# Programming Assignment 2

## Instructions

In this Assignment you will gradually make a quite simple but functional turn-based role playing style fighting game. You will need to make multiple classes for different kinds of creatures. Each of them will have its own fighting strategy and we will use randomness to keeps things interesting. Follow the instructions carefully and be sure that everything works before you move on to the next step. You are not required to develop your project with Jupyter notebook but if you decide to use it use markdown cells to write your report. If you don't use it provide a pdf file with your report and instructions on how to run every testing script for each exercise. Also, record a video that demonstrates your solution. You can use Zoom to record the videos but you are free to use any other tool you are familiar with. The video format should be mp4. **Deadline:** 8 December, 23:59
**Submission:** Brightspace

| | Exercise 1 (3%) | | | Exercise 2 (3%) | | | Exercise 3 (2%) | | Exercise 4 (2%) | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Task 1 | Task 2 | Task 3 | Task 1 | Task 2 | Task 3 | Task 1 | Task 2 | Task 1 | Task 2 |
| max % : | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

## Exercise One - Basic Creature Classes (3%)

### Task 1 (1%): Create a class with the name `Creature` that has the following characteristics:

- Name. Given as argument in the constructor.
- HP (Health Points). Initialised as in Max HP.
- Max HP. Default value is 10 but can be given as argument in the constructor as well.
- Abilities: Attack: 1, Defence: 5, Speed: 5

You are free to decide in which way you will store the abilities (e.g., with. a dictionary).
The class Creature has the following methods (in addition to the constructor and any accessor/mutator methods you will add):

- `check_life()`. When called, it checks if the creature's HP is less than zero and returns the current HP value. If HP is less than zero, it changes it to zero (negative HP makes no sense and creates problems) and prints an appropriate message (e.g., Name fainted...) before returning zero.
- `attack(target)`. When called the creature needs to attack the target creature (other object). Generate a random number in the range [1,20] (called roll) that will determine the chances of a successful attack. If the randomly generated value is lower than the sum of the target's defence and speed values, the attack is not successful. If the attack is successful, the target's HP is reduced by the sum of the creatures attack value (by default is is one) and a randomly generated number between 1 and 4.

  *Example:* If the attack roll is 6 and the target's defence and speed are five and five respectively, the attack will fail because 6 < 5+5. If the attack roll is 12 the attack is successful because 12>10. For the damage, if the random number is 4 then the attack deals 5 damage (the default value for the creature's attack is 1).

- `auto_select(target_list)`. This method is used by creatures to select the target of their attack. When called, it returns a random creature from the list that has HP $> 0$. If there is no such creature it returns `None`.
- `turn(round_num, target_list)` This methods defines the creature's strategy in battle for each turn. On this class, it uses the auto_select method to find a target and then calls the attack method against the target. You can decide what this method returns.

Write a short script that creates two lists of creatures and makes them fight with each other using their turn methods. Run the fight in a for loop for 20 Rounds but make it possible to stop earlier when either team gets defeated.

*Example:* The output of the previous script should be like this:

```
Round 1:
Pippin attacks Frodo.
Attack hits for 2 damage!
Merry attacks Samwise.
Attack hits for 2 damage!
Frodo attacks Merry.
Attack hits for 1 damage!
Samwise attacks Pippin.
Attack missed...
Round 2:
...
Round 5:
Pippin attacks Frodo.
Attack hits for 1 damage!
Frodo fainted.
```

Task 2 (1%): Create a class with the name `Goblin` that inherits the `Creature` class and has the following updated characteristics:

- Max HP is 15
- Abilities: Attack: 3, Defence: 6, Speed: 6

The class Goblin uses the methods it inherits from the Creature class.

Create a class with the name `Orc` that inherits the Creature class and has the following updated characteristics:

- Max HP is 50.
- Abilities: Attack: 5, Defence: 8, Speed: 3

The class Orc also needs to have the following methods along with the ones that inherits from its parent class:

- `heavy_attack(target)`. When called, this method works like the normal attack method with some changes. If the Orcs abilities haven't been modified he gains 5 points in its attack value and loses 3 points from its defence value (Tip: use a flag variable). Then, proceed like using the normal attack from the creature class. Use appropriate messages to indicate the change in stats (eg. Org is in rage).
- `attack(target)`. When called, this method returns the Orc's attack and defence values to their original values. The changes cannot stack. If called multiple consecutive times the method shouldn't make any further changes to the attack and defence values (Tip: use a flag variable). After these changes the orc makes a normal attack against the target. Use appropriate messages to indicate the change in stats (eg. Orc cooled down).

- `turn(round_num, target_list)` . The Orc's battle strategy repeats every 4 rounds and works like this: In Round 1,2,3 it attacks a target. In Round 4 it uses heavy_attack on a target. Then it goes back to the Round 1 actions in a round robin style. Remember to also use the select_target method that inherits from its parent class.You can decide what this method returns.

Write a short script that creates a list with an orc and a list with a goblin and makes them fight each other using their turn methods. Run the fight in a for loop for 20 Rounds but make it possible to stop earlier when one of them gets defeated.

*Example:* The output of the previous script should be like this:

```
Round 1:
Goblin attacks Orc.
Attack hits for 6 damage!
Orc attacks Goblin.
Attack hits for 13 damage!
...
Round 4:
Goblin attacks Orc.
Attack hits for 7 damage!
Orc is in rage.
Orc attacks Goblin.
Attack missed.
Round 5:
Goblin attacks Orc.
Orc cooled down.
Orc attacks Goblin.
Attack hits for 13 damage!
Goblin fainted.
```

Create a class with the name `Warrior` that inherits the `Creature` class and has the following default values:

- Max HP is 50.
- Abilities: Attack: 5, Defense: 10, Speed: 4

The class Warrior also needs to have the following methods along with the ones that inherits from `Creature`:

- `shield_up()`. When called, this method reduces 4 points from Warrior's Attack and adds it to its defence. If the shield is already up and the method is called again, points are not added again (Tip: use a flag variable).
- `shield_down()`. When called, this method returns the Warrior's attack and defence values to their original values. If the shield is already down and the method is called again, points are not reduced again (Tip: use a flag variable).
- `turn(round_num, target_list)` . The Warrior's battle strategy repeats every 4 rounds and works like that: In Round 1 the warrior attacks the target and then uses shield_up. In Round 2 and 3 the Warrior just attacks the target. In round 4 the Warrior uses shield_down and then attacks the target. Then it goes back to the Round 1 actions. Remember to also use the select_target method that inherits from its parent class.You can decide what this method returns.

You are free to use any other variables you think are necessary for the program to work. Don't forget that some methods and variables are inherited from the Creature class and they may need to be updated. Try to avoid writing multiple times the same pieces of code and use methods instead.

Write a short script that creates a creature and a Warrior and makes them fight each other using their turn methods. Run the fight in a for loop for 20 Rounds but make it possible to stop earlier when one of them gets defeated.

*Example:* The output of the previous script should be like this:

```
Round 1:
Gollum attacks Boromir.
Attack missed...
Boromir attacks Gollum.
Attack hits for 6 damage!
Boromir takes a defensive stance.
Round 2:
Gollum attacks Boromir.
Attack missed...
Boromir attacks Gollum.
Attack hits for 1 damage!
Round 3:
Gollum attacks Boromir.
Attack hits for 1 damage!
Boromir's stance returns to normal.
Boromir attacks Gollum.
Attack hits for 5 damage!
Round 4:
Gollum attacks Boromir.
Attack hits for 1 damage!
Boromir attacks Gollum.
Attack hits for 6 damage!
Golum fainted.
```

Task 3 (1%): Create a class with the name `Archer` that inherits the `Creature` class and has the following default values:

- Max HP is 30
- Abilities: Attack: 7, Defense: 9, Speed: 8

The class Archer also needs to have the following methods along with the ones that inherits from its parent class:

- `power_shot(target)`. When called, this method works like the normal attack method with some changes. For the attack roll, generate two random number in the range [1,20] and use the bigger one. If the Archer has more Speed than the target add the difference to the attack roll. Then, if the archer's abilities haven't been modified he gains 3 points in his attack value and loses 3 points from his defence value (Tip: use a flag variable). Then, proceed like using the normal attack from the creature class but the damage is a random number in the range [1,8] + Archers Attack value instead of the normal ones.
- `attack(target)`. When called, this method returns the Archer's attack and defence values to their original values. If called multiple consecutive times the method shouldn't make any further changes to the attack and defence values (Tip: use a flag variable). After these changes the archer makes a normal attack against the target.
- `auto_select(target_list)`. Modify this method so it choses as target the one with the fewest HP.
- `turn(round_num, target_list)`. The Archer's battle strategy repeats every 4 rounds and works like that: In Round 1 the archer attacks a target. In Round 2 and 3 and 4 the Archer uses power_shot on a target. Then it goes back to the Round 1 actions. Remember to also use the select_target method.You can decide what this method returns.

Create a class with the name `Fighter` that inherits the `Creature` class and has the following default values:

- Max HP is 50
- Abilities: Attack: 5, Defense: 8, Speed: 5

The class Fighter also needs to have the following methods along with the ones that inherits from its parent class:

- `auto_select(target_list)`. Modify this method so it choses as target the one with the most HP.
- `turn(round_num, target_list)`. The Fighter makes a total of 3 attacks each turn. The last 2 are made with a -3 penalty on the attack ability. As long as the target isn't defeated it remains the target of the second or third attack. If it gets defeated though before the Fighter finishes all attacks, select the next target with the most HP. Remember to also use the select_target method.You can decide what this method returns.

Write a short script that creates an Archer and a Fighter and makes them fight each other using their turn methods. Run the fight in a for loop for 20 Rounds but make it possible to stop earlier when one of them gets defeated.

*Example:* The output of the previous script should be like this:

```
Round 1:
Legolas's attack rises.
Legolas's defence reduced.
Legolas shoots Aragorn...
Power shot hits for 17 damage!
Aragorn attacks Legolas.
Attack missed...
Aragorn's unleashes a flurry of strikes.
Aragorn attacks Legolas.
Attack hits for 6 damage!
Aragorn attacks Legolas.
Attack hits for 6 damage!
...
Round 4:
Legolas abilities return to normal.
Legolas attacks Aragorn.
Attack hits for 10 damage!
Aragorn fainted.
```

You are free to use any other variables you think are necessary for the program to work. Don't forget that some methods and variables are inherited from the Creature class and they may need to be updated. Try to avoid writing multiple times the same pieces of code and use methods instead.

## Exercise Two - Enemy Classes (3%)

**Task 1 (1%):** Create a class with the name `OrcGeneral` that inherits the Orc and Warrior classes and has the following default values:

- Max HP is 80
- Abilities. Same as the Orc parent class.

The class OrcGeneral also needs to have the following method along with the ones that inherits from its parent classes:

- `turn(round_num, target_list)` . The Orc General's battle strategy repeats every 4 rounds and works like that:
  - In Round 1, it attacks the target and then uses shield_up. In Round 2, the OrcGeneral just attacks the target.
  - In Round 3, the OrcGeneral uses shield_down and then then attacks the target.
  - In Round 4, the OrcGeneral uses heavy_attack on the target.

  Then it goes back to the Round 1 actions. Remember to also use the select_target method that inherits from its parent class.You can decide what this method returns.

**Task 2 (1%):** Create a class with the name `GoblinKing` that inherits the Goblin and the Archer classes and has the following default values:

- Max HP is 50
- Abilities. Same as the Goblin parent class.

`turn(round_num, target_list)` for the GoblinKing class should function in the same way as the turn method of the Archer class.

Write a short script that creates an OrcGeneral and a GoblinKing and makes them fight each other using their turn methods. Run the fight in a for loop for 20 Rounds but make it possible to stop earlier when one of them gets defeated.

**Task 3 (1%):** Create a class with the name `Boss` that inherits the Orc class and has the following default values:

- Max HP is 200
- Abilities: Attack: 5, Defense: 8, Speed: 5

The class Boss also needs to have the following methods along with the ones that inherits from its parent class:

- `auto_select(target_list, mode)`. Modify this method so it choses a target based on 3 different modes. 'Strong' is for the strongest in the list, 'Weak' for the weakest in the list, and 'Random' for randomly between strongest and weakest. Weakest and strongest is defined by HP.
- `turn(round_num, target_list)` . The Boss battle strategy is a mix between the Orc and the Fighter. In round 1 it makes three attacks just like the Fighter but the first target is selected using the 'Weak' mode and if itfells its target the rest use the 'Random' mode. For turns 2,3,4 it uses heavy attack using the 'Strong' mode. Remember to also use the select_target method using the appropriate mode.You can decide what this method returns.

Write a short script that creates a Boss and an OrcGeneral and makes them fight each other using their turn methods. Run the fight in a for loop for 20 Rounds but make it possible to stop earlier when one of them gets defeated.

## Exercise Three - The Wizard (2%)

**Task 1 (1%):** Create a class with the name `Wizard` that inherits the Creature class and has the following characteristics (instance variables):

- Name. Given as argument in object creation.
- HP (Health Points). Initialised as in Max HP.
- Max HP. Default value is 20 but can be given as argument in constructor as well.
- Abilities: Attack: 3, Defense: 5, Speed: 5, Arcana: 10
- Mana: 100

The Mana variable always need to be in the range [0,100]. If an effect causes the mana to drop below zero, the effect needs to fail. If an effect causes the mana to exceed 100, it becomes 100 the remainder isn't added.
The class Wizard also needs to have the following methods along with the ones that inherits from its parent class:

- `attack(target)`. When called, the wizard attacks the target and then regains 20 Mana points.
- `recharge()`. When called, the wizard regains 30 Mana points.
- `fire_bolt(target)`. Works like the normal attack but adds half the Arcana value to the attack roll (rounded down, do not use floats). It deals damage in range [1, Arcana value]. If the attack hits the target, the wizard gains 10 Mana points.
- `heal(target)`. The wizard spends 20 Mana Points to cause the target to gain some of the HP they lost. The amount healed is a random number from [0,8] increased by half the wizard's Arcana value (rounded down, do not use floats). If the Mana Points are not enough the, method returns without healing anyone.
- `mass_heal(allies)`. The wizard spends 30 Mana Points to cause himself and the allies to regain some of the HP they lost. The amount healed is a random number from [0,10] increased by the wizard's Arcana value. If the Mana Points are not enough the method returns without healing anyone.
- `fire_storm(enemies)`. The wizard spends 50 Mana Points to damage all enemies. If the Mana Points are not enough, the method returns without damaging anyone. If there are enough mana points, the method uses a random number in the range [1,20] and add its speed. If the result is greater or equal to the Wizard's Arcana value, they take half damage and if its not, they take the full damage. The full amount of damage that a target can receive is a random number in the range [5,20] increased by the Wizard's Arcana value.
- `select_target(target_list)`. This method is used by the player on their turn to select the target of their action. Display all the targets in the following format: Index, Name, HP/Max_HP and then take input from the user. The user choses the target by entering a number. The methods returns the target object of the user's choice. Note: If the input given is not valid, ask for input again and again until a correct one is given.

  *Example:* Example output:

  ```
  Select target:
  1: Aragorn, HP: 41/50
  2: Legolas, HP: 30/30
  3: Gandalf, HP: 13/20
  Enter choice:1
  ...
  ```

So far, nothing caused the creatures to regain HP. Now with the wizard class this is possible. Be careful to always check the targets Max HP value as the current HP must never exceed the targets Max HP value.
You are free to use any extra variables and methods you think are necessary for the program to work. Don't forget that some methods and variables are inherited from the Creature class and they may need to be updated. Try to avoid writing multiple times the same pieces of code and use methods instead.

Task 2 (1%): Write a short script that creates a Wizard and a list of two other targets. Use each method on them to check that everything is working properly.

*Example:* Example output:

Using attack:
```
Mana: +20!
Gandalf attacks Goblin King.
Attack missed...
```
Using recharge:
```
Gandalf channels magical energy...
Mana: +30!
```
Using fire bolt:
```
Gandalf fires a fire bolt at Saruman...
Fire bolt hits for 5 fire damage!
Mana is full
```
Using heal:
```
Mana: -20
Gandalf heals Aragorn for 8 HP!
```
Using mass heal:
```
Mana: -30
Gandalf heals Aragorn for 10 HP!
Gandalf heals Legolas for 8 HP!
Gandalf heals Gandalf for 6 HP!
```
Using fire storm:
```
Mana: -50
Fire Storm deals 25 fire damage to Nazgul!
Fire Storm deals 23 fire damage to Goblin King!
```

## Exercise Four - Battle in the Middle Earth (2%)

In this exercise you will need to make the class `Battle` that will simulate the fight between heroes and monsters, with the player taking the role of the wizard. The fight is conducted in rounds. In each round, each creature must take a turn. The order is based on their Speed, from the one with the highest first to the ones with the lowest. When the turn of the Player comes, display messages that instruct what actions are available to the player. The battle should end either when all enemies are defeated or when allies are defeated or when the player gets defeated. Remember that a creature that gets defeated in battle doesn't get a turn in subsequent rounds.

WARNING: Avoid using while loops in this exercise and try to use for loops instead. Be extremely careful you don't create an infinite loop accidentally. The output produced by your program in the jupyter notebook will cause the file to expand till it uses a huge part of your disk space. So, before using Jupyter to conduct the final testing of your code before submitting it to Brightspace, you are advised to test your code in a .py file using the IDE of your choice and run it in the terminal.

Task 1 (1%): Write the class constructor that does the following:

- Creates the list of enemies. Include 1 GoblinKing, 1 OrcGeneral, at least 1 Goblin and at least 1 Orc.
- Creates the list of allies. Include at least 1 Fighter, 1 Archer, 1 Warrior and 1 Creature.
- Create the boss character using the Boss class
- Initialises the player's character using the Wizard class.

Write the method `start()` that simulates the battle but don't add the player yet, neither any action selection. It needs to do the following:

- Define the order in which each creature takes its turn every round based on their Speed value. For example if the fastest creature is the Archer and the slowest is the Orc, every round the Archer will take their turn first and the Orc last.
- Write the loop that executes each round. Display appropriate messages to indicate when a round starts and when it ends.
- When the enemies are about to lose add the boss to the fight. You can decide what is the condition that triggers the boss appearance (e.g only one enemy remains standing). The boss has a lot of HP and its strategy is cunning so it might be too hard for the heroes to win if they have to fight it with more enemies.
- Each creature calls its `turn` method on their turn. If a creature has HP <=0 it does't take a turn.
- Allied and Enemy creatures have different winning and losing conditions so check appropriately if they are fulfilled. The game should terminate either if all enemies are defeated, if all allies are defeated or when the player dies.

*Example:* Example output:

```
THE BATTLE BEGINS
=======================================================
Round 1.
=======================================================
Legolas's attack raised.
Legolas's defense reduced.
Legolas shoots Goblin 2...
Power shot hits for 17 damage!
Goblin 2 fainted.
=======================================================
Balrog attacks Aragorn.
Attack hits for 12 damage!
=======================================================
...
=======================================================
End of round 1.
=======================================================
Round 2.
=======================================================
```

Task 2 (1%): Add the player into the turn. Write the method `player_turn()` that will be called instead of the `turn` method you use for other creatures. It need to do the following:

- Display a User Interface with prints. The information we need as the player are: The Character's name, HP and Max HP values, Mana Points and Max Mana Points, The allies' Names, their HP and Max HP values.

- Use prints to display the actions and spells available to the player and the input they need to type to use them.
- Ask for input by the player to select their move. Assign a keyboard character for each action and spell available and execute the appropriate Wizard method that executes it. You should check that the input given is correct and ask again if the input given is not correct.
- Include a Quit option in the user's input to end the game earlier.

Tip: Try to check both capitalised and lower-cased inputs for better User Experience.

*Example:* Example output:

```
=======================================================
Player: Gandalf HP:13/20 Mana: 100/100
Allies:
 Aragorn HP:41/50
 Legolas HP:30/30
=======================================================
Actions. F: Attack R: Recharge Mana
Spells. 1: Heal 2: Firebolt 3: Mass Heal 4: Fire Storm
To Quit game type: Quit
=======================================================
Enter action: 1
Select target:
1: Aragorn, HP: 41/50
2: Legolas, HP: 30/30
3: Gandalf, HP: 13/20
Enter choice:1
Mana: -20
Gandalf heals Aragorn for 9 HP!
```