**ADVENTURE GAME**

**Contribution of Eman Butt (SP23-BAI-014)**

* Designed the complete **game layout**, including detailed dictionaries for all rooms.
* Developed **7 unique and engaging puzzles**, each with a different solving mechanism.
* Created the **“Solve Puzzle”** function to allow players to interact with and solve puzzles.
* Managed **player movement** with error handling to ensure smooth navigation.
* Implemented the **“Examine”** function, allowing players to inspect specific items.
* Introduced **two interactive NPCs (Bodyless and Dragon)** to enhance player experience and immersion.
* Designed the **“Look”** function to assist players in exploring their surroundings.
* Constructed the **game logic**, ensuring seamless gameplay flow.
* Sketched and integrated the **castle map** to guide players through the game.
* Designed **menu options**, including **character customization, backstory**, and **instructions** for a personalized experience.
* Contributed to the **save/load game** features for continuous gameplay.
* Developed functionality to load the **default game state** when a session ends.
* Conducted extensive **testing, debugging, and error handling** to ensure a smooth and bug-free experience.
* Added **multiple prompts and messages** throughout to keep the gameplay interactive and engaging.

**Contribution of Ayesha Tahir (SP23-BAI-019)**

1. **Inventory System**:
   * I made a simple way for players to check what they’re carrying. If the inventory is empty, the game tells the player there's nothing in it. But if there are items, it lists everything they have collected so far.
2. **Unlocking Rooms with Keys**:
   * I created a system where players can use keys from their inventory to unlock certain doors. It checks if the player has the right key and if it matches the room’s lock.
   * If it’s the correct key, the room unlocks, and the player gets a message saying they successfully opened the door. If the room is already unlocked, the player is told to move ahead.
   * This makes the game feel more interactive since players have to find and use the correct keys in the right places.
3. **Using Items in Different Rooms**:
   * I also added functionality for players to use items in specific rooms. The game checks if the item can be used in the current location. If it’s not the right place, the player gets a hint to try using the item somewhere else.
   * This adds a bit of challenge, making players think about where certain items might be useful.
4. **Picking Up Items**:
   * Players can now pick up items they find in rooms and add them to their inventory. The game checks whether the item is already in their inventory or if it can even be picked up.
   * Once an item is picked up, it’s removed from the room to prevent the player from collecting duplicates.
5. **Dropping Items**:
   * I’ve also made it possible for players to drop items from their inventory back into the current room. This helps them manage what they’re carrying .

**Contribution of Syeda Laiba Shahid (SP23-BAI-052)**

**save\_game():** This function is responsible for saving the current progress of the game.

It stores the player's current room and inventory into a file named 'savefile.txt'

and also saves the game's overall state (variables, settings, etc.) into a separate file

called 'gameState.json'. Once the data is saved, it prints a message confirming that the

game has been saved successfully. This allows players to resume from the same point in

future sessions.

**load\_game():** This function loads the saved game data from the previous session.

It reads the 'savefile.txt' to retrieve the player's room and inventory, and reads

'gameState.json' to restore the overall game state. If a saved file exists, it successfully

restores the game to where the player left off. If no saved game is found, it notifies the

player and starts a new game by calling the gameplay() function.

**defaultState():** This function saves the game's default state, or the initial setup, into a

file called 'defaultState.json'. This default state includes the starting values of various

game variables and settings. By saving this data, the game can later be reset to its original

condition if needed, providing a way to start fresh without losing the default configuration.

**loadDefaultGame():** This function restores the game to its original state by loading data

from the 'defaultState.json' file. This allows the player to reset the game to the default

conditions, as though starting a new game from the beginning, preserving the initial setup

of the game.