COMP 1510 Assignment # 3: Your first SUD!

Christopher Thompson
cthompson98@bcit.ca

Due Sunday November 10th at or before 20:59:59

Welcome!

I hope you've been enjoying the Dungeons and Dragons theme, because we're going to implement something very similar: a multi-user dungeon (a MUD!).

In the "olde dayes" of internet and computing, we didn't have amazing, graphically intense MMOs like WoW. We used to dial into things called multi-user dungeons. These were text-based real-time role-playing adventure games that let users read descriptions about rooms, objects, other players, non-player characters, etc., and perform actions and interact with the virtual world. Oh my gosh, when I think of the hours and hours and hours I spent wandering those dungeons...

Let's implement a simple MUD. Of course, we don't know how to use Python for networking yet, so this will in fact be a single-user dungeon, or a "SUD".

1 Submission Requirements

1. **This take home assignment is due no later than Sunday November 10th at or before 20:59:59.**
2. Late assignments will not be accepted for any reason.
3. This is an individual assignment. I strongly encourage you to share ideas and concepts, but sharing code or submitting someone else's work is not allowed.

2 Project Setup

Please complete the following:

1. Open the PyCharm project you created for the first assignment.
2. For your third assignment, all of your code must go into the A3 folder. After you complete each function, commit and push your change to version control. In order to earn full marks, you must commit and push after each function is complete.

3 Style Requirements

1. Your main method must be placed inside a file called sud.py.
2. You must comment each function you implement with correctly formatted docstrings. Include informative doctests **where necessary** to demonstrate to the user how the function is meant to be used, and what the expected output will be when input is provided.
3. For this assignment, **functions must contain no more than 20 lines of code** (excluding the definition statement and excluding any docstrings or doctests). Functions must only do one logical thing. If a function does more than one logical thing, break it down into two or more functions that work

together. Remember that helper functions may help more than one function, so we prefer to use generic names as much as possible. Don't create functions that wrap around an existing function, i.e., don't create a divide function that just divides two numbers because we already have an operator that does that.

4. Each function must also be tested by a collection of unit tests. Ensure your test cases and functions have descriptive, meaningful names, and ensure that every possible valid input is tested for each function. Remember we want to create disjointed equivalency partitions out of all possible output, and test a representative from each partition.

5. Ensure that the docstring for each function you write has the following components (in this order):

   (a) Short one-sentence description that begins with a verb in imperative tense

   (b) One blank line

   (c) Additional comments if the single line description is not sufficient (this is a good place to describe in detail how the function is meant to be used)

   (d) One blank line

   (e) PARAM statement for each parameter which describes what the user should pass to the function.

   (f) PRECONDITION statement for each precondition which the user promises to meet before using the function

   (g) POSTCONDITION statement for each postcondition which the function promises to meet if the precondition is met

   (h) RETURN statement which describes what will be returned from the function if the preconditions are met

   (i) One blank line

   (j) And finally, the doctests. Here is an example:

```
def my_factorial(number):
    """Calculate factorial.

    A simple function that demonstrates good comment construction.

    :param number: a positive integer
    :precondition: number must be a positive integer
    :postcondition: calculates the correct factorial
    :return: factorial of number

    >>> my_factorial(0)
    1
    >>> my_factorial(1)
    1
    >>> my_factorial(5)
    120
    """
    return math.factorial(number)
```

4 Functional Requirements

Your SUD (single user dungeon) must include the following essential components:

1. **Before you type a single character of code, I would like you to try playing a MUD first**. There are all sorts of resources and lists online for finding MUDS. I've isolated an old and popular MUD for you to try: `http://www.aardwolf.com`. Please spend 30-60 minutes playing Aardwolf. Choose Play Aardwolf from the menu in the upper left, and use the Flash client (it's expedient). Be careful. It can be addictive, especially if you're procrastinating!

2. **Be kind and thoughtful when you are logged in**. Last term the aardwolf mods banned BCIT IP addresses because some students were being inappropriate. Having the ban lifted was not easy. Please do not do this. If you do, you will earn zero on this assignment. Stay in character. Don't swear. Don't talk about BCIT. These people take their role-playing very seriously.

3. **Implement a very simple Single User Dungeon (SUD)**. Using your experience with Aardwolf as an example, construct a text-based adventure experience for me. Your game must include the following elements:

   (a) a whimsical, descriptive, and engaging scenario
   (b) a character that I can control
   (c) the ability to move north, east, south, and west
   (d) a boundary to the dungeon, i.e., I cannot move infinitely in any direction
   (e) each time I move, there is a 25 percent chance that I will encounter a monster
   (f) when I encounter a monster, I would like to be able to run away or fight it. If I fight it, it must be combat to the death. If I run away, there is a 10 percent chance the monster will stab me in the back and do 1d4 damage as I flee.
   (g) the game must end when I type quit

4. **We need some ground rules for health and damage**. Let's decide that every new character has 10 health. A monster has 5 health. When a character strikes a monster, it should inflict some damage, let's say 1d6. When a monster strikes a character, let's decide they always do 1d6 damage too.

5. **How does a character heal?** A character's health should increase two points each time the character moves and does NOT encounter a monster. A character's health should never exceed 10 though.

6. **Separate** your code into different modules. The sud.py file should contain game logic, for example. You may decide to have a monster.py file which contains code that manages the monster(s) in your game. And you probably have a character.py file which contains code that manages the character, etc.

7. That's it. Keep it simple. I want clean code, short functions, lots of comments, and simple game play.

5 Grading

Your third assignment will be marked out of 10. I will randomly select some functions from this assignment and mark them. For full marks, you must:

1. (5) Correctly implement the requirements described in this document.

2. (2) Correctly write doctests and unit tests to thoroughly test your code. Only unit test function input that meets the precondition(s), or input that requires an error message.

3. (1) Correctly commit and push your code to git and GitHub at regular intervals, i.e., after you complete each function.

4. (2) Use good style, minimize code repetition, correctly format and comment your code, eliminate all warnings offered by PyCharm, use good function and variable names, write code that is easy to understand, use whitespace wisely, employ good grammar in your comments, make sure functions are brief and only do one thing, etc.

Please remember that this is an individual assignment. I strongly encourage you to share ideas and concepts (please!), but sharing code or submitting someone else's work is not allowed.

Good luck, and have fun!

PS – If you're tired of dungeons and monsters, create a Hello Kitty online adventure. Instead of monsters, maybe Hello Kitty encounters sweets. Can she stonewall the seductive saccharine samples she spies? Or will she succumb to the siren sugar song and sink into a diabetic stupor, ending the game? Or maybe you have a better idea. Go for it!