5LN715
Advanced programming
for language technologists

Assignment 4 Report

Angeliki Zagoura
Oreen Yousuf
Jae Eun Hong

1. Difficulties

First, it was not an easy task to solve the problems using only class to design the games. We took long time to parse the description file into the proper data structures. Also, keeping the data structures we made was a hassle. Our biggest regret was constructing the config file, as it was built with the primary intention of high readability, but after parsing began, we found that it was difficult to segment the data into the data structures we desired. We also tried to parse the data from *yaml* file, however we believed that even though the *yaml* file was even *more* easily readable, we were more confident in parsing the text file.

2. Good design

- Mysterious Stone

Figure 2: Mysterious Stone (Thunder Stone)

Figure 1: Pikachu and Raichu

If the player takes a stone in the kitchen and has it until they reach the winning room, Pikachu evolves into Raichu. If not, the player finishes the game with Pikachu without the evolution. It gives users more rewards and excitement to play again even though they already won the game. In addition, when it evolves, we added time.sleep(2) to drag the time for the dramatic effects.

3. Data structures and complexities

We used two data structure: *lists* and *dictionaries*.

```
3]: {'Bedroom': [['east', 'Bathroom', 'open'], ['north', 'Library', 'closed']],
     'Bathroom': [['west', 'Bedroom', 'open'],
      ['north', 'Kitchen', 'open'],
      ['east', 'Living', 'closed']],
     'Library': [['south', 'Bedroom', 'closed'], ['east', 'Kitchen', 'closed']],
     'Kitchen': [['south', 'Bathroom', 'open'], ['west', 'Library', 'closed']],
     'Living': [['west', 'Bathroom', 'closed'], ['north', 'Pantry', 'locked']],
     'Pantry': [['south', 'Living', 'locked']]}
```

Figure 3 Parsed door data

First, as shown above, the outer data structure is a *dictionary*. The reason we chose it was because it has the time complexity of *O(1)* on searching. When the player goes through the new room, searching is frequently used. Therefore, we tried to find the best data structure that shows the fastest time in searching which is the *hash table* structure (*dict*-type in python).

```
[5]: ParseConfig.items

[5]: {'key': ['Library', 'USE', 'unlock'],
      'carpet': ['Bedroom', 'MOVE'],
      'statue': ['Bathroom', 'STATIONARY'],
      'stone': ['Kitchen', 'MOVE'],
      'teleporter': ['Living', 'USE', 'shake']}
```

Figure 4 Parsed item data

Secondly, we used the *list* in the value of the *dictionary*. The pros of using *list* structure are that we can find data through index. We thought it was effective because indexing into the list takes no time for all the item features, which was maximum length 3. Also, to change the status of doors when a closed door was opened later on in the code, we had to change the open/closed/locked status of the corresponding door on the other side as well. This was done fairly easy as we could take the index of the status of the door you're trying to open and find the corresponding door in the next room's list of doors. Cons that we thought of: there could be a serious bug if one of the elements in the list is accidently deleted. For example, when the action *USE* is deleted accidently, the next element *unlock* takes over the index [1]. It would cause an error as the index [1] was the spot for the action command.

4. Actions

We added the action *shake* for the transporter. It transports the player to any random room other than the winning room.

We added the action *release* to drop items that the player took. The player can release the item in any room and the item is appended into the room inventory.