Introduction to Computer Science Using Python

Homework Assignment 8

Fall, 2018-2019

Add your code into the attached Python file. Change the name of the file hw8_012345678 - instead of 012345678 write your id number. Please also add your name and ID in a comment at the beginning of the submitted file. Make sure you submit the final version of your homework before the due date, not a draft version.

The exercises are personal and should be solved alone. Students are not allowed to solve exercises together, show solutions to other students or read other students' solutions.

Exercises can be discussed without sharing code.

In this homework, you will write a simulation program for battles in the series "The Lord of the Rings" in a slightly updated version.

The program will include classes of combat participants, as well as a combat unit. The creatures that are to be realized are creatures from two armies - the good ones and the bad ones.

Write the required classes in the exercise file, according to the requirements detailed below.

Use inheritance to avoid duplication of code as much as possible!

Question 1

Write the *Creature* class, which represents a creature in the imaginary world of Lord of the Rings. Every creature is characterized by the following attributes:

- Race (String).
- Quantity of life (a real number).
- Name of weapon (string).
- Weapon strength (a positive integer).

You can build a new *Creature* object by the constructor:

__init __ (self, race, life, weapon_name, weapon_power)

The constructor must check the integrity of the types given to it, as well as the proper value of the weapons' power. In case one of the inputs is not correct, an error message should be printed.

Note: Although the amount of life is a number that is not necessarily an integer, a user can build a creature with an integer amount of life. For the strength of the weapon the user must enter an integer (the input 10.0 is thus not valid).

Every creature can do the following:

- Report its race: by the *get_race (self)* method that returns the creature's race.
- Report the amount of its life: by the method *get_life (self)* that returns the amount of life of the creature.
- Say whether it is alive: by the method *is_alive (self)* that returns *True* if the creature has a positive amount of life which is enough for giving a hit (**more** than 5.0 units of life, as explained below). Otherwise, the method should return *False*.
- Hit another creature: The hit operation is characterized by the following steps:
  1. The hitting creature checks whether both he and the other creature are alive. If one of the creatures is not alive, the hitting operation stops.
  2. The hitting creature reduces its life amount by 5.0 life units.
  3. The hit power is calculated according to the following formula, using the strength of the weapon and the updated amount of life:

$$hit\_power = weapon\_power + \frac{life}{4}$$

     The intensity of the resulting hit is a real number (not necessarily an integer).

  4. The hitting creature activates the absorption of the other creature, using the force calculated in the section above (see below for details on the absorption operation).
  5. The hit action must be executed by the *hit (self, other)* method, where *other* represents the other creature, which is an object of type *Creature* object (or of any type that extends *Creature*, as defined below). You can assume that the input to the method is correct.
- Absorb a hit with the help of *hit_power* units: in this action the absorbing creature loses *hit_power* life units. You need to update the exact amount of life even if a negative amount is obtained after the hit. The absorption operation should be realized by the *absorb(self, hit_power)* method, where *hit_power* is a real number representing the force of the hit You can assume that the input to the method is correct.

Read the following code that contains examples of execution and make sure you understand the various outputs:

```
>>> c1 = Creature("elf", 50, "M-16", 10)
>>> c2 = Creature("hobbit", 6.0, "Glock", 4)
>>> print(c1.get_race(), c1.get_life(), c1.is_alive())
elf 50.0 True

>>> print(c2.get_race(), c2.get_life(), c2.is_alive())

hobbit 6.0 True

>>> c1.hit(c2)

>>> print(c1.get_race(), c1.get_life(), c1.is_alive())
elf 45.0 True

>>> print(c2.get_race(), c2.get_life(), c2.is_alive())
hobbit -15.25 False

>>> c2.hit(c1)

>>> print(c1.get_race(), c1.get_life(), c1.is_alive())
elf 45.0 True

>>> print(c2.get_race(), c2.get_life(), c2.is_alive())

hobbit -15.25 False
```

Question 2

Write the class *GoodCreature*, which represents a good creature. *GoodCreature* contains all the qualities and actions of *Creature*, with the following changes:

- The *get_race (self)* method returns a string that begins with "good" and continues with the name of the creature. For example, the string could be "good elf".
- The intensity of the hit of a *GoodCreature* is calculated according to the following formula (instead of the formula in question 1) :

$$hit\_power = weapon\_power + \frac{life}{2}$$

Read the following code that contains examples of execution and make sure you understand the various outputs:

```
>>> g1 = GoodCreature("dwarf", 25.5, "knife", 2)
>>> c2 = Creature("hobbit", 60.0, "Glock", 4)
>>> g1.hit(c2)
>>> print(g1.get_race(), g1.get_life(), g1.is_alive())
good dwarf 20.5 True
>>> print(c2.get_race(), c2.get_life(), c2.is_alive())
hobbit 47.75 True
```

Question 3

Write the class *BadCreature*, which represents a bad creature. A *BadCreature* contains all the qualities and actions of a *Creature*, plus the following attribute and actions:

- *BadCreature* has an attribute called *shield_power* (non-negative integer), so the constructor also receives this attribute and verifies the integrity of the additional input (and prints an error message in case of an error). The constructor will have the following form:

  __init__(self, race, life, weapon_name, weapon_power, shield_power)

- During the act of absorption, the force of the shield should be reduced from the force of the hit. In other words, the amount of life is calculated according to the formula:

$$life = life - (hit\_power - shield\_power)$$

As an example, read the following code and make sure you understand the various outputs:

```
>>> c1 = Creature("dwarf", 25.5, "knife", 2)
>>> b2 = BadCreature("Ork", 60.0, "poison", 4, 3)
>>> c1.hit(b2)
>>> print(c1.get_race(), c1.get_life(), c1.is_alive())
dwarf 20.5 True
>>> print(b2.get_race(), b2.get_life(), b2.is_alive())
Ork 55.875 True
```

Question 4

Write the *Battle* class. The class contains:

- The constructor *__init__ (self, good_army, bad_army)* that receives two lists, the first represents the good army and the other the bad army. You can assume that the lists are not empty. The *good_army* list should contain creatures of type *Creature* or *GoodCreature*. The bad_army list should contain creatures of type *Creature* or *BadCreature*. The constructor must check the integrity of the lists and print an error message if they contain an invalid object.
- A method called *dual_fight (self, good_index, bad_index)* which receives the index *good_index* of a creature from the good army, and an index *bad_index* of a creature from the bad army and runs a dual fight between them. A dual fight is a series of rounds, which lasts until one of the creatures is not alive anymore. In each round, the evil creature strikes first and then the good creature strikes. The method must return the number of rounds that took place in the duel. It can be assumed that the input is correct, and that both

contestants are alive at the beginning of the battle. Both may die at the end of the battle (think how it is possible).

Read the following code that contains examples of execution and make sure you understand the various outputs:

```
>>> g1 = Creature("elf", 50, "M-16", 10)
>>> g2 = GoodCreature("dwarf", 25.5, "knife", 2)
>>> b1 = Creature("Ork", 25.5, "knife", 2)
>>> b2 = BadCreature("Ork", 60.0, "poison", 4, 3)
>>> b = Battle([g1, g2], [b1, b2])
>>> print(b.dual_fight(0,0))
```

```
1
```

Question 5

Write the class *TotalBattle* that represents a battle. The class contains, in addition to the fields and methods of the class *Battle*, also has the following method:

A method called *fight (self)* that conducts a general battle between the armies. Each time the method chooses in each army the live contender with the lowest index in the list, and runs a duel between the contenders. The method ends when at least one of the armies does not contain any alive beings**.** The method must return the total number of rounds given in all battles, and print one of three possible results for battle: “Good guys won” or “Evil won” or “Draw”**.**

Read the following code that contains examples of execution and make sure you understand the various outputs:

```
>>> g1 = Creature("elf", 50, "M-16", 10)
>>> g2 = GoodCreature("dwarf", 25.5, "knife", 2)
>>> b1 = Creature("Ork", 25.5, "knife", 2)
>>> b2 = BadCreature("Ork", 60.0, "poison", 4, 3)
>>> tb = TotalBattle([g1, g2], [b1, b2])
>>> print(tb.fight())
Evil won 5
```