CMPUT 175 - Winter 2023

Assignment 3

Opened: Wednesday, 1 March 2023, 12:00 AM
Due: Thursday, 6 April 2023, 11:55 PM

Assignment 3 - Magical Castle

Due Date: April 6th, 2023 at 23:55

Percentage overall grade: 7%
Penalties: No late assignments allowed
Maximum Marks: 100

Pedagogical Goal: Refresher of Python and hands-on experience with algorithm coding, input validation, exceptions, linked lists and graphs, and data structures with encapsulation.

Read the whole document before starting to solve the Assignment problem.

Submit only your work. While you can collaborate and help each other, do not share code. If you collaborate, acknowledge your collaborator in the code.

Follow the Software Quality Requirements.

Game description:

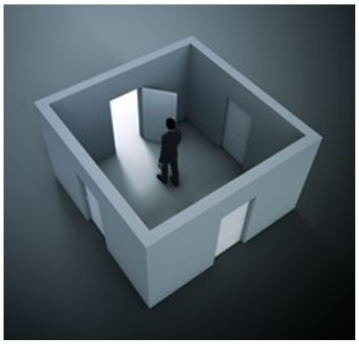
Welcome to an enchanting world of adventure and mystery! You and your friend have embarked on a quest to explore a mysterious castle brimming with diamonds. This castle comprises 25 rooms numbered from 1 to 25, with four magical doors each: North, South, East and West. These doors can lead you to another room or be blocked. The doors are imbued with magical properties that allow them to transport you to adjacent rooms or teleport you to a completely different one. (For instance, the doors in room 1 may lead you to room 25!)

As you explore the castle, you'll come across rooms that may be empty or contain a diamond, wormhole, or portal. The distribution of these treasures and threats is random, so you won't know what's behind each door until you explore it yourself. When you find a room with a diamond, you can collect it and earn points based on the number of diamonds in that room. The room will then become empty.

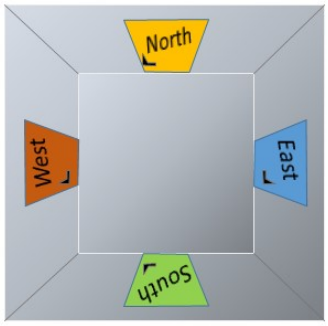
If you enter a room with a wormhole, you'll be teleported to a random room. On the other hand, if you enter a room with a portal inside, you'll be sent back to the initial room with the castle entrance door.

Your ultimate goal is to leave the castle as quickly as possible while collecting the maximum number of points. The castle itself is defined in a file that the program will read to create a data structure representing the castle. Each room will be a node in this structure, with four links representing the doors, each of which could be None if the door is blocked. The node will also indicate whether there are Diamonds, a wormhole, or a portal in the room, or if it is empty.

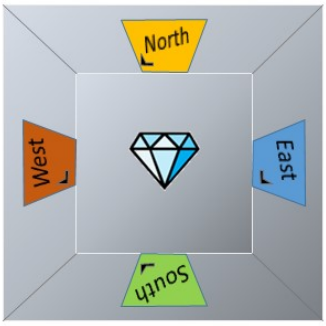
Remember, there is only one entrance and one exit in the castle, and you'll only be able to leave once you've opened the correct door in the exit room. Thus, doors may lead to another room, may be blocked, or may be specifically labeled as the exit or entrance. With these challenges ahead, let the adventure begin!



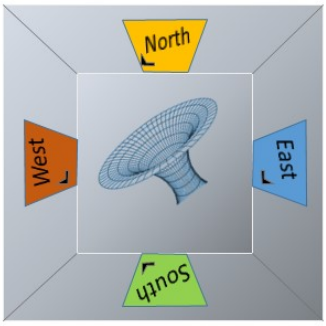
The magical castle has 25 rooms
each with 4 doors: North, South, East and West
A door can be locked or can lead you to any of the 24 other rooms.



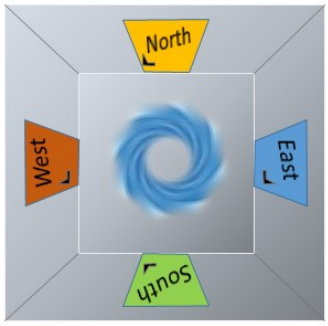
Empty Room



Room with a Diamond



Room with a Wormhole
Takes to a random room



Room with a Portal
Takes to the intial room

Assignment Specifications:

Now that you understand the rules of the game, your objective is to create the following classes that allow you to go through this magical castle. In all of them, you are supposed to raise Exceptions appropriately.

Task 1: Implement the Diamond Class

The Diamond class is going to be a simple class only containing a simple variable showing the number of diamonds. This class should provide a setter and a getter function to retrieve or set the number of diamonds. The number of diamonds cannot be a negative number (Raise an exception for the cases when the number of diamonds is negative). The class should also have a string representation that shows the number of diamonds.

Example output of __str__ method:

Number of Diamonds: 2

The Diamond class should have at least the following methods:

__init__(self, Diamonds: int = 1)

__str__(self)

get_diamonds(self)

set_diamonds(self, Diamonds: int)

Task 2: Implement the Room Class

The Room class represents the nodes of the castle graph. Each room contains an ID, a diamond object (which can be null), a boolean to indicate whether there is a Portal inside the room, a boolean to indicate whether there is a wormhole inside the room, and four links (North, South, East, West) to other rooms. These links should be set appropriately based on the input file. It is important to note that a room can only be in one of four states: empty, containing diamonds, containing a wormhole, or containing a portal. If more than one state is attempted to be set, an exception should be raised. Please note that the room with an Exit or Entrance cannot have a portal or a wormhole.

This class should provide the following functionalities:

- A setter and a getter functions for the diamond object

- A setter and a getter functions for the portal boolean
- A setter and a getter functions for the wormhole boolean
- A function to generate a random room ID when there is a wormhole inside the room (the room ID cannot be equal to the current room ID). Please raise an exception, when this function is called on a room that doesn't have a wormhole inside.
- A setter and a getter functions for the ID, for the setter function if ID is not an integer you should raise and exception
- A setter and a getter functions to change the doors' links based on the input parameter. Both of these functions receive a direction and the getter function should return a value based on the given direction (Please note that the direction should be valid). If the input parameter for the setter is not "east", "west", "south", or "north", it should raise an exception. The values for the doors can be None, other Room objects, "entrance", or "exit". (Other values should raise an exception)
- A function to check whether there is an entrance or exit door inside the room.

The Room class should have at least the following methods:
__init__(self, ID = None, north = None, south = None, east = None, west = None, portal: bool = False, wormhole: bool = False, diamond: Diamond = None)
get_id(self)
set_id(self, ID: int)
generate_random_room_id(self)
get_portal(self)
set_portal(self, portal: bool)
get_wormhole(self)
set_wormhole(self, wormhole: bool)
get_diamond(self)
set_diamond(self, diamond: Diamond)
get_door(self,direction: str)
set_link(self,direction: str, val)
isthere_entrance_exit_door(self)

Task 3: Implement the Castle Class

The Castle class should be a graph of rooms defining the whole structure of the castle and the connectivity of the rooms. The Castle class should have a data structure containing the room objects. Please note that there is only one entrance and one exit at the castle.

This class should provide the following functionalities:

- A function to add a room to the current data structure of the castle. Please note that if the given room already exists in the castle, you should raise an exception.
- A function to get an specific room with an specified ID (Raise an exception if the given room is not inside the castle)
- A function to change an specific room with an specified ID (If ID is out of the range 25 raise an exception)
- A function to get the ID of the room with the entrance door.
- A function to get the ID of the room with the exit door.
- A function which receives a Room ID and a door, and outputs another Room ID which is going to be the room behind the door that is selected or a random room ID which is generated when entering the room with a wormhole. If the room behind the door has a portal inside this function should return the id of the room with the entrance door. For the cases where the chosen door is "entrance" or "exit", this function should return "entrance" or "exit". Please note that to prevent multiple and perpetual teleportation and being stuck in a loop jumping from one room to another when a wormhole transports the player to a room that also has a wormhole, it is essential to ensure that the random room generated by the room containing the initial wormhole does not also have a wormhole inside. You can achieve this by generating a random room ID in a while loop until the generated room ID does not have a wormhole.

The Castle class should have at least the following methods:

__init__(self)

add_room(self, room)

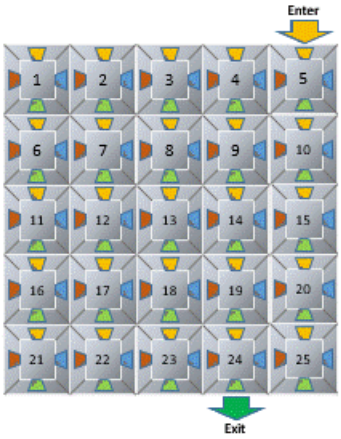
get_room(self, id)

change_room(self, id, new_room)

get_entrance_id(self)

get_exit_id(self)

get_next_room(self, room_id, door)



The 25 rooms are not adjacent.
In the magical castle a door can lead you to any other room. The set of all rooms are organized like a graph where each node has up to 4 links.

Task 4: Implement the Player Class

The player class should represent a player. This class should contain a variable indicating the ID of the room corresponding to the current position of the player, the total number of diamonds gathered so far, a data structure containing all of the previous decisions the player has made so far (Room ID and door ID (North, South, East, West), and an ID for the player which can be 1 or 0. This function should have a string representation that shows the current position of the player and the total number of diamonds the player has gathered so far. When a player faces a portal that sends them back to the initial room, they do not lose their collected diamonds.

Example output of __str__ method:

Player 1. Diamond count: 2

This class should provide the following functionalities:

- A setter and a getter functions for the current position of the player (If ID is out of the range 25 raise an exception)
- A getter function to get the ID of the player.
- A setter and a getter functions for the total number of diamonds player has
- A function to print the path player has traveled so far in the following format:

1 -> East, 2 -> North, 6 -> West, ...

- ~~A function to clear the data structure containing the path (For the case when a player faces a room containing a portal). When a player faces a portal that sends them back to the initial room, they do not lose their collected diamonds.~~
- A function to add the current decision (North, South, East, West) and the Room ID corresponding to the player decision, to the data structure containing the path
- A move function that receives one of the room's IDs and changes the position of the player.

The Player class should have at least the following methods:
__init__(self, player_id)
__str__(self)
get_position(self)
set_position(self, id)
get_player_id(self)
get_diamonds(self)
set_diamonds(self, count)
print_path(self)
~~clear_path(self)~~
add_to_path(self, room_id, door_id)
move(self, room_id)

Task 5: Implement the Game Class

The game class represents the game.

Attributes

- Castle: This is a Castle object. The input file is going to be a txt file in which first two lines describe the exit and entrance, and the rest of this file is the adjacency list. Each line (there are 25) has IDs, door's next rooms, and the room content.

- Players: A list of players in the game. This is a list of Player objects. There are normally 2 players.

- Finished: a list with the exact size as the Players list containing booleans indicating whether each player has finished the game or not.

- Turn: The turn of the game. This is an integer indicating the ID of the player whose turn it is in the list of players.

This class should provide the following functionalities:

- A function to read a file containing the information about the Castle, and then create the Castle based on the information provided. This function should also set the initial position of the players to the room with the entrance.
- A setter and a getter functions for the turn variable.
- ~~A function to create the Castle based on the information provided.~~ [Redundant from above]
- A function to receive information about a room and creates the Room object corresponding to the information provided.
- ~~A function to create the Castle based on the information provided.~~ [Redundant from above]
- A function that asks the user to enter a decision(North, South, East, West) and move the player whose turn it is.
- A function that checks whether the game is finished or not. The game will be finished when both of the players exit from the castle. If it is finished, print the path both players traveled so far.
- A function that updates the number of diamonds the player has based on the current position of the player. Remember that you should set the number of diamonds to zero after visiting the room containing diamonds.
- A function which receives a player ID and returns the Player object corresponds to that given player_ID.

The Game class should have at least the following methods:

__init__(self)

initialize_from_file(self, filename)

get_turn(self)

set_turn(self, turn)

get_player(self, player_id)

build_room(self, room_id, diamond, portal, wormhole)

move(self)

is_finished(self)

update_diamonds(self)

An example of a file showing the adjacency list of the rooms in the castle is as follows. Store this example in a file named "**castle.txt**" and use that to test your code as the input file. You can also use other configurations for testing. The TAs can evaluate your Assignment with yet another configuration.

```
E: 5, N
X: 24, S
1: 0, 21, 2, 0,
2: 0, 9, 19, 9,
3: 0, 9, 19, 1, W
4: 0, 3, 14, 7,
5: E, 11, 0, 9,
6: 1, 22, 2, 0, W
7: 16, 9, 4, 1, W
8: 19, 10, 0, 17,
9: 1, 12, 2, 21, DD
10: 8, 20, 0, 17, D
11: 9, 23, 23, 0, W
12: 21, 11, 5, 15, P
13: 10, 25, 8, 20,
14: 0, 19, 24, 3, D
15: 21, 0, 0, 22, W
16: 0, 7, 4, 0, P
17: 19, 20, 0, 4, DD
18: 13, 25, 0, 20,
19: 14, 17, 10, 2,
20: 17, 23, 13, 11, P
21: 3, 22, 9, 0, DDD
22: 12, 0, 15, 16,
23: 20, 0, 25, 11,
24: 14, X, 0, 3, D
25: 18, 0, 0, 20,
```

The first two lines of the file describe the Entrance (room 5, North door) and Exit (room 24, South door)
Then the rest is an adjacency list. Each line (there are 25) has IDs of 4 rooms (North, South, East, West). When it is 0, the door is blocked. The 4 IDs are followed by the initial content of the room. W is a wormhole, P is a postal, and D a diamond. You may have more than one D.
There is only one room that has a door indicating X. X is for Exit (e.g. room 24 above). Also only one room presents a door with ID E (eg. room 5 above). E stands for Entrance.
Note that with the adjacency list above, there could be many paths leading from E to X. e.g. 5(W)-9(E)-2(E)-19(N)-14(E)-24(S), meaning from room 5 take West door, from room 9 take the East door, etc.

Inside the **main.py** you should create the game object and create the game loop by calling the proper functions of the objects.

General Guidelines

In addition to making sure that your code runs properly, we will also check that you follow good programming practices. For example, divide the problem into smaller sub-problems, and write functions to solve those sub-problems so that each function has a single purpose; use the most appropriate data structures for your algorithm; use concise but descriptive variable names; define constants instead of hardcoding literal values throughout your code; include meaningful comments to document your code, as well as docstrings for all functions; and **be sure to acknowledge any collaborators/references in a header comment at the top of your Python file**.

Restrictions for this assignment: You cannot use break/continue, and you cannot import any modules apart from the files containing other classes. You should import the random module. Importing other modules will result in deductions.

Assignment Deliverables

- You are going to submit 6 Python files, one for each of the described classes:
  - diamond.py
  - room.py
  - castle.py
  - player.py
  - game.py
  - main.py