Course: Advanced Programming

Assignment 4: Report

Kris Farrugia and Eva Elžbieta Sventickaitė

April 2, 2021

1 The Design

In Assignment 4 our task was to develop a text-based game[1]. An instance of class `Game` is created as a variable `game`, with the specification defined in `filename`. As a default configuration we use `config.txt`, yet a completely new specification can be created. A method `boot()` of class Game is called, which starts the game. The game constantly prompts the user for input as defined in the while loop in method `boot()`.

1.1 Changes

This design posed a major challenge during the latter phase of development, as at first our team was creating methods directly for classes `Player` and `Items`, such as `go()` or `drop()`. However, after implementing the game in a way that methods could be accessed directly, as in `player.go('N')`, the team realised that the design is not optimal when considering user input from the terminal as well as is not in accordance with the requirements for OOP-only design. Some smaller changes include using dictionaries in classes instead of lists as well as completely changing attributes of the class `Door`. The `Door` class was first implemented as a two-way door: attributes included `dir1`, `dir2`, `room_1`, `room_2`. As it would be quite challenging to allocate doors to their respective ???chambers having such class architecture, our team decided to have one-way doors instead. However, this architecture would not be suitable in a game, where one room has more than one door per direction, then vanilla version of class `Door` should be used.

1.2 Difficulties and regrets

One of the least efficient and poorly written parts of our code is `Game.__init__()`. Our team did not implement inheritance, but instead created attributes `self.house=House()` and `self.player=Player()` and then manipulated them as object instances during the game. For accessing the object of an item of a player, such as a box, it could be achieved by key-value pairs in `Player` class as an attribute to class `Game`, as in example: `self.player.items['box']`. In turn, for finding the starting location of the item, `self.player.items['box'].location` or `self.house.rooms[name_of_room_where_the_item_is].items['box']` could be accessed. The only difficulty of this implementation is readability, thus, such unnecessary variables as `house=self.house` and `player=self.player` are created in most of the methods of class `Game`.

1.3 Redemption and additional features

Firstly, the most useful (and obvious) feature of this design is attributes as dictionaries. By using lists, one would need to loop through `rooms.doors` or `room.items` and use additional functions, such as add_door or add_item. This would not pose any problems for a small scale game, such as ours, however it would become laggy as new rooms, doors and items are added to the specification. Secondly, although one would assume that using libraries such as `cmd` for getting user input is a more sophisticated approach, we have found that defining our own method

[1]The architecture of the game is illustrated in UML class diagram

`boot()` is beneficial as it is easier to adapt own code to newly introduced changes. Also, `boot()` uses f-strings to fill in the user-called function name in order to execute it. At first our idea was to create a dictionary, but it proved to be cumbersome, as some of the functions use arguments, the arguments differ, etc. Some other honourable mentions Player has an optional name and so the game can be personalised; there is an Easter Egg[2].

1.4 Thoughts on large scale and future updates

This task proved to be not only useful for acquiring a better grasp of OOP, but also very entertaining. Ideas for future updates by ascending level of difficulty: 1) create improved version `sit` with `sit_on`, which would specify the item, which the user wants to sit on, instead of randomising it; add eat and drink methods 2) generate a map of the house 3) create a storyline (a "breakout room") and add the items or functions required by it.

[2]A pun, because it's an Easter Easter Egg