TP3 - LA BATAILLE

Pour jouer à la bataille [1], ou à tout autre jeu de cartes, il est nécessaire de pouvoir comparer des cartes entre elles. On crée deux méthodes, l'une pour "valeur strictement supérieure", l'autre pour "valeur égale".

I Objets Carte

Exercice n° 1 :
Créer la méthode `estEgale` sur le même principe que `estSupérieure`.

Utiliser les "squelettes" de classe qui sont dans le fichier `NSI_CH4_TP2_eleve.py`.

```
class Carte:
    """Carte d'un paquet de cartes, pour jouer à différents jeux.
    On reste dans les paquets 32/52/54 cartes ou tarot

    Attributs :
        - couleur : chaine de caractères, en général coeur/carreau/pique/trèfle,
        mais peu aussi être plus exotique batons/coupes/deniers/épées.
        On peut aussi avoir "atout" ou "joker"
        - hauteur : chaine de caractères, en général de "as" à "roi",
            variantes "un" à "vingt et un" pour les atouts, "aucune" pour les jokers
        - valeur : entier (en général), dépend du jeu.
            Dans un langage fortement typé c'est un flottant (valeurs 0.5 au tarot)

    Méthodes :
        init()
        getCouleur()
        getHauteur()
        getValeur()
        setValeur()
        estSuperieure(autre)  : renvoie un booléen vrai si l'objet
        Carte est de valeur supérieure à celle d'un autre objet Carte
        estEgale(autre)
    """

    def __init__(self,couleur,hauteur,valeur = 0):
        self._couleur = couleur
        self._hauteur = hauteur
        self._valeur = valeur

    # méthodes
    def estSuperieure(self,autre):
        return self._valeur > autre.getValeur()

    def estEgale(self,autre):


    # méthodes getters/setters
```

1. D'après le Cours de Frédéric Mandon

```
    def getCouleur(self):


    def getHauteur(self):
        return self._hauteur

    def getValeur(self):


    def setValeur(self,nouvValeur):


    #def __repr__(self):
        #return f'{self._hauteur} de {self._couleur}, valeur {self._valeur}'

roiCarreau = Carte('carreau','roi',13)
septPique =  Carte('pique','sept',7)
print(roiCarreau.estSuperieure(septPique))
print(septPique.estSuperieure(roiCarreau))
print(roiCarreau)
```

Exercice n° 2 :

- Compléter les méthodes de la classe.
- Que fait la méthode `__repr__` ?

II Objet composé d'objets

Dans un paquet de cartes, il y a plusieurs cartes. On va créer donc la classe `Paquet`.
On remarque que les listes des valeurs et des hauteurs possibles sont définies avant le constructeur. Ces variables sont partagées par toutes les instances de la classe : il n'y a qu'une seule copie de ces variables, créee lors du chargement de la classe.

```
class PaquetCartes:
    """
    Paquet de cartes
    Attributs:
        - nom : nom du paquet, de préférence correspondant au nom du jeu pour
            lequel il va être utilisé
        - paquet : liste des cartes
    """
    _hauteurs = ["as","deux","trois","quatre","cinq","six","sept","huit","neuf","dix",
              "valet","dame","roi"]
    _couleurs = ["coeur","pique","carreau","trèfle"]

    def __init__(self,nom,nbCartes = 32):
        """Constructeur du paquet de cartes"""
        self._nom = nom
        self._nbCartes = nbCartes
        self._paquet = []
        if nbCartes == 32:
```

```
            for i in range(6,len(hauteurs)):
                for j in range(len(self._couleurs)):
                    self._paquet.append(Carte(self.couleurs[j],hauteurs[i],i + 1))
            for j in range(len(self._couleurs)):
                self._paquet.append(Carte(self._couleurs[j],self._hauteurs[0],14))
        else:
            for i in range(1,len(self._hauteurs)):
                for j in range(len(self._couleurs)):
                    self._paquet.append(Carte(self._couleurs[j],self._hauteurs[i],i+1))
            for j in range(len(self._couleurs)):
                self._paquet.append(Carte(self._couleurs[j],self._hauteurs[0],14))

    def getPaquet(self):
        return self._paquet


paquetBataille = PaquetCartes('bataille', 52)
```

Exercice n° 3 :
Testez le code précédent avec 32 cartes, et corrigez les erreurs.

Remarque : le paramètre `nbCartes` est renseigné par défaut à 32, il n'est pas forcément nécessaire de le préciser lors de l'appel du constructeur. S'il est absent, il y aura 32 cartes dans le paquet, sinon il y aura le nombre de cartes précisé lors de l'appel.

III Afficher les cartes

Quand on joue aux cartes, c'est assez important de savoir ce que l'on a en main, ou au moins de savoir quelle est la valeur de la carte jouée ! Affichons donc les 10 premières cartes avec le code suivant.

```
print(paquetBataille)
for i in range(10):
    print(paquetBataille.getPaquet()[i])
```

Exercice n° 4 :

- Suivant que vous avez ou non défini la méthode `__repr__` dans la classe `Carte`, que constatez-vous ?
- Si nécessaire, modifiez le le code des classes `Carte` et `PaquetCartes`, pour afficher les attributs de chaque carte et du paquet (utilisez `__repr__(self)`).
  Vérifiez également que la valeur des as est correcte

IV Avec les objets composés d'objets, les méthodes "enchainées"

On peut écrire des instructions appliquant une méthode sur le résultat d'une autre méthode :

```
paquetBataille.getPaquet()[-3].getValeur()
```

Exercice n° 5 :

Pour pouvoir jouer, il nous manque encore quelques méthodes dans notre classe `PaquetCartes`.
Créer :

- la méthode `melange(self)` qui mélange le paquet comme son nom l'indique.
  Cette méthode renvoie `self`, elle modifie l'attribut `_paquet`.
  On utilisera la méthode `shuffle(tableau_à_mélanger)` de la bibliothèque `random`
- la méthode `distribution(nbJoueurs,nbADistribuer = 0)` qui renvoie une `donne`, une liste de listes de `Cartes`. Il y a autant de listes que de joueurs.
  Si `nbADistribuer = 0`, on distribue tout le paquet de manière équitable. Sinon, on donne le nombre de cartes indiquées à chaque joueur. Le nombre de cartes à distribuer et le nombre de joueurs doivent être compatibles avec la taille du paquet de cartes. S'il reste des cartes après distribution, une dernière liste sera ajoutée avec celles-ci.
- Faire une méthode `distribution(nbJoueurs)` beaucoup plus simple, où le paquet est divisé en deux (`donne` contient alors 2 listes).
  Pour l'instant l'objectif est de jouer à la bataille et rien de plus complexe.

Remarque : l'affichage des donnes de chaque joueur relève plutôt de la classe gérant le jeu. En effet dans certains jeux les cartes sont inconnues du joueur.

V La classe "jeu de la bataille"

A vous de jouer, puisque vous allez créer la classe `JeuBataille`, dont les spécifications sont données ci-dessous. L'ordinateur joue contre lui-même.

Rappel des règles :

- deux joueurs se partagent le paquet
- la donne de chaque joueur est posée face cachée devant lui
- chaque joueur tire en même temps une carte. Le "en même temps" est en fait un tirage successif, d'abord le joueur numéro 1 puis le 2. Ceci pour des raisons d'ordre dans lequel on va ranger les cartes par la suite
- En cas d'inégalité des cartes, le joueur ayant la carte la plus forte l'emporte. Il met les deux cartes sous son paquet, face retournée.
- En cas d'égalité, on itère le processus. Lorsqu'un joueur retourne une carte plus forte que celle de son adversaire, il remporte tout le tas, qu'il retourne et place en dessous de son paquet
- Un joueur a perud lorsqu'il n'a plus de cartes à retourner.
- Il est théoriquement possible d'avoir un match nul

Une notion importante fait son apparition dans ce jeu. Il s'agit de la `__file__` de cartes (et non pile comme on aurait tendance à le dire dans le langage courant).

Les opérations sur les files ont des noms spécifiques :

- `fileVide()` : crée une file vide
- `tete(file)` : renvoie l'élément en tête de la file, la file étant non vide
- `enfiler(element, file)` : insère `element` en fin de file
- `defiler(file)` : supprime l'élément en tête de la file. On peut le renvoyer éventuellement
- `estFileVide(file)` : teste si la file est vide

- On peut éventuellement fixer une taille maximale à une file

Exercice n° 6 :

Compléter la classe `File` en suivant les spécifications ci-dessous, vous l'utiliserez dans "JeuBataille". Quelques tests sont proposés, vous pouvez en rajouter (essayez d'utiliser des `assert`).

```
class File:
    """
    Gère les files FIFO
    Attributs :
        - file : liste d'éléments à priori du même type
        - nb_elements : taille de la file
        - premier : premier élément de la file
        - dernier : dernier élément de la file
    Méthodes :
        - __init__(liste = []) : constructeur, renvoie une file vide si liste
        n'est pas renseigné. Sinon renvoie une file constituée des
        éléments de la liste
        - tete : renvoie l'élément en tête de la file, la file étant non vide
        - enfiler(element) : insère element en fin de file
        - defiler : supprime l'élément en tête de la file.
        On peut le renvoyer éventuellement.
            Si la file est vide, renvoie None
        - estFileVide : teste si la file est vide
        - getters pour nb_elements, premier et dernier
    """
    def __init__(self,liste = None):
        """Constructeur de la file
        Remarque : écrire self._file = liste copie l'adresse de liste dans self._file.
        Ceci peut poserdes problèmes.
        En effet, si par la suite on modifie liste, alors on modifiera aussi self._file
        """
        self._file = []
        if liste == None:

        else:


    def estFileVide(self):


    def enfiler(self,element):


    def defiler(self):


    # getters
    def premier(self):
        return self._premier
    def dernier(self):
        return self._dernier
```

```
    def getNb_elements(self):
        return self._nb_elements

    def printFile(self):
        print("Contenu de la file : ")
        for element in self._file:
            if isinstance(element,Carte):
                print(element)
            else:
                print(element, "n'est pas une carte")

ma_file = File([11,22,33,44,55])
ma_file.printFile()
ma_file.enfiler(66)
print(ma_file.premier(),ma_file.dernier(),ma_file.getNb_elements())
long_ma_file = ma_file.getNb_elements()
for i in range(long_ma_file):
    print(ma_file.defiler())
print(ma_file.defiler())
ma_file2 = File()
print(ma_file2.premier(),ma_file2.dernier(),ma_file2.getNb_elements())
```

Exercice n° 7 :

Compléter la méthode `jouer` de la classe suivante.

```
class JeuBataille:
    """
    Jeu de bataille
    Attributs:
        - nom_joueur1 : chaine
        - nom_joueur2 : chaine
        - paquetBataille : liste des cartes
        - cartes_j1 : pile des cartes du joueur 1
        - cartes_j2 : pile des cartes du joueur 2
        - defausse : pile des cartes de la défausse
        - nb_tours : entier, nombre de tours de jeu
        - nb_batailles : entier, nombre de cas d'égalité lors des tirages simultanés

    Méthodes :
        - __init__
        - jouer : jeu de l'oridnateur contre lui-même. Il est conseillé:
            soit de mettre très peu de cartes (8 au total max)
            soit de préciser un nombre maximal de tours de jeu
            Renvoie :
                match_nul : booléen au nom explicite
                gagnant : chaîne de caractères
                self.nb_batailles
                self.nb_tours
    """

    def __init__(self, nom_joueur1 = 'ordi1', nom_joueur2 = 'ordi2'):
        """Constructeur du jeu"""
```

```
        self._nomjoueur1 = nom_joueur1
        self._nomjoueur2 = nom_joueur2
        self._paquetBataille = PaquetCartes('bataille',6)
        donne = self._paquetBataille.melange().distribution(2)
        self._cartesj1 = File(donne[0])
        self._cartesj2 = File(donne[1])
        self._defausse = File()
        self._nb_tours = 0
        self._nb_batailles = 0
        print("Cartes j1")
        self._cartesj1.printFile()
        print("Cartes j2")
        self._cartesj2.printFile()

    def jouer(self):
        # jouez avec très peu de cartes (4 à 10).
        # Fixez un maximum de nombre de tours de jeu

        return (match_nul,gagnant,self._nb_batailles,self._nb_tours)


baston = JeuBataille()
(mat,gagnant,nb_batailles,tours) = baston.jouer()
if mat:
    print("match nul, ce n'est pas fréquent")
else:
    if gagnant == None:
        print("trop de tours de jeu. Il y a eu ",nb_batailles," batailles")
    else:
        print(gagnant," a gagné en ",tours," tours de jeu, et ",nb_batailles," batailles.")
```

Exercice n° 8 :

Organiser toutes ces classes "proprement" :

- un fichier `Cartes.py` pour les classes `Carte` et `PaquetCartes`,
- un fichier `Files.py` pour la classe `File`,
- un fichier `Bataille.py` pour la classe `JeuBataille` et le `main` de test.