Jeu du puissance 4

Thèmes : théorie des jeux, jeux d'accessibilité, algorithme min-max

Le *puissance 4* est un jeu à deux joueurs, le joueur rouge (1) et le joueur jaune (2), se jouant sur une grille de 6 lignes et 7 colonnes. Chacun à son tour, les joueurs laissent tomber un jeton, en haut d'une des 7 colonnes, le jeton vient alors s'empiler en haut des jetons déjà présents dans la colonne. Le joueur qui parvient à créer un alignement de 4 jetons de sa couleur gagne la partie.

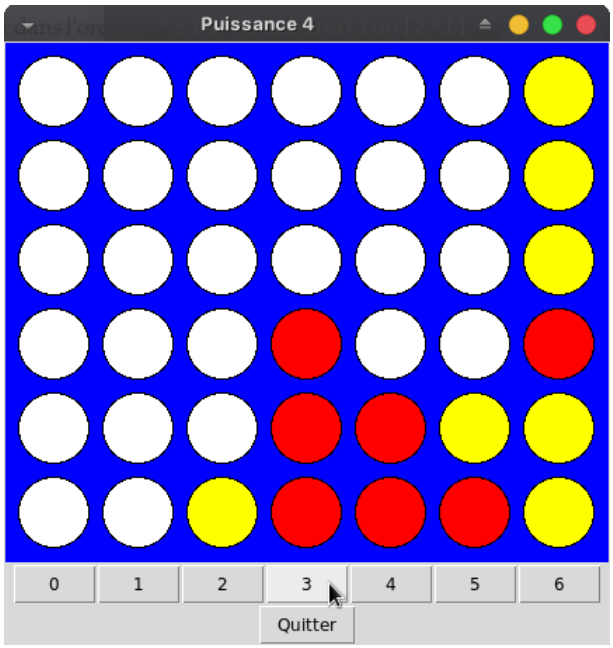

Figure 1 Jeu du puissance 4

1 Programmation du jeu

On modélise la grille de jeu par une matrice d'entiers `numpy` de 6 lignes et 7 colonnes. La ligne d'indice 0 sera la ligne située tout en haut de la grille de puissance 4; ainsi la ligne d'indice 5 est la ligne du bas.

Les entiers de la matrice représentent le contenu des cases selon le code suivant :

```
VIDE = 0
A = 1 // rouge
B = 2 // jaune
```

Un *coup* pour un joueur est un entier entre 0 et 6 représentant le numéro de colonne choisi.

Question 1

Écrire une fonction `est_libre(grille, j)` retournant un booléen indiquant s'il est possible de jouer dans la colonne `j` avec la `grille` donnée.

Question 2

Écrire une fonction `coups_possibles(grille)` retournant la liste des coups qu'il est possible de jouer avec la `grille` donnée.

Question 3

Écrire une fonction `jouer(grille, joueur, j)` qui modifie la `grille` donnée en entrée, en faisant jouer le joueur `joueur` en colonne `j`. À l'aide d'une assertion, on pourra annoter en pré-condition que le coup proposé est jouable.

On donne la fonction suivante :

```
def diagonale1(grille, joueur):
    for i in range(3):
        for j in range(4):
            if (grille[i][j] == joueur
                and grille[i+1][j+1] == joueur
                and grille[i+2][j+2] == joueur
                and grille[i+3][j+3] == joueur
                ):
                return True
    return False
```

Question 4

Quel est le rôle de la fonction `diagonale1` ? Écrire de même les fonctions :
a. `ligne`
b. `colonne`
c. `diagonale2`

Question 5

En déduire une fonction `gagne(grille, joueur)` qui retourne `True` si et seulement si dans la position `grille` le joueur `joueur` a gagné la partie.

On pourra tester le jeu en humain contre humain en exécutant la dernière cellule du fichier fourni. Attention, le code de l'interface du jeu a été programmé pour vous, il n'est pas nécessaire de le modifier.

2 Intelligence artificielle : algorithme min-max

On souhaite maintenant avoir la possibilité de jouer au puissance 4 contre un *bot*. On programmera ce *bot* à l'aide de l'algorithme min-max vu en cours. On rappelle que celui-ci repose sur l'utilisation d'une fonction *heuristique* qui permet d'estimer les chances de gain de chaque joueur. On considérera qu'une heuristique positive signifie que le joueur A a l'avantage. L'heuristique vaudra $+\infty$ (resp. $-\infty$) lorsque la position est gagnée pour le joueur A (resp. le joueur B).

On donne la fonction heuristique basique suivante :

```
def h1(grille):
    if gagne(grille, A):
        return +float('inf')
    elif gagne(grille, B):
        return -float('inf')
    else:
        return 0
```

Question 6

Compléter la fonction suivante qui retourne une évaluation de la `grille`, dans la situation où c'est au tour du joueur `joueur`. Cette évaluation reposera sur l'algorithme *min-max* utilisé avec la profondeur de recherche `profondeur` et la fonction heuristique `heuristique`.

```
def eval(grille, joueur, profondeur, heuristique):
    coups = coups_possibles(grille)
    if profondeur==0 or est_final(grille):
        return (...)
    if joueur == A: # A veut maximiser
        max_eval = -float('inf')
        for c in coups:
            ng = grille.copy()
            jouer(ng, joueur, c)
            e = eval(ng, B, profondeur-1, heuristique)
            if e > max_eval:
                (...)
        return max_eval
    if joueur == B: # B veut minimiser
        (...)
```

Question 7

Écrire une fonction

```
minmax(grille, joueur, profondeur, heuristique)
```

basée sur le code de la fonction `eval` mais qui retourne cette fois-ci un couple `(eval, best)` où `eval` est l'évaluation de la position et `best` est le meilleur coup à jouer pour le `joueur`.

On pourra tester le jeu contre la machine en assignant la valeur `True` à la variable `joueurIA`.

Pour aller plus loin... Pour améliorer votre IA, vous pouvez définir une meilleure fonction heuristique. Par exemple, on pourra regarder chaque alignement de 4 cases et accorder 1 point si le joueur a placé 2 jetons et que les autres cases sont libres, 3 points s'il a placé 3 jetons.