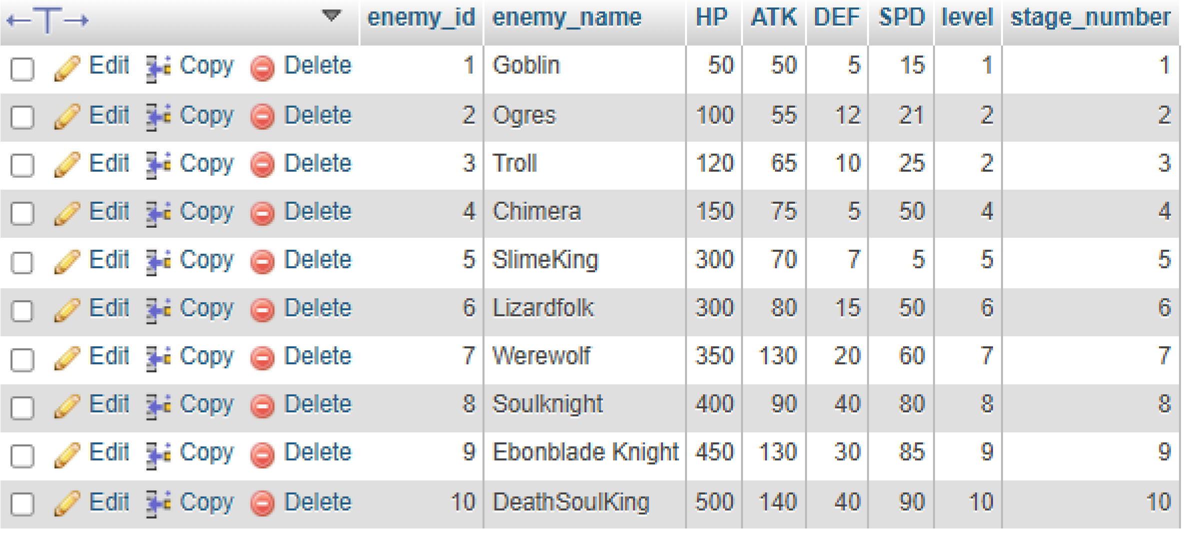
Spell table



	#	Name	Type	Collation	Attributes	Null	Default	Extra
□	1	spell_id	int(11)			No	*None*	AUTO_INCREMENT
□	2	spell_name	varchar(50)	latin1_swedish_ci		No	*None*	
□	3	value	int(11)			No	*None*	
□	4	cost	int(11)			No	*None*	
□	5	Type	text	latin1_swedish_ci		Yes	*NULL*	

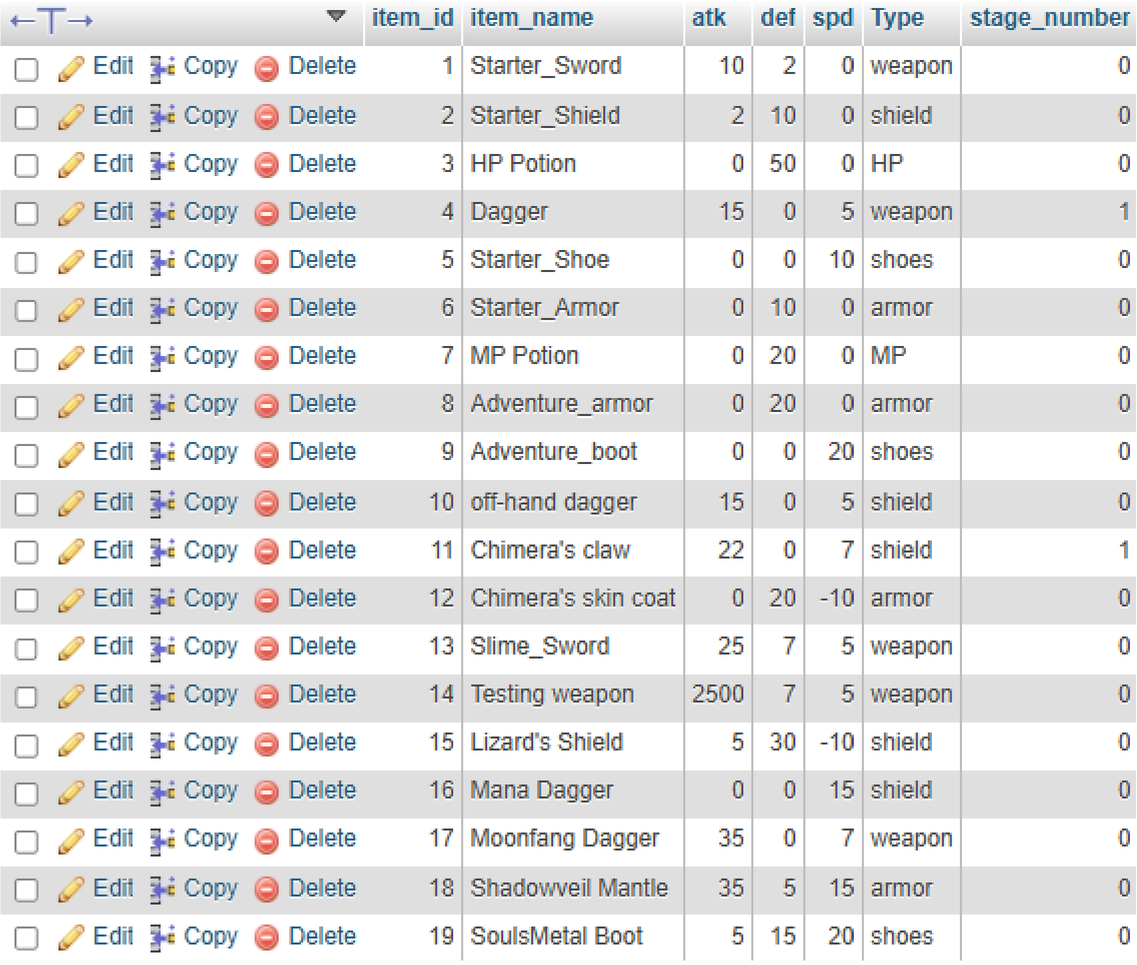
Initial value stored in the database

Enemy table



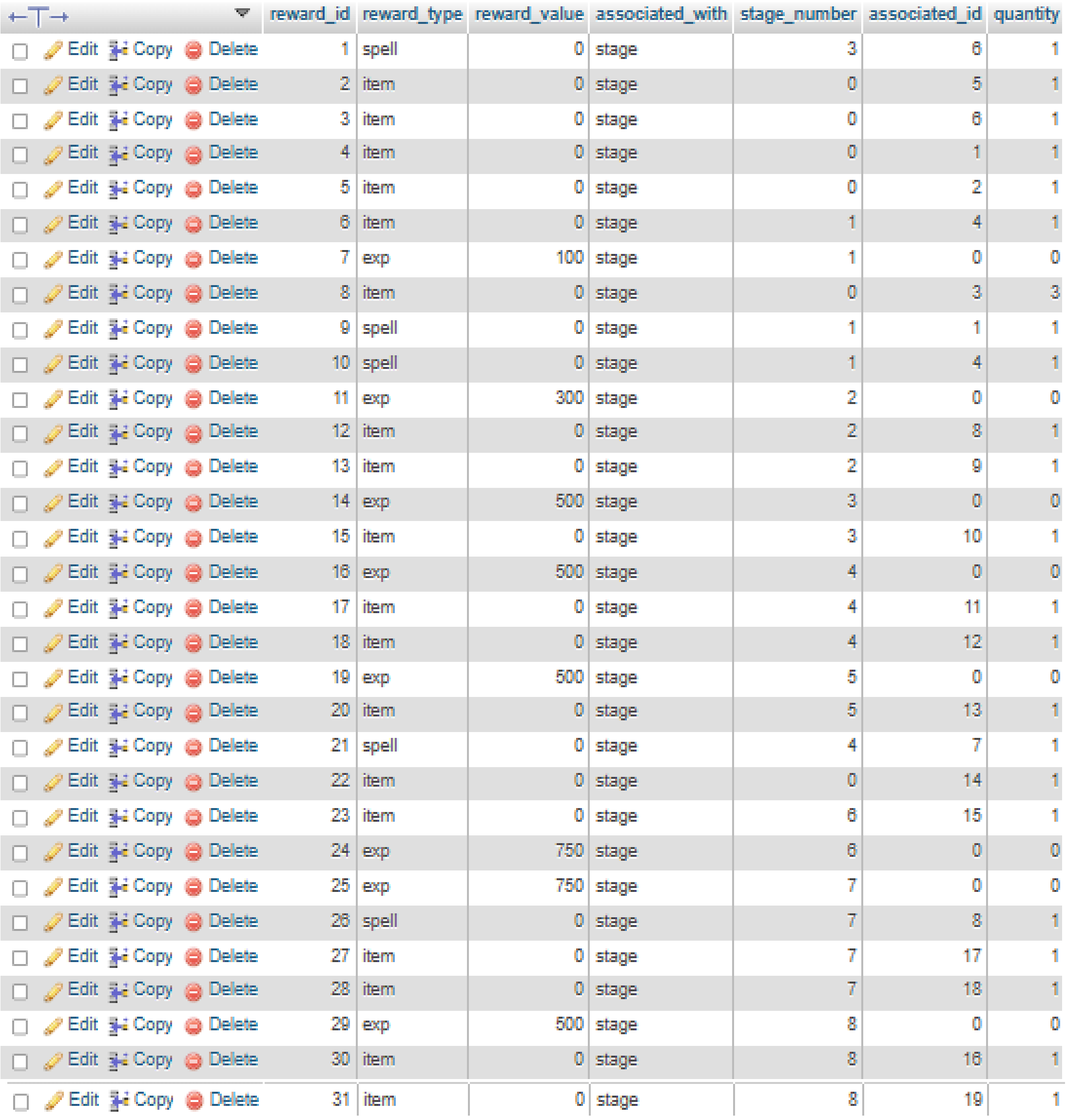
←T→				enemy_id	enemy_name	HP	ATK	DEF	SPD	level	stage_number
□	Edit	Copy	Delete	1	Goblin	50	50	5	15	1	1
□	Edit	Copy	Delete	2	Ogres	100	55	12	21	2	2
□	Edit	Copy	Delete	3	Troll	120	65	10	25	2	3
□	Edit	Copy	Delete	4	Chimera	150	75	5	50	4	4
□	Edit	Copy	Delete	5	SlimeKing	300	70	7	5	5	5
□	Edit	Copy	Delete	6	Lizardfolk	300	80	15	50	6	6
□	Edit	Copy	Delete	7	Werewolf	350	130	20	60	7	7
□	Edit	Copy	Delete	8	Soulknight	400	90	40	80	8	8
□	Edit	Copy	Delete	9	Ebonblade Knight	450	130	30	85	9	9
□	Edit	Copy	Delete	10	DeathSoulKing	500	140	40	90	10	10

Item table



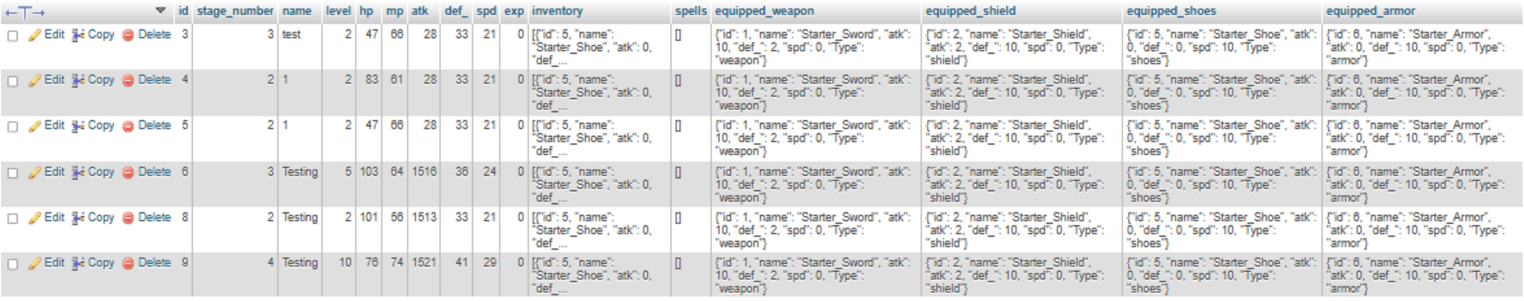
←T→				item_id	item_name	atk	def	spd	Type	stage_number
□	Edit	Copy	Delete	1	Starter_Sword	10	2	0	weapon	0
□	Edit	Copy	Delete	2	Starter_Shield	2	10	0	shield	0
□	Edit	Copy	Delete	3	HP Potion	0	50	0	HP	0
□	Edit	Copy	Delete	4	Dagger	15	0	5	weapon	1
□	Edit	Copy	Delete	5	Starter_Shoe	0	0	10	shoes	0
□	Edit	Copy	Delete	6	Starter_Armor	0	10	0	armor	0
□	Edit	Copy	Delete	7	MP Potion	0	20	0	MP	0
□	Edit	Copy	Delete	8	Adventure_armor	0	20	0	armor	0
□	Edit	Copy	Delete	9	Adventure_boot	0	0	20	shoes	0
□	Edit	Copy	Delete	10	off-hand dagger	15	0	5	shield	0
□	Edit	Copy	Delete	11	Chimera's claw	22	0	7	shield	1
□	Edit	Copy	Delete	12	Chimera's skin coat	0	20	-10	armor	0
□	Edit	Copy	Delete	13	Slime_Sword	25	7	5	weapon	0
□	Edit	Copy	Delete	14	Testing weapon	2500	7	5	weapon	0
□	Edit	Copy	Delete	15	Lizard's Shield	5	30	-10	shield	0
□	Edit	Copy	Delete	16	Mana Dagger	0	0	15	shield	0
□	Edit	Copy	Delete	17	Moonfang Dagger	35	0	7	weapon	0
□	Edit	Copy	Delete	18	Shadowveil Mantle	35	5	15	armor	0
□	Edit	Copy	Delete	19	SoulsMetal Boot	5	15	20	shoes	0

Reward table



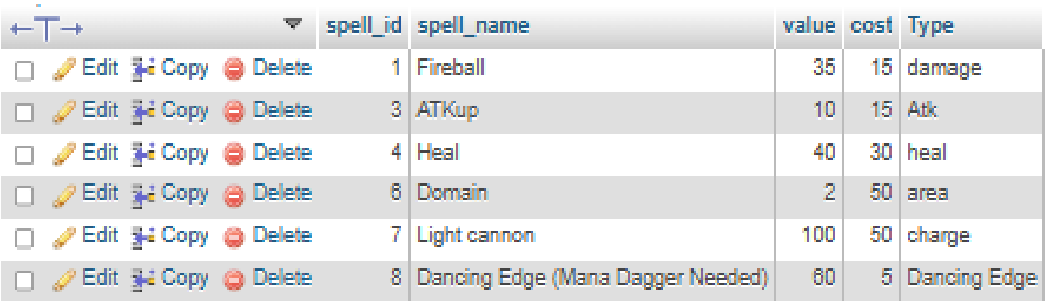
←T→	reward_id	reward_type	reward_value	associated_with	stage_number	associated_id	quantity
□ Edit Copy Delete	1	spell	0	stage	3	6	1
□ Edit Copy Delete	2	item	0	stage	0	5	1
□ Edit Copy Delete	3	item	0	stage	0	6	1
□ Edit Copy Delete	4	item	0	stage	0	1	1
□ Edit Copy Delete	5	item	0	stage	0	2	1
□ Edit Copy Delete	6	item	0	stage	1	4	1
□ Edit Copy Delete	7	exp	100	stage	1	0	0
□ Edit Copy Delete	8	item	0	stage	0	3	3
□ Edit Copy Delete	9	spell	0	stage	1	1	1
□ Edit Copy Delete	10	spell	0	stage	1	4	1
□ Edit Copy Delete	11	exp	300	stage	2	0	0
□ Edit Copy Delete	12	item	0	stage	2	8	1
□ Edit Copy Delete	13	item	0	stage	2	9	1
□ Edit Copy Delete	14	exp	500	stage	3	0	0
□ Edit Copy Delete	15	item	0	stage	3	10	1
□ Edit Copy Delete	16	exp	500	stage	4	0	0
□ Edit Copy Delete	17	item	0	stage	4	11	1
□ Edit Copy Delete	18	item	0	stage	4	12	1
□ Edit Copy Delete	19	exp	500	stage	5	0	0
□ Edit Copy Delete	20	item	0	stage	5	13	1
□ Edit Copy Delete	21	spell	0	stage	4	7	1
□ Edit Copy Delete	22	item	0	stage	0	14	1
□ Edit Copy Delete	23	item	0	stage	6	15	1
□ Edit Copy Delete	24	exp	750	stage	6	0	0
□ Edit Copy Delete	25	exp	750	stage	7	0	0
□ Edit Copy Delete	26	spell	0	stage	7	8	1
□ Edit Copy Delete	27	item	0	stage	7	17	1
□ Edit Copy Delete	28	item	0	stage	7	18	1
□ Edit Copy Delete	29	exp	500	stage	8	0	0
□ Edit Copy Delete	30	item	0	stage	8	16	1
□ Edit Copy Delete	31	item	0	stage	8	19	1

Savedgame (This is the view of after few times of testing the saving function)



←T→	id	stage_number	name	level	hp	mp	atk	def_	spd	exp	inventory	spells	equipped_weapon	equipped_shield	equipped_shoes	equipped_armor
□ Edit Copy Delete	3	3	test	2	47	66	28	33	21	0	[{"id": 5, "name": "Starter_Shoe", "atk": 0, "def_...	[]	{"id": 1, "name": "Starter_Sword", "atk": 10, "def_": 2, "spd": 0, "Type": "weapon"}	{"id": 2, "name": "Starter_Shield", "atk": 2, "def_": 10, "spd": 0, "Type": "shield"}	{"id": 5, "name": "Starter_Shoe", "atk": 0, "def_": 0, "spd": 10, "Type": "shoes"}	{"id": 6, "name": "Starter_Armor", "atk": 0, "def_": 10, "spd": 0, "Type": "armor"}
□ Edit Copy Delete	4	2	1	2	83	61	28	33	21	0	[{"id": 5, "name": "Starter_Shoe", "atk": 0, "def_...	[]	{"id": 1, "name": "Starter_Sword", "atk": 10, "def_": 2, "spd": 0, "Type": "weapon"}	{"id": 2, "name": "Starter_Shield", "atk": 2, "def_": 10, "spd": 0, "Type": "shield"}	{"id": 5, "name": "Starter_Shoe", "atk": 0, "def_": 0, "spd": 10, "Type": "shoes"}	{"id": 6, "name": "Starter_Armor", "atk": 0, "def_": 10, "spd": 0, "Type": "armor"}
□ Edit Copy Delete	5	2	1	2	47	66	28	33	21	0	[{"id": 5, "name": "Starter_Shoe", "atk": 0, "def_...	[]	{"id": 1, "name": "Starter_Sword", "atk": 10, "def_": 2, "spd": 0, "Type": "weapon"}	{"id": 2, "name": "Starter_Shield", "atk": 2, "def_": 10, "spd": 0, "Type": "shield"}	{"id": 5, "name": "Starter_Shoe", "atk": 0, "def_": 0, "spd": 10, "Type": "shoes"}	{"id": 6, "name": "Starter_Armor", "atk": 0, "def_": 10, "spd": 0, "Type": "armor"}
□ Edit Copy Delete	6	3	Testing	5	103	64	1516	36	24	0	[{"id": 5, "name": "Starter_Shoe", "atk": 0, "def_...	[]	{"id": 1, "name": "Starter_Sword", "atk": 10, "def_": 2, "spd": 0, "Type": "weapon"}	{"id": 2, "name": "Starter_Shield", "atk": 2, "def_": 10, "spd": 0, "Type": "shield"}	{"id": 5, "name": "Starter_Shoe", "atk": 0, "def_": 0, "spd": 10, "Type": "shoes"}	{"id": 6, "name": "Starter_Armor", "atk": 0, "def_": 10, "spd": 0, "Type": "armor"}
□ Edit Copy Delete	8	2	Testing	2	101	56	1513	33	21	0	[{"id": 5, "name": "Starter_Shoe", "atk": 0, "def_...	[]	{"id": 1, "name": "Starter_Sword", "atk": 10, "def_": 2, "spd": 0, "Type": "weapon"}	{"id": 2, "name": "Starter_Shield", "atk": 2, "def_": 10, "spd": 0, "Type": "shield"}	{"id": 5, "name": "Starter_Shoe", "atk": 0, "def_": 0, "spd": 10, "Type": "shoes"}	{"id": 6, "name": "Starter_Armor", "atk": 0, "def_": 10, "spd": 0, "Type": "armor"}
□ Edit Copy Delete	9	4	Testing	10	76	74	1521	41	29	0	[{"id": 5, "name": "Starter_Shoe", "atk": 0, "def_...	[]	{"id": 1, "name": "Starter_Sword", "atk": 10, "def_": 2, "spd": 0, "Type": "weapon"}	{"id": 2, "name": "Starter_Shield", "atk": 2, "def_": 10, "spd": 0, "Type": "shield"}	{"id": 5, "name": "Starter_Shoe", "atk": 0, "def_": 0, "spd": 10, "Type": "shoes"}	{"id": 6, "name": "Starter_Armor", "atk": 0, "def_": 10, "spd": 0, "Type": "armor"}

Spell table

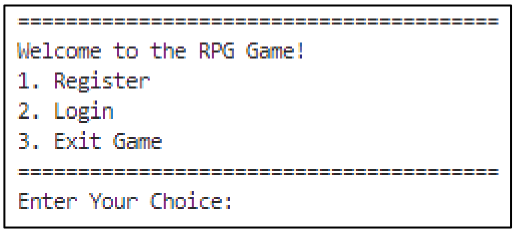


←T→ ▼	spell_id	spell_name	value	cost	Type
□ Edit Copy Delete	1	Fireball	35	15	damage
□ Edit Copy Delete	3	ATKup	10	15	Atk
□ Edit Copy Delete	4	Heal	40	30	heal
□ Edit Copy Delete	6	Domain	2	50	area
□ Edit Copy Delete	7	Light cannon	100	50	charge
□ Edit Copy Delete	8	Dancing Edge (Mana Dagger Needed)	60	5	Dancing Edge

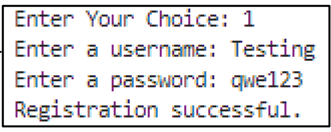
Research and development of new skills and/or knowledge (Database)

Screenshots of the game working

Situation: The player is the first time playing this game and beat the game

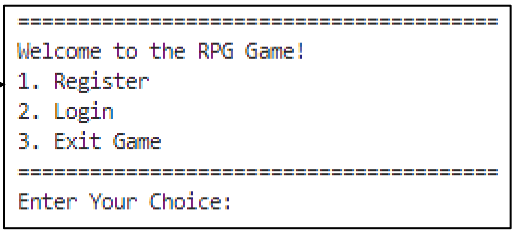


```
========================================
Welcome to the RPG Game!
1. Register
2. Login
3. Exit Game
========================================
Enter Your Choice:
```



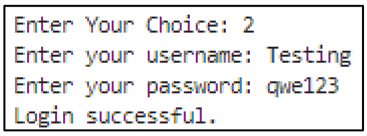
```
Enter Your Choice: 1
Enter a username: Testing
Enter a password: qwe123
Registration successful.
```

The player selects option 1 to register and create a new account.



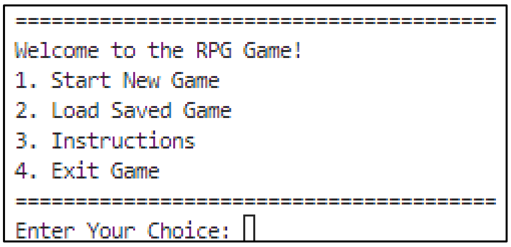
```
========================================
Welcome to the RPG Game!
1. Register
2. Login
3. Exit Game
========================================
Enter Your Choice:
```

After registering, the game will return to this screen, allowing the player to choose between logging in or exiting.



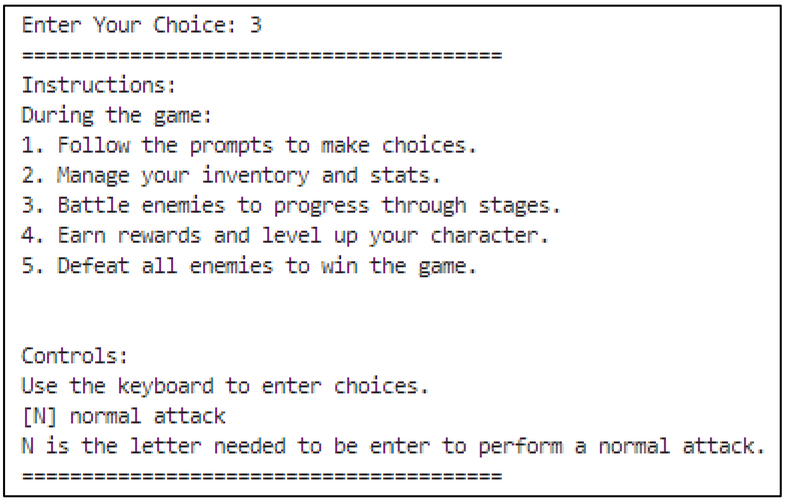
```
Enter Your Choice: 2
Enter your username: Testing
Enter your password: qwe123
Login successful.
```

The player selects option 2 to log in and access their account.



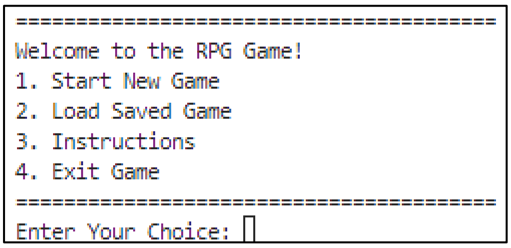
```
========================================
Welcome to the RPG Game!
1. Start New Game
2. Load Saved Game
3. Instructions
4. Exit Game
========================================
Enter Your Choice: ▯
```

After the player logs into their account, the main menu will appear, giving them the option to choose their next action.



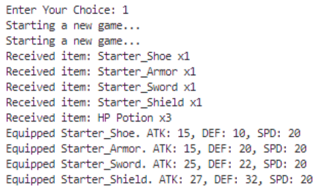
```
Enter Your Choice: 3
========================================
Instructions:
During the game:
1. Follow the prompts to make choices.
2. Manage your inventory and stats.
3. Battle enemies to progress through stages.
4. Earn rewards and level up your character.
5. Defeat all enemies to win the game.


Controls:
Use the keyboard to enter choices.
[N] normal attack
N is the letter needed to be enter to perform a normal attack.
========================================
```

Since the player doesn't know how to play the game, they select option 3 to open the instructions and learn how the game works.



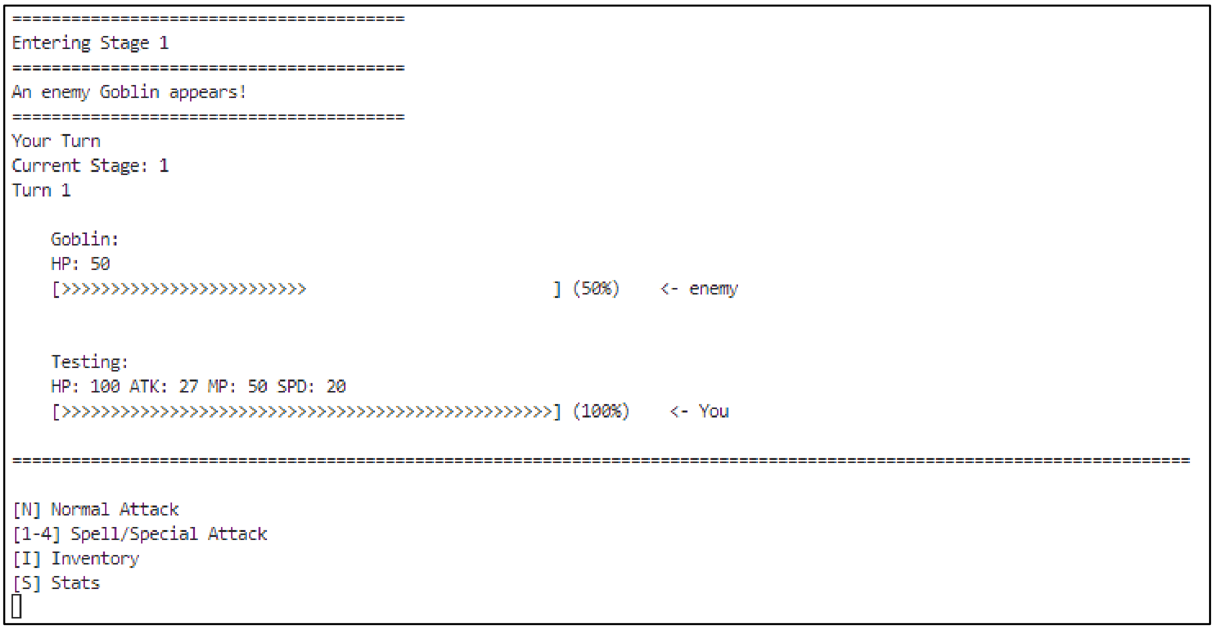
```
========================================
Welcome to the RPG Game!
1. Start New Game
2. Load Saved Game
3. Instructions
4. Exit Game
========================================
Enter Your Choice: ▯
```

After reading the instructions, the main menu will automatically reopen, offering the player options to choose their next step.



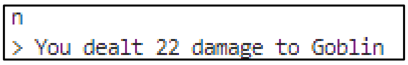
```
Enter Your Choice: 1
Starting a new game...
Starting a new game...
Received item: Starter_Shoe x1
Received item: Starter_Armor x1
Received item: Starter_Sword x1
Received item: Starter_Shield x1
Received item: HP Potion x3
Equipped Starter_Shoe. ATK: 15, DEF: 10, SPD: 20
Equipped Starter_Armor. ATK: 15, DEF: 20, SPD: 20
Equipped Starter_Sword. ATK: 25, DEF: 22, SPD: 20
Equipped Starter_Shield. ATK: 27, DEF: 32, SPD: 20
```

Before the battle interface opens, a log will display the items the player has obtained and the item they have equipped.



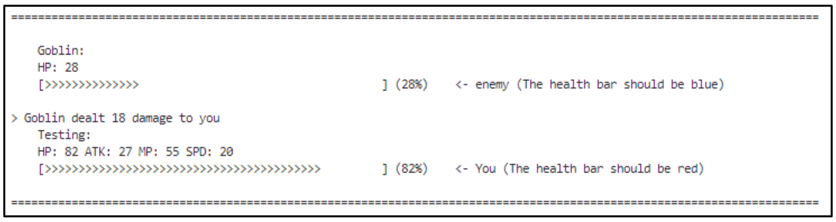
```
========================================
Entering Stage 1
========================================
An enemy Goblin appears!
========================================
Your Turn
Current Stage: 1
Turn 1

    Goblin:
    HP: 50
    [>>>>>>>>>>>>>>>>>>>>>>>>>                         ] (50%)    <- enemy


    Testing:
    HP: 100 ATK: 27 MP: 50 SPD: 20
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (100%)    <- You

========================================================================================================================

[N] Normal Attack
[1-4] Spell/Special Attack
[I] Inventory
[S] Stats
▯
```

Since the player doesn't have a saved game, they can only select option 1 to start a new game. After choosing to start a new game, the battle interface will open, displaying the player and enemy HP bars, stats, and the available actions the player can choose.



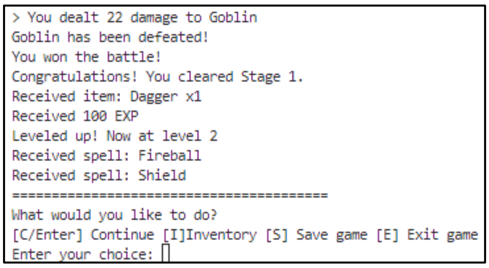
```
n
> You dealt 22 damage to Goblin
```

The player chooses to attack with a normal attack, and the system will display the amount of damage dealt to the enemy.



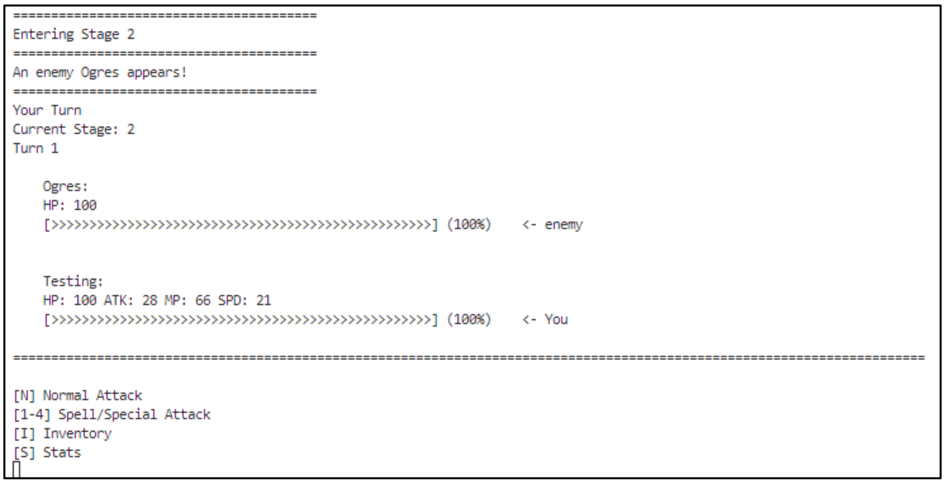
```
==========================================================================================


   Goblin:
   HP: 28
   [>>>>>>>>>>>>>                                    ] (28%)    <- enemy (The health bar should be blue)

> Goblin dealt 18 damage to you
   Testing:
   HP: 82 ATK: 27 MP: 55 SPD: 20
   [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>        ] (82%)    <- You (The health bar should be red)

==========================================================================================
```

After the player's turn, it will be the enemy's turn.



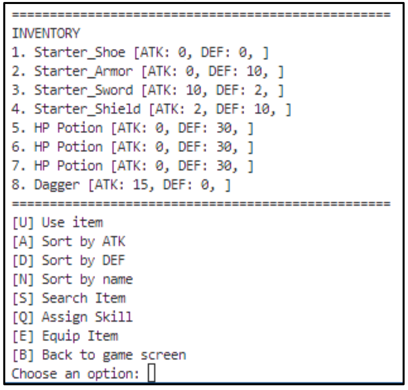
```
> You dealt 22 damage to Goblin
Goblin has been defeated!
You won the battle!
Congratulations! You cleared Stage 1.
Received item: Dagger x1
Received 100 EXP
Leveled up! Now at level 2
Received spell: Fireball
Received spell: Shield
========================================
What would you like to do?
[C/Enter] Continue [I]Inventory [S] Save game [E] Exit game
Enter your choice:
```

After a few more turns, the player defeats the stage 1 enemy. Four options will appear for the player to choose from.



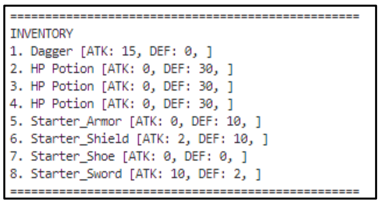
```
========================================
Entering Stage 2
========================================
An enemy Ogres appears!
========================================
Your Turn
Current Stage: 2
Turn 1

    Ogres:
    HP: 100
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (100%)    <- enemy


    Testing:
    HP: 100 ATK: 28 MP: 66 SPD: 21
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (100%)    <- You

========================================================================================================================

[N] Normal Attack
[1-4] Spell/Special Attack
[I] Inventory
[S] Stats
```

The player chooses to continue the game, and stage 2 is generated.



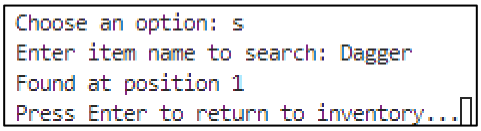
```
==================================================
INVENTORY
1. Starter_Shoe [ATK: 0, DEF: 0, ]
2. Starter_Armor [ATK: 0, DEF: 10, ]
3. Starter_Sword [ATK: 10, DEF: 2, ]
4. Starter_Shield [ATK: 2, DEF: 10, ]
5. HP Potion [ATK: 0, DEF: 30, ]
6. HP Potion [ATK: 0, DEF: 30, ]
7. HP Potion [ATK: 0, DEF: 30, ]
8. Dagger [ATK: 15, DEF: 0, ]
==================================================
[U] Use item
[A] Sort by ATK
[D] Sort by DEF
[N] Sort by name
[S] Search Item
[Q] Assign Skill
[E] Equip Item
[B] Back to game screen
Choose an option:
```

The player chooses to open the inventory to check the items they received from the last stage reward.



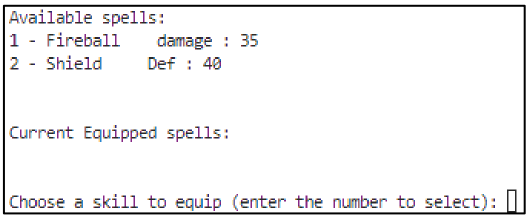
```
==================================================
INVENTORY
1. Dagger [ATK: 15, DEF: 0, ]
2. HP Potion [ATK: 0, DEF: 30, ]
3. HP Potion [ATK: 0, DEF: 30, ]
4. HP Potion [ATK: 0, DEF: 30, ]
5. Starter_Armor [ATK: 0, DEF: 10, ]
6. Starter_Shield [ATK: 2, DEF: 10, ]
7. Starter_Shoe [ATK: 0, DEF: 0, ]
8. Starter_Sword [ATK: 10, DEF: 2, ]
==================================================
```

The player has sort the inventory by name



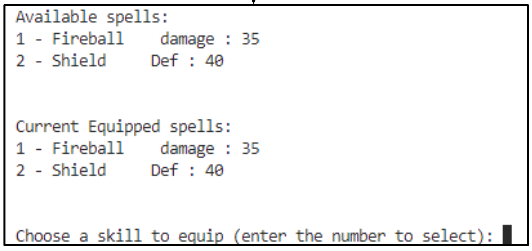
```
Choose an option: s
Enter item name to search: Dagger
Found at position 1
Press Enter to return to inventory...
```

The player wants to find the dagger in their inventory, so they use the search item feature and enter "dagger."



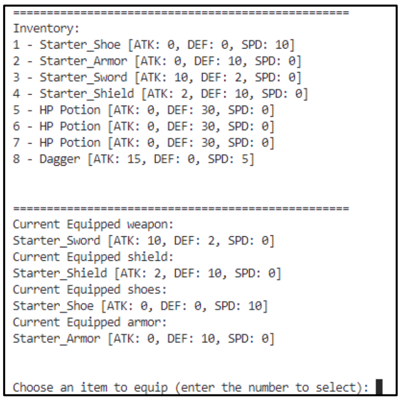
```
Available spells:
1 - Fireball     damage : 35
2 - Shield      Def : 40


Current Equipped spells:


Choose a skill to equip (enter the number to select):
```

After returning to the inventory, the player wants to assign the spells they own to the spell slots.



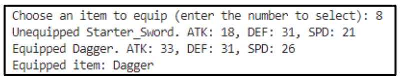
```
Available spells:
1 - Fireball     damage : 35
2 - Shield      Def : 40


Current Equipped spells:
1 - Fireball     damage : 35
2 - Shield      Def : 40


Choose a skill to equip (enter the number to select):
```

The player has equipped “Fireball” and “Shield”

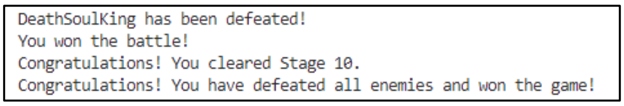


```
==================================================
Inventory:
1 - Starter_Shoe [ATK: 0, DEF: 0, SPD: 10]
2 - Starter_Armor [ATK: 0, DEF: 10, SPD: 0]
3 - Starter_Sword [ATK: 10, DEF: 2, SPD: 0]
4 - Starter_Shield [ATK: 2, DEF: 10, SPD: 0]
5 - HP Potion [ATK: 0, DEF: 30, SPD: 0]
6 - HP Potion [ATK: 0, DEF: 30, SPD: 0]
7 - HP Potion [ATK: 0, DEF: 30, SPD: 0]
8 - Dagger [ATK: 15, DEF: 0, SPD: 5]


==================================================
Current Equipped weapon:
Starter_Sword [ATK: 10, DEF: 2, SPD: 0]
Current Equipped shield:
Starter_Shield [ATK: 2, DEF: 10, SPD: 0]
Current Equipped shoes:
Starter_Shoe [ATK: 0, DEF: 0, SPD: 10]
Current Equipped armor:
Starter_Armor [ATK: 0, DEF: 10, SPD: 0]


Choose an item to equip (enter the number to select):
```

The player checks what item he are equipping and equip the dagger

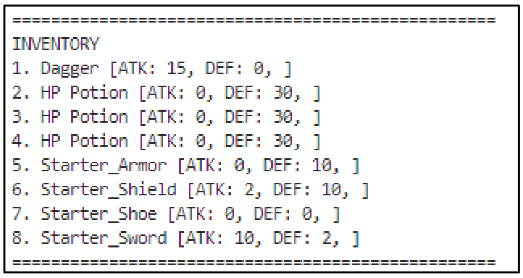


```
Choose an item to equip (enter the number to select): 8
Unequipped Starter_Sword. ATK: 18, DEF: 31, SPD: 21
Equipped Dagger. ATK: 33, DEF: 31, SPD: 26
Equipped item: Dagger
```



```
DeathSoulKing has been defeated!
You won the battle!
Congratulations! You cleared Stage 10.
Congratulations! You have defeated all enemies and won the game!
```

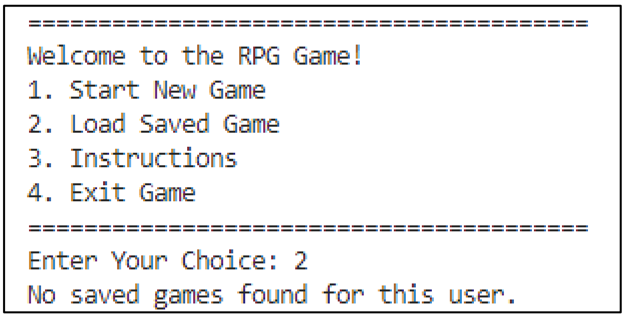
After 9 stages, the player defeats the final boss of the game, and the system displays the winning message.



```
==================================================
INVENTORY
1. Dagger [ATK: 15, DEF: 0, ]
2. HP Potion [ATK: 0, DEF: 30, ]
3. HP Potion [ATK: 0, DEF: 30, ]
4. HP Potion [ATK: 0, DEF: 30, ]
5. Starter_Armor [ATK: 0, DEF: 10, ]
6. Starter_Shield [ATK: 2, DEF: 10, ]
7. Starter_Shoe [ATK: 0, DEF: 0, ]
8. Starter_Sword [ATK: 10, DEF: 2, ]
==================================================
```

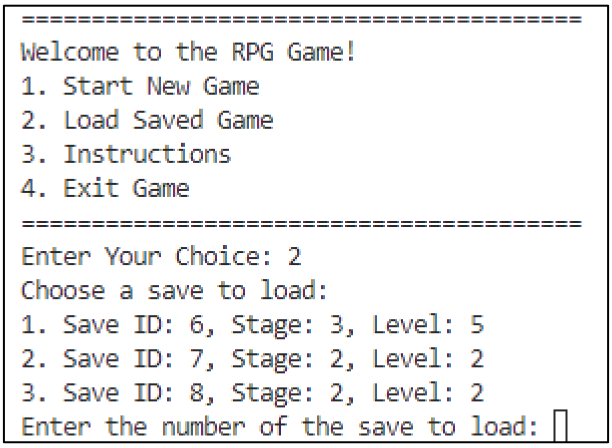
The battle interface view after equipping “Fireball” and “Shield”

Save / Load game function



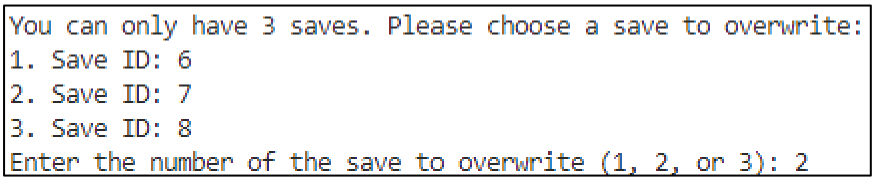
```
========================================
Welcome to the RPG Game!
1. Start New Game
2. Load Saved Game
3. Instructions
4. Exit Game
========================================
Enter Your Choice: 2
No saved games found for this user.
```

This is how the interface will look when a player’s account doesn't have any saved games, but the player still chooses to load a game.



```
========================================
Welcome to the RPG Game!
1. Start New Game
2. Load Saved Game
3. Instructions
4. Exit Game
========================================
Enter Your Choice: 2
Choose a save to load:
1. Save ID: 6, Stage: 3, Level: 5
2. Save ID: 7, Stage: 2, Level: 2
3. Save ID: 8, Stage: 2, Level: 2
Enter the number of the save to load:
```

This is how the interface will look when a player chooses to load a game, and their account has multiple saved games.



```
You can only have 3 saves. Please choose a save to overwrite:
1. Save ID: 6
2. Save ID: 7
3. Save ID: 8
Enter the number of the save to overwrite (1, 2, or 3): 2
```

If the player wants to save their game progress but already has 3 saved games, this interface will appear, asking which saved game they want to overwrite.

Testing the solution

Final test plan

Functional requirements

Functional requirements	Tested/ Achieved
1. User Interface	
1.1 Display a text-based menu system for navigating through the game options.	**Yes**
1.2 Provide options for starting a new game, loading a saved game, viewing the instruction, and exiting the game.	**Yes**
2. Player Management	
2.1 Maintain player stats, including HP, attack, defense, speed, and experience points.	**Yes**
2.2 Track player inventory and allow item usage during battles.	**Yes**
3. Stage and Enemy Management	
3.1 Implement a series of stages with increasing difficulty by the main stage.	**Yes**
3.2 Each stage will have a predefined set of enemies with unique stats.	**Yes**
4. Combat System	
4.1 Implement a turn-based combat system where the player and enemy take turns attacking based on their speed.	**Yes**
4.2 Include basic attack options, special skills, and item usage (e.g., healing potions).	**Yes**
4.3 Display the outcome of each battle and update player stats accordingly.	**Yes**
5. Database and Data Management	
5.1 Use a database to store player profiles, enemy information, and stage data.	**Yes**
5.2 Retrieve data in real time during gameplay.	**Yes**
6. Sorting and Searching	
6.1 Implement sorting algorithms (e.g., bubble sort, insertion sort) for organizing player inventory.	**Yes**
6.2 Use binary search for quick lookup of player items and enemy data during battles.	**Yes**
7. Saving and Loading	
7.1 Allow players to save their progress after completing a stage.	**Yes**
7.2 Load saved game data to continue from the last saved point.	**Yes**

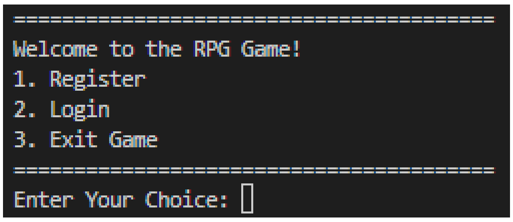
7.3 Handle saved files securely to prevent data loss.	**Yes**
8. Login System	
8.1 User Registration	
8.1.1 Players can create a new account by entering a unique username and password.	**Yes**
8.1.2 The system will check that the username does not already exist in the file and save the username and password securely.	**Yes**
8.2 User Authentication	
8.2.1 Players will log in using their username and password.	**Yes**
8.2.2 The system will verify the entered credentials by checking the file, and if they match, it will load the player's saved game data.	**Yes**

End User Requirements

Functional requirements	**Tested/ Achieved**	**Comment**
1. Simple and Clean Interface Design		
1.1 A game screen that is not cluttered, making it easy to focus on playing.	**Yes**	The game screen that is displayed is simple and clean that the player could easily know which part of the screen is the focus point.
2. Easy Navigation		
2.1 Clear and simple menus to help players find what they need quickly.	**Yes**	The menu has been design that just have a simple text and what letter needed to type in to choose the option in the menu
3. Straightforward Battle System		
3.1 A combat system that is easy to learn and play, with clear instructions.	**Yes**	The combat system has been design that the mechanics of the game will be introduce one by one during the progress of the game stages
4. Clear Stage Progression		
4.1 Easy-to-follow paths between game stages, showing players where to go next.	**Yes**	After the player finishes a stage a option menu will be pop up to let the player organize before starting another new stage or just start another stage straight away
5. User-Friendly Controls		
5.1 Easy-to-use controls	**Yes**	Most of the control input that the player needs to input is just one single letter input.

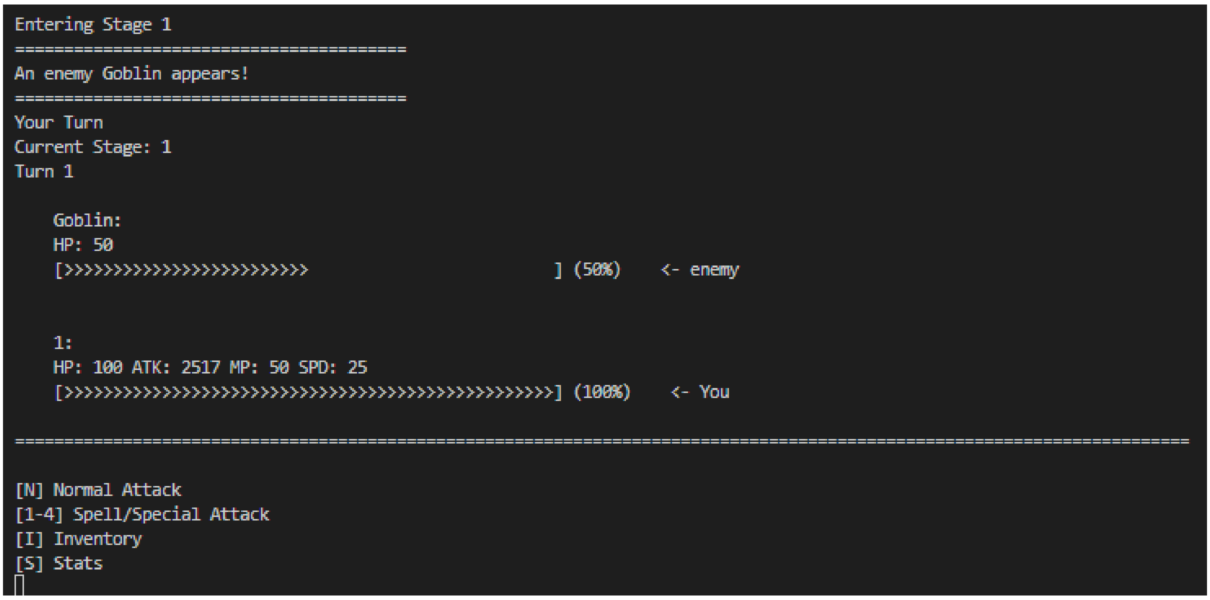
Evidence of Testing

1.1 Display a text-based menu system for navigating through the game options.



```
========================================
Welcome to the RPG Game!
1. Register
2. Login
3. Exit Game
========================================
Enter Your Choice: 
```

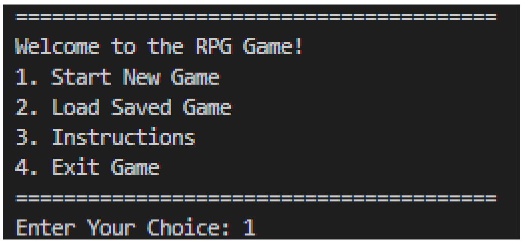
Having text-based menu for register and login



```
Entering Stage 1
========================================
An enemy Goblin appears!
========================================
Your Turn
Current Stage: 1
Turn 1

    Goblin:
    HP: 50
    [>>>>>>>>>>>>>>>>>>>>>>>>>                         ] (50%)    <- enemy


    1:
    HP: 100 ATK: 2517 MP: 50 SPD: 25
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (100%)    <- You

========================================================================================================================

[N] Normal Attack
[1-4] Spell/Special Attack
[I] Inventory
[S] Stats
```

Having clean and tidy text-based menu options for the battle interface

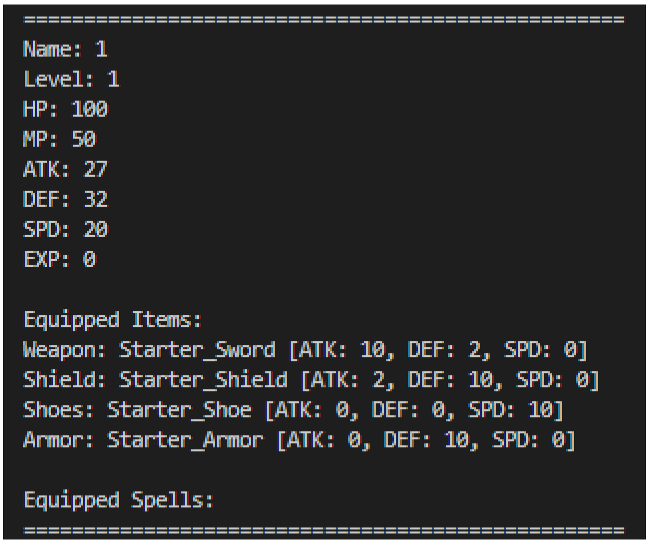
1.2 Provide options for starting a new game, loading a saved game, viewing the instruction, and exiting the game.



```
========================================
Welcome to the RPG Game!
1. Start New Game
2. Load Saved Game
3. Instructions
4. Exit Game
========================================
Enter Your Choice: 1
```

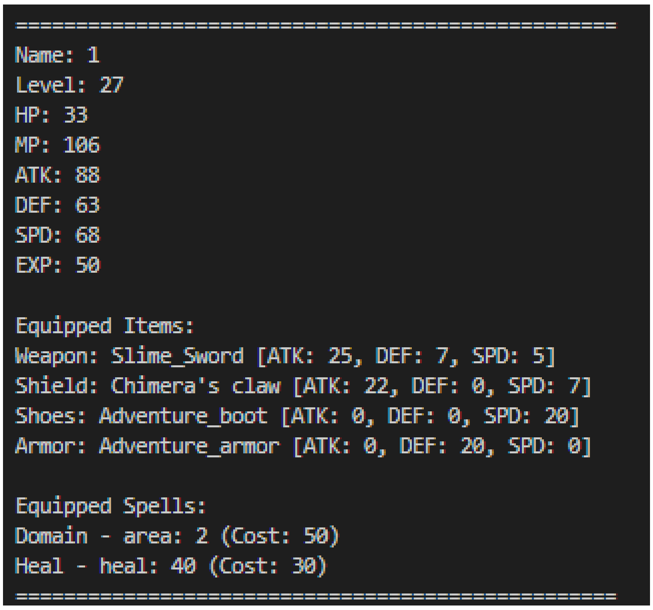
Having clear menu to provide all options before staring playing the game

2.1 Maintain player stats, including HP, attack, defense, speed, and experience points.



```
==================================================
Name: 1
Level: 1
HP: 100
MP: 50
ATK: 27
DEF: 32
SPD: 20
EXP: 0

Equipped Items:
Weapon: Starter_Sword [ATK: 10, DEF: 2, SPD: 0]
Shield: Starter_Shield [ATK: 2, DEF: 10, SPD: 0]
Shoes: Starter_Shoe [ATK: 0, DEF: 0, SPD: 10]
Armor: Starter_Armor [ATK: 0, DEF: 10, SPD: 0]

Equipped Spells:
==================================================
```

This is the status of the player at stage 1 with starter equipment



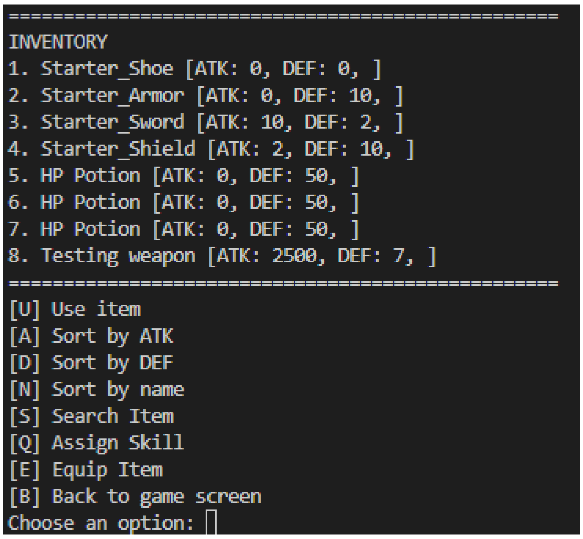
```
==================================================
Name: 1
Level: 27
HP: 33
MP: 106
ATK: 88
DEF: 63
SPD: 68
EXP: 50

Equipped Items:
Weapon: Slime_Sword [ATK: 25, DEF: 7, SPD: 5]
Shield: Chimera's claw [ATK: 22, DEF: 0, SPD: 7]
Shoes: Adventure_boot [ATK: 0, DEF: 0, SPD: 20]
Armor: Adventure_armor [ATK: 0, DEF: 20, SPD: 0]

Equipped Spells:
Domain - area: 2 (Cost: 50)
Heal - heal: 40 (Cost: 30)
==================================================
```

This is the status of the player at stage 7 and have swap out some of the equipment, also have equip some spells.

By these evidence, this could show that the program could maintain the player stats and equipped equipment and spells.

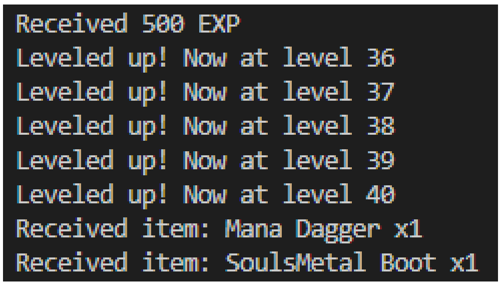
2.2 Track player inventory and allow item usage during battles.

Track player inventory



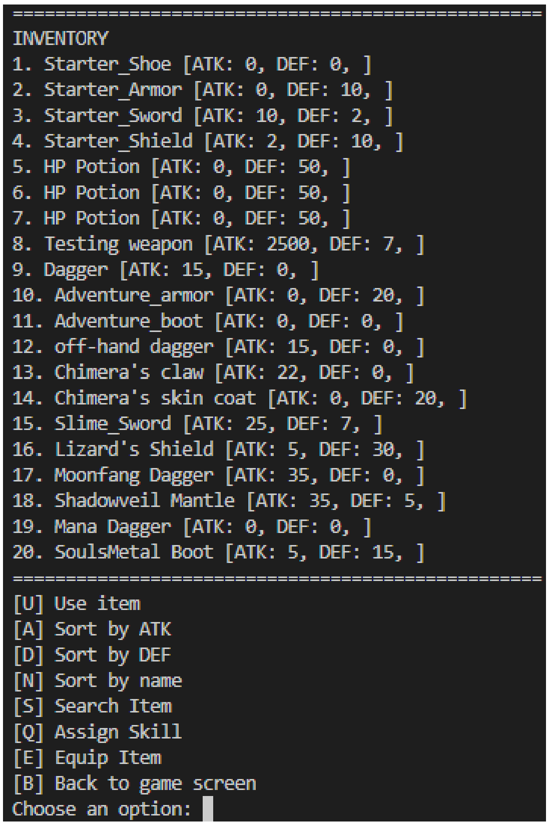
```
====================================================
INVENTORY
1. Starter_Shoe [ATK: 0, DEF: 0, ]
2. Starter_Armor [ATK: 0, DEF: 10, ]
3. Starter_Sword [ATK: 10, DEF: 2, ]
4. Starter_Shield [ATK: 2, DEF: 10, ]
5. HP Potion [ATK: 0, DEF: 50, ]
6. HP Potion [ATK: 0, DEF: 50, ]
7. HP Potion [ATK: 0, DEF: 50, ]
8. Testing weapon [ATK: 2500, DEF: 7, ]
====================================================
[U] Use item
[A] Sort by ATK
[D] Sort by DEF
[N] Sort by name
[S] Search Item
[Q] Assign Skill
[E] Equip Item
[B] Back to game screen
Choose an option:
```

This is the inventory when the player at stage 1



```
Received 500 EXP
Leveled up! Now at level 36
Leveled up! Now at level 37
Leveled up! Now at level 38
Leveled up! Now at level 39
Leveled up! Now at level 40
Received item: Mana Dagger x1
Received item: SoulsMetal Boot x1
```

A log that tells the player that a item has putted into their inventory



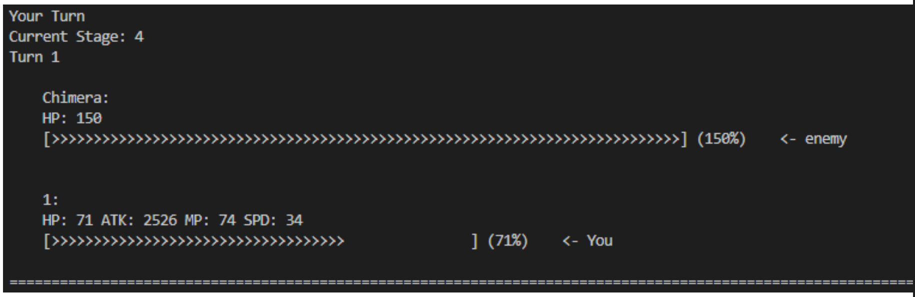
```
====================================================
INVENTORY
1. Starter_Shoe [ATK: 0, DEF: 0, ]
2. Starter_Armor [ATK: 0, DEF: 10, ]
3. Starter_Sword [ATK: 10, DEF: 2, ]
4. Starter_Shield [ATK: 2, DEF: 10, ]
5. HP Potion [ATK: 0, DEF: 50, ]
6. HP Potion [ATK: 0, DEF: 50, ]
7. HP Potion [ATK: 0, DEF: 50, ]
8. Testing weapon [ATK: 2500, DEF: 7, ]
9. Dagger [ATK: 15, DEF: 0, ]
10. Adventure_armor [ATK: 0, DEF: 20, ]
11. Adventure_boot [ATK: 0, DEF: 0, ]
12. off-hand dagger [ATK: 15, DEF: 0, ]
13. Chimera's claw [ATK: 22, DEF: 0, ]
14. Chimera's skin coat [ATK: 0, DEF: 20, ]
15. Slime_Sword [ATK: 25, DEF: 7, ]
16. Lizard's Shield [ATK: 5, DEF: 30, ]
17. Moonfang Dagger [ATK: 35, DEF: 0, ]
18. Shadowveil Mantle [ATK: 35, DEF: 5, ]
19. Mana Dagger [ATK: 0, DEF: 0, ]
20. SoulsMetal Boot [ATK: 5, DEF: 15, ]
====================================================
[U] Use item
[A] Sort by ATK
[D] Sort by DEF
[N] Sort by name
[S] Search Item
[Q] Assign Skill
[E] Equip Item
[B] Back to game screen
Choose an option:
```

This is the inventory when the player at stage 9

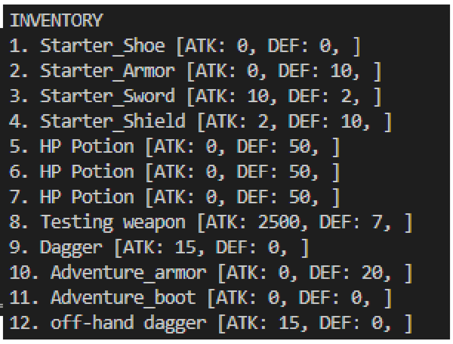
From these two evidence, this could show that the program could keep track on the item in and out in the inventory.

Item Usage

For now the just have one item could be used while in a battle, it's HP potion, so I will test three time to see is the potion heals the correct value (50%) also the potion should be used up and disapper in the inventory.



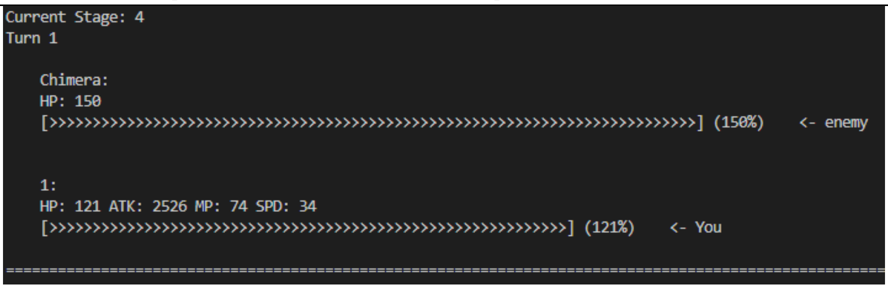
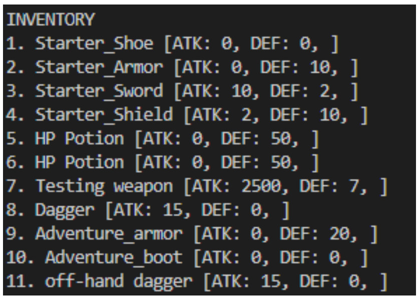
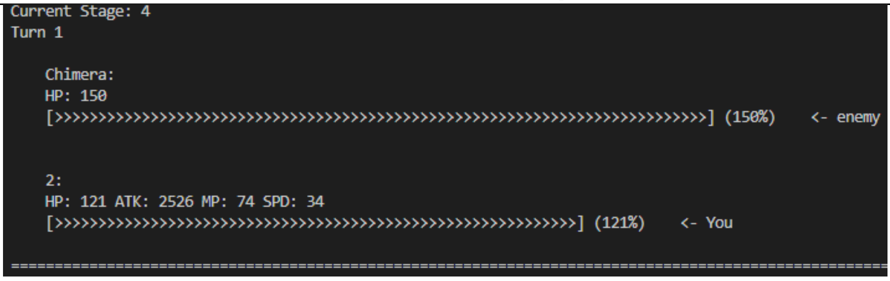
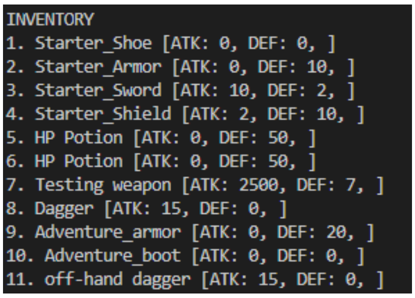
```
Your Turn
Current Stage: 4
Turn 1

    Chimera:
    HP: 150
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (150%)    <- enemy


    1:
    HP: 71 ATK: 2526 MP: 74 SPD: 34
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>               ] (71%)    <- You

==============================================================================================================
```

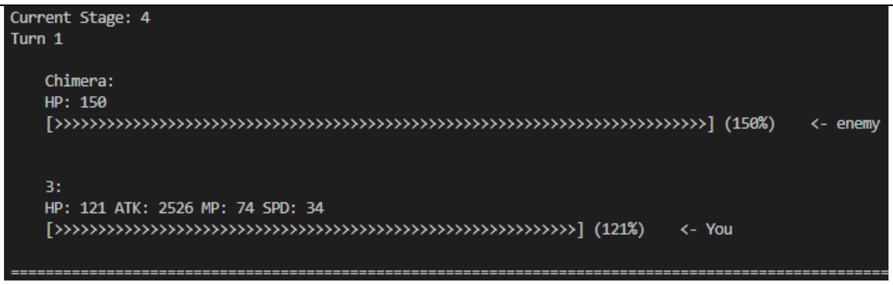
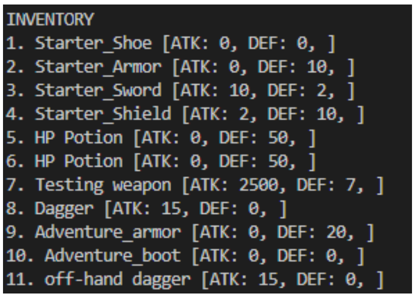


```
INVENTORY
1. Starter_Shoe [ATK: 0, DEF: 0, ]
2. Starter_Armor [ATK: 0, DEF: 10, ]
3. Starter_Sword [ATK: 10, DEF: 2, ]
4. Starter_Shield [ATK: 2, DEF: 10, ]
5. HP Potion [ATK: 0, DEF: 50, ]
6. HP Potion [ATK: 0, DEF: 50, ]
7. HP Potion [ATK: 0, DEF: 50, ]
8. Testing weapon [ATK: 2500, DEF: 7, ]
9. Dagger [ATK: 15, DEF: 0, ]
10. Adventure_armor [ATK: 0, DEF: 20, ]
11. Adventure_boot [ATK: 0, DEF: 0, ]
12. off-hand dagger [ATK: 15, DEF: 0, ]
```

This is the original HP of the player (71%)

The inventory view before using the potion

Testing times	Result (expected to be heal up to 121%)
1st	 Current Stage: 4 Turn 1 Chimera: HP: 150 [>>>] (150%) <- enemy 1: HP: 121 ATK: 2526 MP: 74 SPD: 34 [>>] (121%) <- You == **(121%)**  INVENTORY 1. Starter_Shoe [ATK: 0, DEF: 0,] 2. Starter_Armor [ATK: 0, DEF: 10,] 3. Starter_Sword [ATK: 10, DEF: 2,] 4. Starter_Shield [ATK: 2, DEF: 10,] 5. HP Potion [ATK: 0, DEF: 50,] 6. HP Potion [ATK: 0, DEF: 50,] 7. Testing weapon [ATK: 2500, DEF: 7,] 8. Dagger [ATK: 15, DEF: 0,] 9. Adventure_armor [ATK: 0, DEF: 20,] 10. Adventure_boot [ATK: 0, DEF: 0,] 11. off-hand dagger [ATK: 15, DEF: 0,] **Yes the potion has been used up**
2nd	 Current Stage: 4 Turn 1 Chimera: HP: 150 [>>>] (150%) <- enemy 2: HP: 121 ATK: 2526 MP: 74 SPD: 34 [>>] (121%) <- You == **(121%)**  INVENTORY 1. Starter_Shoe [ATK: 0, DEF: 0,] 2. Starter_Armor [ATK: 0, DEF: 10,] 3. Starter_Sword [ATK: 10, DEF: 2,] 4. Starter_Shield [ATK: 2, DEF: 10,] 5. HP Potion [ATK: 0, DEF: 50,] 6. HP Potion [ATK: 0, DEF: 50,] 7. Testing weapon [ATK: 2500, DEF: 7,] 8. Dagger [ATK: 15, DEF: 0,] 9. Adventure_armor [ATK: 0, DEF: 20,] 10. Adventure_boot [ATK: 0, DEF: 0,] 11. off-hand dagger [ATK: 15, DEF: 0,] **Yes the potion has been used up**

3rd	 Current Stage: 4 Turn 1 Chimera: HP: 150 [>>>] (150%) <- enemy 3: HP: 121 ATK: 2526 MP: 74 SPD: 34 [>>] (121%) <- You == **(121%)**  INVENTORY 1. Starter_Shoe [ATK: 0, DEF: 0,] 2. Starter_Armor [ATK: 0, DEF: 10,] 3. Starter_Sword [ATK: 10, DEF: 2,] 4. Starter_Shield [ATK: 2, DEF: 10,] 5. HP Potion [ATK: 0, DEF: 50,] 6. HP Potion [ATK: 0, DEF: 50,] 7. Testing weapon [ATK: 2500, DEF: 7,] 8. Dagger [ATK: 15, DEF: 0,] 9. Adventure_armor [ATK: 0, DEF: 20,] 10. Adventure_boot [ATK: 0, DEF: 0,] 11. off-hand dagger [ATK: 15, DEF: 0,] **Yes the potion has been used up**

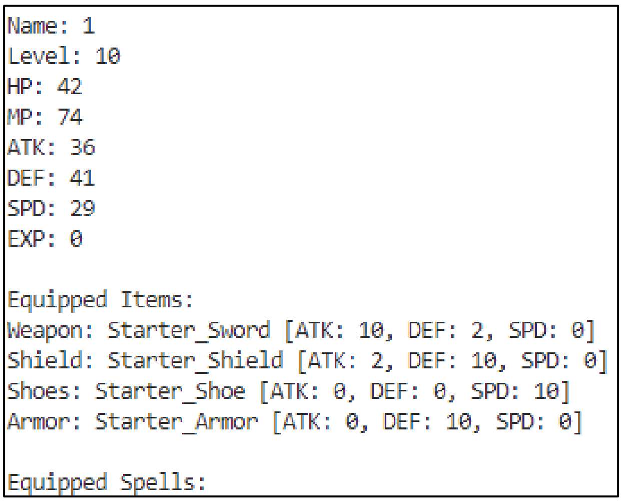
From these 3 result, I could see that the potion is working correctly. It’s healing the player by 50hp (50%) per one potion also the potion will be used up after using it to heal.

Equipping Items

I will test that is the status changing when a item is equipped, also the equipped equipment name displayed in the status page changes.

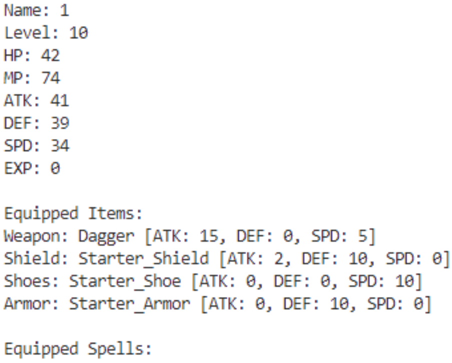
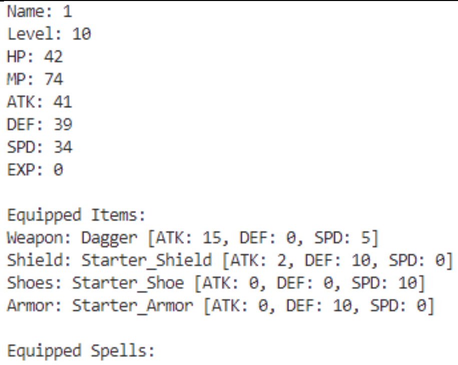
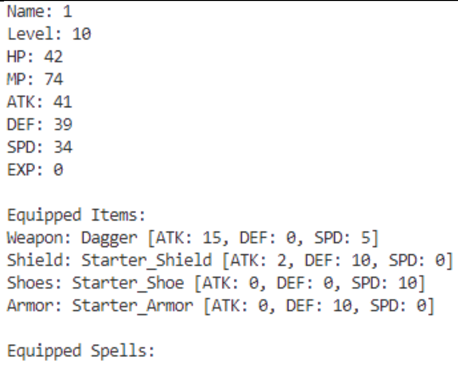
For this testing I will just swap out the “Starter_Sword” to “Dagger”, only “ATK” , “DEF” and “SPD” will change after the change of equipment. Also I will test 3 times.

I expect the “ATK” will increase to **41**, “DEF” will decrease to **39** and “SPD” will increase to **34**, and the displayed weapon name will be change to “Weapon: Dagger [ATK: 15, DEF: 0, SPD: 5]”



```
Name: 1
Level: 10
HP: 42
MP: 74
ATK: 36
DEF: 41
SPD: 29
EXP: 0

Equipped Items:
Weapon: Starter_Sword [ATK: 10, DEF: 2, SPD: 0]
Shield: Starter_Shield [ATK: 2, DEF: 10, SPD: 0]
Shoes: Starter_Shoe [ATK: 0, DEF: 0, SPD: 10]
Armor: Starter_Armor [ATK: 0, DEF: 10, SPD: 0]

Equipped Spells:
```

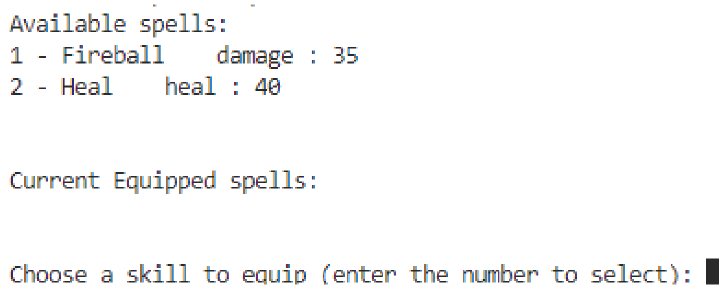
Original status

Testing times	**Result**
1st	 Name: 1 Level: 10 HP: 42 MP: 74 ATK: 41 DEF: 39 SPD: 34 EXP: 0 Equipped Items: Weapon: Dagger [ATK: 15, DEF: 0, SPD: 5] Shield: Starter_Shield [ATK: 2, DEF: 10, SPD: 0] Shoes: Starter_Shoe [ATK: 0, DEF: 0, SPD: 10] Armor: Starter_Armor [ATK: 0, DEF: 10, SPD: 0] Equipped Spells:
2nd	 Name: 1 Level: 10 HP: 42 MP: 74 ATK: 41 DEF: 39 SPD: 34 EXP: 0 Equipped Items: Weapon: Dagger [ATK: 15, DEF: 0, SPD: 5] Shield: Starter_Shield [ATK: 2, DEF: 10, SPD: 0] Shoes: Starter_Shoe [ATK: 0, DEF: 0, SPD: 10] Armor: Starter_Armor [ATK: 0, DEF: 10, SPD: 0] Equipped Spells:
3rd	 Name: 1 Level: 10 HP: 42 MP: 74 ATK: 41 DEF: 39 SPD: 34 EXP: 0 Equipped Items: Weapon: Dagger [ATK: 15, DEF: 0, SPD: 5] Shield: Starter_Shield [ATK: 2, DEF: 10, SPD: 0] Shoes: Starter_Shoe [ATK: 0, DEF: 0, SPD: 10] Armor: Starter_Armor [ATK: 0, DEF: 10, SPD: 0] Equipped Spells:

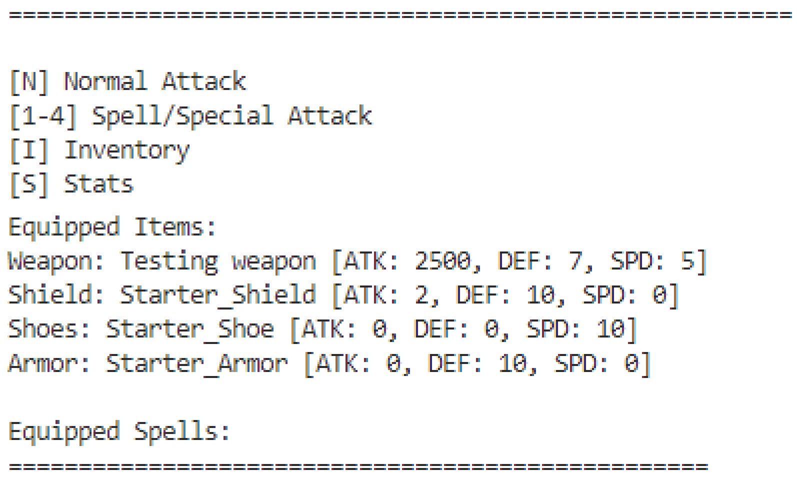
Equipping Spells

I will test that the equipping spell/skill function, I will check that after equipping the spell will the battle interface spell part will display the out the spell that equipped and has the status has display out the spell that equipped.

For the testing I will equip the two available spells “Fireball” and “Heal”, and check that has the battle interface and status page has changed.



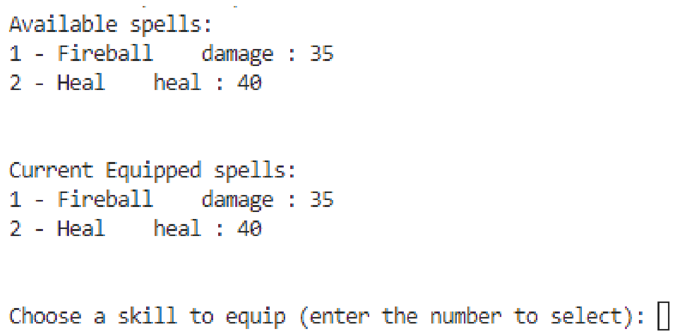
```
Available spells:
1 - Fireball    damage : 35
2 - Heal    heal : 40


Current Equipped spells:


Choose a skill to equip (enter the number to select): █
```



```
=========================================================

[N] Normal Attack
[1-4] Spell/Special Attack
[I] Inventory
[S] Stats
Equipped Items:
Weapon: Testing weapon [ATK: 2500, DEF: 7, SPD: 5]
Shield: Starter_Shield [ATK: 2, DEF: 10, SPD: 0]
Shoes: Starter_Shoe [ATK: 0, DEF: 0, SPD: 10]
Armor: Starter_Armor [ATK: 0, DEF: 10, SPD: 0]

Equipped Spells:
===================================================
```

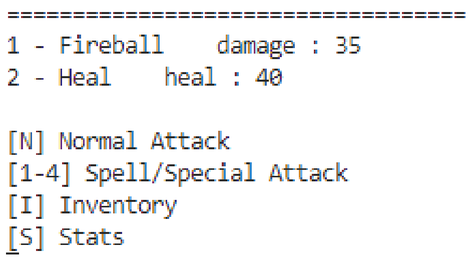
This is what the status and battle interface showing before equipping the spells

Equipping Spells (continue)



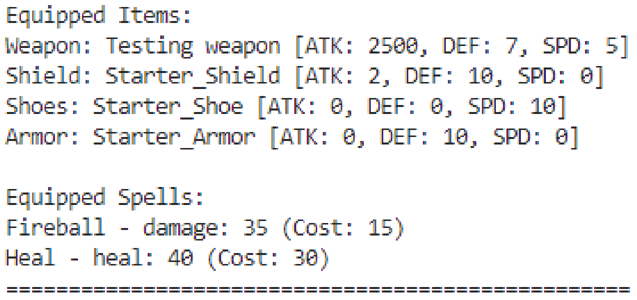
```
Available spells:
1 - Fireball    damage : 35
2 - Heal    heal : 40


Current Equipped spells:
1 - Fireball    damage : 35
2 - Heal    heal : 40


Choose a skill to equip (enter the number to select): 
```

After I equipped the two spells, this is how the assign spell interface looks. The two equipped spell is listed under “Current Equipped spells:”



```
===================================
1 - Fireball    damage : 35
2 - Heal    heal : 40

[N] Normal Attack
[1-4] Spell/Special Attack
[I] Inventory
[S] Stats
```

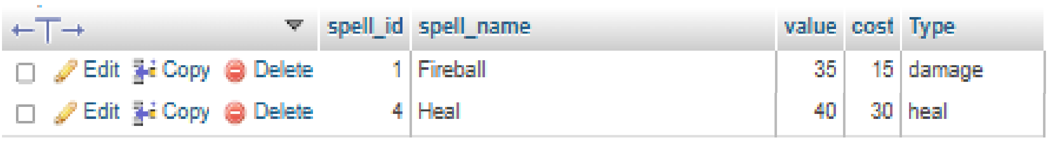
After equipping the spells, the battle interface has display out the spell equipped with the index of the spell.



```
Equipped Items:
Weapon: Testing weapon [ATK: 2500, DEF: 7, SPD: 5]
Shield: Starter_Shield [ATK: 2, DEF: 10, SPD: 0]
Shoes: Starter_Shoe [ATK: 0, DEF: 0, SPD: 10]
Armor: Starter_Armor [ATK: 0, DEF: 10, SPD: 0]

Equipped Spells:
Fireball - damage: 35 (Cost: 15)
Heal - heal: 40 (Cost: 30)
==================================================
```

After equipping the spells, the status page has listed out the spell with their function, damage and the cost of the spell.

Check if the spell displayed is matched with the database



←T→	spell_id	spell_name	value	cost	Type
□ Edit Copy Delete	1	Fireball	35	15	damage
□ Edit Copy Delete	4	Heal	40	30	heal

3.1 Implement a series of stages with increasing difficulty by the main stage.

To ensure a balanced and engaging gameplay experience, I implemented a series of stages with progressively increasing difficulty. This feature was designed to challenge players as they advance through the game while maintaining a fair learning curve.

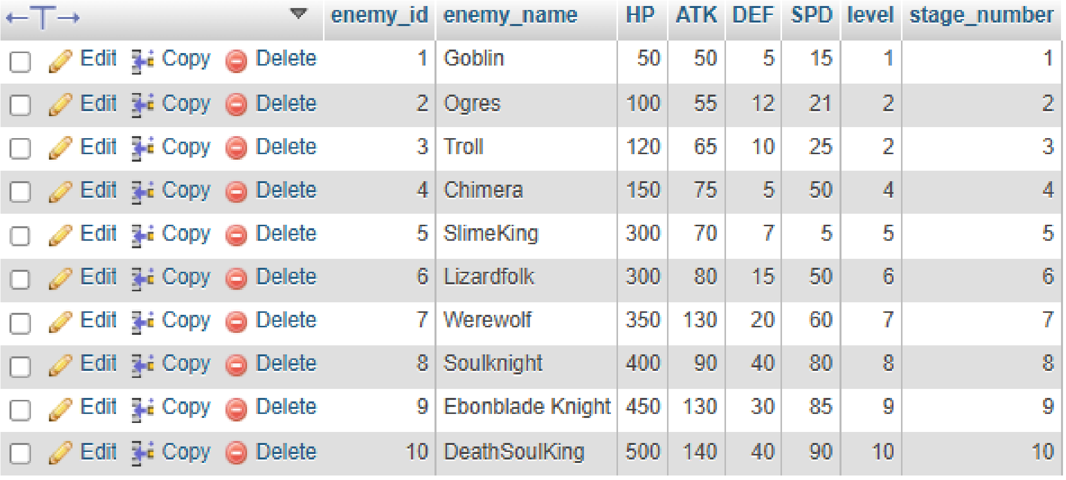
Design concept of the difficulty

For my thinking of designing the difficulty, I want to design the difficulty curve just be a slope just going up, I want it to be sloping up to a point, the game will introduce a new spell or mechanic to the game, this could make the game more fun to play and also may lower the difficulty a bit, but it will rise back up until another spell or mechanic is introduced.

3.2 Each stage will have a predefined set of enemies with unique stats.

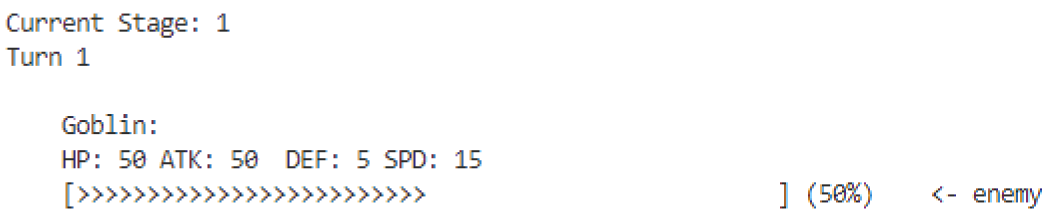
For testing the predefined set of enemies with unique stats, I will check do the enemy in the program matches the data in the database by each stage.

(For testing this I have added some code to playerTurn.py to let the battle interface display the enemy stats)



←T→				enemy_id	enemy_name	HP	ATK	DEF	SPD	level	stage_number
□	Edit	Copy	Delete	1	Goblin	50	50	5	15	1	1
□	Edit	Copy	Delete	2	Ogres	100	55	12	21	2	2
□	Edit	Copy	Delete	3	Troll	120	65	10	25	2	3
□	Edit	Copy	Delete	4	Chimera	150	75	5	50	4	4
□	Edit	Copy	Delete	5	SlimeKing	300	70	7	5	5	5
□	Edit	Copy	Delete	6	Lizardfolk	300	80	15	50	6	6
□	Edit	Copy	Delete	7	Werewolf	350	130	20	60	7	7
□	Edit	Copy	Delete	8	Soulknight	400	90	40	80	8	8
□	Edit	Copy	Delete	9	Ebonblade Knight	450	130	30	85	9	9
□	Edit	Copy	Delete	10	DeathSoulKing	500	140	40	90	10	10

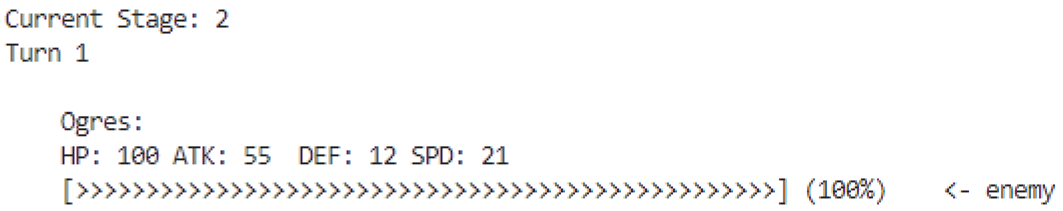
Stage 1



```
Current Stage: 1
Turn 1

    Goblin:
    HP: 50 ATK: 50  DEF: 5 SPD: 15
    [>>>>>>>>>>>>>>>>>>>>>>>>>                         ] (50%)    <- enemy
```

Do the data matches (Yes/No)					
Name	**HP**	**ATK**	**DEF**	**SPD**	**Appear in the correct stage**
Yes	Yes	Yes	Yes	Yes	Yes

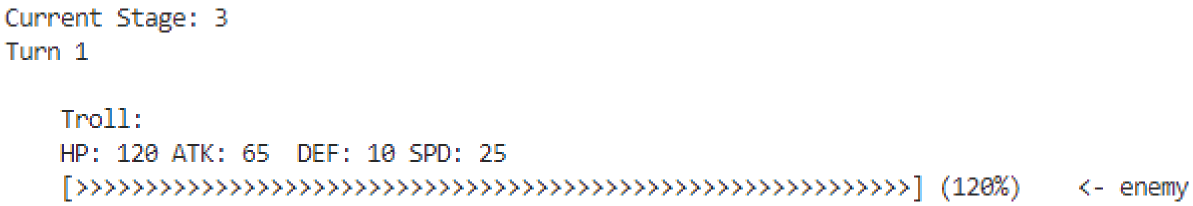
Stage 2



```
Current Stage: 2
Turn 1

    Ogres:
    HP: 100 ATK: 55  DEF: 12 SPD: 21
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (100%)    <- enemy
```

Do the data matches (Yes/No)					
Name	**HP**	**ATK**	**DEF**	**SPD**	**Appear in the correct stage**
Yes	Yes	Yes	Yes	Yes	Yes

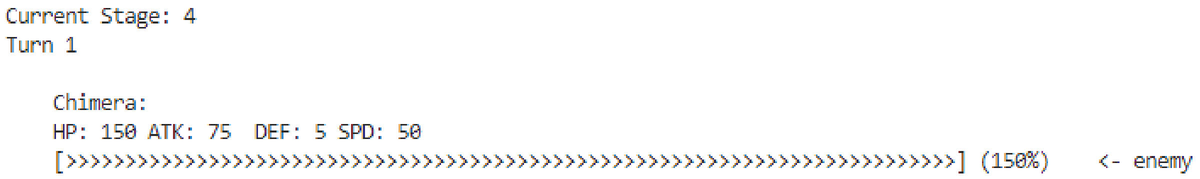
Stage 3



```
Current Stage: 3
Turn 1

    Troll:
    HP: 120 ATK: 65  DEF: 10 SPD: 25
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (120%)    <- enemy
```

Do the data matches (Yes/No)					
Name	**HP**	**ATK**	**DEF**	**SPD**	**Appear in the correct stage**
Yes	Yes	Yes	Yes	Yes	Yes

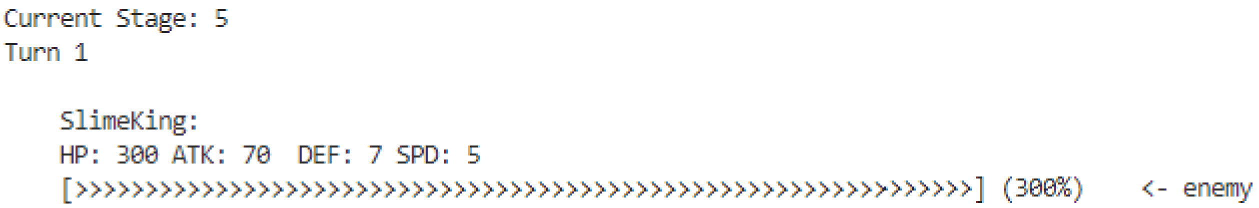
Stage 4



```
Current Stage: 4
Turn 1

    Chimera:
    HP: 150 ATK: 75  DEF: 5 SPD: 50
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (150%)    <- enemy
```

Do the data matches (Yes/No)					
Name	**HP**	**ATK**	**DEF**	**SPD**	**Appear in the correct stage**
Yes	Yes	Yes	Yes	Yes	Yes

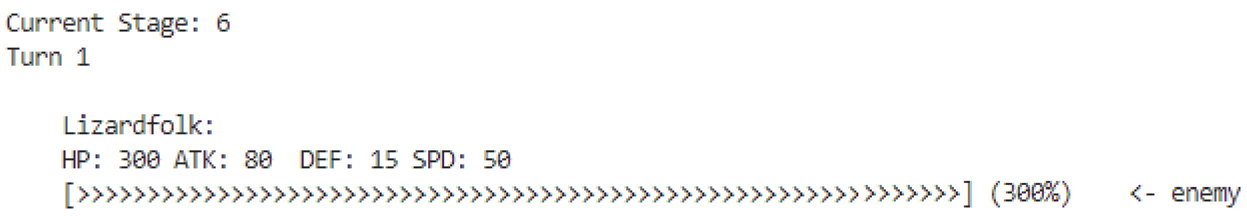
Stage 5



```
Current Stage: 5
Turn 1

    SlimeKing:
    HP: 300 ATK: 70  DEF: 7 SPD: 5
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (300%)    <- enemy
```

Do the data matches (Yes/No)					
Name	**HP**	**ATK**	**DEF**	**SPD**	**Appear in the correct stage**
Yes	Yes	Yes	Yes	Yes	Yes

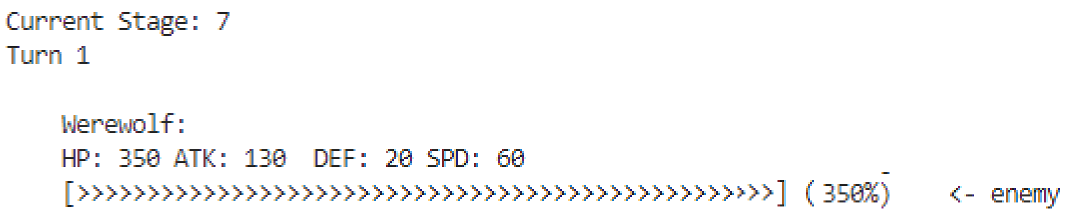
Stage 6



```
Current Stage: 6
Turn 1

    Lizardfolk:
    HP: 300 ATK: 80  DEF: 15 SPD: 50
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (300%)    <- enemy
```

Do the data matches (Yes/No)					
Name	**HP**	**ATK**	**DEF**	**SPD**	**Appear in the correct stage**
Yes	Yes	Yes	Yes	Yes	Yes

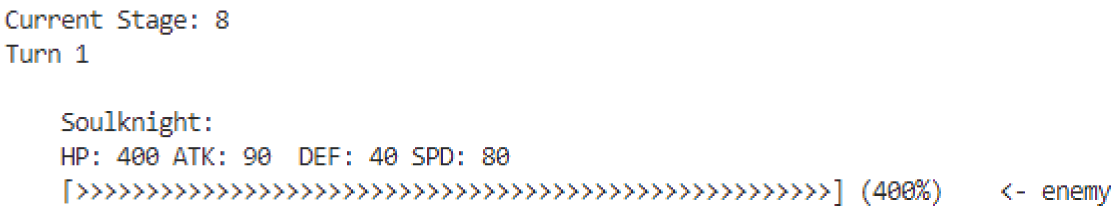
Stage 7



```
Current Stage: 7
Turn 1

    Werewolf:
    HP: 350 ATK: 130  DEF: 20 SPD: 60
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] ( 350%)    <- enemy
```

Do the data matches (Yes/No)					
Name	**HP**	**ATK**	**DEF**	**SPD**	**Appear in the correct stage**
Yes	Yes	Yes	Yes	Yes	Yes

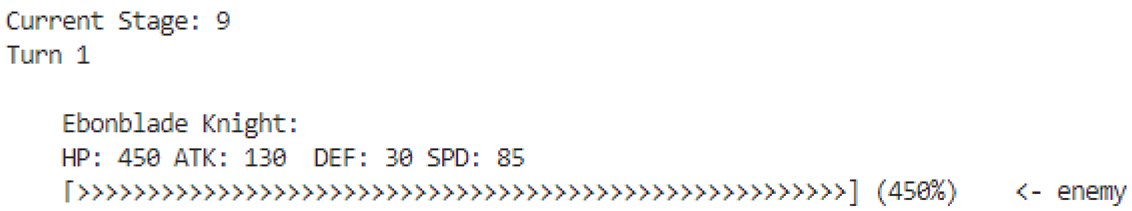
Stage 8



```
Current Stage: 8
Turn 1

    Soulknight:
    HP: 400 ATK: 90  DEF: 40 SPD: 80
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (400%)    <- enemy
```

Do the data matches (Yes/No)					
Name	**HP**	**ATK**	**DEF**	**SPD**	**Appear in the correct stage**
Yes	Yes	Yes	Yes	Yes	Yes

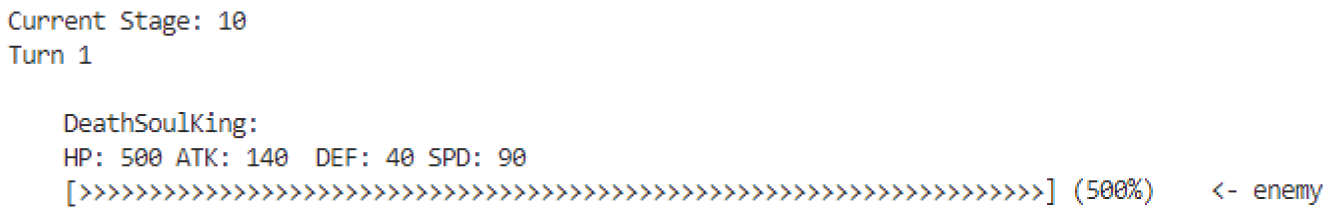
Stage 9



```
Current Stage: 9
Turn 1

    Ebonblade Knight:
    HP: 450 ATK: 130  DEF: 30 SPD: 85
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (450%)    <- enemy
```

Do the data matches (Yes/No)					
Name	**HP**	**ATK**	**DEF**	**SPD**	**Appear in the correct stage**
Yes	Yes	Yes	Yes	Yes	Yes

Stage 10



```
Current Stage: 10
Turn 1

    DeathSoulKing:
    HP: 500 ATK: 140  DEF: 40 SPD: 90
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (500%)    <- enemy
```

Do the data matches (Yes/No)					
Name	**HP**	**ATK**	**DEF**	**SPD**	**Appear in the correct stage**
Yes	Yes	Yes	Yes	Yes	Yes

From all 10 stage of testing, I could confirm that all the stat, name and enemy appear stage is matching to the preset setting in the database.

4.1 Implement a turn-based combat system where the player and enemy take turns attacking based on their speed.

I will test this by testing the program with 3 different situation :

1. Test when the player's speed is greater than the enemy's speed.
2. Test when the player's speed is equal to the enemy's speed.
3. Test when the player's speed is less than the enemy's speed.

Player's speed is greater than the enemy's speed.

(Expected Player will go first)

Player's speed is equal to the enemy's speed.

(Expected Player will go first)

player's speed is less than the enemy's speed.

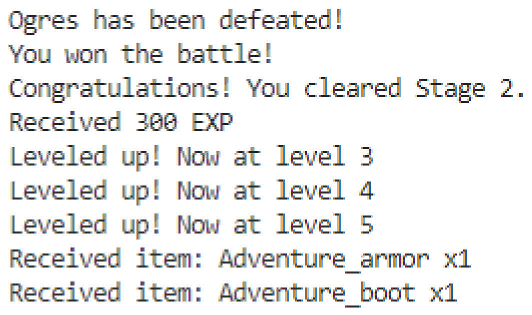
(Expected enemy will go first)

4.2 Include basic attack options, special skills, and item usage (e.g., healing potions).

Item usage I have already tested in **2.2** so here I will just test the basic attack and the spell is them working as I want.

4.3 Display the outcome of each battle and update player stats accordingly.

For each battle ends, the system will display a congrats message out. The system will calls the reward functions and fetch the rewards. The system will display out the items that the player gets and if the player level up the system will display a message out.

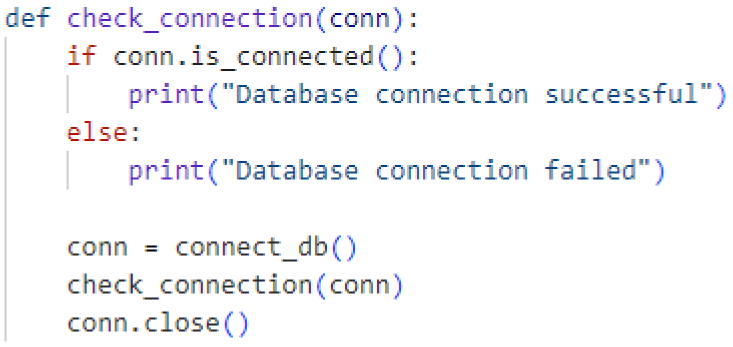


```
Ogres has been defeated!
You won the battle!
Congratulations! You cleared Stage 2.
Received 300 EXP
Leveled up! Now at level 3
Leveled up! Now at level 4
Leveled up! Now at level 5
Received item: Adventure_armor x1
Received item: Adventure_boot x1
```

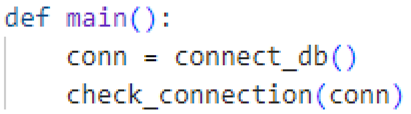
(This is the screen when the player finish a stage)

5.1 Use a database to store player profiles, enemy information, and stage data.

To test the database functionality, I will verify whether the game can connect to the database successfully. Additionally, I will test if users can save and load their data (covered in Section 7.1). The correctness of stage data retrieval has already been tested in Section 3.2



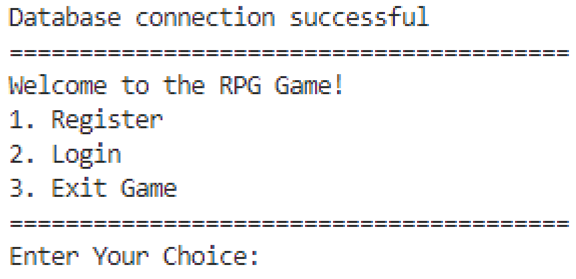
```python
def check_connection(conn):
    if conn.is_connected():
        print("Database connection successful")
    else:
        print("Database connection failed")

    conn = connect_db()
    check_connection(conn)
    conn.close()
```



```python
def main():
    conn = connect_db()
    check_connection(conn)
```

I have added these lines of code in side main.py and database_connection.py to check is the database connected.

Testing result :



```
Database connection successful
========================================
Welcome to the RPG Game!
1. Register
2. Login
3. Exit Game
========================================
Enter Your Choice:
```

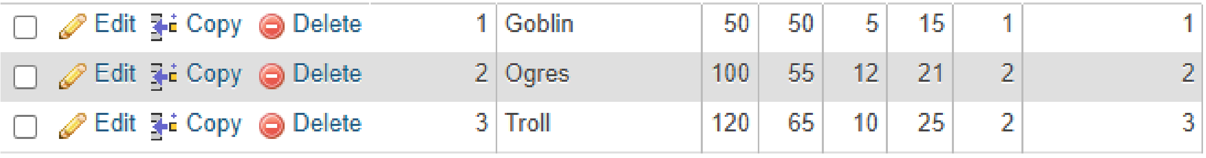
After testing the connection for 3 time I have got the same result.

5.2 Retrieve data in real time during gameplay.

For this part I will do 3 testing:

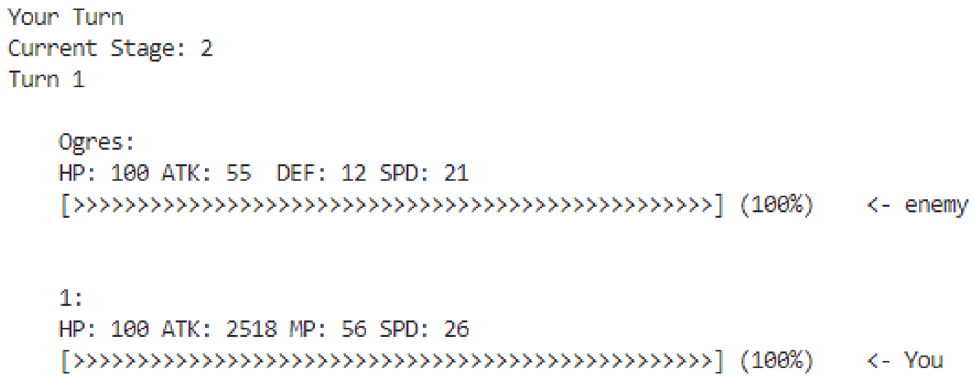
- try to change the value in stage 2 enemy during gameplay, and see is there anything changed
- Try to change the damage of a damage spell, and see do the damage has changed
- Try to change the ATK of a item, and see do the player stat changed

Original :



□	Edit	Copy	Delete	1	Goblin	50	50	5	15	1	1
□	Edit	Copy	Delete	2	Ogres	100	55	12	21	2	2
□	Edit	Copy	Delete	3	Troll	120	65	10	25	2	3

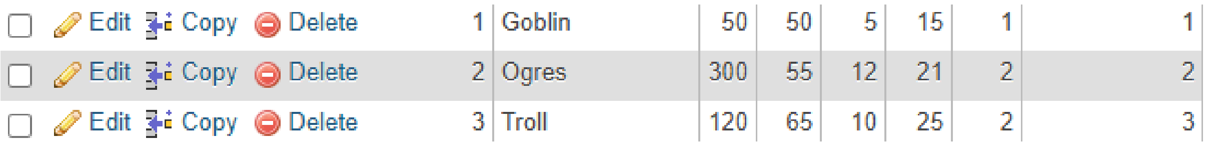
I will change the Ogres Value during the program is running



```
Your Turn
Current Stage: 2
Turn 1

    Ogres:
    HP: 100 ATK: 55  DEF: 12 SPD: 21
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (100%)    <- enemy


    1:
    HP: 100 ATK: 2518 MP: 56 SPD: 26
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (100%)    <- You
```

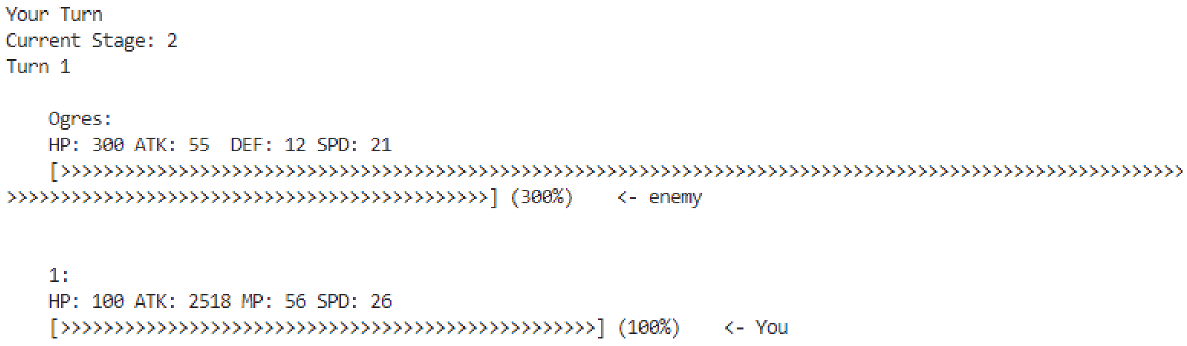
This is the Screen before changing the values

Testing :



□	Edit	Copy	Delete	1	Goblin	50	50	5	15	1	1
□	Edit	Copy	Delete	2	Ogres	300	55	12	21	2	2
□	Edit	Copy	Delete	3	Troll	120	65	10	25	2	3

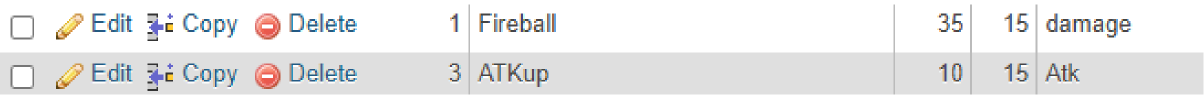
I have changed the hp of the Ogres from 100 to 300



```
Your Turn
Current Stage: 2
Turn 1

    Ogres:
    HP: 300 ATK: 55  DEF: 12 SPD: 21
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (300%)    <- enemy


    1:
    HP: 100 ATK: 2518 MP: 56 SPD: 26
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (100%)    <- You
```

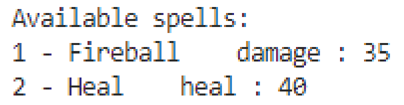
I have re-entered the same stage the hp of the Ogres has changed 100 to 300

Original :



□	Edit	Copy	Delete	1	Fireball	35	15	damage
□	Edit	Copy	Delete	3	ATKup	10	15	Atk

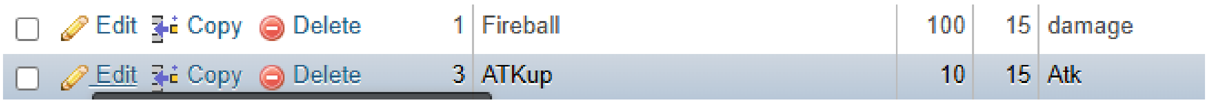
I will change the Fireball damage from 35 to 100



```
Available spells:
1 - Fireball    damage : 35
2 - Heal    heal : 40
```

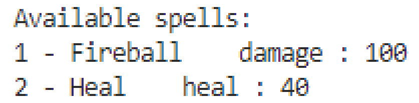
This is the damage that shown in the inventory screen

Testing :



□	Edit	Copy	Delete	1	Fireball	100	15	damage
□	Edit	Copy	Delete	3	ATKup	10	15	Atk

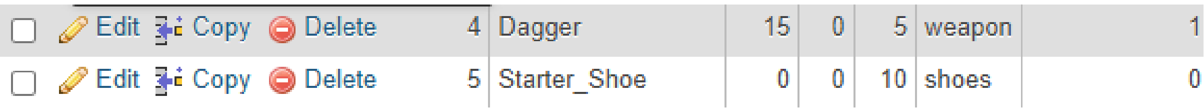
I have changed the Fireball damage to 100



```
Available spells:
1 - Fireball    damage : 100
2 - Heal    heal : 40
```

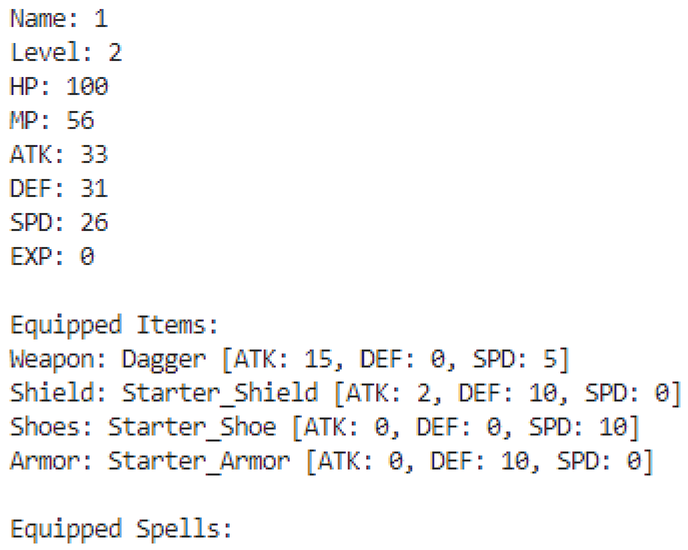
After restarting the game (not the program, just starting a new game), the fireball damage shown in the inventory has changed.

Original :



□	Edit	Copy	Delete	4	Dagger	15	0	5	weapon	1
□	Edit	Copy	Delete	5	Starter_Shoe	0	0	10	shoes	0

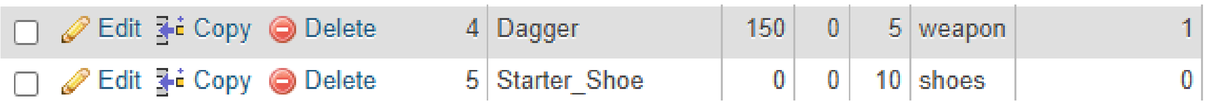
I will change the ATK of the dagger from 15 to 150



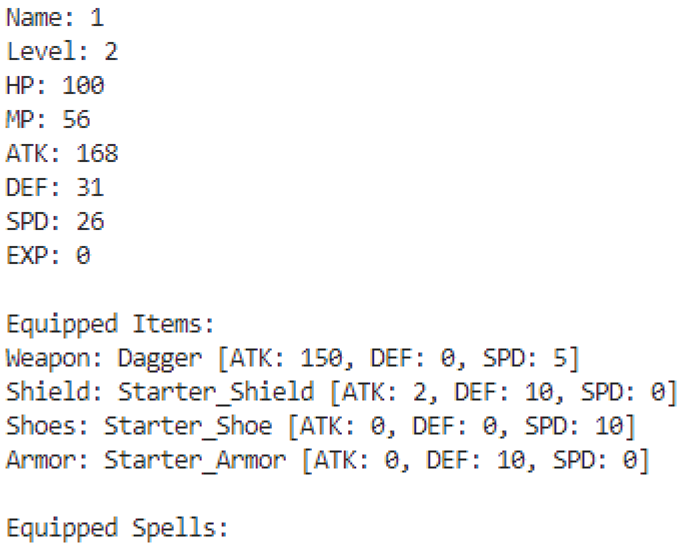
```
Name: 1
Level: 2
HP: 100
MP: 56
ATK: 33
DEF: 31
SPD: 26
EXP: 0

Equipped Items:
Weapon: Dagger [ATK: 15, DEF: 0, SPD: 5]
Shield: Starter_Shield [ATK: 2, DEF: 10, SPD: 0]
Shoes: Starter_Shoe [ATK: 0, DEF: 0, SPD: 10]
Armor: Starter_Armor [ATK: 0, DEF: 10, SPD: 0]

Equipped Spells:
```

This is the status before changing the value

Testing :



□	Edit	Copy	Delete	4	Dagger	150	0	5	weapon	1
□	Edit	Copy	Delete	5	Starter_Shoe	0	0	10	shoes	0



```
Name: 1
Level: 2
HP: 100
MP: 56
ATK: 168
DEF: 31
SPD: 26
EXP: 0

Equipped Items:
Weapon: Dagger [ATK: 150, DEF: 0, SPD: 5]
Shield: Starter_Shield [ATK: 2, DEF: 10, SPD: 0]
Shoes: Starter_Shoe [ATK: 0, DEF: 0, SPD: 10]
Armor: Starter_Armor [ATK: 0, DEF: 10, SPD: 0]

Equipped Spells:
```

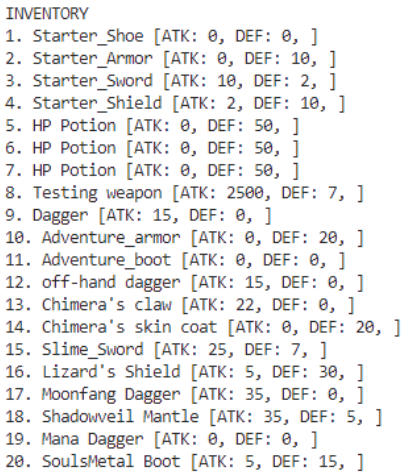
From these 3 testing result, I could show that data is retrieve in real time during gameplay

6.1 Implement sorting algorithms (e.g., bubble sort, insertion sort) for organizing player inventory.

Here I will test all the bubble sort and check is it sorting the inventory in the right way.

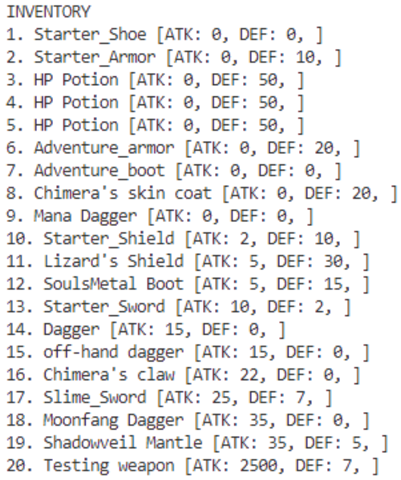
- Sort by ATK of a item
- Sort by DEF of a item
- Sort by Name (Ascending A-Z)

Original inventory's order (Testing in Stage 9) :



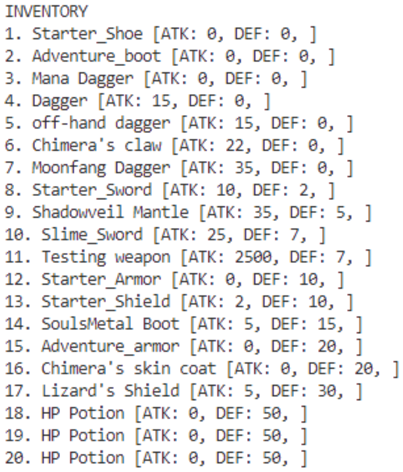
```
INVENTORY
1. Starter_Shoe [ATK: 0, DEF: 0, ]
2. Starter_Armor [ATK: 0, DEF: 10, ]
3. Starter_Sword [ATK: 10, DEF: 2, ]
4. Starter_Shield [ATK: 2, DEF: 10, ]
5. HP Potion [ATK: 0, DEF: 50, ]
6. HP Potion [ATK: 0, DEF: 50, ]
7. HP Potion [ATK: 0, DEF: 50, ]
8. Testing weapon [ATK: 2500, DEF: 7, ]
9. Dagger [ATK: 15, DEF: 0, ]
10. Adventure_armor [ATK: 0, DEF: 20, ]
11. Adventure_boot [ATK: 0, DEF: 0, ]
12. off-hand dagger [ATK: 15, DEF: 0, ]
13. Chimera's claw [ATK: 22, DEF: 0, ]
14. Chimera's skin coat [ATK: 0, DEF: 20, ]
15. Slime_Sword [ATK: 25, DEF: 7, ]
16. Lizard's Shield [ATK: 5, DEF: 30, ]
17. Moonfang Dagger [ATK: 35, DEF: 0, ]
18. Shadowveil Mantle [ATK: 35, DEF: 5, ]
19. Mana Dagger [ATK: 0, DEF: 0, ]
20. SoulsMetal Boot [ATK: 5, DEF: 15, ]
```

Sort by ATK of a item



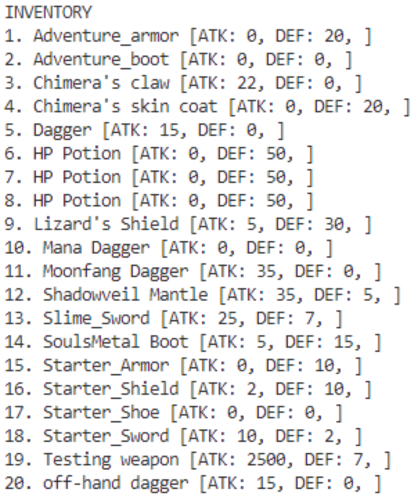
```
INVENTORY
1. Starter_Shoe [ATK: 0, DEF: 0, ]
2. Starter_Armor [ATK: 0, DEF: 10, ]
3. HP Potion [ATK: 0, DEF: 50, ]
4. HP Potion [ATK: 0, DEF: 50, ]
5. HP Potion [ATK: 0, DEF: 50, ]
6. Adventure_armor [ATK: 0, DEF: 20, ]
7. Adventure_boot [ATK: 0, DEF: 0, ]
8. Chimera's skin coat [ATK: 0, DEF: 20, ]
9. Mana Dagger [ATK: 0, DEF: 0, ]
10. Starter_Shield [ATK: 2, DEF: 10, ]
11. Lizard's Shield [ATK: 5, DEF: 30, ]
12. SoulsMetal Boot [ATK: 5, DEF: 15, ]
13. Starter_Sword [ATK: 10, DEF: 2, ]
14. Dagger [ATK: 15, DEF: 0, ]
15. off-hand dagger [ATK: 15, DEF: 0, ]
16. Chimera's claw [ATK: 22, DEF: 0, ]
17. Slime_Sword [ATK: 25, DEF: 7, ]
18. Moonfang Dagger [ATK: 35, DEF: 0, ]
19. Shadowveil Mantle [ATK: 35, DEF: 5, ]
20. Testing weapon [ATK: 2500, DEF: 7, ]
```

Sort by DEF of a item



```
INVENTORY
1. Starter_Shoe [ATK: 0, DEF: 0, ]
2. Adventure_boot [ATK: 0, DEF: 0, ]
3. Mana Dagger [ATK: 0, DEF: 0, ]
4. Dagger [ATK: 15, DEF: 0, ]
5. off-hand dagger [ATK: 15, DEF: 0, ]
6. Chimera's claw [ATK: 22, DEF: 0, ]
7. Moonfang Dagger [ATK: 35, DEF: 0, ]
8. Starter_Sword [ATK: 10, DEF: 2, ]
9. Shadowveil Mantle [ATK: 35, DEF: 5, ]
10. Slime_Sword [ATK: 25, DEF: 7, ]
11. Testing weapon [ATK: 2500, DEF: 7, ]
12. Starter_Armor [ATK: 0, DEF: 10, ]
13. Starter_Shield [ATK: 2, DEF: 10, ]
14. SoulsMetal Boot [ATK: 5, DEF: 15, ]
15. Adventure_armor [ATK: 0, DEF: 20, ]
16. Chimera's skin coat [ATK: 0, DEF: 20, ]
17. Lizard's Shield [ATK: 5, DEF: 30, ]
18. HP Potion [ATK: 0, DEF: 50, ]
19. HP Potion [ATK: 0, DEF: 50, ]
20. HP Potion [ATK: 0, DEF: 50, ]
```

Sort by name



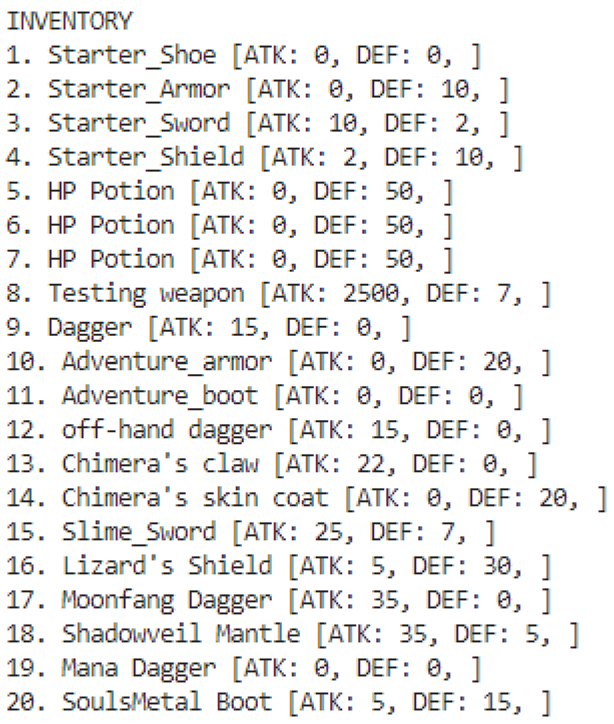
```
INVENTORY
1. Adventure_armor [ATK: 0, DEF: 20, ]
2. Adventure_boot [ATK: 0, DEF: 0, ]
3. Chimera's claw [ATK: 22, DEF: 0, ]
4. Chimera's skin coat [ATK: 0, DEF: 20, ]
5. Dagger [ATK: 15, DEF: 0, ]
6. HP Potion [ATK: 0, DEF: 50, ]
7. HP Potion [ATK: 0, DEF: 50, ]
8. HP Potion [ATK: 0, DEF: 50, ]
9. Lizard's Shield [ATK: 5, DEF: 30, ]
10. Mana Dagger [ATK: 0, DEF: 0, ]
11. Moonfang Dagger [ATK: 35, DEF: 0, ]
12. Shadowveil Mantle [ATK: 35, DEF: 5, ]
13. Slime_Sword [ATK: 25, DEF: 7, ]
14. SoulsMetal Boot [ATK: 5, DEF: 15, ]
15. Starter_Armor [ATK: 0, DEF: 10, ]
16. Starter_Shield [ATK: 2, DEF: 10, ]
17. Starter_Shoe [ATK: 0, DEF: 0, ]
18. Starter_Sword [ATK: 10, DEF: 2, ]
19. Testing weapon [ATK: 2500, DEF: 7, ]
20. off-hand dagger [ATK: 15, DEF: 0, ]
```

From the 3 sorting result, I could say that the bubble sort is working as I expected.

6.2 Use binary search for quick lookup of player items and enemy data during battles.

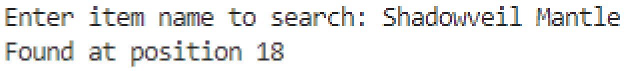
Here I will test the binary search with 3 different input normal, extreme and erroneous input

(Also the testing will be test in stage 9)



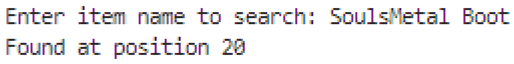
```
INVENTORY
1. Starter_Shoe [ATK: 0, DEF: 0, ]
2. Starter_Armor [ATK: 0, DEF: 10, ]
3. Starter_Sword [ATK: 10, DEF: 2, ]
4. Starter_Shield [ATK: 2, DEF: 10, ]
5. HP Potion [ATK: 0, DEF: 50, ]
6. HP Potion [ATK: 0, DEF: 50, ]
7. HP Potion [ATK: 0, DEF: 50, ]
8. Testing weapon [ATK: 2500, DEF: 7, ]
9. Dagger [ATK: 15, DEF: 0, ]
10. Adventure_armor [ATK: 0, DEF: 20, ]
11. Adventure_boot [ATK: 0, DEF: 0, ]
12. off-hand dagger [ATK: 15, DEF: 0, ]
13. Chimera's claw [ATK: 22, DEF: 0, ]
14. Chimera's skin coat [ATK: 0, DEF: 20, ]
15. Slime_Sword [ATK: 25, DEF: 7, ]
16. Lizard's Shield [ATK: 5, DEF: 30, ]
17. Moonfang Dagger [ATK: 35, DEF: 0, ]
18. Shadowveil Mantle [ATK: 35, DEF: 5, ]
19. Mana Dagger [ATK: 0, DEF: 0, ]
20. SoulsMetal Boot [ATK: 5, DEF: 15, ]
```

Normal input (where the target is present in the list.)



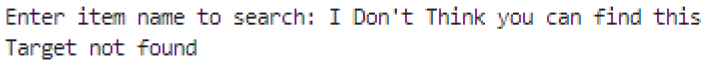
```
Enter item name to search: Shadowveil Mantle
Found at position 18
```

Extreme input (where the target is first, last in the list.)



```
Enter item name to search: SoulsMetal Boot
Found at position 20
```

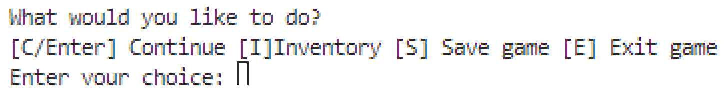
Erroneous input (where the target is invalid or unexpected data that a program is not designed to handle)



```
Enter item name to search: I Don't Think you can find this
Target not found
```

7.1 Allow players to save their progress after completing a stage.

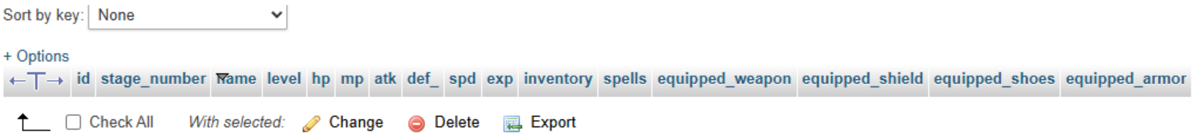
For the player finishing each stage, the program should shows up some option with one option that could let the player to save their game progress data into the database. Also need to test that is the data saved into the database correctly.



```
What would you like to do?
[C/Enter] Continue [I]Inventory [S] Save game [E] Exit game
Enter your choice: 
```

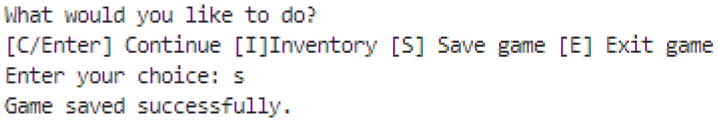
The program has display out the option for letting the user to save game.

Before Saving progress, the database is looking like this

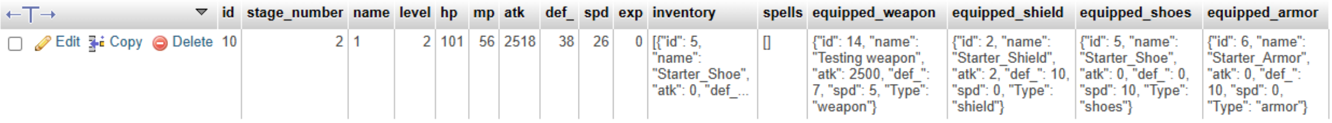


id	stage_number	name	level	hp	mp	atk	def_	spd	exp	inventory	spells	equipped_weapon	equipped_shield	equipped_shoes	equipped_armor

After Saving progress



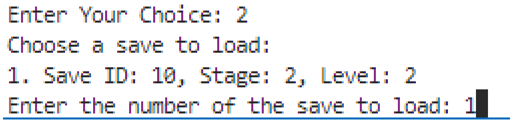
```
What would you like to do?
[C/Enter] Continue [I]Inventory [S] Save game [E] Exit game
Enter your choice: s
Game saved successfully.
```



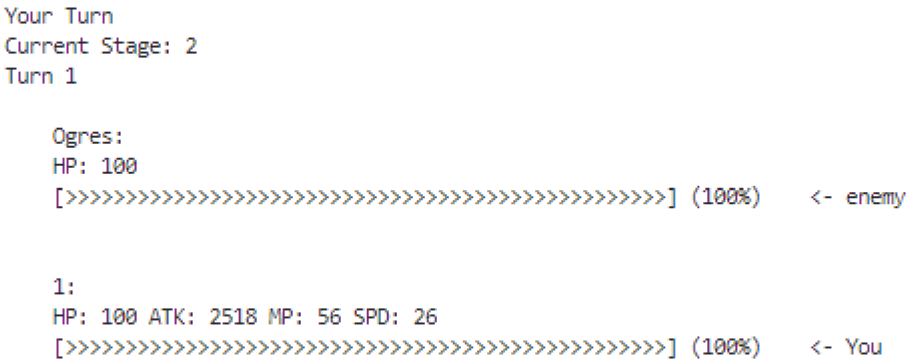
	id	stage_number	name	level	hp	mp	atk	def_	spd	exp	inventory	spells	equipped_weapon	equipped_shield	equipped_shoes	equipped_armor
□ Edit Copy Delete	10	2	1	2	101	56	2518	38	26	0	[{"id": 5, "name": "Starter_Shoe", "atk": 0, "def_...	[]	{"id": 14, "name": "Testing weapon", "atk": 2500, "def_": 7, "spd": 5, "Type": "weapon"}	{"id": 2, "name": "Starter_Shield", "atk": 2, "def_": 10, "spd": 0, "Type": "shield"}	{"id": 5, "name": "Starter_Shoe", "atk": 0, "def_": 0, "spd": 10, "Type": "shoes"}	{"id": 6, "name": "Starter_Armor", "atk": 0, "def_": 10, "spd": 0, "Type": "armor"}

7.2 Load saved game data to continue from the last saved point.

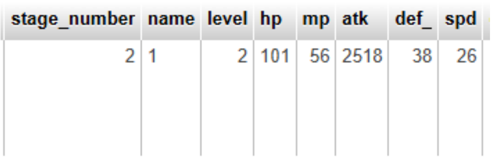
For the player in the main menu will have a option for the player to load their saved progress, so here will need to test that could the database saved progress be fetched and load correctly.



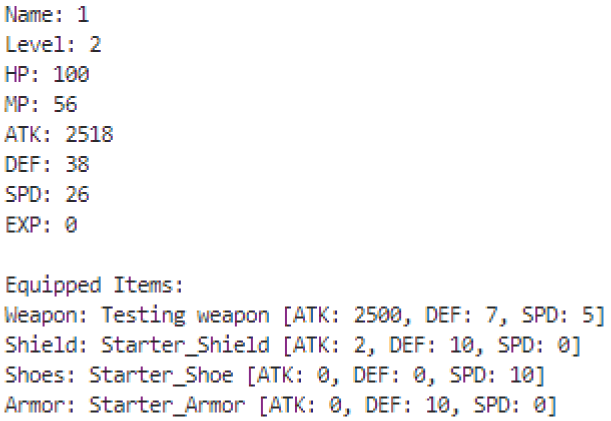
```
Enter Your Choice: 2
Choose a save to load:
1. Save ID: 10, Stage: 2, Level: 2
Enter the number of the save to load: 1
```



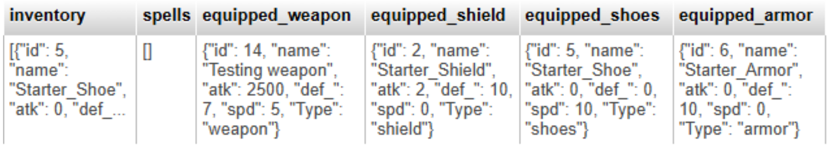
```
Your Turn
Current Stage: 2
Turn 1

    Ogres:
    HP: 100
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (100%)    <- enemy


    1:
    HP: 100 ATK: 2518 MP: 56 SPD: 26
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] (100%)    <- You
```



stage_number	name	level	hp	mp	atk	def_	spd
2	1	2	101	56	2518	38	26



```
Name: 1
Level: 2
HP: 100
MP: 56
ATK: 2518
DEF: 38
SPD: 26
EXP: 0

Equipped Items:
Weapon: Testing weapon [ATK: 2500, DEF: 7, SPD: 5]
Shield: Starter_Shield [ATK: 2, DEF: 10, SPD: 0]
Shoes: Starter_Shoe [ATK: 0, DEF: 0, SPD: 10]
Armor: Starter_Armor [ATK: 0, DEF: 10, SPD: 0]
```

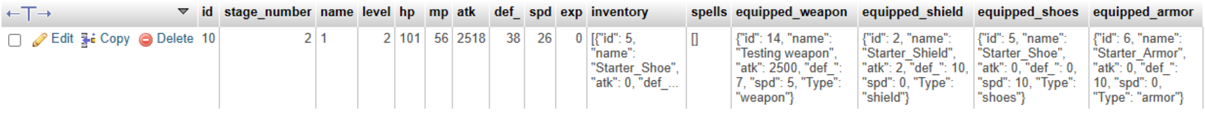


inventory	spells	equipped_weapon	equipped_shield	equipped_shoes	equipped_armor
[{"id": 5, "name": "Starter_Shoe", "atk": 0, "def_...	[]	{"id": 14, "name": "Testing weapon", "atk": 2500, "def_": 7, "spd": 5, "Type": "weapon"}	{"id": 2, "name": "Starter_Shield", "atk": 2, "def_": 10, "spd": 0, "Type": "shield"}	{"id": 5, "name": "Starter_Shoe", "atk": 0, "def_": 0, "spd": 10, "Type": "shoes"}	{"id": 6, "name": "Starter_Armor", "atk": 0, "def_": 10, "spd": 0, "Type": "armor"}

From the 2 evidence I could see that the loaded progress data has matched the progress that is saved in **7.1**

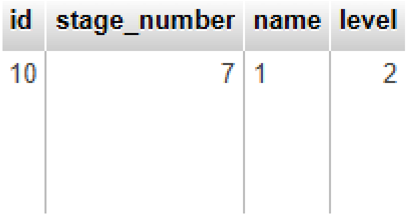
7.3 Handle saved files securely to prevent data loss.

For this part I will tested that is the saved game progress will be the same when it's loaded, also I will try to manually edit the saved game file to see could the program handle it correctly.



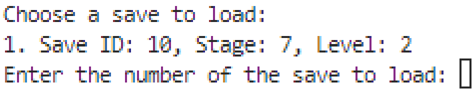
←T→	id	stage_number	name	level	hp	mp	atk	def_	spd	exp	inventory	spells	equipped_weapon	equipped_shield	equipped_shoes	equipped_armor
□ Edit Copy Delete	10	2	1	2	101	56	2518	38	26	0	[{"id": 5, "name": "Starter_Shoe", "atk": 0, "def_...	[]	{"id": 14, "name": "Testing weapon", "atk": 2500, "def_": 7, "spd": 5, "Type": "weapon"}	{"id": 2, "name": "Starter_Shield", "atk": 2, "def_": 10, "spd": 0, "Type": "shield"}	{"id": 5, "name": "Starter_Shoe", "atk": 0, "def_": 0, "spd": 10, "Type": "shoes"}	{"id": 6, "name": "Starter_Armor", "atk": 0, "def_": 10, "spd": 0, "Type": "armor"}

I will change the stage_number in this progress data from stage 2 to stage 7

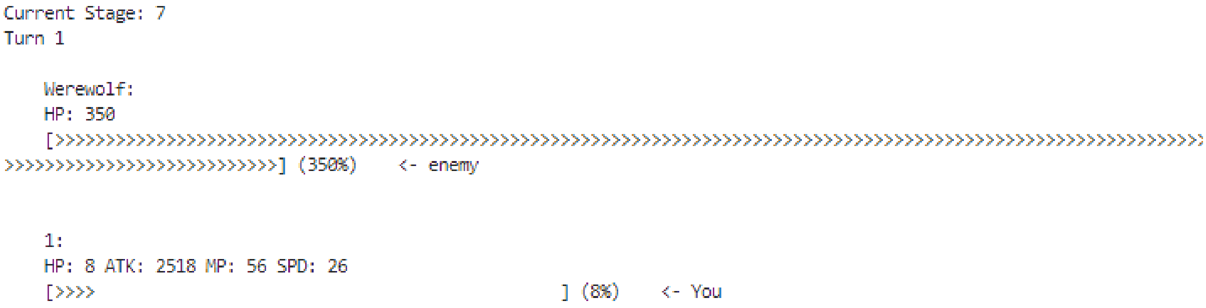


id	stage_number	name	level
10	7	1	2

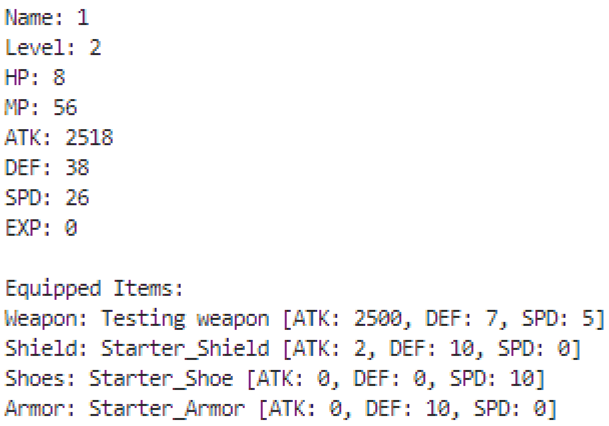
The table view after changing data from stage_number



```
Choose a save to load:
1. Save ID: 10, Stage: 7, Level: 2
Enter the number of the save to load: 
```



```
Current Stage: 7
Turn 1

    Werewolf:
    HP: 350
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>] (350%)    <- enemy


    1:
    HP: 8 ATK: 2518 MP: 56 SPD: 26
    [>>>>                                              ] (8%)    <- You
```



```
Name: 1
Level: 2
HP: 8
MP: 56
ATK: 2518
DEF: 38
SPD: 26
EXP: 0

Equipped Items:
Weapon: Testing weapon [ATK: 2500, DEF: 7, SPD: 5]
Shield: Starter_Shield [ATK: 2, DEF: 10, SPD: 0]
Shoes: Starter_Shoe [ATK: 0, DEF: 0, SPD: 10]
Armor: Starter_Armor [ATK: 0, DEF: 10, SPD: 0]
```

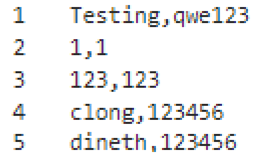
From the 3 evidence we just can see that only the stage_number has been edit, other data is all same with the progress data before editing.

And the progress could be loaded normally without any error.

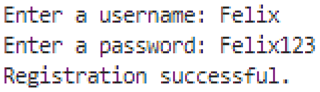
8.1.1 Players can create a new account by entering a unique username and password.

Here I will try to create 3 new account and check is the account detail has been stored into the txt file that store these details.

The txt file before doing this testing :

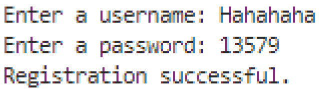


```
Testing,qwe123
1,1
123,123
clong,123456
dineth,123456
```



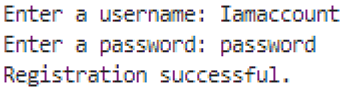
```
Enter a username: Felix
Enter a password: Felix123
Registration successful.
```

Created account 1



```
Enter a username: Hahahaha
Enter a password: 13579
Registration successful.
```

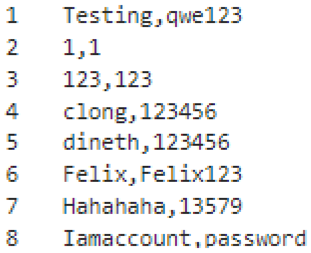
Created account 2



```
Enter a username: Iamaccount
Enter a password: password
Registration successful.
```

Created account 3

The txt file after doing this testing :

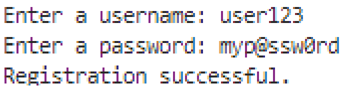


```
Testing,qwe123
1,1
123,123
clong,123456
dineth,123456
Felix,Felix123
Hahahaha,13579
Iamaccount,password
```

From this 3 testing, we could see that the registration part of the program is working fine, the entered username and password has been saved into the txt file.

Also I will test the with different input.

Normal Input

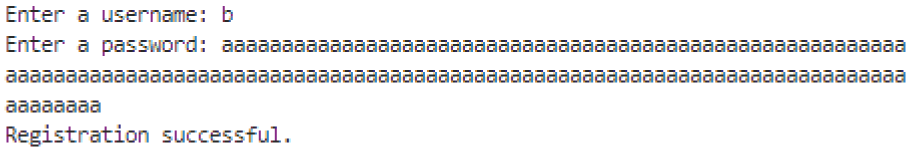


```
Enter a username: user123
Enter a password: myp@ssw0rd
Registration successful.
```

Testing the function with standard alphanumeric username and password with special characters

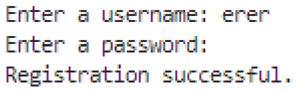
Extreme input

Testing the function with Minimal username length and Very long password



```
Enter a username: b
Enter a password: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaa
Registration successful.
```

Erroneous input



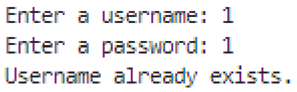
```
Enter a username: erer
Enter a password:
Registration successful.
```

Testing the function with no password

From these 3 input test this could show that the program could handle most of the situations without breaking.

8.1.2 The system will check that the username does not already exist in the file and save the username and password securely.

Here I will try to create a new account with a repeated username to check will the system let me pass or will it reject me and need me to create a new account.



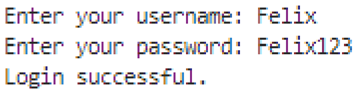
```
Enter a username: 1
Enter a password: 1
Username already exists.
```

The program could handle the situation that the user enter already exists username perfectly

8.2.1 Players will log in using their username and password.

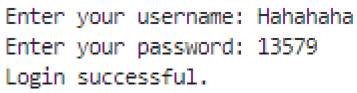
8.2.2 The system will verify the entered credentials by checking the file, and if they match, it will load the player's saved game data.

Here I will test these 2 point together, I will test that could I log in back to the 3 account that I have created in **8.1.1**



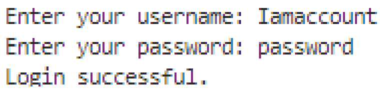
```
Enter your username: Felix
Enter your password: Felix123
Login successful.
```

The user "Felix" could log back in successfully



```
Enter your username: Hahahaha
Enter your password: 13579
Login successful.
```

The user "Hahahaha" could log back in successfully



```
Enter your username: Iamaccount
Enter your password: password
Login successful.
```

The user "Iamaccount" could log back in successfully

From this 3 testing, we could see that the login part of the program is working fine, the entered username and password could load back the account from the txt file.

Evaluation

Fitness for Purpose

Requirement Specification Matching

The development of the text based RPG game has meet all the requirements that is listed at the analysis requirement specification section of the project. Each of the key features is implemented successfully, ensuring that the features are functioning as I expected, below is an evaluation that I ensure the solution is meeting the functional requirements well.

1. **User Interface**
   - The games has a clean and simple interface that allow the player to navigate the games screen easily.
   - All the option in menu that is talked about at design section, I have implemented them and they are functioning well

2. **Player Management**
   - The system has successfully maintains and updates of the player status, like player's hp, attack, defense etc.
   - The inventory management has been well implemented, the player could use, equip and organize the inventory effective.

3. **Stage and Enemy Management**
   - The difficulty scaling progression system is well implemented, this ensure that the enemy increase in difficulty as the player being stronger
   - The database has successfully stored and retrieved the enemy data, and could ensure that the preset stats for the stages is loaded in the correct value

4. **Combat system**
   - The turn-based battle system works as I expected. The turn order is determined by the speed stats of the player and enemy
   - The player could attack, use skills or apply items, all the function that's about combat has been implement correctly and working well
   - The system will updated the player stats, give the reward item to the player and display the stage result to the player.

5. **Database and data management**
   - The database system has been successfully implemented that could store the players profiles, enemy data and all other item/spells data.
   - Data has been retrieve in real time, this ensures seamless gameplay

6. **Sorting and Searching**
   - Sorting algorithms has been implemented correctly that it could sort the inventory by the criteria that I have set (Like sort by ATK, DEF or Name)
   - The binary search function is functioning as I want this allow the player to do some quick lookups of items

7. **Saving and Loading**
   - The save function has been correctly implemented it could correctly store the progress to the save progress table in the database.
   - The load function has been correctly implemented, the system will let the player to choose which save file they want to load and could retrieve the data accurately
   - The system could prevent data corruption

8. **Login & registration system**
   - The registration system has been implemented correctly that it could store the player account data into the txt file correctly also the login system has been implemented correctly, also could retrieve the detail in the txt file and load the correct account.
   - The system could handle will situation like very long username/ password or duplicate username while the player registering.

Testing result

A comprehensive test plan was executed, covering all functional and end-user requirements. The results confirm that:

- The game interface has implemented correctly and it’s simple and user-friendly
- Also the combat system has being operates correctly, maintaining consistency
- Stage progression works as expected, the difficult scaling is working as I designed giving the player a challenging but playable experience
- Also The inventory system supports sorting, searching, and proper item usage
- The game correctly saves and loads progress without data corruption
- User authentication is robust, preventing unauthorized access and duplicate accounts

Future Maintainability and Robustness

1. **Code Structure and Readability**
   - The program has uses a organized structure, using OOP to organize class and method to store data in it.
   - Modular coding practices has been applied to the program, making that future editing will be easier.
   - Well use of recognizable-named functions and variables to improve the readability and maintainability of the code.

2. **Scalability**
   - The game could be expand be just adding stage with designing new enemies, item, skills or even other mechanics.
   - The database is designed for future expansion, it’s easy to add new item to the tables in the database.

3. **Error Handling and Robustness**
   - The game has implemented input validations that could prevent crashes due to unexpected user input.
   - The save/load system could prevent data corruption
   - Error messages provide useful feedback, making that fixing the problem be easier.

4. **Potential Future Enhancements**
   - Implementing a graphical interface to enhance user experience
   - Introducing move set to the enemy with a smarter AI that could make the game more interesting and difficult.
   - Introduce story or event to the gameplay to increase the play time.

Conclusion

The developed solution successfully meets all the functional and end-user requirements outlined in the specification. The system is well-structured, scalable, and robust, ensuring future maintainability. The testing phase confirms that the game operates correctly, with all features functioning as expected. The current version meets the objectives set for the project.