TP D'INFORMATIQUE N°7
IA pour Puissance 4

L'objectif de ce TP est de programmer un jeu de puissance 4 permettant, en entrant les coups dans la console, à deux joueurs humains de s'affronter, puis à un joueur humain d'affronter une IA.

1 Implémentation du jeu de Puissance 4

La base du jeu est donnée dans le fichier `puissance4.py` disponible sur cahier de prepa. La fonction `PvP` permet de lancer une partie entre deux joueurs humains. On remarquera qu'une configuration du jeu est représentée à la fois par une grille de dimensions $n \times m$, dont les cases valent 0 (case inoccupée), 1 (occupée par le joueur 1) ou 2 (occupée par le joueur 2), et (pour une raison d'efficacité) par une liste `hauteur` donnant pour chaque colonne le nombre de coups déjà joués dans cette colonne.

1. Quelle est la complexité de la fonction `jouer` ? Quelle serait sa complexité si on ne prenait pas `hauteur` en argument ?
2. Compléter la fonction `victoire` pour qu'elle renvoie `True` si le joueur numéro `p` a gagné sur `grille`, et `False` sinon.

2 Implémentation de l'algorithme min-max

On cherche à présent à écrire une fonction calculant automatiquement un coup pertinent dans une configuration donnée, en se basant sur l'algorithme Min-max.
Lors du parcours de l'arbre des coups possibles, pour obtenir la configuration fille d'une configuration donnée, on jouera le coup considéré avec la fonction `jouer`. Il faudra alors effacer ce coup une fois que le score de la configuration fille aura été calculé.

1. Écrire une fonction `enlever_coup` prenant en argument `grille`, `hauteur` et un entier `k`, et modifiant `grille` **et** `hauteur` pour enlever le dernier coup joué sur la colonne `k` (on suppose qu'il y a au moins un coup sur cette colonne). on pourra s'inspirer de la fonction `jouer`.
2. Il faut également disposer d'une heuristique évaluant une configuration du point de vue d'un joueur `p`. Pour cela, en notant `a` l'adversaire de `p`, on compte le score d'une série de 4 cases consécutives (ne contenant pas 4 coups d'un même joueur) de la façon suivante :
   - Si cette série ne contient aucun coup de `a`, on ajoute $x/(4-x)$, où x est le nombre de coups de `p` dans cette liste ;
   - Si cette série ne contient aucun coup de `p`, on retranche $x/(4-x)$, où x est le nombre de coups de `a` dans cette liste ;

   Le score d'une configuration est alors la somme des scores de chaque série de 4 coups, horizontale, verticale ou en diagonale, dans la grille.
   Par exemple, la grille `ex_grille` dont l'affichage graphique est :

```
  0 1 2 3 4 5 6
0| | | | | | | |0
1| | | | | | | |1
2| | | | | | | |2
3| | | |o| | | |3
4| | | |x| | | |4
5| |x|o|o| | | |5
  0 1 2 3 4 5 6
```

   a un score de 3 pour le joueur `o` (c'est-à-dire le joueur 1).

   (a) Écrire une fonction `eval_serie` prenant en argument une liste de 4 coups et un numéro de joueur, et renvoyant le score de ce joueur sur cette série.
   Par exemple :

- `eval_serie([1,0,0,1], 1)` doit renvoyer 1 ;
- `eval_serie([1,2,0,1], 1)` doit renvoyer 0 ;
- `eval_serie([2,2,0,2], 1)` doit renvoyer -3 ;

(b) En déduire une fonction `eval_grille` prenant en argument une grille et un joueur, et renvoyant le score de cette grille pour ce joueur. On pourra reprendre la structure de la fonction `victoire` pour considérer chaque série de 4 cases consécutifs. On pourra tester cette fonction sur la grille `ex_grille`.

3. Écrire une fonction `IA` prenant en argument `grille`, `hauteur`, `p` et une profondeur `prof`, et renvoyant un coup pour le joueur `p` en appliquant l'algorithme Min-max pour la profondeur `prof`. On suivra le pseudo-code suivant, similaire à celui vu pour le cas général en cours :

```
fonction IA(grille, hauteur, p, prof):
    soit opp l'adversaire de p

    fonction coup_max(prof):
    #renvoie le score de p sur le noeud courant, alors que c'est à p de jouer
        si opp a déjà gagné, on renvoie -infini
        si la grille est remplie, on renvoie 0
        si prof = 0, on renvoie eval_grille(grille,p)
        sinon, pour chaque coup possible k:
            on joue ce coup pour p dans la grille
            on évalue le score résultant avec coup_min(prof-1)
            on supprime le coup
        puis on renvoie le maximum des scores ainsi calculés

    fonction coup_min(prof):
    #renvoie le score de p sur le noeud courant, alors que c'est à opp de jouer
        si p a déjà gagné, on renvoie infini
        si la grille est remplie, on renvoie 0
        si prof = 0, on renvoie eval_grille(grille,p)
        sinon, pour chaque coup possible k:
            on joue ce coup pour opp dans la grille
            on évalue le score résultant avec coup_max(prof-1)
            on supprime le coup
        puis on renvoie le minimum des scores ainsi calculés

    #Corps de la fonction IA, similaire à coup_max
    pour chaque coup possible k:
        on joue ce coup pour p dans la grille
        on évalue le score résultant avec coup_min(prof-1)
        on supprime le coup
    puis on renvoie le coup réalisant le score maximal parmi les scores ainsi calculés
```

4. Tester cette fonction `IA` grâce à la fonction `PvIA` qui permet à un joueur humain de jouer une partie contre l'ordinateur jouant ses coups avec la fonction `IA`.
5. Optimiser la fonction `IA` en mémoïsant les scores des configurations explorées.
6. Écrire une fonction `IAvIA` permettant à deux IA de s'affronter. Observer le résultat de parties entre IA utilisant des profondeurs différentes.
7. Déterminer une meilleure heuristique à utiliser dans l'IA.