Projet informatique S3
Les Rois du Ring

1 - Objectif

Le but du projet est de programmer le jeu « Les Rois du Ring » dans lequel 2 à 4 joueurs « virtuels » s'affronteront (l'ordinateur les fera jouer).

Les règles du jeu sont données en annexe dans la section 8 ci–après.

2 - Affichage du plateau de jeu

L'affichage du plateau de jeu sera du type suivant :

Affichage effectif

```
   | A B C D E |
---- - - - - - |
 1 | X . . . . |
 2 | . . . . . |
 3 | . . . . . |
 4 | . . . . . |
 5 | . . . . O |
---------------
```

Affichage avec les espaces visibles

```
␣␣␣|␣A␣B␣C␣D␣E␣|
----␣-␣-␣-␣-␣-␣|
␣1␣|␣X␣.␣.␣.␣.␣|
␣2␣|␣.␣.␣.␣.␣.␣|
␣3␣|␣.␣.␣.␣.␣.␣|
␣4␣|␣.␣.␣.␣.␣.␣|
␣5␣|␣.␣.␣.␣.␣O␣|
----------------
```

Les pions des joueurs seront représentés par les lettres : `O`, `X`, `Y` et `Z`.
Les cases non occupées seront représentées par un point.

3 - « Spécifications »

- Le nombre de joueurs est choisi par l'utilisateur (entre 2 et 4).
- L'ordre des cartes est standard : 2 à 10, puis Valet, Dame, Roi et As (le 1).
- Une carte du jeu sera représentée par une chaîne de 2 caractères, le premier pour sa valeur (1 à 9, 0 pour 10, V pour valet, D pour Dame, R pour Roi), le deuxième la couleur de la carte (P pour pique, T pour trèfle, C pour cœur et K pour carreau). Les 2 jokers seront représentés par les chaînes de caractères `J1` et `J2`. *Exemples* : 6♥ ⟶ `6C` ; 10♠ ⟶ `0P` ; R♦ ⟶ `RK` ; 1♣ ⟶ `1T`.
- La position d'un pion sera représentée par une chaîne de 2 caractères : sa colonne suivie de sa ligne. *Exemples* : `A1` ; `E5` ; `D3` ; `D4`.
- La pioche, la défausse et la main des joueurs seront représentées par des listes de cartes. *Exemple* : 8♥, R♥, V♦, 1♣, 2♣ ⟶ `['8C', 'RC', 'VK', '1T', '2T']`.
- Pour les listes de cartes (notamment la pioche et la défausse), on pourra considérer que les cartes sont empilées de gauche à droite, en particulier la carte au sommet de la pile se trouve en dernière position de la liste.

- Un joueur sera stocké sous forme de dictionnaire avec 4 clés : « `pion` » pour la lettre le représentant sur le plateau, « `pos` » pour sa **pos**ition sur le plateau, « `pv` » pour son endurance (« **p**oint de **v**ie ») et « `cartes` » pour sa main de cartes. « `pv` » sera un entier compris entre 0 et 10. Le type « dictionnaire » est abordé dans la section 9 ci–après.
  Exemple :
  `joueur1 = {'pion':'X', 'pos':'A1', 'pv':6, 'cartes':['8C', 'RC', 'VK', '1T', 'J2']}`
- Après chaque tour de jeu d'un joueur, on affichera :
  - La lettre du joueur « actif » suivie des cartes qui ont été jouées (et dans l'ordre où elles ont été jouées),
  - le plateau de jeu (en prenant en compte les cartes jouées),
  - la main et les points d'endurance de chaque joueur, précédés de sa lettre (en prenant en compte les cartes jouées).
- Quand un joueur est éliminé (son endurance est à 0), on retire son pion du plateau (on pourra par exemple lui donner la position `00`) et il ne jouera plus les tours suivants. Le programme détectera la fin de partie et donnera le gagnant.
- Quand un joueur joue son tour il joue entre 1 et 3 cartes si c'est possible (nombre déterminé aléatoirement). Au moment de jouer une carte :
  - S'il n'a pas encore joué une carte et qu'il ne peut pas en jouer (il n'a que des cartes rouges et pas d'adversaire à proximité par exemple), il en défausse 2 (de plus faible valeur possible) et il finit son tour.
  - S'il a déjà joué une carte (exactement), mais ne peut plus en jouer, il défausse une carte (de plus faible valeur possible) et il finit son tour.
  - S'il a déjà joué deux cartes, mais ne peut plus en jouer, il finit son tour.
  - Il joue par ordre de priorité :
    1. un joker
    2. une carte pour attaquer un adversaire
    3. s'il n'est pas au centre du plateau, une carte pour se déplacer (vers le centre du plateau si possible)
  - Dans le cas où plusieurs solutions conviennent pour jouer (une carte, un déplacement ou une attaque), il choisit aléatoirement parmi celles-ci.
  - Pour une attaque, il choisit une poussée si le joueur adverse est au centre du plateau, sinon il choisit un coup.
  - Quand il finit son tour, il pioche 2 cartes (car avec les instructions précédentes il lui restera au maximum 3 cartes).
- Pendant le tour d'un autre joueur, un joueur fait toujours une parade si c'est possible (avec la carte de plus faible valeur convenant).

4 - Organisation et travail attendu

- Le projet se fera par groupe de 3 ou 4.
- Votre programme principal se nommera `rois.py`. Vous écrirez toutes vos fonctions dans un fichier `utils.py`. Pour pouvoir utiliser vos fonctions vous ferez évidemment un import dans votre programme principal (`from utils import *`).
- Le fichier `utils.py` devra contenir les tests effectués pour vérifier que les fonctions font bien ce qui est attendu. (Voir la section 10 - Modules.)
- Vous rendrez vos fichiers sources (les « .py »).

5 - Fonctions minimales à programmer

Votre fichier `utils.py` devra **au minimum** contenir les fonctions ci-dessous. Ces fonctions ne seront pas suffisantes et il vous faudra en créer d'autres ... Ces premières fonctions à programmer sont seulement une aide au démarrage.

- `affiche_plateau(L_joueurs)`

  La fonction a pour paramètre d'entrée une liste de joueurs (c'est à dire une liste de dictionnaires) et ne retourne rien.
  Elle affiche le plateau de jeu à l'écran comme indiqué dans la section 2.

- `affiche_cartes(L_cartes)`

  La fonction a pour paramètre d'entrée une liste de cartes et ne retourne rien. Elle affiche les cartes de la liste.
  Par exemple pour `['8C', 'RC', 'VK', '1T', 'J2']`, elle affiche : `8C, RC, VK, 1T, J2`.

- `valeur(carte)`

  La fonction a pour paramètre d'entrée une carte et retourne un entier correspondant à la valeur de la carte.
  Les figures ont les valeurs suivantes : Valet : 11 ; Dame : 12 ; Roi : 13 ; As : 14 ; Joker : 15.

- `KO(joueur)`

  La fonction a pour paramètre d'entrée un joueur et retourne un booléen. Le booléen retourné sera « Faux » si la valeur de la clé « `pv` » du joueur est strictement positive, et « Vrai » sinon.

- `nb_KO(L_joueurs)`

  La fonction a pour paramètre d'entrée une liste de joueurs et retourne un entier correspondant au nombre de joueurs éliminés.

- `voisins(direction,case)`

  La fonction a 2 paramètres d'entrée : `direction` une chaîne de 4 caractères représentant une direction (`orth` pour orthogonale et `diag` pour diagonale), et `case` est une chaîne de 2 caractères représentant une case du plateau de jeu (exemple : `A1`, `D3`). Elle retourne une liste de chaînes de caractères représentant les cases voisines de « `case` » dans la direction « `direction` ».

  Exemples :

  - `voisins('orth','E4')` retourne `['E3', 'D4', 'E5']` ;
  - `voisins('diag','B2')` retourne `['A1', 'C1', 'A3', 'C3']`.

- `est_voisin(j1,j2,direction)`

  La fonction a 3 paramètres d'entrée : `j1` et `j2` deux dictionnaires représentants deux joueurs, et `direction` une chaîne de 4 caractères représentant une direction (`orth` pour orthogonale et `diag` pour diagonale). Elle retourne un booléen : « Vrai » si les pions des joueurs sont voisins suivant la direction donnée et « Faux » sinon.

- `init_pioche()`

  La fonction n'a pas de paramètre d'entrée et retourne une liste de toutes les cartes rangées dans un ordre aléatoire.

- `init_defausse()`

  La fonction n'a pas de paramètre d'entrée et retourne une liste vide.

- `defausse2pioche(defausse,pioche)`

  La fonction a 2 paramètres d'entrée : `defausse` et `pioche` deux listes de cartes. Elle **modifie** les deux listes : les cartes de la `defausse` sont ajoutées à la `pioche` et cette dernière est mélangée de manière aléatoire, puis la `defausse` est vidée de toutes ses cartes.

- `est_vide(paquet)`

  La fonction a pour paramètre d'entrée une liste de carte et retourne un booléen. Le booléen retourné sera « Vrai » si la liste est vide, et « Faux » sinon.

- `pioche_cartes(n,L_cartes)`

  La fonction a 2 paramètres d'entrée : `n` un entier et `L_cartes` une liste de carte. Elle retourne une liste de cartes constituée des `n` dernières valeurs de `L_cartes`. `L_cartes` est modifiée : ses `n` dernières valeurs sont supprimées.

- `init_joueurs(n,pioche)`

  La fonction a 2 paramètres d'entrée : `n` un entier (qui devra être entre 2 et 4) et `pioche` une liste de cartes. Elle retourne une liste de `n` joueurs dans l'ordre de jeu.

  - Les lettres représentant les pions des joueurs doivent être parmi : `X`, `O`, `Y`, `Z`.
  - Les positions des joueurs sont :
    - en `A1` et `E5` s'il y a 2 joueurs,
    - en `A1`, `E1` et `E5` s'il y a 3 joueurs,
    - dans les 4 coins s'il y a 4 joueurs.
  - L'endurance des joueurs est initialement de 10.
  - La main de chaque joueur est constituée en prenant des cartes dans la `pioche`.

6 - Module random

Vous pouvez utiliser le module `random` pour ce qui est de l'aléatoire, en particulier la fonction `shuffle`.

`https://docs.python.org/fr/3.6/library/random.html`

7 - Questions bonus ...

Pour les plus rapide, après cette version de base du jeu, je vous propose d'aller un peu plus loin en ajoutant des fonctionnalités à votre jeu ...

Avant de vous lancer dans ces « questions bonus », votre programme doit être fini. Une fois votre programme de base a été rendu, je vous donnerai des compléments d'information sur les ajouts proposés.

1. Permettre au programme de faire jouer un utilisateur.
2. Proposer une autre « IA » plus efficace.
3. Ajouter des règles (« optionnelles »).

8 - Annexe 1 : Les règles du jeu « Les Rois du Rings »

Source : `http://bobouest.canalblog.com/archives/2008/04/27/8973619.html`

Le matériel

Les Rois du Ring se joue avec

- un jeu de 34 cartes (32 cartes + 2 Jokers, les cartes de 2 à 6 ne sont pas utilisées),
- des pions ou des figurines pour représenter chaque combattant (boxeur, gladiateur . . .)
- un plateau (ring, arène...) de 5 cases par 5 cases,
- 10 jetons de couleurs par combattant pour représenter leurs points d'endurance.

Mise en place

En jeu à un contre un, les figurines débutent la partie sur une case dans deux coins opposés. En multijoueurs, faites au mieux, en commençant en priorité par les coins.

On distribue 5 cartes à chaque joueur.

Pioche et défausse

Prendre la défausse et la mélanger pour constituer une nouvelle pioche une fois celle-ci épuisée.

Qui commence ?

Chaque joueur tire une carte du paquet, le plus fort sera le premier joueur, puis on joue dans le sens horaire. Remettre les cartes au hasard dans la pioche.

Séquence de jeu

- Les joueurs jouent en alternance.
- À son tour, un joueur peut jouer de 0 à 5 cartes, et/ou défausser le nombre de cartes qu'il désire.
- Un joueur peut piocher jusqu'à 2 cartes à la fin de son tour de jeu.
- Un joueur ne peut jamais avoir plus de 5 cartes en main.
- Une carte jouée représente une action de base : déplacer un personnage d'une case, porter un coup à un adversaire.

Symbolique des couleurs :

- Cartes noires (bitume) : déplacements
- Cartes rouges (sang) : attaques/parades

Symbolique des directions :

- Carreaux et Trèfles (symboles en « croix ») : à la verticale ou l'horizontale
- Cœurs et Piques : en diagonale

Ou dit autrement :

- Cartes piques : déplacement en diagonale
- Cartes Trèfles : déplacement horizontal ou vertical
- Cartes Carreaux : attaque/parade horizontale ou verticale
- Cartes Cœurs : attaque/parade en diagonale

La valeur des cartes n'intervient pas sur le jeu, sauf pour les parades (voir ci-dessous) seul le symbole importe (cœur ou trèfle par exemple).

Effets des attaques

Au choix de l'attaquant, une attaque peut être :

- soit un coup : elle retire un point d'endurance à un adversaire,
- soit une poussée : elle fait reculer l'adversaire dans le sens de la poussée (s'il a la place de reculer bien sûr).

Parades

Lorsque la figurine d'un joueur subit un coup, le joueur qui la contrôle peut parer le coup en jouant une carte du même symbole ET de valeur supérieure.

Règles spéciales

- *La place du champion* : La case centrale est la place du champion, la figurine qui est dessus retire 2 points d'endurance à l'adversaire lorsqu'elle frappe.
- *Coup bas* : Un joueur lors de son tour peut défausser 3 cartes pour obliger un joueur adverse à en défausser 2 de son choix.
- *Doigt dans l'œil* : La carte Joker, jouée contre un personnage situé dans une case adjacente, permet de piocher une carte dans le jeu de l'adversaire.
- *Tomates* : Un joueur qui ne fait aucune action de jeu (c'est à dire ne joue pas de carte) mécontente le public qui lui fait savoir en lui balançant des tomates. Le joueur perd alors 1 point d'endurance.

Fin de la partie

C'est le joueur dont le dernier personnage est encore présent sur le ring qui gagne.

9 - Annexe 2 : Le type dictionnaire

Un dictionnaire peut être vu comme une liste mais indexée par des objets immuables quelconques à la place des entiers de 0 à n. Un dictionnaire s'écrit comme une liste, entre accolades, d'éléments de type « `<cle>:<valeur>` » séparés par des virgules. Les clés seront les index et doivent donc être immuables. Le dictionnaire vide s'écrit `{}`.

Attention

Comme pour les listes, les dictionnaires sont des objets **mutables**.

Exemple d'utilisation d'un dictionnaire :

```
In [1]: annuaire = {'bob': 25, 'alice': 20, 'marc': 28}
In [2]: annuaire
Out[2]: {'alice': 20, 'bob': 25, 'marc': 28}
In [3]: annuaire = dict([('bob',25),('alice', 20),('marc', 28)])
In [4]: annuaire
Out[4]: {'alice': 20, 'bob': 25, 'marc': 28}
In [5]: annuaire = dict(bob=25,alice=20,marc=28)
In [6]: annuaire
Out[6]: {'alice': 20, 'bob': 25, 'marc': 28}
In [7]: annuaire['bob'] # L'accès aux données est similaire aux listes
Out[7]: 25
In [8]: annuaire['bob']=30 # Un dictionnaire est mutable
In [9]: annuaire
Out[9]: {'alice': 20, 'bob': 30, 'marc': 28}
In [10]: del annuaire['marc']
In [11]: annuaire
Out[11]: {'alice': 20, 'bob': 30}
In [12]: annuaire['zoe']=8 # Ajoute le couple <cle;valeur> au dictionnaire
In [13]: annuaire
Out[13]: {'alice': 20, 'bob': 30, 'zoe': 8}
In [14]: 'alice' in annuaire
Out[14]: True
In [14]: 'marc' in annuaire
Out[14]: False
In [15]: annuaire.items() # Liste des couples
Out[15]: dict_items([('bob', 30), ('alice', 20), ('zoe', 8)])
In [16]: annuaire.keys() # Liste des clés
Out[16]: dict_keys(['bob', 'alice', 'zoe'])
In [17]: annuaire.values() # Liste des valeurs
Out[17]: dict_values([30, 20, 8])
In [18]: len(annuaire)
Out[18]: 3
```

Vous pouvez aussi suivre le tutoriel :

`https://docs.python.org/fr/3.6/tutorial/datastructures.html#dictionaries`

10 - Annexe 3 : Modules

Il est possible de créer ses propres modules regroupant des fonctions spécifiques et de les utiliser via un « import » comme d'autres modules existant (`math`, `random`, ...).

Il suffit pour cela, après avoir sauvegarder son fichier servant de module, de l'importer avec la commande `import` et le nom du fichier.

Exemple 1.

Télécharger sur moodle les deux fichiers suivants et les enregistrer dans un même dossier.
Puis exécuter `projet_autre_module.py`.

- `projet_mon_module.py`

```
def fproduit(a,b):
    return a*b

print('Test de fproduit avec 2 et 3, attendu : 6')
valeur = fproduit(2,3)
print('fproduit(2,3)={} : '.format(valeur),end="")
if  valeur == 6 :
    print("OK")
else :
    print("NOK")
```

- `projet_autre_module.py`

```
import projet_mon_module as mm

a=int(input("Donner une premiere valeur entiere : "))
b=int(input("Donner une deuxieme valeur entiere : "))
prod=mm.fproduit(a,b)
print("{} x {} = {}".format(a,b,prod))
```

Pour éviter le problème d'exécution de code non désirée lors de l'import d'un module, on peut utiliser l'instruction « `if __name__ == "__main__":` » qui spécifie à Python de n'exécuter ce qui suit qui si le module dans lequel on se trouve est le programme principal (donc pas un « import »).

Il faudrait modifier le programe `projet_mon_module.py` de la façon suivante :

```
def fproduit(a,b):
    return a*b

if __name__ == "__main__":
    print('Test de fproduit avec 2 et 3, attendu : 6')
    valeur = fproduit(2,3)
    print('fproduit(2,3)={} : '.format(valeur),end="")
    if  valeur == 6 :
        print("OK")
    else :
        print("NOK")
```