Moonlit Marrakech Hotel

I - Description of the game

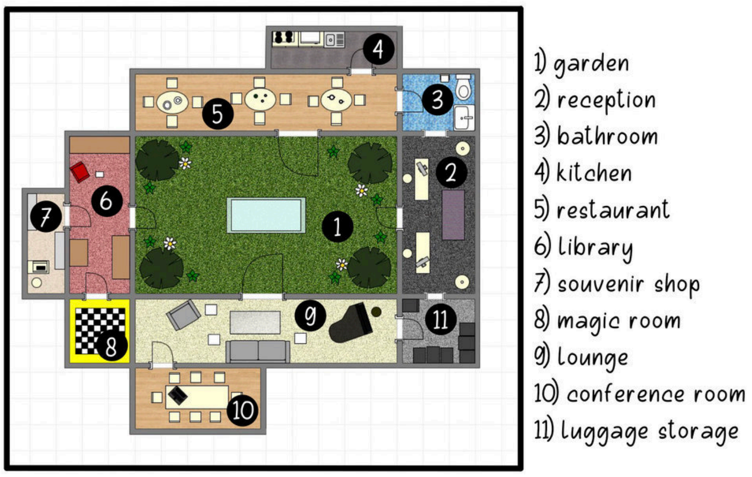
• User level description :

The ’Moonlit Marrakech Hotel’ is a text-based game, inspired by the framework of ‘The Word of Zuul Adventure’. As it operates through player-entered commands, the game aims to rescue characters trapped in chaos following a violent earthquake that has just struck Morocco. The player ‘sis to find specific items these characters need to escape and then seek refuge in the garden. The game map comprises common types of rooms that you can find in a hotel. In this game, there are 11 rooms and a total of 7 items spread among all rooms. Some of them are collectible, while others are not, presenting a challenge as the player is constrained by a limited inventory weight of 3030 grams.

• Implementation description :

In this game there are 8 classes. First, there are Command, CommandWords and Parser, these classes aim to read input typed by the player in the terminal and interpret it as a command. Secondly, there are the classes Item, Character and Room, which handle the game’s structure. Thirdly, there is the Player class which holds all the characteristics of the player such as ; its inventory, a boolean canTakeItem to know whether or not the player can hold an item, a boolean isCarrying to know whether or not the player is currently holding an item and to methods that allow to drop or give an item. Finally, the main class of the game, the Game class, which creates and initializes all the components of the game structure, allows the player to start the game with the play method and executes all commands received.

Here is the hotel floor plan, that I created myself using the Room Arranger application :



II - Base tasks completed

• Base task 1 ’The game has at least 6 locations/rooms’ :

I created 11 rooms in the createRooms method of the Game class with their names and exits, like in the ‘The Word of Zuul Adventure’ . I also added them in the allRooms Array List to be able to use all the rooms more easily in other methods of the game.

- Base task 2 ’There are items in some rooms. Every room can hold any number of items. Some items can be picked up by the player, others can’t’ :

I created an Item class that holds the name the items. Then, I created the initializeItems method in the Room class that adds items into rooms and gives them a name. Finally, I added a hold method command (that I implemented in the game like other commands such as quit or help) that allows the player to carry items while checking if there are equal to the names of the items that I allow the player to carry with the condition statement ‘if ‘.

- Base task 3 ’The player can carry some items with them. Every item has a weight. The player can carry items only up to a certain total weight ’ :

For this task, I created the Player class that holds the player’s inventory and the weight limit variable. I also add a weight to every items in the initializeItems method. And finally, in the hold method command I added the taken item in the inventory if the weight limit is respected, to check this condition I created the boolean method canTakeItem and then mark the item as carried with the boolean method isCarrying. I implement both methods in the Player class.

- Base task 4 ’The player can win. There has to be some situation that is recognized as the end of the game where the player is informed that they have won. Furthermore, the player has to visit at least two rooms to win’ :

I created a checkVictory method in the Game class that checks if the three booleans are true for the three items to be given to distressed (stationary) characters and if the player is in the garden. If these four conditions are met the player has won ! For the booleans related to the items that need to be given, I created them in the give command method, which changes their status to true when the item has been given to the corresponding character.

- Base task 5 ’ Implement a command back that takes you back to the last room you’ve been in. The back command should keep track of every move made, allowing the player to eventually return to it’s starting room’ :

For this task, I created an ArrayList that keeps track of all the rooms visited by the player. Then, I created a back command method that subtracts 1 from the list of visited rooms and sets the current room to this value.

- Base task 6 ’ Add at least three new commands (in addition to those that are present in the base code); The back command will not count as a new command.’ :

I chose to add the hold command, which works as explained earlier, drop, which follows the same logic as hold, allowing the player to place an item if it matches the items they are allowed to carry and if they are currently carrying that item. Finally, the give command that allows the player to give an item to a character, while adhering to specific conditions for this action – meaning a particular item can only be given to a specific character. For these methods, I had to implement additional auxiliary methods in my code to check various conditions.

III - Challenge tasks completed

- Challenge task 1 ' Add at least three characters to your game. Characters are people or animals or monsters – anything that moves, really. Characters are also in rooms (like the player and the items). Unlike items, characters can move around by themselves ' :

I created a moveCharacterRandomly method, making the character move randomly within the hotel. I make them move throughout the hotel using the getAvailableExit method that I implemented in the Room class and that returns the list of available exits, and finally using the random instruction to choose one randomly, setting the character's nextRoom to that room. I invoke this method in the go method command (back one as well), thus when the player enters 'go' to move around the hotel the player received output in the terminal indicating real-time movements of the three mobile characters.

- Challenge task 2 'Extend the parser to recognize three-word commands. You could, for example, have a command give bread dwarf to give some bread (which you are carrying) to the dwarf' :

To complete this task, I simply used the code of the 'The Word of Zuul Adventure' game and added a third field, thirdWord, where needed, particularly in the Command class.

- Challenge task 3 'Add a magic transporter room – every time you enter it you are transported to a random room in your game' :

For this final task, I added a method in the Game class, checkMagicWord, which checks if the description of the room where the player is currently located contains the word 'magic', and if the word is detected this method randomly teleports the player within the hotel. To achieve this, I used the array list of allRooms, which is a list of all the rooms in the hotel. Then, I once again use the random instruction to choose one randomly. To use this method, I implemented it in the go method command so that every time the player moves, the program checks if the player enter this magic room.

IV - Code quality considerations

- Coupling :

On the one hand, the Player class handles interactions with Item objects, managing the player's inventory, and checking weight limits. On the other hand, the Item class manages its own attributes and functionalities (such as isCarried, isDropped) related to its status, while the Player class controls the player's actions regarding items, this demonstrates good coupling between the Item and Player classes.

- Cohesion :

A good example where I considered cohesion in my program is the Room class. I organized this class into 3 main sections, each consisting of methods related to the functionalities of the rooms in my hotel. The first one deals with room descriptions with getShortDescription and getLongDescription methods. Then, everything related to room exits with setExit, getExit and getAvailableExit methods. Finally, the section that maintains functionalities related to items and characters in the context of a room, like addItems, removeItems, initializeItems and getItems methods that handle items and characters, along with the following methods : addCharacters, removeCharacters, initializeCharacters and getCharactersRoom.

- Responsibility-driven design :

I considered responsibility-driven design when I moved the checkMagicWord method, which aims to move the player to a random room in the hotel when they enter the magic room. Initially, I placed it in the Player class, but I realized it would be challenging to access all the rooms to construct this method. So, I ultimately placed it in the Game class, allowing easier access to the data needed for this task.

- Maintainability :

I considered maintainability when I modified how I moved the mobile characters. Initially, I had created three different methods for the three mobile characters in my game. Each of them moved each character individually using the random instruction to assign it a random room. However, I changed this by creating a single method that generates a random piece for any character. Then, I simply called this method for my three characters, making this functionality easier to extend. This means that if I want to add a new mobile character, I just need to initialize it, create it in the Game class, and call the MoveCharacterRandomly method on it.

V - Walk-through of the game

There are multiple ways to win the game, depending on the item chosen to carry first or the character to whom you want to give the required item first. Here is one way to complete the game :

1) go west
2) go south
3) go west
4) go west
5) hold glasses
6) go east
7) give glasses person
8) go east
9) go south
10) go east
11) hold cane
12) go west
13) give cane oldman
14) go south
15) hold baby
16) go north
17) go north
18) go north
19) give baby mother
20) go south