Atelier 4 - UE S2 Projet

1 Travail à faire

A la fin de cet atelier, un utilisateur doit pouvoir jouer une partie "joueur contre joueur" et "joueur contre ordinateur".

Vous rendrez un unique fichier .py qui contient les tests unitaires aux fonctions, les fonctions demandées ainsi que le code principal qui permet de jouer.

Les fichiers ne doivent pas mentionner votre nom. Ils ne doivent pas contenir la suite du programme.

Le programme doit bien entendu poursuivre le travail des ateliers précédents : les fonctionnalités demandées en atelier 2 et 3 doivent se retrouver.

Il est possible de "s'inspirer" des ateliers 2 et 3 corrigés lors de l'évaluation par les pairs, sans copier. Les codes des ateliers 2, 3 et 4 seront comparés par nos soins à l'aide de l'outil moss.

2 Questions

2.1 Menu pour l'utilisateur

Afin de simplifier l'affichage pour le joueur (et l'évaluateur), vous proposerez un menu qui permettra de choisir de jouer "joueur contre joueur" ou "joueur contre ordinateur" et de choisir de jouer dans les configurations de début, milieu et de fin de partie. Votre menu proposera également de lancer la fonction générale de tests.

2.2 Joueur contre joueur

En reprenant le travail de l'atelier précédent, vous devez définir une ou plusieurs fonctions qui permettent de jouer une partie joueur contre joueur :

- Initialiser la partie (grille, joueurs, cases, pions...) au moment choisi (début, milieu ou fin de partie).
- Proposer des tours de jeu tant que la partie n'est pas finie. A chaque fin de tour, changer de joueur.
- Proclamer le vainqueur.

Cette fois encore l'accent n'est pas sur l'interface graphique mais le joueur doit savoir si c'est à lui de jouer, quels pions sont les siens (au moins en début de partie).

Selon les règles du jeu, on pourra ajouter l'affichage du nombre de pions capturés/retournés, le nombre de coups déjà joué...

2.3 Joueur contre Ordinateur "version naïve"

L'approche proposée ici est de remplacer un des deux joueurs par l'ordinateur. L'ordinateur jouera *aléatoirement*[1]. Tout le travail consiste à "construire" ces choix.

Généralement, il vaut mieux limiter les choix à des choix applicables : plutôt que de choisir au hasard une case puis vérifier qu'il y a un "bon pion" et qu'on peut lui appliquer un "bon"

1. Dans l'atelier 5, l'ordinateur choisira "intelligemment"

déplacement... on fera d'abord la liste des pions que l'on peut déplacer afin de choisir au hasard dans cette liste.

Très grossièrement, voici ce qu'on vous propose :

```
Créer autant de liste que de types de déplacements possibles
Pour chaque case de la grille :
  Si elle contient un pion du joueur :
    Pour chaque type de déplacement possible
      Ajouter les couples départ/destination dans sa liste de déplacements possibles (et
          qui respectent la règle)
Choisir au hasard un type de déplacement*
Choisir au hasard un couple dans la liste correspondant à ce déplacement et l'appliquer
Tant que le type de déplacement choisi autorise un enchaînement
   Choisir au hasard si on souhaite essayer de l'appliquer
   Si oui
    Calculer tous les couples départ/arrivée possibles (en excluant le coup précédent)
    Choisir au hasard parmi ce couple et l'appliquer
    Recommencer
  Si non
    sortir.
```

Remarque : Dans l'atelier 5, vous aurez à programmer le même genre d'algorithme où les choix ne seront plus "au hasard" mais guidés par ce qu'on appelle une heuristique. Il y aura donc pas mal de "récupération" de fonctions.

Attention, le choix dans le type de déplacement n'est pas toujours possible selon les règles (déplacement simple après un enchaînement par exemple).

2.4 Code principal

Classiquement, votre code principal devra consister simplement à proposer le menu, et en fonction du choix appeler soit la fonction du joueur contre joueur (à la configuration choisie), soit appeler la fonction du joueur contre ordinateur (à la configuration choisie).

2.5 Tests unitaires

Vous aurez un grand nombre de fonctions à écrire, commencez par les définir les fonctions de tests (test driven development) et faites-les appeler par une fonction générale de tests afin de gagner en efficacité. La fonction générale de tests fera des affichages pour indiquer les fonctions testées/valides.

3 Evaluation

Voici une liste non exhaustive des critères sur lesquels vous serez évalués :

- Menu (approx 1 point)
- Jouabilité "Joueur contre Joueur" (approx 5 points) :
  - Respect des règles
  - Gestion des erreurs (le programme ne s'interrompt pas)
  - Les actions (déplacement, prise/capture, retournement, passer, abandonner...) sont toutes possibles selon les règles

    - Le dialogue est pratique
    - Le vainqueur est déterminé/annoncé
- Jouabilité "Joueur contre Ordinateur" (approx 5 points) :
    - Mêmes critères que "Joueur contre Joueur"
- IA Naïve (approx 5 points) :
    - Tous les pions sont testés
    - Toutes les actions (par pion) sont testées
    - Le choix aléatoire est judicieusement appliqué
- Clean code (approx 3 points) :
    - les fonctions entre "Joueur contre Joueur" et "Joueur contre Ordinateur" sont bien décomposées et réutilisées
    - le nom des variables est explicite et respecte les conventions
    - le programme est composé de fonctions courtes et explicites
    - le programme est commenté pour améliorer la lisibilité et compréhension
    - il y a un code principal qui s'exécute au lancement
- Tests (approx 2 points) : la fonction générale de tests est structurée, faits des affichages pertinents et comprend des tests pertinents pour la majorité des fonctions.
- ...

4 Conseils

4.1 Le choix aléatoire

Pour générer un choix d'entier aléatoire entre a et b (inclus) :

```
#En début de code principal (une seule fois) les 2 lignes suivantes :
import random
random.seed()
...
#à  chaque besoin de choix aléatoire
choix = random.randint(a,b)
```

Vous pouvez consulter la doc de random qui propose d'autres fonctions (random.choice(sequence) par exemple) : `https://docs.python.org/3.5/library/random.html`

4.2 Itérations successives

D'une manière générale, et suivant votre aisance à programmer, il est conseillé de travailler par "itération successive". Par exemple d'abord écrire des fonctions de déplacement qui ne vérifient pas la validité vis à vis de la règle (on fait confiance aux joueurs) afin de pouvoir programmer le tour du jeu. Ainsi, dans la fonction de déplacement vous pouvez utiliser des fonctions auxiliaires de validité renvoyant toujours `True` que vous compléterez plus tard. Bien sûr n'oubliez pas de les compléter.

4.3 L'encodage

Lors de l'évaluation de l'atelier 3, de nombreux étudiants n'ont pas réussi à exécuter les programmes qui contenaient des accents/caractères spéciaux. Vous pouvez ajouter la ligne `#coding : utf-8` en début de fichier ou supprimer tous les caractères spéciaux.

4.4 Outil de développement

Il est fortement conseillé d'utiliser des logiciels de développement tels que PyCharm, spyder, IDLE, eclipse... repl.it est adapté pour écrire une ou deux fonctions mais n'est pas assez puissant pour un tel projet. De plus, sur repl.it votre code est public.

4.5 Fraudes

Les programmes seront tous testés à l'aide de l'outil moss. S'il s'avère que des programmes sont identiques (ou très similaires), vous serez sanctionnés au delà du simple 0 à l'atelier.