Ishika Arora - K21038125 - "Mission Diffuse" - PPA Assignment 2

Game Description

The name of my text-based game is "Mission Diffuse". The player of this game is a bomb disposal specialist who has been sent by their department leader to a hotel where there is a bomber hiding on the rooftop terrace. To win the game, the player must find the bomber and diffuse the bomb. However, the terrace can only be accessed with a key and upon arrival the receptionist can't find the key at reception. The player is told by the receptionist that the manager has a duplicate key to the terrace. The player must then search the hotel for the manager. When the manager is found, he informs the player that he has left the duplicate key in the pool room, but he also has a large cut on his leg. The player can help the manager by finding and giving him a bandage before continuing the search for the terrace key. After helping the manager, the player must secure the key, unlock the terrace room, and win the game by diffusing the bomb.

Game Function

The game is structured based on a command system; a user will input certain commands which are then processed by the game allowing for an action to be carried out as a reaction to the input.

Base Tasks

"The game has at least 6 locations/rooms."
Given the fact that, 5 rooms had already been created in the createRooms method, I created another instance of the Room class and changed the existing ones' names and descriptions to match the context of my game. I also set each room's exits accordingly. The rooms in my game are: outside, reception, spa, pool, restaurant and terrace.

"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 started by creating a new class called Item. Within this class, I made fields for the item's name (String), description (String), weight (int) and whether the item can be picked (Boolean).

I invoked a constructor for items in the Game class which creates items related to the context of my game.

Since the items must be placed in rooms, I created a new method inside the Room class that would enable me to add items to a room (object of the room class). This method takes an instance of the Item class as a parameter. Since the task asked for there to be items in 'some' rooms, I only added items to the spa, pool and restaurant rooms. I also ensured that I included a mixture of items, some that can be picked up and some that can't.

Then in the Game class, I invoked the above method to add items to different rooms.

Within the Item class, I created a method called getItemInfo which returns all the information about the item. Then in the Room class, I created a method called getItemsInfo which returns a list of items in a room and all the information about them. This method contains a for each loop that iterates through the ArrayList of items in a room. If the room doesn't have any items, it returns an empty string. Next, I modified the getLongDescription method in the Room class so that upon entry into each room, the list of items and all the information about them is printed on the game console so the user of the game can see which items they have the option to pick from in each room and whether the item is pickable.

"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 decided it would be best to create a Player class to store the weight of the items a player is carrying. Firstly, I made fields for the player's name (String) and the maximum amount of weight that a player can carry (int). Secondly, I implemented an ArrayList to store the list of items that a player is carrying.

Then I created a method called calculateTotalItemWeight that returns the total weight of all the items the player currently holds. This method uses a for each loop to calculate the total weight.

Next, I created a pick method which enables the player to pick an item (add it list to the list of items) depending upon whether the total weight of the items that the player is carrying has exceeded the maximum weight that a player can carry (initialised in the constructor). If the total weight of all the items has exceeded the maximum amount of weight that a player can carry, then an error message is displayed, and the item won't be added to the list of items.

I also created a drop method which allows the player to drop an item (remove it from the list of items). This method is a mutator method which uses the remove method from the ArrayList class.

Finally, I created an inventory method which displays which items a player is carrying and the total weight of all the items being carried on the game console using the print function. This method can help the player keep track of the total weight of all the items they are carrying.

"The player can win. There must be some situation that is recognised 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."

To win the game, the player must diffuse the bomb. Once the player is on the terrace with the bomb and the user of the game types "diffuse" into the terminal, the diffuse command is executed and a line congratulating the player on winning the game is printed on the game console. Since the terrace room (the room where the bomber and bomb are in) cannot be accessed without a key, to win the game, the player must visit both the room containing the key (pool room) and the terrace room to win.

The goRoom method in the Game class ensures that the player must visit the room with the key before being able to enter the terrace room and win the game. Within this method, there is an if statement that will return an error message if the key item is not in the ArrayList of items when trying to go up into the terrace thus forcing the player to visit at least two rooms before they can win.

"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 its starting room."

To implement the back command, I decided to use a Stack to keep track of every room that the player visits. I created a method called setCurrentRoom in the Player class which invokes the push method to store the order of the rooms visited by the player. Then in the Game class I created a method called back which enables the player to go to the previous room they were in via the back command. This method uses the Stack from the player instance and each time the back command is used the pop method of the Stack class is being invoked. If the back command is used repeatedly, the player will eventually return to its starting room.

"Add at least four new commands (in addition to those that are present in the base code)."

In addition to the commands present in the base code, I added the following six commands: back, pick, drop, inventory, diffuse and talk. Before implementing these commands, I defined them all in validCommands in the CommandWords class.

The functionality and the implementation of the back command is explained above.

The pick command enables the player to add items to the inventory array. There is an explanation of the functionality and implementation of the pick method in base task three.

The drop command enables the player to remove items from the inventory array. There is an explanation of the functionality and implementation of the pick method in base task three.

The inventory command prints how many items the player is carrying, the total weight of all the items and the list of all the items' name and weight. When this command is used, the inventory method in the Game class calls the printInventory method in the Player class allowing the above to be displayed on the game console.

Diffuse is the command that will allow the player to diffuse the bomb. This command has a condition on it which means it can only be called when the player is inside the terrace room of the game with the suicide bomber. I implemented this feature of the command in the diffuse method in the Game class. The diffuse method will return true if the player is on the terrace resulting in a message congratulating the player on winning the game being printed and the quit method will also be invoked (processCommand method will invoke quit). However, the diffuse method will return false if the player is not on the terrace resulting in an error message being printed and the game not being quit.

The talk command works in conjunction with the challenge task (which requires at least three characters to be added) since it allows the player to interact with characters. When the talk command is used, dialogue of the Character instance that is in the same room as the player is returned and displayed on the game console.

<u>Challenge Tasks</u>

"Add at least three characters to your game. Characters are also in rooms (like the player and the items). Unlike items, characters can move around by themselves."

I started this task by creating an entirely new Characters class. I made fields for the characters' name (String), dialogue (Stack) and whether the character can move randomly between rooms (Boolean).

Then I invoked a constructor for characters in the Game class which creates characters related to the context of my game. I created four characters: receptionist, manager, bomber and cat.

Since the characters must be placed in rooms, I created a new method inside the Room class that would enable me to add characters to the room (object of the room class). This method takes an instance of the Character class as a parameter.

Then in the Game class, I invoked the above method to add characters to different rooms.

In the Game class I implemented a method called randomCharacterMove which takes a character as a parameter. This method uses a random method from the java.util Random class to randomly choose a room for the character to be moved to from reception any time a command is inputted by the user of the game.

For my game to make sense, I confined the receptionist to the reception room, the manager to the spa room and the bomber to the terrace room. The only character that can move randomly is the cat.

"Extend the parser to recognise three-word commands. You could, for example, have a command give bread dwarf to give some bread (which you are carrying) to the dwarf."

Firstly, I added an example of a three-word command in the validCommands in the CommandWords class so that a player can understand how to correctly use a three-word command.

Then, I introduced a thirdWord field in the command class to store the third word of a command.

Next, in the Parser class I modified the getCommand method to handle three-word commands. If the command is made up of three words, it will return a command object with three words.

Finally, in the Game class, I modified the drop method to take three-word commands. The first word is expected to be the command word "drop", the second word is expected to be a character and the third word is expected to be an item e.g., drop manager bandage. It will also print that the item has been dropped to a character on the game console.

"Add a magic transporter room."
First, I created a transporter room in Game class and assigned its exits to every room except the terrace (as a key is required to enter this room). Then I implemented a getRandomRoom method in the Game class. This method uses the random method from the java.util package which randomly selects and returns a room from the exits. In the goRoom method, the player is moved to the new randomly selected room.

Code Quality Considerations

Loose Coupling: Introducing the Player class rather than storing player related information in the Game and Room class is an example of assisting loose coupling between classes as we can make changes to the Player class without breaking any other class.

High Cohesion: Whilst completing base task two, instead of adding the fields itemName, itemDescription, itemWeight and pickable to the Room class, I created a separate Item class so that the room class could continue to maintain high cohesion by only representing one single, well-defined entity.

Responsibility-driven design: I followed the principle of responsibility-driven design when I decided to create a Player class to keep track of all the information about a player rather than splitting the responsibility of handing player related data (rooms, items) between the Room and Game class.

Maintainability: I considered maintainability when I decided to create the Character class, Item class and Player class as if a maintenance programmer wanted to change the characteristics of characters, items or players, they can easily recognise where to start reading code and implementing changes. The use of these classes enables all the classes in the game to remain highly cohesive which makes the code more maintainable.

Game Walkthrough (Read the left column then the right column)
Welcome to Mission Diffuse. You are bomb disposal specialist who has been sent by your department leader to a hotel where there is a bomber hiding on the rooftop terrace. To win the game, you must find the bomber and diffuse the bomb. However, the terrace can only be accessed with a key. You must find this key first. Beware, you may encounter challenges along the way! Tip: If there is a character in a room, type "talk" to receive a clue. Type 'help' if you need help.

```
You are outside the main entrance of the hotel.
Exits: inside
User: go inside

You are at the reception desk of the hotel.
Exits: left outside up down transporter
Characters in the room: Receptionist, Cat
Type talk to interact.
Cat has moved to up
User: talk
'The terrace key is missing from reception. Please
find the manager, there is a duplicate key with him.'
Cat has moved to down
User: go up

Please find and pick the key to enter the terrace.
Cat has moved to up
User: go left

You are in the hotel spa room.
Exits: left right transporter

Items in this room:
Name: Chair
Description: Large and heavy massage chair.
Weight: 2000
This item can not be picked up.

Name: Bandage
Description: Bandage that can save the manager's leg.
Weight: 400
This item can be picked up.

Characters in the room: Manager
Type talk to interact.
Cat has moved to left
User: talk
'Ahh my leg is bleeding so much, please give me the
bandage to save me from bleeding out! Oh and sorry,
I left the duplicate key in the pool room!'
Cat has moved to outside
User: pick bandage
Bandage has been picked up.
Cat has moved to up
User: drop manager bandage
Bandage has been dropped to manager.
Cat has moved to down
User: go left

You are in the hotel swimming pool room.
Exits: right transporter

Items in this room:
Name: Key
Description: The terrace key needed to unlock
the door to the rooftop where the bomber is hiding.
Weight: 300
This item can be picked up.

Name: PoolWater
Description: All the water in the pool.
Weight: 10000
This item can not be picked up.

Name: Tile
Description: Broken tile with sharp edges.
Weight: 300
This item can not be picked up.

Cat has moved to left
User: pick key
Key has been picked up.
Cat has moved to outside
User: back

You are in the hotel spa room.
Exits: left right transporter

Items in this room:
Name: Chair
Description: Large and heavy massage chair.
Weight: 2000
This item can not be picked up.

Name: Bandage
Description: Bandage that can save the manager's leg.
Weight: 400
This item can be picked up.

Characters in the room: Manager Cat
Type talk to interact.
Cat has moved to up
User: back

You are at the reception desk of the hotel.
Exits: left outside up down transporter
Characters in the room: Receptionist, Cat
Type talk to interact.
Cat has moved to outside
User: go up

You are on the hotel's rooftop terrace.
Exits: down
Characters in the room: Bomber, Cat
Type talk to interact.
Cat has moved to down
User: talk
'I didn't want to do this! I was forced!
Please diffuse the bomb!'
Cat has moved to down
User: diffuse
Congratulations, you have successfully diffused
the bomb and won the game!
Cat has moved to left
Thank you for playing. Goodbye.
```