Dungeon Adventure — Program Design

Classes

`Room`

- __init__(id, location) expects an integer (id) and a tuple (location) should do the following:
  - Store id and location as private variables
  - For self.__health_p and self.__vision_p, generate a different random number for each one and have them be `True` 10% of the time, `False` otherwise.
  - self.__pit should follow the same logic as the potions, but instead of `True`, generate another random number between 1 and 20.
  - self.__n, self.__w, self.__e, and self.__s should all be initialized as `None`.
  - self.__pillar, self.__exit, and self.__has_player should be initialized as `False`.
- enter(adventurer) expects a reference to the player and is called when they enter the room.
  - If the room has the Exit, call adventurer.exit()
  - If there's a Pillar, the method should call adventurer.earn_pillar(self.__pillar), then set self.__pillar to `False`.
  - If any of the potion variables are `True`, they should become `False`, and the method should call adventurer.add_health_potion() and/or adventurer.add_vision_potion() as appropriate.
  - If there's a pit, call adventurer.take_damage(self.__pit, "pit trap")
  - Set self.__has_player to `True`.
- leave() sets self.__has_player to `False`.
- link(other_room, direction) expects a `Room` object and a string "n", "w", "e", or "s".
  - Check if other_room's complimentary direction (e.g. if direction is "n" then check other_room's south link) is `None`.
    - If so, then store room in the variable indicated by direction. (Switch statements are your friend here.) Else, set the link indicated by direction to `False`.
- wall(direction) expects a string "n", "w", "e", or "s" and sets the corresponding direction to `False`.
- Write "getters" for each of id, location, n, w, e, and s using the naming convention "get_whatever()"
- set_pillar(pillar) expects a string "A", "I", "E", or "P" and sets self.__pillar to that value.
- clear_room() removes all pit and potions (i.e. sets them to `False`)
- set_as_exit() should set self.__exit to `True`, then calls self.clear_room()
- __str__() should create a graphical representation of the room as per the assignment description:

- Must contain a _ _ str _ _ () method that builds a 2D Graphical representation of the room (NOTE: you may use any graphical components that you wish). The (command line) representation is as follows:
  - * - * will represent a north/south door (the - represents the door). If the room is on a boundary of the maze (upper or lower), then that will be represented with ***
  - East/west doors will be represented in a similar fashion with the door being the | character as opposed to a -.
  - In the center of the room you will display a letter that represents what the room contains. Here are the letters to use and what they represent:
    - M - Multiple Items
    - X - Pit
    - i - Entrance (In)
    - O - Exit (Out)
    - V - Vision Potion
    - H - Healing Potion
    - <space> - Empty Room
    - A, E, I, P - Pillars

(Addendum to assignment: if self.__has_player is `True`, display the "@" symbol rather than the other non-door, non-wall symbols)

Example: Room 1,1 might look like

```
* - *
| P |
* - *
```

`Adventurer`

- `__init__`(name, game) expects a string (name) and a reference to `DungeonAdventure` (game) and should do the following:
  - Store name and game as private variables
  - self.__pillars should be initialized as an empty list
  - self.__vision_p and self.health_p should both be 0
  - self.__location should be initialized as `None`. # not sure if we actually need this
  - Use a random number to make self.__hp between 75 and 100
    - For funsies, come up with a way to make health lower on higher difficulties. Difficulty is an integer from 1-3. You can get it with game.get_difficulty()
- earn_pillar(pillar) expects either "A", "E", "I", or "P".
  - If the letter passed is already in self.__pillars, raise an exception
  - If the adventurer was passed something that is not one of those letters, raise an exception
  - Append the letter to self.__pillars
  - Make a notification to the player using self.__game.announce("your message here")
- add_health_potion(), use_health_potion(), add_vision_potion(), and use_vision_potion() should all increase or decrease the relevant potion count by 1.
  - The "use" methods should check if the adventurer has potions to use before decrementing the count. Return `True` if they do, and `False` otherwise.
  - These methods should never decrease potion count to below 0.
  - use_health_potion() should generate a random integer between 5-15 and add it to self.__hp, but only if a potion was actually used.
  - Each of these methods should make an announcement using DungeonAdventure's announce() method (as above). If one of the "use" methods was called and the potion count was 0, the announcement should inform the player they have no potions left.
- take_damage(damage, source) expects an integer damage amount and a string naming the source of the damage.
  - Decrease self.__hp by the given amount. This can put the adventurer below 0 HP.
  - Make an announcement listing the damage, the name of the damage source, and the player's new HP total.
    - You don't need to do anything special if their HP reaches <=0. DungeonAdventure will make a separate announcement when the game ends.
- exit() should check if the player has all the pillars and call self.__game.end_game() if so.
  - An easy way to do that is just get the length of self.__pillars
- __str__() returns a string listing name, hp, potion totals for each type of potion, and pillars found.

`Dungeon`

- `__init__`(difficulty, game) expects an integer between 1-3 and a reference to `DungeonAdventure`:
  - Store difficulty and game as private variables.
  - Sets self.__size to 5 + (2 * difficulty)
  - Initialize self.__entrance and self.__player_location to `None`.
  - Initialize self.__room_count to 0.
- generate(adventurer):
  - Creates the maze by making self.__entrance a new `Room` object and generating from there.
    - Use the room's .link() and .wall() functions to connect or block rooms.
  - The entrance room should be empty (use its .clear_room() method). Once the maze is generated, start the adventurer there by setting self.__player_location equal to self.__entrance, then calling self.__entrance.enter(adventurer).

    - Distribute the Pillars and the Exit across the dungeon such that none are in the same room.
- __validate() returns `True` or `False` depending on whether the dungeon is valid for play.
    - Per assignment requirements, it is mandatory to include a traversal algorithm as part of this.
- __str__() should generate a string based on the __str__() function of all contained `Room` objects.
- display(range) expects an integer and should return a reduced version of self.__str__() based on the player's location.
    - Ideally we'd have vision potions interact with the range value, but we'll see how that goes.
- move_player(adventurer, direction) expects an `Adventurer` reference and a string "n", "w", "e", or "s"
    - If direction is not one of the expected values, it raises an exception
    - Use the self.__player_location reference and getter functions (e.g. .get_n()) to determine whether the desired move is valid.
        - If so, call the current room's .leave() function and the .enter(adventurer) function of the destination room, then update self.__player_location with a reference to the new room.
        - The move should fail otherwise.
    - Make an announcement using self.__game.announce("your message here") with the result of the attempted move.

`DungeonAdventure`

- Creates and updates GUI with dungeon, controls, and message log. # TKinter package, maybe?
- Runs main game loop
- Captures player input
- Constructor:
    - initializes self.__dungeon, self.__adventurer, and self.__difficulty as `None`.
    - Initializes self.__game_over to `False`.
- start_game() method that creates a `Dungeon` object and enables game controls.
- announce(message) adds messages to the message log.
- end_game() sets self.__game_over() to `True`.
- update_gui() updates the GUI based on game state