COMP260 – Distributed programming **Worksheet 1**

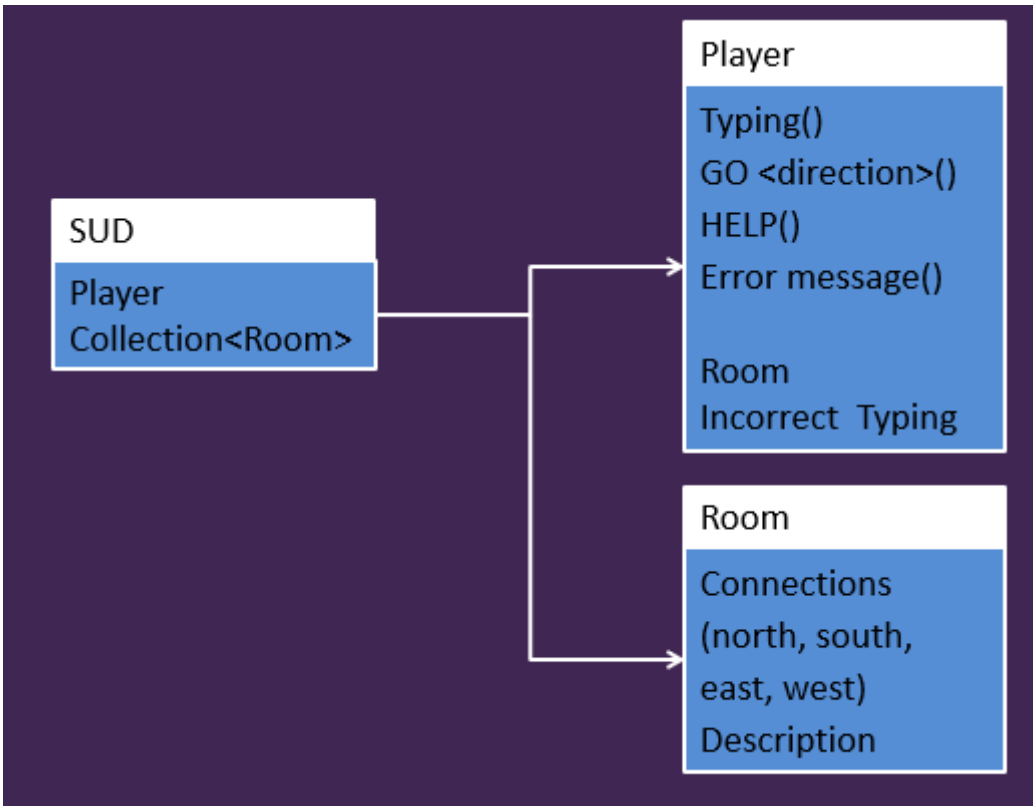
Developing a single user Dungeon (SUD) in Python

Introduction

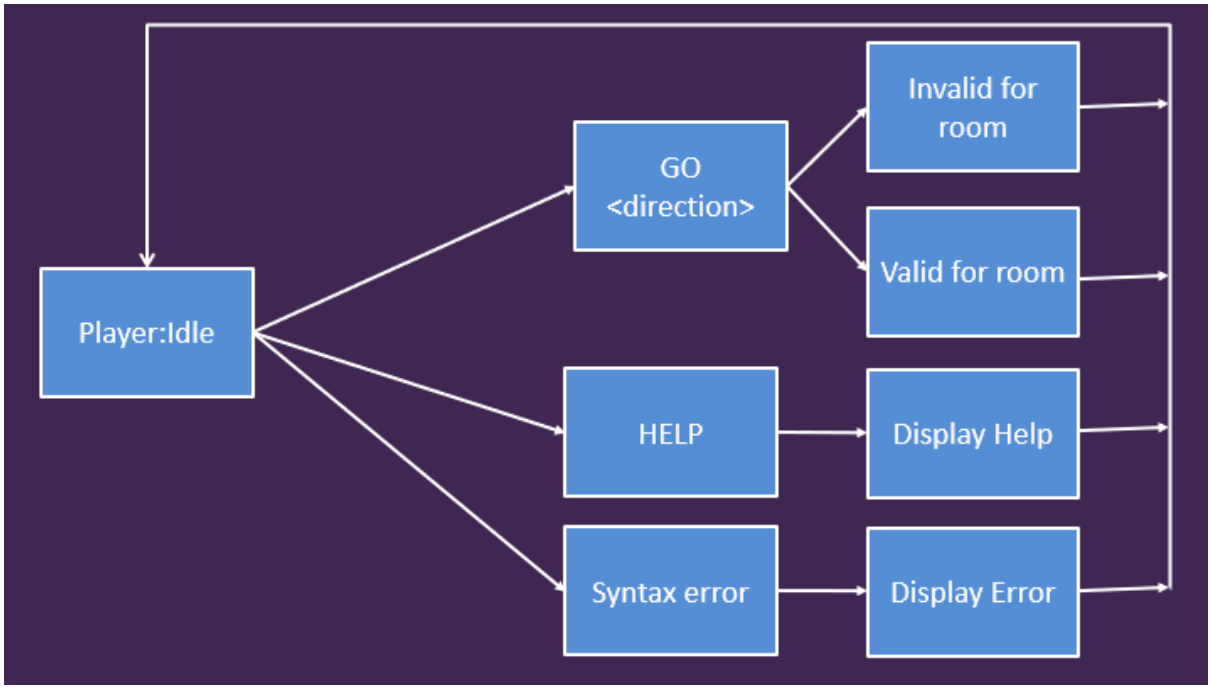
The goal of this worksheet is to create a dungeon application in Python that will run from PyCharm. A player can traverse the rooms in the dungeon using text commands of GO <direction>. You can design and implement other commands for the player.

Design

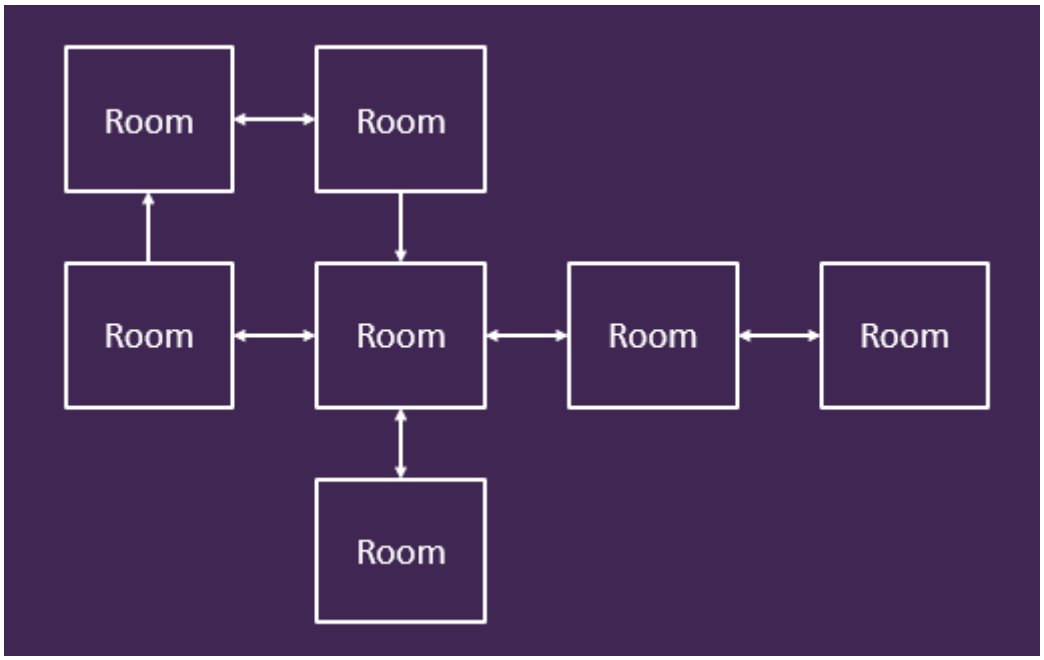
From the lecture, we developed a class hierarchy and a state diagram for the SUD. It's worth remembering that these UML diagrams are based on our understanding of the creative design of the SUD application and it's common for them to change as we move from design to implementation, so don't feel that your application must absolutely follow the design.



The class hierarchy shows three classes, a SUD application, a player and a room. The SUD contains the player instance and the dungeon as a collection of rooms.



The state diagram for the SUD shows the player in an idle state with 3 outcomes to their typed commands; a GO command that will take the player to a new room (if there is a valid connection), a HELP command and a syntax error for anything the player types that can not be resolved as being a GO or HELP command.



The dungeon also requires a dungeon, that is a collection of rooms that are connected (north, south, east & west). From the lecture material, it will make sense to implement the dungeon collection as a dictionary of strings that look up rooms, that way we can use the name of a room for the room connections which should make it easier to layout and manage the dungeon.

Implementation advice

1. Create a new Python project in PyCharm and create an application stub for launching your app using `if __name__ == '__main__'`.

2. The dungeon will be a collection of rooms implemented as a dictionary `self.dungeon = {}`

3. Each room will have a unique name and will map in the direction from name to a room object, `self.dungeon["room 1"] = Room("room 1", <description>, <connections>).`

4. When the rooms are configured, use '' to indicate that there is no connection in that direction.

5. It's worth designing the dungeon on paper to ensure that the rooms are connected as you think

6. The player object will need to have a current room when the dungeon is started, this will be stored as a string

7. The main game loop will ask the player to enter their command using Python's `input` function which returns a string

8. The string returned can be split into an array of strings using `user_input = split(' ')` to remove the spaces between words

9. The first element of the array will be the user's command, `user_input[0].` This can be tested against the valid commands of 'help' & 'go'

10. Using `lower()` on user input and testing against lower case 'help' & 'go' will remove any case issues

11. For the go command, use the player's current room to see if the move is valid. If it isn't valid, inform the player, if it is valid, move the player

Taking it further

Once you have a basic dungeon crawler, it's worth looking at the following:

1. Add more commands for the player to type

2. Expand the room class so that the player will receive different descriptions for a room on subsequent visits

3. Add objects to the world that the player can interact with

4. Add NPCs to the game

5. Look at integrating the SUD into a visualisation library like PyGame or Curses or PyQT