

La bataille

1. Présentation du projet

Le jeu de la bataille en console

1.1. La bataille, les règles du jeu

On distribue les 52 cartes aux joueurs (la bataille se joue généralement à deux) qui les rassemblent face cachée en paquet devant eux.

Chacun tire la carte du dessus de son paquet et la pose face visible sur la table.

Celui qui a la carte la plus forte ramasse les autres cartes.






















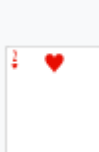
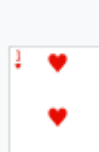





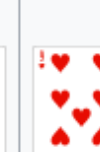











L'as est la plus forte carte, puis roi, dame, valet, 10, etc.

Lorsque deux joueurs posent en même temps deux cartes de même valeur il y a "bataille". Lorsqu'il y a "bataille" les joueurs tirent la carte suivante et la posent, face cachée, sur la carte précédente. Puis, ils tirent une deuxième carte qu'ils posent cette fois-ci face découverte et c'est cette dernière qui départagera les joueurs. Celui qui a la valeur la plus forte, l'emporte.

Le gagnant est celui qui remporte toutes les cartes du paquet.

1.2. La bataille simplifiée

Un paquet de cartes de 40 cartes organisées en deux couleurs : noir et rouge et en quatre enseignes : trèfle, carreau, cœur, pique

Piques										
Carreaux										
Cœurs										
Trèfles										

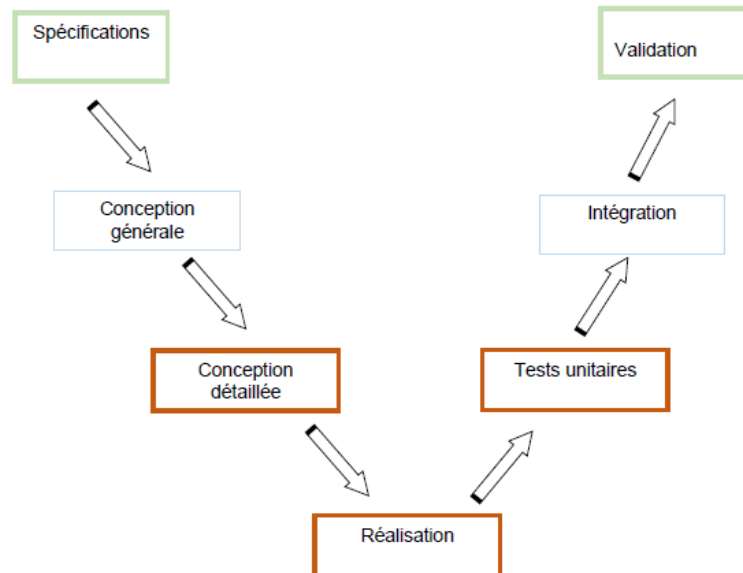
On mélange le paquet et on distribue les 40 cartes aux joueurs (la bataille se joue généralement à deux) qui les rassemblent face cachée en paquet devant eux : une main.

Chacun tire la carte du dessus de son paquet et la pose face visible sur la table. Celui qui a la carte la plus forte ramasse les autres cartes et la place sur un tas. Le 10 est la plus forte carte, puis le 9, le 8, etc. Lorsque

deux joueurs posent en même temps deux cartes de même valeur il y a "bataille". Lorsqu'il y a « bataille », les deux joueurs poseront alors directement la carte suivante face apparente et celui qui disposera de la valeur la plus élevée emportera la manche (et donc les 4 cartes placées)
Lorsqu'un joueur n'a plus de carte en main, le gagnant est celui qui dispose de plus de carte, tas et main cumulés.

2. Conduite du projet

La conduite de projet suivra la description ci-dessous



2.1. Spécifications du jeu

Il s'agit de définir les spécifications du jeu tel que nous souhaitons le réaliser. Le programme réalisé devra simuler 25 parties consécutives d'un jeu de bataille simplifiée et afficher le résultat dans la console.

```

>>> %Run bataille.py

Partie 1 : Le joueur 2 a gagné la partie
Partie 2 : Le joueur 2 a gagné la partie
Partie 3 : Le joueur 2 a gagné la partie
Partie 4 : Le joueur 2 a gagné la partie
Partie 5 : C'est un match nul
Partie 6 : Le joueur 1 a gagné la partie
Partie 7 : Le joueur 2 a gagné la partie
Partie 8 : Le joueur 1 a gagné la partie
Partie 9 : Le joueur 1 a gagné la partie
Partie 10 : Le joueur 2 a gagné la partie
Partie 11 : Le joueur 2 a gagné la partie
Partie 12 : Le joueur 2 a gagné la partie
Partie 13 : Le joueur 2 a gagné la partie
Partie 14 : Le joueur 2 a gagné la partie
Partie 15 : C'est un match nul
Partie 16 : Le joueur 2 a gagné la partie
Partie 17 : Le joueur 2 a gagné la partie
Partie 18 : Le joueur 2 a gagné la partie
Partie 19 : Le joueur 2 a gagné la partie
Partie 20 : Le joueur 2 a gagné la partie
Partie 21 : Le joueur 2 a gagné la partie
Partie 22 : Le joueur 2 a gagné la partie
Partie 23 : C'est un match nul
Partie 24 : Le joueur 2 a gagné la partie
Partie 25 : Le joueur 2 a gagné la partie
  
```

Remarque : Dans la bataille simplifiée le jeu s'arrête lorsqu'un des deux joueurs n'a plus de carte en main. A cet instant on compte l'ensemble des cartes main et tas ... il peut donc y avoir égalité.

2.2. Conception générale

On commence par construire un algorithme général du jeu

```
# créer la structure de donnée : paquet de carte
# créer les structures de données : les mains des joueurs
# Répéter 25 fois :
    # mélanger le paquet de carte
    # distribuer les cartes
    # Jouer une partie c'est faire des plis pendant que la partie n'est pas terminée.
    # attendre fin_des_plis
    # Compter les cartes et désigner le vainqueur
    # afficher la partie et le vainqueur
```

On voit ici apparaître une décomposition fonctionnelle. Chaque fonction peut être écrite par un concepteur différents : c1, c2 ...

Il est nécessaire avant tout que les concepteurs choisissent une structure de donnée pour le paquet et les mains des joueurs.

☞ Choisir une structure de donnée pour le paquet.
C'est un tableau d'entiers de dimension 40

```
# creer le paquet de carte
paquet = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, \
          1, 2, 3, 4, 5, 6, 7, 8, 9, 10, \
          1, 2, 3, 4, 5, 6, 7, 8, 9, 10, \
          1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

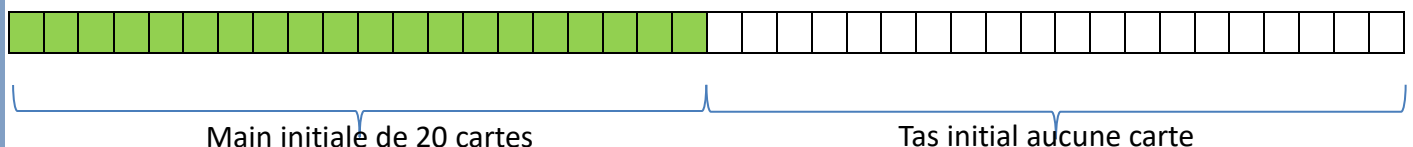
On doit choisir une structure de donnée pour la main du joueur et le tas.

- 2 structures par joueur ?
- 1 structure par joueur ?

☞ Le choix d'une seule structure semble à ce stade plus simple à gérer. Nous faisons ce choix.
C'est un tableau d'entier de dimension 40, un peu particulier. Les 20 premiers éléments constituent la main initiale du joueur, les 20 derniers le tas.

```
# creer les mains des joueurs
joueur_1 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

joueur_2 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```



Main initiale de 20 cartes

Tas initial aucune carte

2.3. Conception détaillée

Il s'agit de donner les signatures des fonctions nécessaires. Il sera peut-être nécessaire d'écrire d'autres fonctions *internes* pour exécuter certaines tâches, rendre le code plus lisible...

```
melanger(tab:list)->list
distribuer(tab:list, tab1:list, tab2:list)->None
fin_plis(j1:list, j2:list) ->None
quel_vainqueur (j1:list, j2:list)->int
afficher(vainqueur:int)->None
```

2.4. Réalisation

Il s'agit maintenant de coder les fonctions envisagées. Pour chaque fonction, le concepteur crée un programme, documente la docString et valide sa fonction avec un jeu de test.

2.4.1. Concepteur c1.py

```
melanger(tab:list)->list
```

☞ La longueur du tableau est disponible à l'aide d'une fonction `len(tab)`

Il faut pouvoir choisir une valeur au hasard, dans le domaine `[0 : len(tab)[` pour désigner la carte à déplacer.

Réflexion :

- Répéter piocher dans le tas qui diminue et créer un tas mélangé.
- Retirer la carte supposerait de décaler puis redimensionner tout le tableau à chaque fois. C'est une stratégie coûteuse.
- Une autre solution peut consister à créer un autre tableau de booléen qui indique si une carte a été tirée ou pas, puis répéter le tirage ...il y a une probabilité non nulle de chercher à tirer toujours la même carte.

Algorithme retenu :

On considère un tableau de `n` éléments indicés de 0 à `n-1`.

Pour mélanger les éléments du tableau

- Tirer un nombre `i` au hasard entre 0 et `n-1` et échanger les éléments qui sont dans les positions `i` et `n-1` ;
- Tirer un nombre `i` au hasard entre 0 et `n-2` et échanger les éléments qui sont dans les positions `i` et `n-2` ;
- ...
- Tirer un nombre `i` au hasard entre 0 et 1 échanger les éléments qui sont dans les positions `i` et 1.

On voit apparaître la nécessité d'échanger deux éléments d'un tableau, ce qui nécessitera l'écriture d'une nouvelle fonction :

```
echanger(tab:list, pos1:int, pos2:int)->None
```

Jeu de test :

```
>>> paquet = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> paquet_m = melanger(paquet)
>>> print(paquet_m)
[1, 2, 9, 4, 6, 5, 7, 10, 3, 8]
```

2.4.2. Concepteur c2.py

```
distribuer(tab:list, tab1:list, tab2:list)->None
```

Les valeurs du tableau passé en argument doivent être alternativement placées dans les tableaux 1 ou 2 passés en argument.

Les tableaux 1 et 2 peuvent être modifiés en place ou renvoyé en retour, ce qui ne semble pas nécessaire.

☞ Le cahier des charges ne précise pas le choix du joueur pour la première carte distribuée. On choisit ici arbitrairement de commencer par le joueur 1.

Jeu de test :

```
>>> tab = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> tab1 = [0 for i in range(10)]
>>> tab2 = [0 for i in range(10)]
>>> distribuer(tab, tab1, tab2)
>>> tab1
[1, 3, 5, 7, 9, 0, 0, 0, 0, 0]
>>> tab2
[2, 4, 6, 8, 10, 0, 0, 0, 0, 0]
```

2.4.3. Concepteur c3.py

```
fin_plis(j1:list, j2:list)->bool
```

☞ Afin de faciliter les tests, on recalculera la position du tas dans cette fonction.

☞ Proposition d'algorithme

```
# Initialiser indice carte, nombre de bataille
# Tant que un joueur a des cartes à jouer
# il n'est pas dans le tas
    Si la ième carte du joueur 1 est supérieure à la ième carte du joueur 2
        # Jouer 1 gagnant
        ramasser(j1, j2, i_carte, nb_bataille)
        nb_bataille ← 0
    SinonSi la ième carte du joueur 1 est inférieure à la ième carte du joueur 2
        # Jouer 2 gagnant
        ramasser(j2, j1, i_carte, nb_bataille)
        nb_bataille ← 0
    else:
        # bataille
        Incréments nb_bataille de 1
        # Jouer carte suivante
# fin des plis
Renvoyer rien
```

On implantera une fonction

```
def ramasser(gagnant:list, perdant:list, tour:int, n:int)->None:
```

Le rôle de la fonction ramasser est de faire passer les cartes de la main du perdant vers le tas du gagnant.

Pour la tester on pourra fabriquer une fonction dédiée

```
def generateur(taille:int):
    '''
    Renvoyer une main et un tas
    Args :
    taille: int le nombre total carte et tas
    Returns :
    main : list une main tirée au hasard et un tas vide
    CU:
    taille est paire
    '''
```

Quelques exemples de ce que l'on souhaite obtenir avec la fonction ramasser.

Exemple 1 :

Donne

Joueur 1 [4, 7, 3, 9, 2, 0, 0, 0, 0, 0]

Joueur 2 [8, 8, 1, 7, 4, 0, 0, 0, 0, 0]

Fin

Joueur 1 [0, 0, 3, 9, 0, 0, 0, 1, 7, 0]

Joueur 2 [8, 8, 0, 0, 4, 4, 7, 0, 0, 2]

Exemple 2 :

Donne

Joueur 1 [4, 9, 4, 7, 8, 0, 0, 0, 0, 0]

Joueur 2 [1, 4, 4, 1, 3, 0, 0, 0, 0, 0]

Fin

Joueur 1 [4, 9, 4, 7, 8, 1, 4, 4, 1, 3]

Joueur 2 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Exemple 3 :

Donne

Joueur 1 [1, 9, 5, 7, 1, 0, 0, 0, 0, 0]

Joueur 2 [2, 4, 7, 7, 1, 0, 0, 0, 0, 0]

Fin

Joueur 1 [0, 9, 0, 7, 1, 0, 4, 0, 0, 0]

Joueur 2 [2, 0, 7, 7, 1, 1, 0, 5, 0, 0]

2.4.4. Concepteur c4.py

```
quel_vainqueur (j1:list, j2:list)->int
```

Cette fonction doit désigner le vainqueur.

Dans cette version simplifiée de la bataille, on peut imaginer une situation d'égalité. On peut choisir par exemple de renvoyer un entier de valeur 0, 1 ou 2 selon le résultat : 0 symbolisant l'égalité, 1 ou 2 désignant le gagnant.

On voit apparaître l'intérêt de créer une fonction de comptage des cartes de chaque joueur :

```
def compter(joueur:list)->int
```

Jeu de test :

```
>>> quel_vainqueur([0, 0, 3, 9, 0, 0, 0, 1, 7, 0], [8, 8, 0, 0, 4, 4, 7, 0, 0, 2])
2
>>> quel_vainqueur([4, 9, 4, 7, 8, 1, 4, 4, 1, 3], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
1
>>> quel_vainqueur([4, 0, 5, 0, 8, 3, 0, 2, 0, 0], [0, 9, 0, 8, 8, 0, 6, 0, 7, 0] )
0
```

3. Intégration

Les travaux des différents concepteurs sont réunis. Le programme bataille.py peut être implanté en important les travaux réalisés.

bataille.py

c1.py

c2.py

c3.py

c4.py

c5.py

4. Validation du projet

Jeu de test permettant la validation du projet :

Le jeu étant mélangé de façon aléatoire on pourra vérifier une partie quelconque, par exemple :

```
Le paquet mélangé
[2, 10, 3, 6, 1, 10, 7, 1, 5, 10, 7, 6, 9, 8, 5, 8, 4, 8, 2, 8, 1, 7, 2, 4, 5, 6, 7, 9, 9, 10, 5, 3, 1, 4, 3, 6, 3, 9, 2, 4]
La main du joueur 1
[2, 3, 1, 7, 5, 7, 9, 5, 4, 2, 1, 2, 5, 7, 9, 5, 1, 3, 3, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
La main du joueur 2
[10, 6, 10, 1, 10, 6, 8, 8, 8, 8, 7, 4, 6, 9, 10, 3, 4, 6, 9, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Après la partie
Les cartes du joueur 1
[0, 0, 0, 7, 0, 7, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 1, 0, 6, 8, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0]
Les cartes du joueur 2
[10, 6, 10, 0, 10, 0, 0, 8, 8, 8, 7, 4, 6, 9, 10, 0, 4, 6, 9, 4, 2, 3, 1, 0, 5, 0, 0, 5, 4, 2, 1, 2, 5, 7, 9, 0, 1, 3, 3, 2]
Le joueur 2 a gagné la partie
```

5. Travail demandé

Chaque groupe devrait réaliser un document présentant :

- les spécifications
- la conception générale,
- la conception détaillée,
- le partage des tâches entre les membres du groupe.

Ce travail a déjà été réalisé dans ce projet. Vous devrez vous en inspirer dans vos prochains projets.

Chaque groupe devra produire un dossier compressé contenant les fichiers du programme :

- Les codes des programmes devront être commentés.
- Les fonctions auront une *docstring* détaillée
- Un jeu de tests sera construit pour chaque fonction

Chaque fichier commencera toujours par les lignes

```
# Author(s) name (Individual, Team or corporation)
# Date
# Title of program/source code
# Code version
# Type (e.g. main program, module, source code)
# Web address or publisher (e.g. program publisher, URL)
```

6. Compétences évaluées

- ✗ Analyser et modéliser un problème
- ✓ Décomposer un problème en sous problèmes
- ✓ Concevoir des solutions algorithmiques
- ✗ Mobiliser les concepts et les technologies
- ✓ Traduire un algorithme dans un langage de programmation
- ✓ Développer des capacités d'abstraction et de généralisation