

Mini-projet : Batailles



1. Description globale

Le but de ce projet est de développer un système de bataille automatique (pour un jeu de type *auto battler*, un peu comme [super auto pets](#) mais en beaucoup moins sophistiqué et sans la partie jeu de construction de l'armée étape par étape) en utilisant les classes en Python.

À l'aide d'instructions de départ, nous commencerons par de simples duels, avec des classes de combattants de base.

Puis nous formerons des armées qui se battront l'une contre l'autre.

Nous ajouterons ensuite quelques classes de combattants supplémentaires, certaines avec des capacités spéciales.

Enfin, nous donnerons la possibilité aux unités d'avoir des armes (ou objets) supplémentaires.

Votre but, une fois les instructions de départ réalisées, sera de développer, en groupe et selon vos envies, ce système de bataille, en ajoutant des classes de combattants, des objets, chacun avec éventuellement des nouvelles capacités.

2. Instructions de départ

Nous allons introduire petit à petit les éléments initiaux du mini-projet :

a. Premiers combattants et duels

1. Créer une classe **Guerrier** (combattant de base, dont les autres seront dérivés).

Un objet de cette classe doit avoir 2 attributs dont les valeurs initiales sont :

- **pv** : 50 (points de vie) ;
- **atq** : 5 (points d'attaque).

Définir également dans cette classe la méthode **est_vivant** qui retourne **True** si le nombre de points de vie est strictement positif, **False** sinon.

2. Créer une classe **Chevalier** qui hérite de la classe **Guerrier** (voir [cette page sur le sujet de l'héritage en Python](#)).

La seule différence avec la classe **Guerrier**, c'est qu'un **Chevalier** a une attaque (**atq**) qui vaut 7.

3. Créer une fonction **duel** qui prend en argument deux combattants (deux éléments de la classe **Guerrier** ou qui hérite de cette classe) et qui effectue le combat selon le principe suivant :

À chaque tour, le premier combattant frappe le second et le second perd autant de points de vie que l'attaque du premier. Après cela, s'il est toujours en vie, le second attaque le premier selon le même principe, et ainsi de suite, jusqu'à ce que l'un des deux meurt.

Si le combat est gagné par le premier combattant, la fonction **duel** retourne **True**, sinon elle retourne **False**.

Exemple (tests à passer)

```
chuck = Guerrier()
bruce = Guerrier()
carl = Chevalier()
dave = Guerrier()

assert duel(chuck, bruce)
assert not duel(dave, carl)

assert chuck.est_vivant()
assert not bruce.est_vivant()
assert carl.est_vivant()
assert not dave.est_vivant()
```

b. Armées et batailles

1. Ajouter une classe **Armee**. À l'initialisation, définir l'attribut **combattants** comme étant une liste vide. Il s'agit de la liste des combattants.
2. Dans la classe **Armee**, définir une méthode **ajoute** qui prend en arguments une classe de combattant et un entier et ajoute à la liste des combattants de l'armée le nombre de combattants souhaité de la classe donnée.

Les premiers combattants ajoutés sont en tête de liste (indice 0 pour le premier), les suivants s'ajoutent après dans la liste. Autrement dit, le premier combattant de la liste est en tête de l'armée et sera le premier à combattre. La liste des combattants peut être vue comme une file (FIFO).

3. Définir une fonction **bataille** qui prend en argument deux armées et les fera se combattre l'une contre l'autre jusqu'à ce que l'une des deux soit anéantie.

Le principe des batailles est le suivant :

Les deux premiers combattants des deux armées font un duel. Une fois que l'un des deux meurt, il est remplacé par le suivant de son armée qui combat le survivant de l'armée opposée, ainsi de suite jusqu'à ce que l'une des armées n'ait plus de combattant vivant.

Si la première armée gagne, la fonction **bataille** retourne **True**, sinon elle retourne **False**.

Exemple (tests à passer)

```
armee1 = Armee()
armee1.ajoute(Chevalier, 3)

armee2 = Armee()
armee2.ajoute(Guerrier, 3)

armee3 = Armee()
armee3.ajoute(Guerrier, 20)
armee3.ajoute(Chevalier, 5)

armee4 = Armee()
armee4.ajoute(Guerrier, 30)

assert bataille(armee1, armee2)
assert not bataille(armee3, armee4)
```

Il peut être conseillé de supprimer de la liste de chaque armée tout combattant qui vient de mourir. Ainsi, c'est toujours le combattant d'indice 0 qui combat.

c. D'autres unités

1. (a) Ajouter à la classe **Guerrier** un attribut **defense** valant 0.
- (b) Définir ensuite une classe **Defenseur**, héritée de la classe **Guerrier**.
Ses valeurs d'attributs spécifiques sont :

- pv : 60
- atq : 3
- defense : 2

Pour les duels, il faudra maintenant prendre en compte cet attribut de la manière suivante :

Quand quelqu'un attaque un défenseur, ce dernier perd des points de vie selon la formule (attaque de l'adversaire - défense personnelle) si cette valeur est positive, sinon il ne perd rien.

Exemple (tests à passer)

```
armee1 = Armee()
armee1.ajoute(Defenseur, 1)

armee2 = Armee()
armee2.ajoute(Guerrier, 2)

armee3 = Armee()
armee3.ajoute(Guerrier, 1)
armee3.ajoute(Defenseur, 1)

armee4 = Armee()
armee4.ajoute(Guerrier, 2)

assert not bataille(armee1, armee2)
assert bataille(armee3, armee4)
```

2. (a) Ajouter dans la classe **Guerrier** un attribut **max_pv** copiant la valeur de l'attribut **pv** initial.

On pourra, pour minimiser les changements à faire, considérer que la valeur de l'attribut **pv** est donnée en argument lors de l'initialisation d'une unité, et qu'elle vaut 50 par défaut (celle d'un Guerrier de base). Pour les unités ayant un nombre de points de vie initial différent, on initialisera la classe avec la valeur qui leur correspond. Comme l'attribut **max_pv** copiera la valeur de l'attribut **pv**, elle ne sera pas à redéfinir dans les autres classes.

Voir par exemple [cette page](#) pour comprendre le fonctionnement des arguments par mot-clé avec valeurs par défaut.

Aucun combattant ne peut avoir un nombre de points de vie dépassant son maximum qui, sauf changement ultérieur, est toujours son nombre de points de vie initial.

- (b) Définir dans la classe **Guerrier** le nouvel attribut **vampirisme** dont la valeur initiale est 0. Il s'agit du pourcentage de points de vie que le combattant gagne au maximum sur le nombre points de vies qu'il a fait perdre à son adversaire (sans dépasser son nombre de points de vie maximum). Il faudra arrondir à l'entier inférieur dans le cas où la valeur n'est pas entière.
- (c) Définir alors une classe **Vampire** qui a une valeur 50 de **vampirisme**.

Ainsi par exemple, si un vampire fait perdre 4 points de vie à son adversaire (après diminution éventuelle de la défense de celui-ci), alors le vampire peut gagner jusqu'à 50% de ces 4 points, soit 2 points de vie.

Il faudra bien entendu mettre à jour la fonction `duel` afin de prendre en compte cet attribut.

Valeurs d'attributs spécifiques du vampire :

- pv : 40
- atq : 4
- vampirisme : 50

Exemple (tests à passer)

```
armee1 = Armee()
armee1.ajoute(Defenseur, 2)
armee1.ajoute(Vampire, 2)
armee1.ajoute(Guerrier, 1)

armee2 = Armee()
armee2.ajoute(Guerrier, 2)
armee2.ajoute(Defenseur, 2)
armee2.ajoute(Vampire, 3)

armee3 = Armee()
armee3.ajoute(Guerrier, 1)
armee3.ajoute(Defenseur, 4)

armee4 = Armee()
armee4.ajoute(Vampire, 3)
armee4.ajoute(Guerrier, 2)

assert not bataille(armee1, armee2)
assert bataille(armee3, armee4)
```

3. (a) Ajouter à la classe `Guerrier` un attribut `perce` qui vaut initialement 0.

Certains combattants peuvent blesser deux adversaires à la fois. Le deuxième adversaire, qui suit l'adversaire principal dans la file de l'armée adverse, est blessé à hauteur de la valeur indiquée par l'attribut `perce` (diminuée de l'éventuelle défense de ce deuxième adversaire).

- (b) Créer une classe `Lancier` hérité de `Guerrier`.

Valeurs d'attributs spécifiques du lancier :

- pv : 50
- atq : 6
- perce : 3

Il faudra bien entendu mettre à jour la fonction `duel`. Elle devra pouvoir accepter deux arguments supplémentaires (par défaut `None`), à savoir les armées des combattants, afin de pouvoir prendre en compte les combattants en deuxième position dans la file de chacune, pour que dans le cas où un des combattants ait un attribut `perce` non nul, les dégâts puissent être infligés en conséquence.

Exemple (tests à passer)

```
armee1 = Armee()
armee1.ajoute(Defenseur, 2)
```

```

armee1.ajoute(Vampire, 2)
armee1.ajoute(Lancier, 4)
armee1.ajoute(Guerrier, 1)

armee2 = Armee()
armee2.ajoute(Guerrier, 2)
armee2.ajoute(Lancier, 2)
armee2.ajoute(Defenseur, 2)
armee2.ajoute(Vampire, 3)

Armee3 = Armee()
Armee3.ajoute(Guerrier, 1)
Armee3.ajoute(Lancier, 1)
Armee3.ajoute(Defenseur, 2)

Armee4 = Armee()
Armee4.ajoute(Vampire, 3)
Armee4.ajoute(Guerrier, 1)
Armee4.ajoute(Lancier, 2)

assert bataille(armee1, armee2)
assert not bataille(Armee3, Armee4)

```

4. (a) Ajouter à la classe **Guerrier** un attribut **soin** qui vaut initialement 0.

Le soin est le nombre de points de vie que peut ajouter le guerrier à l'allié qui le précède dans la file de l'armée (sans dépasser le nombre de points de vie maximum) uniquement lorsque celui-ci attaque un adversaire.

- (b) Créer une classe **Soigneur** hérité de **Guerrier**.

Valeurs d'attributs spécifiques du soigneur :

- pv : 60
- atq : 0
- soin : 2

Il faudra bien entendu mettre à jour la fonction `duel`, dans le cas où un soigneur se trouverait tout de suite après le combattant qui attaque.

Exemple (tests à passer)

```

armee1 = Armee()
armee1.ajoute(Defenseur, 2)
armee1.ajoute(Soigneur, 1)
armee1.ajoute(Vampire, 2)
armee1.ajoute(Lancier, 2)
armee1.ajoute(Soigneur, 1)
armee1.ajoute(Guerrier, 1)

armee2 = Armee()
armee2.ajoute(Guerrier, 2)
armee2.ajoute(Lancier, 4)
armee2.ajoute(Soigneur, 1)
armee2.ajoute(Defenseur, 2)

```

```

armee2.ajoute(Vampire, 3)
armee2.ajoute(Soigneur, 1)

armee3 = Armee()
armee3.ajoute(Guerrier, 1)
armee3.ajoute(Lancier, 1)
armee3.ajoute(Soigneur, 1)
armee3.ajoute(Defenseur, 2)

armee4 = Armee()
armee4.ajoute(Vampire, 3)
armee4.ajoute(Guerrier, 1)
armee4.ajoute(Soigneur, 1)
armee4.ajoute(Lancier, 2)

assert not bataille(armee1, armee2)
assert bataille(armee3, armee4)

```

d. Objets

1. Créer une classe `Objet` dont la fonction prendra des arguments par mots clé avec une valeur par défaut égale à 0. Les arguments sont identiques aux caractéristiques (valeurs d'attributs) des combattants :

pv atq defense vampirisme perce soin

Les valeurs données pourront être positives, mais également négatives.

Ces valeurs sont enregistrées à l'initialisation dans des attributs du même nom.

2. Créer dans la classe `Guerrier` une méthode `equipe` qui permet de modifier les caractéristiques du combattant en respectant cependant les règles suivantes :
 - Sauf cas indiqué plus bas, les valeurs des caractéristiques de l'objet sont ajoutées à celles du combattant (si une valeur est négative, cela contribue donc à diminuer celle du combattant), sachant qu'une caractéristique ne peut pas devenir négative, elle doit être au minimum de 0.
 - La valeur de `pv` modifie à la fois les points de vie du combattant et son maximum de points de vie (`max_pv`).
 - Les caractéristiques de `vampirisme` et de `soin` ne peuvent pas être modifiées si elles sont nulles dès le départ. Autrement dit, un combattant qui n'en a pas les capacités au départ ne peut pas obtenir celles de vampire ou de soigneur.
3. Créer les classes d'objets standards suivants, dérivés de la classe `Objet` :
 - `Epee` : pv +5, atq +2
 - `Bouclier` : pv +20, atq -1, defense +2
 - `Hache` : pv -15, atq +5, defense -2, vampirisme +10
 - `Katana` : pv -20, atq +6, defense -5, vampirisme +50
 - `Extension` : pv -20, atq +6, defense -5, perce +3
 - `BatonMagique` : pv +30, atq +3, soin +3

Il est possible d'équiper plusieurs fois le même combattant avec plusieurs objets. Noter toutefois que certains diminuent les points de vie, donc que cela peut tuer le combattant.

Exemple (tests à passer)

```
armee1 = Armee()
armee1.ajoute(Chevalier, 4)

armee2 = Armee()
armee2.ajoute(Vampire, 1)

super_arme = Objet(pv=50,atq=2,vampirisme=35,soin=8)

armee2.combattants[0].equipe(super_arme)

assert armee2.combattants[0].soin == 0
assert not bataille(armee1, armee2)
```

3. Extension

Les instructions de départ étant toutes exécutées, c'est ensuite à vous de développer le mini-projet avec vos idées.

Dans les instructions de départ, il a toujours été demandé d'ajouter des attributs et des méthodes dans la classe `Guerrier` qui est la classe de base des combattants, afin de permettre un traitement homogène des duels.

Il vous est cependant possible de définir des attributs ou méthodes spécifiques à d'autres classes de combattants. Bien sûr, il sera alors nécessaire, dans la fonction `duel` en particulier, de prendre en compte ces spécificités en fonction des combattants.

Certaines fonctions Python permettent de vérifier si un objet donné est d'une classe particulière ou a un attribut particulier :

- La fonction `type` appliqué à un objet, dont on prend l'attribut `__name__` qui est de type `str` donne le nom de la classe de l'objet :

```
assert type(Vampire()).__name__ == 'Vampire'
```

- On peut aussi utiliser la fonction `isinstance` pour vérifier la classe d'un objet :

```
assert isinstance(Vampire(),Vampire)
```

- On peut utiliser la fonction `hasattr` pour vérifier la présence d'un attribut :

```
assert hasattr(Vampire(),'vampirisme')
```

Vous donnerez des exemples de tests d'utilisation de vos extensions.

Quelles que soient les extensions effectuées, les principes de base doivent être conservés, les tests initiaux, disponibles dans un fichier fourni, doivent donc toujours passer.

Bien entendu, il est possible d'ajouter toute fonction, attribut, méthode, voire même classes qui vous seront nécessaires.

Le code, en particulier pour les extensions, devra être commenté pour expliquer les intentions et le fonctionnement.

4. Suggestions

On pourra par exemple :

- Ajouter des unités spéciales supplémentaires comme :
 - * Des squelettes ou plus généralement des combattants mort-vivants, desquels les vampires ne peuvent pas voler de points de vie.
 - * Un seigneur de guerre qui réorganise après chaque duel toute son armée, afin d'avoir par exemple, tant qu'il y en a, un lancier en tête ou toute autre unité forte, au combat,

suivi immédiatement si possible par un soigneur, et qui se place lui-même en queue de file (si le seigneur meurt, l'armée ne peut plus être réorganisée).

* Des unités d'artillerie.

* etc.

- Ajouter des objets permettant des actions spéciales, ne faisant pas qu'ajouter des valeurs aux caractéristiques des combattants qui les possèdent.
- Ajouter un mode « verbeux » (géré par exemple par une ou plusieurs variables booléennes) qui, s'il est activé, permet d'avoir le détail des duels et/ou des batailles.