WORLD OF ZUUL ASSIGNMENT

OMAR KASSAM

MODULE : PPA K-NUMBER: 23170974 UNIVERSITY: King's College London

28th of November 2024

• INTRODUCTION

The "Underground world" is an adventure text based game with the objective of defeating the 'evil orb' entity that has put you there. The game requires you to explore, collect items and interact with characters in order to escape and 'win'.

• FULL GAME UNDERSTANDING/WALKTHROUGH

GAME CONDITIONS

Every game starts off in a "dark pit" in which you can travel in different directions. There are items in rooms later on with the items being mentioned in the description printed. You have a health value which can increase and decrease given scenarios. To win the game you must have a sword and enter the room ladder.

ITEMS

You can equip 2 weapons (a "Bow" and a "Sword") however you can only equip one at a time you must decide which one to choose with the help of items like the "Ancient Book". If you have a sword equipped and you enter the room with the 'evil orb' you will win the game. There is also a "Healing Potion" item given to you by a character which restores you to full health. You can only carry items up to a weight of 3. Here is the list of items and their weights.

Item List: "SWORD OF VICTORS", 2 "HEALING POTION", 1 "GOLDEN BOW", 1 "ANCIENT BOOK", 1

CHARACTERS

You may also encounter 2 characters Ernie and the Dwarf.

ERNIE:

Ernie is stationed in a room named 'store' and is considered a friend who offers you the "HEALING POTION" to give you increased health.

DWARF:
The Dwarf however is considered ‘evil’ and can appear in a list of different rooms and may appear in any random one in any playthrough. As well as this he changes the room he is situated in randomly every time the player moves room as long as the player has a weapon(and so as long as the player has a win condition the dwarf will iterate into random rooms to try and stop you.)

You can also interact with the Dwarf in three ways. You can ‘fight’ him, you can ‘give’ him an item or you can shoot an arrow at him (‘shoot arrow dwarf’). He also has the ability to kill you if your health is too low and/or you type the wrong inputs.

ROOMS (WALKTHROUGH INCLUDED)

NOTE- The colours represent the different routes, winning route is given at end

1. entrance - This is the default start to the game
2. tunnel - After going “north” from entrance
3. armory - After going “north” from the tunnel, 1 item (“SWORD OF VICTORS”)
4. chest - After going “east” from armory, 1 item (“GOLDEN BOW”)
5. cupboard - After going “west” from armory
6. book - After going “down” from cupboard, 1 item (“ANCIENT BOOK”)
7. store - After going “south” from entrance, 1 item (“HEALING POTION”), 1 character (“Ernie”)
8. ladder- After going “up” from entrance
9. glow- After going “up” from ladder, LOSE health if Sword not equipped, WIN the game if sword is equipped.
10. door- After going “east” from entrance
11. fountain - After going “east” from the door, GAIN full health once entered
12. TELEPORT - After going “down” from the entrance, TELEPORTED to a random room in the game (excluding itself and the store).

WINNING ROUTE- 1, 2, 3, 2, 1, 8 Once SWORD is equipped and you return to the ladder you win.

COMMANDS

- go - GIVEN already only change is ‘down’ and ‘up’ are now directions
- quit - GIVEN
- help - GIVEN
- equip - Once typed and if items are available in this room and you have below 3 weight you will ‘equip’ an item which adds weight and is displayed in the terminal.
- unequip - Opposite of equip(removes items and reduces weight)
- fight - If SWORD is equipped and the Dwarf is present this will remove the Dwarf from characters ArrayList permanently.
- back - Will return you to the room you were in previously until you return to the start. **Note**: if you teleport the ‘start’ is considered to be the room you were teleported to.
- heal - Restores health to 100(Max).
- give - A command usable when interacting with the Dwarf results in your death if you are tricked into giving the Dwarf something.
- shoot (with arrow dwarf) - Only three letter command initiated once “shoot” is typed and if the next two words fulfill conditions (are “arrow” and “dwarf") will initiate a fight with the dwarf resulting in you winning but losing half of your health.

BASE TASKS

- Have at least 6 rooms.
  Game has 12 rooms implemented the same way original rooms were.
- 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.
  Items are stored in an array list(and so can hold any number) and are a part of the room description, implemented by changing Room class so that an object of type room has a parameter of items.
- 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.
  The player carries items after equipping the items in a given room, it is then displayed in the terminal after each room (done so via Room class printing items in its getLongDescription).

- The player can win. There has to be some situation that is recognised as the end of the game where the player is informed that he/she has won.
  Win condition for the player is if they have the item “SWORD OF VICTORS” and they enter the ladder room (in which contains the ‘evil orb’). I used a method called gameWinChecker() which contains instance field win and if above conditions are met win is set to true. This win boolean is a condition of a while loop and if win is true it will exit the loop resulting in a unique win message.

- 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.
  This command took me some time hence the long explanation. There is an ArrayList footsteps which is implemented in the goRoom method. After you type a valid direction the footsteps ArrayList adds that ‘direction’ String to its List.

  Now, in the goBack method a local boolean variable is created and set to false. It also checks if footsteps are empty in order to prevent a null pointer exception.

  Then in the main block of code the while loop (under condition error == false) creates another local variable footstep which stores the latest direction string added to footsteps by doing footsteps.getLast(). It then removes the latest direction string from the list.

  After this the loop contains a series of if statements which checks if the footstep is any of the directions(“north”, “south” e.t.c) and if it is it will REVERSE its direction so for example “up” will now be “down”.

  Finally, using the same code from goRoom you will move into the previous room using this new direction(the reverse direction hence you go into the previous room).

- Add at least four new commands (in addition to those that were present in the code you got from us).
  New commands “shoot”, “fight”, “give”, “heal”, “back”, “unequip” and “equip” all implemented using same methods given (explained more in commands section)

- CHALLENGE TASKS

- Add 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.
  Characters are parameters of Room objects(so rooms can hold any number of characters). If a character is met it is stored in the characters ArrayList which is printed in each room’s description.

  Ernie does not move(he is stuck in the room store) the Dwarf does though (explained more in the characters section).

- 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.
  The command “shoot arrow dwarf” is further explained in the commands section.

- Add a magic transporter room – every time you enter it you are transported to a random room in your game.
  Uses an if statement to check if current room description matches the teleport room then uses the same logic as the dwarf movement in which it uses the rooms ArrayList which stores a number of rooms and then uses the random library to randomise an integer. This random integer is then used in getRoom(integer) and set to currentRoom and the rest is the same implementation as goRoom.

● CODE QUALITY

COUPLING
All Classes apart from Game class are loosely coupled and so make little or no references to each other which ensure they are readable and easily understandable. However the Game class containing all the new code is quite bad(tightly coupled) as it heavily depends on other classes and so for example if I were to add a parameter to the rooms in the Room class, several errors would occur in the Game class due to the creating rooms method which does not have the same number of parameters.
COHESION
Unfortunately, the cohesion of this product is not good at all, considering all information about the player and all other tasks (characters, items ,health) are all coded and executed in the Game and Room class means the code has low cohesion. This means that are a series of unrelated tasks (or tasks better off to be completed in separate classes) all put into a single class. Another possible problem is the constant use of checker to see what room we are in which does not **directly** link to our methods intention(shouldn’t apply a checker to see what room we are in a method that wants to equip items doing two things here)

NOTE - This is unfortunately because by the time I realised there should be separate classes for Player, Items e.t.c I had already implemented most tasks and I really struggled to write code to complete these tasks with separate classes (as the way I completed the tasks before was using the fact that everything was in the same class).
RESPONSIBILITY-DRIVEN DESIGN
Designing methods for commands such as equip and unequip separately helps to ensure each method does separate things(not more than one thing at a time). Also, having most methods have several conditions in their respective if statements helps prevent the code from functioning in a way we did not intend.(for example if I am in a room with no weapons I **should not** be able to call equip and **actually equip something**).
MAINTAINABILITY
In the goBack command goRoom command and shootArrowDwarf command all include if statements to prevent crashes from occurring ,such as checking if footstep.isEmpty() which prevents null pointer exception. Also checking if commands have second or third words before implementing them prevents random inputs having an effect.