# Assignment 2 - Part 2 - COMP2152

## Part 2 - Explain your Code (50%)

Write the answers in the doc provided and save and submit Part 2 as a PDF. You do not have to write or submit any new code for this section. I want you to understand how you could work on a piece of code that already exists (as is the case when working with Open Source code), and how to improve it. You can **type** in your answers *or* complete it **by** **hand** (handwriting MUST be legible) and then scan your submission.

1. How have we used classes for our project to reuse code?  
     
   I have defined common properties and methods for the Hero and Monster characters in our project using classes. For instance, both classes inherit from the Character class and share the combat\_strength, health\_points, and attack functions. This eliminates code duplication and facilitates code maintenance by enabling us to write common functionality only once and reuse it for other characters.

1. Provide 1 line of code, as one of many examples, where code is shared between the monster class and the hero class?

monster.take\_damage(self.combat\_strength)  
Regardless of whether the character is the hero or the monster, this line demonstrates shared functionality where both classes use the take\_damage() method to inflict damage.

1. What is the benefit of using complex getters and setters?

Complex getters and setters have the advantage of giving us control over how properties are accessed and changed. For instance, we can enforce criteria like making sure health\_points is never negative or combat\_strength is always positive. This enhances data integrity and avoids invalid states by adding a layer of validation and error-checking to our software.

1. If we didn’t use try-except blocks, what would be the problem?

If the user entered erroneous data (such as a non-integer for the number of dream levels), the software would crash without try-except blocks. This would prevent the game from continuing and result in a bad user experience. We can catch mistakes, show the user helpful warnings, and let the game continue to function properly by utilizing try-except blocks.

1. How could we use the name of the **operating system or the version of python** in your game to prevent errors? Choose just 1 of the above.  
     
   By making sure that certain features or libraries that require particular Python versions are only used if the correct version is recognized, i might leverage the Python version to avoid mistakes. For instance, we might check the user's version and advise them to upgrade if a feature requires Python 3.7 or higher, or we could offer an alternate option if the version is incompatible.

if platform.python\_version\_tuple()[0] == '3' and int(platform.python\_version\_tuple()[1]) >= 7:

# Execute feature that requires Python 3.7+

else:

print("Please upgrade to Python 3.7 or higher to use this feature.")

1. What’s another piece of information we could save inside of the save.txt file? (Remember, we load this information every time we start a new game, so that we can keep track of all of the games you have played so far. )  
     
   To monitor the hero's progress over several gaming sessions, we may store their level or experience points in the save.txt file. When the game loads again, this would enable the user to continue from their previous level or progress.
2. **New Feature:**
   1. Think of **1** **new feature** you can add to the game that could use list comprehension and nested conditional statements. For now just write 1 sentence that describes the feature:  
        
      Now add your new feature description here:  
      We could add a feature where the hero can collect items like weapons or health potions during the game. The collected items could be saved in a list, and the player could use them during the game to heal or increase their combat strength.

**Examples:**

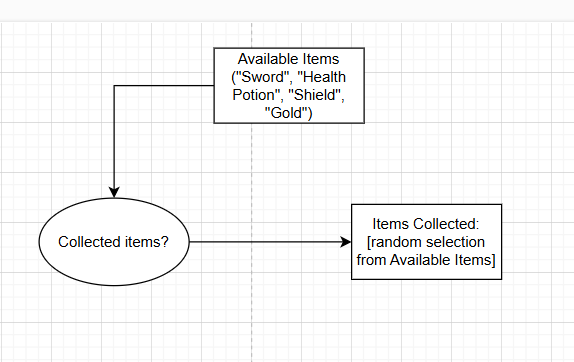
Below are the examples to show you that you can be very creative, and you should have fun with this exercise. You must use an idea that is NOT directly on the list below:  
  
**eg a)** Add another monster so that the hero can fight 2 monsters at once

**eg b)** Create a digital board game, that shows the hero moving around to different towns on a map  
  
**eg c)** Add a dog that runs in front of the hero and discovers features about the world

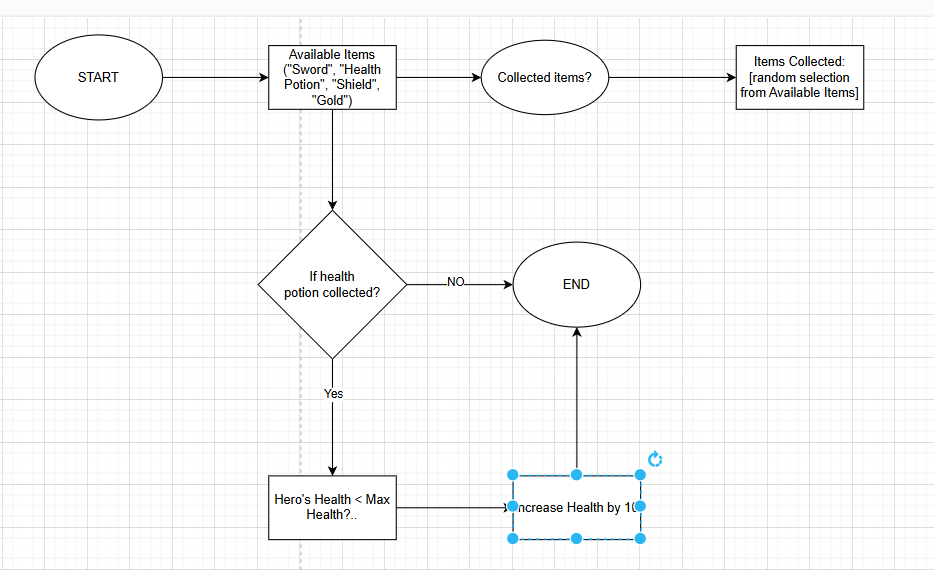
* 1. Give the new feature you created a short 2-3 -word a title:   
       
     Now write your Title here:  
     Item Collection \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Examples:**  
**eg a**) Multiple Monsters  
**eg b)** Roam Towns  
**eg c)** Dog Scout

* 1. Explain how you could implement the idea you chose. You must explain how you would use both of the control structures below. Draw a diagram, map, sketch for each (you can use any software for this, e.g. Draw.io). You don’t have to match the style of diagram I have here, just use a visual to describe your idea. Note, you must have loops and conditional statements diagrammed below as needed:  
     1. **Using a list comprehension loop**  
        We may make a list of the things the hero gathers while traveling by using list comprehension. For instance, the hero may choose up a random object when they come across a treasure chest.



* + 1. **Using nested conditional statements**  
       Nestled conditionals could be used to ascertain an item's effect. The hero should gain more health points, for instance, if they gather a health potion, but only if their health falls below a specific level.



**Example:**  
**eg b)** Roam Towns

* + 1. **Using a list comprehension loop**  
       Every time in the loop, move one square in 1 direction, (N, E, S, W). Have a variable that keeps track of the Hero’s location by saving values of the board. We can have 2 nested for loops and store the map as a 2D array.  
         
       Eg.   
       Hero location is currently at Row 3, Column D.   
       Town 2 location is at Row 4, Column G.   
       Town 1 location is at Row 1, Column A .  
         
       **Diagram:**  
         
        A B C D E F G H I  
       1 (town1\_loc)  
       2  
       3 (player\_loc)  
       4 (town2\_loc)
    2. **Using a nested Conditional Statement**  
       **If** the hero is in Town 2, **then** allow the hero to buy armor but not sell. Otherwise, the hero can sell armor but cannot buy.   
         
       Create an array of armor options available in Town 2. He could also trade some of his loot based on the value of the loot he has.

carbon fiber breastplate,   
steel helmet,   
 bionic gloves

armour\_town\_2

array

can\_sell

true/false

boolean

RAM

if hero is at town 1

hero can’t sell  
 hero can buy

Display armor that the hero can buy and their respective prices

else

hero can sell

Display armor that the hero can buy and their respective prices

hero can’t buy

can\_buy