ASSIGNMENT 2: ZUUL RESCUE

Title: Assignment 2 – "Zuul Rescue"

Author: Alexandra-Maria Anastase

Student ID: 20022789

Due date: 02.12.2020

Zuul Rescue

Name: Zuul Rescue

Description: Zuul Rescue is a game consisting of several locations in a castle. The game's purpose is saving the prince of the World of Zuul, who is held captive, and get him outside. First, the player must gather items for two different purposes: an eventual fight with a dragon and unlocking several rooms. If the player gets defeated by the dragon, he dies and loses the game. After that, the player needs to find the prince of Zuul and get him to the courtyard in order to win the game.

Implementation Description

Game is the main class, which contains the play() and the processCommand(Command) methods. In the latter, there are four boolean variables. 'wantToQuit' will be assigned 'true' if the user chooses to quit the game, 'gameIsLost' will be assigned 'true' if the player loses the fight with the dragon (the aftermath of the 'attack' command) and 'gameIsWon' will be assigned 'true' if the player gets the prince to the courtyard (aftermath of the goRoom(Command) or transportToRandomRoom(Room, Room) methods). The 'quit' variable will be assigned 'true' if any of these variables is true and will thus end the game. This method also makes calls to UserCommands. The class UserCommands is responsible for the implementation of the commands. Some of the methods from that class make calls to the methods in class RoomTasks which is responsible for what happens in each room. The Game class creates some objects of the other classes and passes them as parameters for constructors of the classes which need access to them in order to function. It is important to note that some rooms are locked and the method of unlocking them varies.

Base Tasks:

• The game has several locations/rooms. – I created a class called Rooms which creates and stores all the rooms in the game, and a class called Room which represents one room in the game.

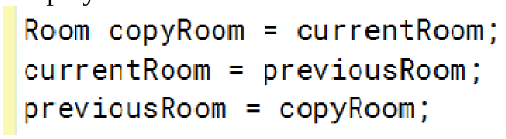
• The player can walk through the locations. – In order to let the player walk through the rooms, I used the existing code and added other exits and directions (down, up).

• 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 a class called Items which creates and stores all the items in the game and a class Item which represents an item in the game. Every object of class Room can store a HashSet of Item. One of the parameters of the Item class is 'itemAvailability', which sets the availability of the item (true if it can be picked up and false otherwise). Once an item is picked up, its availability is set to false, so the player can't pick it up again, and when it is disposed of, its availability is set to true and the item added to the HashSet of the current room. The methods from Room which do this are addItemToRoom(Item) and removeItemFromRoom(Item).

• The player can carry some items with him. Every item has a weight. The player can carry items only up to a certain total weight. – I created a class Player which stores a HashSet of Item called 'inventory' and added two new command words ('take' – for picking up the item, 'throw' – for disposing of the item). When the item is added to the inventory, the inventory weight is increased and when the item is disposed of, the weight of the inventory decreases (methods used are addToInventory(Item) and removeFromInventory(Item)). When the command 'take' is called, first it checks whether the inventory is full. If it surpasses the maximum weight, then the item will not be added to the inventory.

• The player can win. – if the player manages to get the prince from Royalty Room to the courtyard, they win. I did this by adding two methods called setTaskCompletion(boolean) and getTaskCompletion() in class Room. So, each room has a task and certain things happen only if those tasks are completed/not completed. In order to win the game, the task for the Royalty Room needs to be completed. If that task is completed and the player gets to the courtyard, they win the game. If the player gets the prince to the courtyard, the variable ‘gameIsWon’ from processCoomand(Command) in Game will be assigned true and the method will return true to the method play() and the game will end.

• Implement a command “back” that takes you back to the last room you’ve been in. – I added ‘back’ to CommandWords and then created a new method in UserCommands. The class Game passes several parameters to the constructor of UserCommands. The goBack(Command) method sets the previous room to the current room and the current room to the previous room. Thus, the current room becomes the last room the player has been in.



```
Room copyRoom = currentRoom;
currentRoom = previousRoom;
previousRoom = copyRoom;
```

If the player chooses to use this command repeatedly, they would switch back between these two rooms. This method also checks to see if the previous room was the Transporter, in which case the command will not be available. and also checks to see that the previous room is not null.

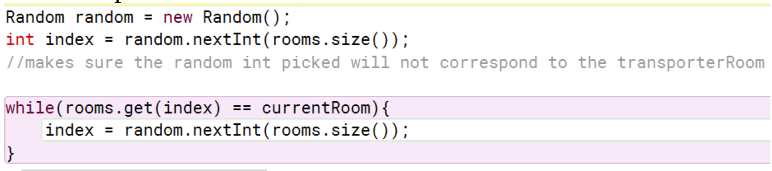
• Add at least four new commands. – The commands which I added are "take" (for putting an item into the inventory), "throw" (for removing an item from the inventory), "listInventory" (for listing the items in the inventory), "back", "answer"(to answer the sphinx’s riddle), "attack"(to attack), "use"(to use a key), "ingest"(for eating food/drinking potions).

Challenge Tasks

• Add characters to your game. – I created a new class called Character which represents a character in the game and a class Characters which creates many objects of class Character and stores them in a HashSet ‘characters’. The Room class contains a HashSet of Character characters which contains all the characters in that particular room. There are also two methods addCharacter(Character) and removeCharacter(Character) which add a character to that room and remove them, respectively. The most important method is the moveCharacter(Character, Room) method, which is used in order to move a character from the room they are in to another room. I made it so that when the player meets either the character spy or prince in a room, those characters will follow him to the other rooms. I created a class CharacterMovement which is responsible for the movement of all characters and uses the method formerly mentioned. UserCommands and RoomTasks will use the method moveCharacters(Room currentRoom, Room previousRoom) from that class whenever the player changes rooms. In order to prove that characters can indeed move, I decided that for every room, the characters will be printed (this was done by calling the listCharacters() method of the class Room).

• Extend the parser to recognize three-word commands.– In Parser, I added another String, ‘word3’, then checked to see if the user input has a third word using the tokenizer and if that is the case, word3 will assigned that word. Then. I modified the Parser so it would return a three-word command.

• Add a magic transporter room. – I created a new room called ‘transporterRoom’. I used a Random object in order to generate a random integer using the size of the ArrayList<Room> ‘rooms’ which holds all of the rooms in the game. In order to make sure that the index corresponding to the Transporter will not be the one picked, I made a while which loops as long as the index chosen is the one corresponding to the Transporter.



```
Random random = new Random();
int index = random.nextInt(rooms.size());
//makes sure the random int picked will not correspond to the transporterRoom

while(rooms.get(index) == currentRoom){
    index = random.nextInt(rooms.size());
}
```

• **My own challenges.**

1. Different types of items. – I created two types of items by creating two boolean variables in class Item, 'isWeapon' for weapons and 'isConsumable' for consumables (items which can be used only once and then discarded). In the case of 'poisonPotion', this item is a weapon, as well as a consumable.

2. Dragon fight - The Character class contains an integer instance 'healthBar' which receives a value from a parameter passed through the constructor. Then, the method getHealthBar() returns the 'healthBar' of the character, deductFromHealthBar(int) deducts a value from the health bar and addToHealthBar(int) adds a value to the 'healthBar'. The minimum a 'healthBar' can be is 0 and the maximum is 100. These two methods include a check so that the value does not surpass these limits. There is also a method setAttackPoints(int) which sets the 'attackPoints' of the character (the damage they would cause in a fight for one hit). I added the command "attack" in CommandWords ('attack (adversary) (weapon)') and then created a public method in UserCommands called doDamageToAdversary(Command). There, it checks whether the item input by the user is indeed a weapon and whether the character the player wants to attack is attackable (the 'isAttackable' instance from class Character). Every time the user attacks, the adversary's 'healthBar' will decrease by the item's 'attackPoints'. If the 'healthBar' of the adversary is less than 0, then the adversary died, and the player can continue the game. If it's greater than 0, then the adversary can "attack" the player too. I did this by creating a private method takeDamageFromAdversary(), which is called by the other, public method. If the player's 'healthBar' becomes less or equal to 0 in the process, the game ends with the loss of the player.

3. Sphinx's Riddle – One of the items I made unavailable was the 'bronzeKey' from the Bronze Chamber. When the player reaches the Sphinx Room, they need to answer a riddle in order to be able to pick up the key, which will be needed later on in the game. I created a new command "answer" and implemented a method in UserCommands, answerCommand(Command), which calls a method from RoomTasks called answerForSphinxRoom(String) by sending the second word of the command as a parameter. There, we check whether the answer is correct and if it is, we then set the availability of the 'bronzeKey' to true.

4. Locked rooms– In addition to making items available/unavailable, I decided to do the same thing to some of the rooms. In one of the cases, the player needs to have in his/her inventory all the three keys: 'bronzeKey', 'silverKey' and 'goldKey'. If this is achieved, then the dungeon will be unlocked. In the other case, the user needs to pick up a key ('diamondKey') and then utilize the command "use diamondKey" in order to unlock the next room. This is done by having a boolean instance in Room called 'isLocked' and methods for setting and getting this variable.

5. Player losing – The player loses if he gets defeated in the battle with the dragon. I created a boolean variable in the method processCommand(Command) in Game called 'gameIsLost' and initialized it to false. When the "attack" command is processed, it will return a value (true if the player lost and false otherwise). If 'gameIsLost' is true, then variable 'quit' will be assigned true and the processCommand(Command) method will return this to the 'finished' variable of the method play(), thus causing the game to end.

Code quality considerations

1. Coupling – There are no two classes which depend on each other (in the class diagram, there are no classes whose arrows point to one another) and all instance fields are private. An example of loose coupling would be the fact that the class 'Rooms' does not depend directly on the 'Item' and 'Character' classes.

2. Cohesion – An example of high cohesion from my program is the Character class, in which each method does a single, specific thing: setting the name of the character, returning the name of the character, returning the 'healthBar', adding and deducting points from it, setting the attack points and setting and getting the 'isAttackable' status.

3. Responsibility-driven design – An example of responsibility-driven design in my code is the class Player, which stores the information about the player ('inventory', 'healthBar', 'inventoryWeight', 'maximumInventoryWeight') and is responsible for processing that said information through its various methods (addToInventory(Item), removeFromInventory(Item), getInventory() etc.)

4. Maintainability – An example of maintainability in my code is the fact that the characters and items are not hardcoded. Instead, they could be easily added in later thanks to the methods in the classes Character and Item and be used with the existing commands.

Walk-through of the game

The player starts in the courtyard of the castle. They have to gather weapons and items for restoring their health. After that, they should head to the sphinx Room, answer the riddle and go to the Bronze Chamber and take the bronze key. Then, they should head to the guard room where they meet Spy (a character), go to the Silver Chamber and pick up the silver key, head to the kitchen where they can pick up a magic mushroom (item for restoring health). Next, the player needs to head to the Gold Chamber. Once the gold key is put in their inventory, the dungeon will be unlocked. In the dungeon, the player needs to fight a dragon. There are combinations of items they can choose to use. Some of them may not assure his survival in the fight. The principle of the fight is that each time the player attacks the dragon, the dragon will then fight back, taking 25 points from their health every time. If the player dies in the fight, they lose the game. (The player can get out of the room or enter it during the fight as much as they want to.) Otherwise, they gain access to the Diamond Chamber where they can pick up the diamond key and use it to unlock the Royalty Room where the prince is held captive. From there on, the prince will follow them. In order to win the game, the player needs to get the prince to the courtyard. They can either retrace their steps or use the transporter in order to teleport to a random room. The player wins when they get the prince to the courtyard.

Set of commands

One possible set of commands which can be used to win the game is : go west, take sword, go east, go south, answer man, go east, take bronzeKey, go west, go west, go south, take silverKey, go east, go east, take goldKey, go down, attack dragon sword (x 4 times), go east, take diamondKey, use diamondKey, go down, go east and depending on where the player teleports to they need to enter a few more commands to get to the courtyard. (In our case, we did not need health restoring items, because the sword was enough to defeat the dragon.)