COMP 1510 Assignment #4:
Make me a game, please!

Christopher Thompson
chris_thompson@bcit.ca

Due **Friday March 11th at or before 23:59:59 PM**

Games

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 (MUDs). MUDs 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...

For your fourth assignment, you will implement a simple multi-user dungeon. 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".

Any project of this magnitude requires some planning. Your fourth assignment this term begins with some planning. Let's get started.

1 Submission requirements

1. This assignment is due no later than **Friday March 11th at or before 23:59:59 PM**
2. Late submissions 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 another person code is not allowed.

2 Grading scheme

1. A detailed grading rubric for this project has been appended to the end of this document. It includes a list of required elements you must include in your project.

3 Assignment 4 prep

1. Follow this link to accept the assignment from GitHub Classroom, and copy the PyCharm template I created to your personal list of repositories:

   `https://classroom.github.com/a/15wAutJu`.

2. Clone the project to your local machine.
3. Add your name and student number to the README.md.
4. Commit and push. Hello Chris!

5. **Stop what you're doing. Before you type a single character of code, you must try playing a MUD first**. There are all sorts of resources and lists online for finding MUDS. I've chosen an old and venerated MUD for you to try called **Aardwolf**. An aardwolf is a kind of hyena that eats insects. I have no idea why the developers chose this name.

6. **Start by downloading Mudlet**, a MUD client I've chosen for you `https://www.mudlet.org/`. It is available for Windows and macOS. Download and install it please.

7. Start Mudlet. When Mudlet opens, a modal window invites you to select an existing game or create a new profile. **Create and save a new Profile for the Aardwolf MUD**:

   (a) Profile name: Aardwolf, of course!

   (b) Server address: aardwolf.org

   (c) Port: 23

8. Then log in and play!

9. Take your time, read everything, and follow the instructions. You will start by typing NEW for new character, and then you will be asked for your character name and a password. Don't lose these!

10. **Play Aardwolf until you reach level 10**. You must add a screenshot to your A4 project (in PDF format or some other easily-viewed format) to prove your character has reached 'Level 10'. You will understand what this mean after you have played for a few minutes. And how cool is this – you have to play a game for marks! **Add the PDF to your PyCharm project and commit and push using the commit message: "ACHIEVED LEVEL 10, CHRIS!"**.

11. Be kind and thoughtful when you are logged in. A few terms ago, the Aardwolf mods banned DTC BCIT IP addresses because some students were being inappropriate while playing from campus. Having the ban lifted was not easy. Please do not do this. These people take their role-playing very (perhaps too!) seriously.

12. Based on your experience with Aardwolf, apply what you have learned so far in COMP 1510 and design a very simple game. Examine the list of game requirements and the list of Python language requirements in the next section. **Develop a PDF flowchart that captures the important elements of the game**. Show me what happens when I play your game. Use indirection. Pass arguments to functions. Return values from functions. Identify where the required Python elements will go. This flowchart will be large. If you select a large page size, perhaps you can fit it all on one page. Go for it. I have a huge monitor. **Add the PDF to your PyCharm project, and commit and push using the commit message: "COMPLETED GAME DESIGN, CHRIS!"**.

4 Include these mandatory elements

Implement an interesting Single User Dungeon (SUD). Using your experience with Aardwolf as an example and your personal gaming, literature, or popular culture interests as a starting point, construct a text-based adventure experience for me.

1. **All Python source code must remain in game.py.**

2. **Your game must include the following elements**:

   (a) A flowchart called **game.pdf** that is updated to correctly show me EXACTLY what is happening in the final version of your game.

   (b) a whimsical, descriptive, and engaging scenario

   (c) a 10 x 10 **or larger** grid-based environment

   (d) some sort of ASCII map or partial map to guide my adventure

   (e) a character who has a name, health/hit points aka HP, a class, a level, and class-based attacks that improve when the character levels up

(f) I can MOVE my character in the four cardinal directions: NORTH, EAST, SOUTH, and WEST

(g) each time my character moves, there is a twenty (20) percent chance that I will encounter some sort of foe when I arrive at my next location

(h) when I encounter a foe, I would like to be able to run away or fight my foe. If I fight my foe, it **IS NOT** a fight to the death. I may choose to flee at the beginning of any round of combat during the encounter. If I run away, there is a twenty (20) percent chance the foe will do damage as I flee. And. There is a twenty (20) percent chance at the end of each round that your foe will run away. You, being the kind adventurer that you are, will not shank them as they flee.

(i) the game ends when I kill a boss who has a fixed position

(j) the game must also end if I type quit instead of choosing a direction.

3. **The character needs a name**. The user must choose the name.

4. **The character also needs a class**. The class the character chooses determines the number of hit points they start with, and how much damage they can commit during attacks. Create four classes. For example, you may choose to create:

   (a) A sorcerer class that uses magic with a low probability of success and an extremely high damage score

   (b) A thief class that specializes in throwing butterknives with shocking accuracy, applying steady low level damage with high probability

   (c) An Amazon class that uses archery, swordplay, and a touch of magic to apply a reliable and consistent amount of damage

   (d) A fighter class that uses freakish natural strength, any weapon they can find, and sheer blind luck to deal stunning damage sometimes and a lot of damage the rest of the time.

5. **Develop a simple levelling scheme** for each of your four 'classes':

   (a) My character should start at level 1, and be able to reach level three (3).

   (b) Each level needs a name, and a certain amount of experience points need to be accrued before the character reaches that level.

   (c) When a character reaches a new level, two things should happen – the maximum HP should increase some reasonable amount, and the amount of damage the character does should also increase.

   (d) When a character reaches level 3, they should finally be able to take on the boss with a reasonable expectation for success.

6. **Create a coherent, rich ecosystem of foes**. Perhaps foes become more challenging as the character level increases. Maybe your game space will have regions for different levels. Be creative.

7. If a character dies, the game must end.

8. **Your game must incorporate the following Python elements**:

   (a) one or more tuples used in a manner that makes sense

   (b) one or more lists used in a thoughtful and correct manner

   (c) one or more list/dictionary comprehensions

   (d) selection using if-statements

   (e) repetition using the for-loop and/or the while loop

   (f) the membership operator where it makes sense

   (g) the range function

   (h) one or more functions from itertools

   (i) the enumerate function

   (j) the filter or map function

(k) the random module

(l) function annotations

(m) doctests and/or unit tests for every single function (that is, every function needs doctests or unit tests or doctests and unit tests).

(n) ALL output must be formatted using f-strings and/or str.format and/or %-formatting

9. That's it. Keep it simple. I want clean code, short functions, lots of comments, and simple game play.
10. No really, I mean it. I will withhold marks for solutions that are unnecessarily complicated. Simplicity and clarity trump everything.
11. No global variables. Global constants are permitted. Absolutely no single letter variable names.

Style Requirements

1. You must observe the code style warnings produced by PyCharm. **You must style your code so that it does not generate ANY warnings.**

   When code is underlined in red, PyCharm is telling us there is an error. Mouse over the error to (hopefully!) learn more.

   When code is underlined in a different colour, we are being warned about something. We can click the warning triangle in the upper right hand corner of the editor window, or we can mouse over the warning to learn more.

2. You must comment each function you implement with correctly formatted docstrings.
3. **Include informative doctests where necessary and possible** to demonstrate to the user how the function is meant to be used, and what the expected reaction will be when input is provided.
4. **Include a separate unit test file for each function that requires unit tests**. If a function CAN be tested with unit tests, it MUST be tested with unit tests.
5. For this assignmend/midterm, functions must only do one logical thing and **functions must be no longer than 15 lines of code**. 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 hard code values, either. Make your functions as flexible as possible.
6. Don't create functions that just wrap around and rename 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 called / .
7. **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 and ends in a period

   (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. Avoid describing HOW the function works, just tell us WHAT it does)

   (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: int) -> int:
    """Calculate factorial.

    A simple function that demonstrates good comment construction.

    :param number: a positive integer
    :precondition: number must be a positive integer equal to or
                   greater than zero
    :postcondition: calculates the correct factorial
    :return: factorial of number as an integer

    >>> my_factorial(0)
    1
    >>> my_factorial(1)
    1
    >>> my_factorial(5)
    120
    """
    if number == 0:
        return 1
    else:
        return number * factorial(number - 1)
```

8. Use markdown language to include a table in the README.md file that contains two columns:

   (a) Column 1: Required element

   (b) Column 2: Location (line number)

9. This table must contain a row for each of the following required elements:

   (a) Tuple

   (b) List

   (c) An example of a dictionary or list comprehension

   (d) A remarkable use of selection using the if-statement

   (e) A clever use of repetition with the for or while loop

   (f) the membership operator (in) where it makes sense

   (g) the range function

   (h) one or more functions from the itertools module

   (i) the enumerate function

   (j) the filter or map function

   (k) the random module

   The second column must contain an integer, the line number for each required element.

10. If an element is present in more than one location, list two or three line numbers (at most!) in order of decreasing splendour. That is, if you used a list comprehension in two places, and one is an amazing implementation that you want me to look at and use for the grading, then its location (line number) should be first in a list of length two, and the second slightly less stupendous list comprehension's location should be second. Your backup for me to grades, if you will.

Hints

1. The main function should invoke the game function. And the game function should run the game. This is where the main loop goes.
2. There is absolutely nowhere in this assignment where recursion makes sense. Functions should not endlessly call other functions. Things need to return to the main loop in the game function.
3. The character needs to store a lot of information:
   (a) Name
   (b) Class
   (c) Level
   (d) Current number hit points
   (e) The maximum hit points this class and level can have when at full health
   (f) Current number of experience points

   I'd store this in a dictionary.
4. You need to define four classes. I need you to draw on your experiences as a gamer and a consumer and creator of popular culture. Be creative! Each class needs to have the following information stored somewhere, somehow:
   (a) Name of the class
   (b) The names for the three levels of the class, i.a., acolyte, novice, brother for a clerical class
   (c) Some sort of interesting attack that is unique
   (d) The number of experience points required to reach levels 2 and 3
   (e) The maximum number of hit points a character may have for each level
   (f) The maximum number of damage points a character may commit during each level

   How can you store this information in such a way that you can easily retrieve it when needed by the functions you are implementing?
5. Your foes should be interesting. There are no limits here. The only requirement is that I must encounter a final boss at the end of the game. It must be reasonable for me to defeat the boss, I do not want to be marking your game for hours and hours! When I defeat the boss the game must end. Gracefully. With fanfare. Maybe some cool ASCII art.
6. Devising unique situations for each of 10 x 10 = 100 game cells (locations) in a huge list is not something I want you to do. Be creative. Create a data structure that maps coordinates to functions. The functions can autogenerate some sort of location, maybe using randomly selected bits of different lists of environment descriptions... Just thinking out loud. I did not say that a person needs to be able to trace their steps. The only thing that must remain in a fixed position is the boss. Everything else is up to you. That's it. Good luck, and have fun!
7. The highest marks will be reserved for exemplary submissions that demonstrate elegance and parsimony of code, i.e., short functions aka the fewest number of functions possible aka the fewest number of lines of code in game.py.
8. Do not make me type things. I want to select from numbered lists as much as possible.

If you're not interested in Dungeons and Dragons, make me a Hello Kitty Online Adventure. Or a Star Trek Adventure. Or a RuPaul's Drag Race Adventure. Be creative. The only limit is your imagination.

Good luck, and have fun!

Print Rubric

GitHub	Excellent 3 points	Good 2 points	Needs improvement 1 point	Unsatisfactory or missing 0 points	Criterion Score
Evidence of correct GitHub use including regular pushes and meaningful commit comments. Each commit message is relevant. Each commit is for a single unit of code, or a single fix. Overlarge commits will be penalized,					/ 3

Code style	Satisfactory 1 point	Unsatisfactory 0 points	Criterion Score
Identifiers are in the correct case, are not misspelled, are meaningful, and help me understand the code. No. Single. Letter. Variables.			/ 1
Indentation and use of white space is correct and contributes to the code's consistency and readability. There are NO STYLE WARNINGS offered by PyCharm.			/ 1
Functions contain 15 or fewer lines of code. No exceptions.			/ 1
No. Global. Variables. Constants are enclosed in protective functions with appropriate UPPER_CASE names or are encapsulated in immutable data structures. Constants are grouped together immediately below import statements.			/ 1
All functions are annotated where it makes sense for them to be annotated.			/ 1

Testing	Excellent 3 points	Good 2 points	Needs improvement 1 point	Unsatisfactory or missing 0 points	Criterion Score
Doctests are included for all functions that can be tested with doctests.					/ 3
Doctests correctly demonstrate how to use the function --one or two important cases is enough!					/ 3
Unit tests are included for all functions that can be tested with unit tests.					/ 3
Unit tests correctly and thoroughly demonstrate how each function will react to each legal disjointed equivalency partition or equivalent.					/ 3

Implementation	Satisfactory 1 point	Unsatisfactory 0 points	Criterion Score
Code compiles and the program executes without errors or warnings.			/ 1
The game board is 10 x 10 or larger and is managed efficiently. There does not have to be a 2D list. There does not have to be a dictionary. It can be virtual!			/ 1
Gameplay ends correctly when the character dies or kills the final boss.			/ 1
Character movement and restrictions on character movement are correct, that is, the character can move north south east and west, and cannot cross the gamespace boundaries.			/ 1
Combat (or equivalent) between the user and foes is correct. Foes are generated randomly 20% of the time when a character moves to a new spot in the environment grid.			/ 1
Health (or equivalent) is managed correctly, i.e., increases between encounters, correctly edited during encounters.			/ 1
There is a class system and there are four classes to choose from, each with unique backstory, leveling scheme, damage and HP levels.			/ 1
There is a coherent and rich ecosystem of foes. The foes are not all the same. Some are more dangerous than others. It is not impossible for me to win. Things "scale" nicely.			/ 1
One or more tuples is used in a correct and sensible manner.			/ 1
One or more lists is used in a correct and appropriate manner.			/ 1
There is one or more list and/or dictionary comprehensions used in a manner that is correct and makes your code shorter and easier to read.			/ 1
There is selection using if-statements.			/ 1
There is repetition using the for-loop and/or the while loop.			/ 1
The membership operator is used correctly.			/ 1
The range function is used correctly to make your code more efficient.			/ 1
One or more functions from itertools is thoughtfully used at least once in a correct way, and the enumerate( ) functions is used at least once in a correct way.			/ 1
The enumerate function is used correctly.			/ 1
The filter or map function is used correctly.			/ 1
The random module is used correctly.			/ 1
All output for the user is nicely formatted f-strings and/or str.format and/or %-formatting.			/ 1

Close

learn.bcit.ca/d2l/common/dialogs/nonModal/blank.d2l?d2l_body_type=1&d2l_nonModalDialog_cb=previewRubric_04919076762145559&d...

Overall Impression	Outstanding 5 points	Very good 4 points	Satisfactory 3 points	Unsatisfactory 2 points	Poor 1 point	Not submitted 0 points	Criterion Score
OVERALL IMPRESSION: A slightly subjective but mostly objective overall assessment of the elegance, scale, and sophistication of design presented. This includes consistency of code style and commenting, quality of tests, parsimony and excellent in code.							/ 5

Flowchart PDF	Excellent 4 points	Satisfactory 3 points	Unsatisfactory 2 points	Poor 1 point	Not submitted 0 points	Criterion Score
Flowchart is included as a single page PDF which might be enormous. Is it easy to read? Is it well organized? Does it correctly represent what I see in the code?						/ 4

Documentation	Complete 3 points	Some omissions or errors 2 points	Incomplete 1 point	Not submitted 0 points	Criterion Score
README.md contains required information in tabular form.					/ 3

Total	/ 52

Overall Score

Excellent 47 points minimum	Very good 42 points minimum	Satisfactory 34 points minimum	Pass 26 points minimum	Unsatisfactory 0 points minimum

Close