Assignment 4

Klaudia Biczysko & Justyna Sikora

April 2, 2021

1 Introduction & Data Structures

Our code consists of 6 classes: Player, Castle, Game, Room, Door, Item. We included 2 examples of inheritance: class Castle is a child of class Player and a parent to class Game. Classes Room, Door and Item are responsible for creating instances of rooms, doors, items, which are being read from the txt file. For instance, class Game uses the methods (f.ex. commands, show, quit, take, fill, drink, release, open) from class Player.

The data structures, we used, are lists and dictionaries. Rooms are stored in a dictionary, where the value is a room and the key is a name of the room. Thanks to that, we can easily access the instance of the Room class by using its name. Every room contains a linked rooms dictionary (value: room, key: direction), which is used to store possible passages. This implementation allows us to execute the method "go" - when the user chooses a direction, where he/she wants to go, the program checks if the direction exists in the linked room dictionary of the current room.

The other structure, we implemented, is a list. Every room contains a list of items available in that room. In consequence, we append instances of the class Item to the respective room while reading the configuration file. Due to storing items in lists, we cannot access them by using their names, because they are visible only as instances of class. Hence, we had to create a dictionary with all the items in the game, where the key is a name and the value is an instance. Therefore, it is not an optimal solution, because we store items twice and if game would become very large, it would require a lot of data space.

2 Added actions

- fill - this method takes an item as an attribute and checks if it can be filled (f.ex. with liquid) and in case it can, the item is filled. Here we use a try/except clause. Thanks to that, if user wants to fill an item, which he does not have in his inventory, he will get an error message "You have nothing like that". We also check if the item can be filled in the room, where the player is located.
- drink - a method that allows player to drink. In order to be able to implement this method we have added a new attribute - state, that indicates if the item that player wants to drink from is empty or full.

3 Difficulties

One of the difficulties that we stumbled upon regarded methods open/unlock. To solve our problem we decided to divide each door into passages to and from a given room and treat each passage separately. Consequently, when we read in the door "door E-W open Hall Lab" from a .txt file, two "passages" are created - one from "Hall" to "Lab" and the second from "Lab" to "Hall". We store passages in a dictionary, where the key is the name of the passage. The name of the passage consist of the name of the room

from which we can enter the second room and the name of the second room, e.g. "LabHall", "HallLab". It allowed us to easily access both "sides" of a door while executing methods unlock/open.

When it comes down to the parts of our design that we would solve differently, we could read in all data at once. Currently, we are first loading rooms and then items and doors, which is not optimal since we need to open and close the file multiple times.

In the beginning, we have implemented more list than the final code contains. During the coding, we realized it would be easier to store some data in dictionaries where the items can be accessed just by the e.g name.