Le paradigme de programmation orienté objet

I . Introduction : avant les objets

Objectif : créer en Python un jeu de rôles évolutif :

Cahier des charges initial :
- chaque joueur crée en début de jeu un Personnage héritant de 100 points de vie. Le Personnage créé peut être de type Guerrier ou Archer. Certains attributs du personnage dépendent de son type (points d'attaque , points de défense, etc..)
- à tour de rôle, chaque joueur fait agir son personnage en saisissant une commande dans la console Python (actions possibles : attaquer un autre personnage, se reposer, etc. (cf. évolution du jeu))

- Lorsqu'un joueur n'a plus de points de vie, il est éliminé

Voici un exemple simple de programme, utilisant des listes :

Jeu_de_roles1_avec_listes.py

Questions :

1. Un point du cahier des charges n'est pas respecté : lequel ?

Rectifiez-le

2. On lance le jeu avec deux joueurs :
a. que saisir dans la console pour réaliser une attaque du joueur 1 par le joueur 2 ?

b. que saisir dans la console pour réaliser une attaque du joueur 2 par le joueur 1 ?

c. réaliser plusieurs attaques jusqu'à ce que l'un des joueurs soit éliminé

3. Ajouter l'action « se reposer » (elle rajoute des points de vie)

4. En cours de partie, un troisième joueur souhaite intégrer le jeu .

   a. Est-ce possible ?

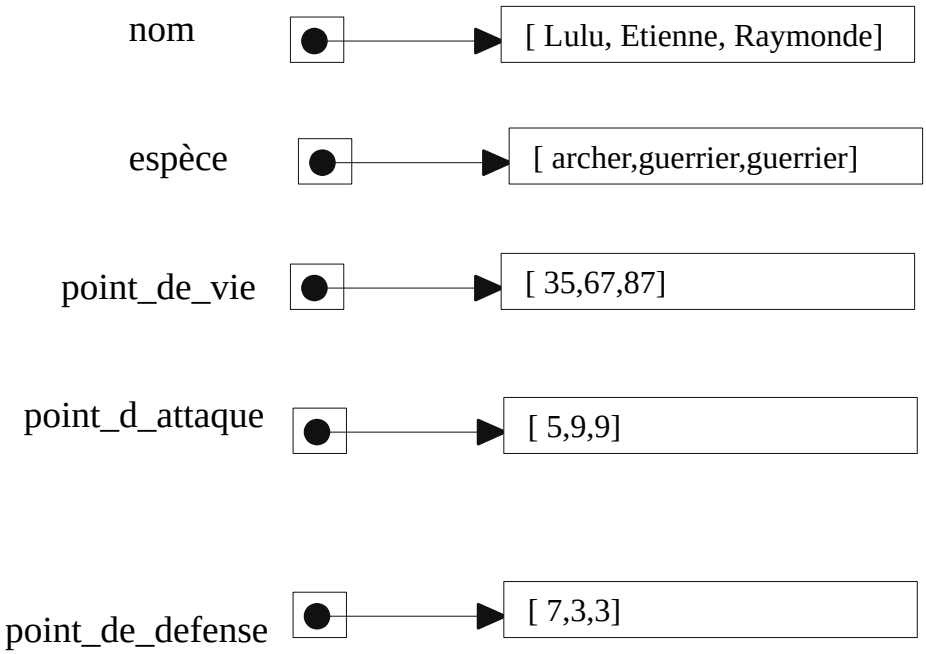
   b. Quelles commandes avez-vous saisi dans la console ?

c. Tester le jeu avec ces trois joueurs .

5. On suppose que l'on veut rajouter une centaine de joueurs en cours de partie . Qu'en pensez-vous ? Quelle solution proposez-vous ?

II. La notion d'objet :

On a vu dans le programme de la partie I que les variables correspondants à chacun des personnages se trouvent disséminées dans plusieurs listes , son nom dans la liste nom, ses points de vie dans la liste points de vie , etc ….



Ainsi, si par exemple, on veut rajouter des joueurs en cours de partie, il va falloir ressaisir à la console plusieurs fois le même code, ou rajouter au programme initial une partie de programme permettant de rajouter des joueurs en cours de partie, ce qui est fastidieux et redondant.
Autre exemple, si un joueur est éliminé et qu'on désire supprimer ses variables de la mémoire, il va falloir « manuellement » supprimer de chaque liste la valeur correspondante, ce qui peut devenir très vite compliqué et source d'erreurs.

Pour éviter ses problèmes, on peut procéder autrement :
on peut d'abord observer que ce jeu est constitué d'entités (ici, les personnages), qui interagissent les uns avec les autres (ici en s'attaquant) , en déclenchant des résultats précis (en l'occurrence perte ou gain de points de vie).
De tels systèmes sont en fait très nombreux dans le monde qui nous entoure, et c'est en observant cela qu' a été créé dans les années 60 un nouveau mode de programmation (on parle de paradigme de programmation) , appelé Programmation Orienté Objet (POO).

Concrètement, un objet est constitué de 3 caractéristiques :

— Un type (ou classe), qui identifie le rôle de l'objet (int, str et list sont des exemples de types ou de classes d'objets) ;

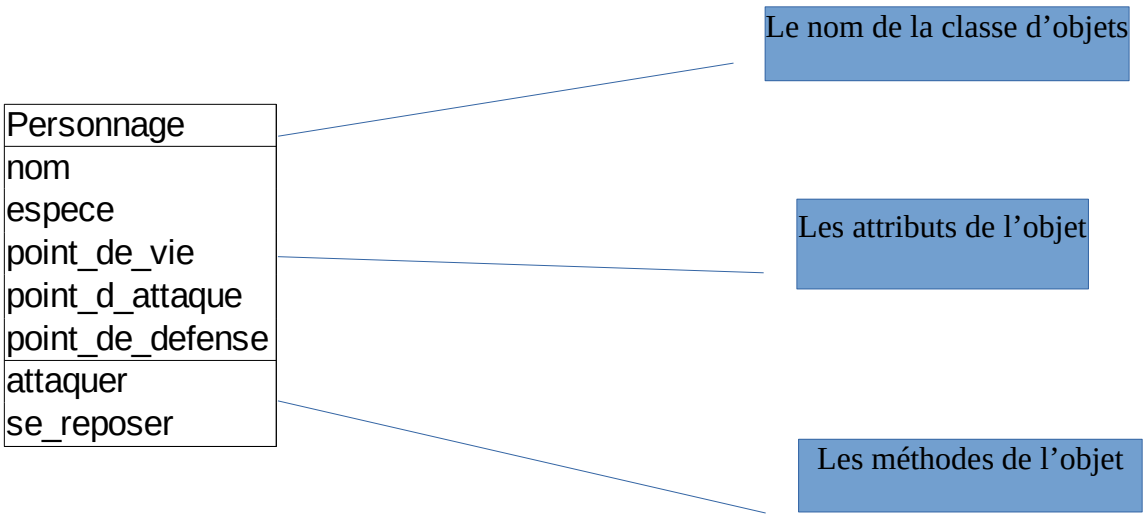
— Des attributs, qui sont les propriétés de l'objet ;

— Des méthodes, les opérations qui s'appliquent sur l'objet.

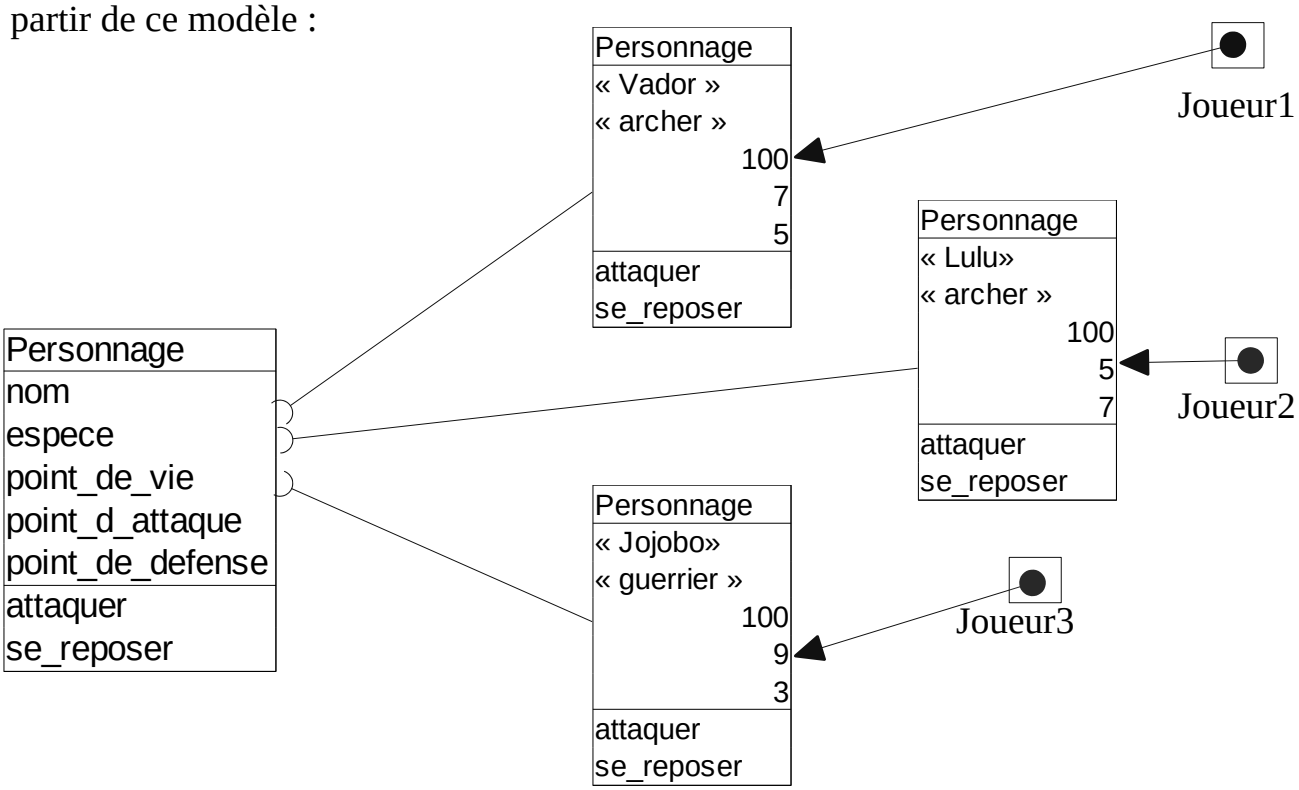
Dans notre cas, ces objets seront les personnages du jeu , et comme ils n'existent pas par défaut en Python, nous allons donc les créer:

- classe : Personnage
- attributs :
    - nom
    - espèce
    - points de vie
    - points d'attaque
    - points de défense
- méthodes :
    - attaquer
    - se reposer

Chaque classe d'objets sera schématisée par un rectangle à trois zones :



Une fois notre classe Personnage définie, on pourra directement créer des personnages à partir de ce modèle :



Pour créer une classe, on utilise le code suivant : [classe_Personnage.py]

Nom de la classe

Docstring de la classe

Attribut commun à tous les objets de cette classe

Le constucteur de la classe, où sont initialisés tous les attributs spécifiques à l'objet instancié

La méthode de la classe Personnage

```
class Personnage:

    """ personnage de type générique """

    points_de_vie=100

    def __init__(self, nom):
        self.nom = nom
        self.points_attaque = 5
        self.points_defense = 3

    def attaquer(self,adversaire):
        adversaire.points_de_vie-=(self.points_attaque - adversaire.points_defense)
        print(adversaire.points_de_vie)
```

- Remarques :
  - Le nom de la classe doit obligatoirement commencer par une majuscule.
  - On expliquera le rôle de self un peu plus loin.

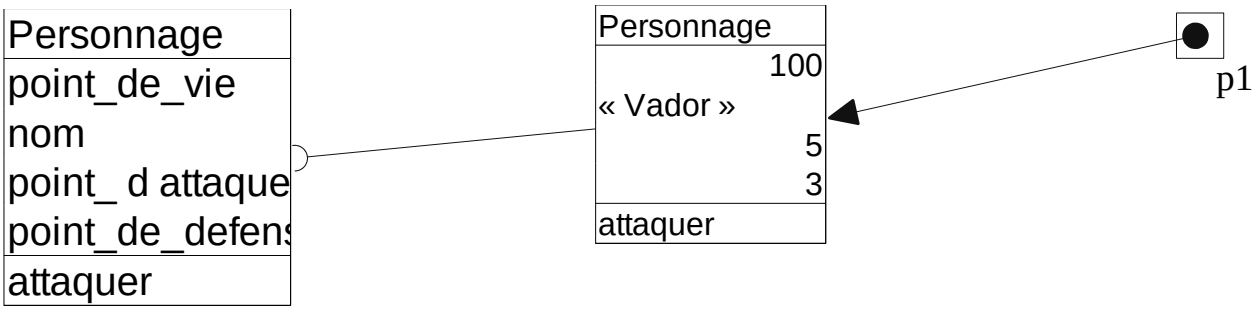
Une fois la classe définie, on crée un objet de type Personnage avec la commande :

p1 = Personnage(« Vador »).

On dit que l'on a *instancié* le personnage p1 . Le verbe instancier est un anglicisme, à partir du mot instance, qui signifie exemple, exemplaire . On a en fait créé en mémoire un exemplaire d'objet de la classe Personnage.

```
>>>
*** Console de processus distant Réinitialisée ***
>>> p1=Personnage("Vador")
>>> p1.nom
'Vador'
>>> p1.points_attaque
5
>>> p1.points_de_vie
100
>>>
```

On a ainsi instancié un objet p1, de type Personnage, dont le nom est Vador.
Il a 100 points de vie et 5 points d'attaque. Il a une méthode, attaquer .



Questions :

1. Créer de la même manière un personnage p2 appelé Lulu .
Combien a-t-il de points de vie ? De points d'attaque , de points de défense ?

2. Utiliser la méthode :

Saisir dans la console : p1.attaquer(p2)
Ce faisant, vous appelez la méthode attaquer de l'objet p1 . Comme le premier paramètre de cette méthode est self, Python comprend que le premier paramètre de la fonction attaquer doit être remplacé par p1, et exécute alors la fonction attaquer (p1,p2)

a. Que retourne la méthode appelée ?

b. Modifier la méthode pour qu'elle retourne plutôt une phrase de type :

« Il reste 96 points de vie au joueur Lulu »

```
def attaquer(self,adversaire):
        adversaire.points_de_vie-=(self.points_attaque - adversaire.points_defense)
        print(adversaire.points_de_vie) « « «  à modifier « « «
```

3. Observez qu'à ce stade de la conception il n'a pas été tenu compte de l' « espèce » de Personnage.

a. Que proposez-vous pour y remédier?

b. Rajouter le paramètre espece dans le constructeur de la classe Personnage et compléter le code pour que le nombre de points d'attaque et de défense initiaux du personnage soient adaptés à son espèce. Tester .

4. Rajouter la méthode se_reposer.