

TP Morpion - Positions gagnantes et stratégies

Dans le jeu du morpion ou tic-tac-toe , 2 joueurs doivent remplir alternativement une case de la grille avec un symbole :

- X pour le joueur 1,
- O pour le joueur 2. Le gagnant est le premier à aligner trois symboles identiques horizontalement, verticalement, ou en diagonale.

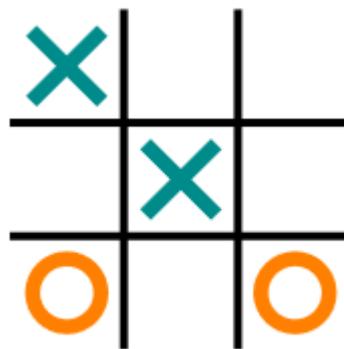
Le joueur 1 commence.

Modélisation informatique

On choisit de représenter une grille de jeu de morpion par une liste de listes en 3×3 comportant uniquement des 0, 1, 2.

- 0 pour une case libre
- 1 pour X (case jouée par le joueur 1)
- 2 pour O (case jouée par le joueur 2)

Ainsi, la grille de jeu



est représentée en Python par la liste `grille1` suivante :

```
Entrée[1]: ▶ 1 grille1 = [[1, 0, 0],  
2             [0, 1, 0],  
3             [0, 0, 2]]
```

Dans le script ci-après, les fonctions suivantes sont fournies :

- `afficher(grille)` qui réalise un affichage textuel de la grille,
- `gagnant(grille)` qui renvoie le numéro d'un joueur s'il est gagnant, 0 sinon,
- `grilleVersTuple(grille)` qui renvoie un tuple de taille 9 dont les éléments sont ceux de la grille énumérés par ligne,
- `tupleVersGrille(grille)` qui renvoie une grille dont les éléments sont ceux du tuple de taille 9 énumérés par ligne.
- `copie(grille)` qui crée et renvoie une copie de la grille, sans la modifier.

Entrée[79]: ▶

```
1 grille1 = [[1, 0, 0], [0, 1, 0], [2, 0, 2]]
2
3 def afficher(grille):
4     """ afficher( grille : list )
5         affiche la grille
6         entrees : grille, liste, representant l'etat du jeu
7     """
8     #conversion = {0:' ', 1:'X', 2:'•'}
9     conversion = {0:' ', 1:'X', 2:'O'}
10    print(' _ _ _')
11    for ligne in grille:
12        print('|', end='') #end='' pour ne pas revenir a la ligne
13        for elem in ligne:
14            print(conversion[elem], end='|')
15        print()
16    print(' - - -')
17
18 def gagnant(grille):
19    """ gagnant( grille : list ) ->int
20        entrees : grille, liste, representant l'etat du jeu
21        sortie : entier qui vaut 0, 1, 2 selon si personne, le joueur 1 ou le j
22    """
23    for k in (0, 1, 2): # Analyse des lignes et des colonnes
24        if grille[k][0] == grille[k][1] == grille[k][2]:
25            return grille[k][0]
26        if grille[0][k] == grille[1][k] == grille[2][k]:
27            return grille[0][k]
28    for k in (0, 2) : # Analyse des diagonales
29        if grille[k][0] == grille[1][1] == grille[2 - k][2]:
30            return grille[1][1]
31    return 0
32
33 def grilleVersTuple(grille):
34    """ grilleVersTuple( grille : list ) ->tuple
35        convertit une grille 3x3 en tuple
36        entrees : grille, liste, representant l'etat du jeu
37        sortie : tuple qui correspond à grille
38    """
39    return tuple(grille[0] + grille[1] + grille[2])
40
41 def tupleVersGrille(tuple):
42    """ tupleVersGrille( tuple : tuple ) ->list
43        convertit un tuple en grille 3x3
44        entrée : tuple qui correspond à grille
45        sortie : grille, liste, representant au tuple sous forme de liste de lis
46    """
47    return [[tuple[3 * i+j] for j in range(3)] for i in range(3)]
48
49 def copie(grille):
50    """ copie( grille : list ) ->list
51        renvoie une copie de la grille
52        entrees : grille, liste, representant l'etat du jeu
53        sortie : list qui correspond à une copie de grille
54    """
55    return [ligne.copy() for ligne in grille]
56    # alternative :
57    # return tupleVersgrille(grille2tuple(grille))
58
```

```

Entrée[81]: ▶ 1 afficher(grille1)
                2
                3 print ( gagnant(grille1) )
                4 # affiche 0
                5
                6 grille2 = [[1, 0, 2], [2, 1, 0], [0, 0, 1]]
                7 afficher(grille2)
                8
                9 print( gagnant(grille2) )
            10 # 1
            11 t = grilleVersTuple(grille1)
            12 print(t)
            13 #(1, 0, 0, 0, 1, 0, 2, 0, 2)
            14 g = tupleVersGrille(t)
            15 print(g) #[[1, 0, 0], [0, 1, 0], [2, 0, 2]]
    
```

```

|X|_| |
| |X| |
|0| |0|
|_|_|
    
```

0

```

|X| |0|
|0|X| |
|_|_|X|
    
```

1

```

(1, 0, 0, 0, 1, 0, 2, 0, 2)
[[1, 0, 0], [0, 1, 0], [2, 0, 2]]
    
```

Analyser le code fourni.

1- Écrire une fonction `casesLibres(grille)` qui prend en paramètre la grille de jeu `grille` et renvoie la liste des positions (i, j) des cases vides de `grille` (c'est-à-dire tel que `grille[i][j]` vaut 0).

```

Entrée[82]: ▶ 1 def casesLibres(grille):
                2     """ casesLibres( grille : list ) ->list
                3         entrees : grille, liste, representant l'etat du jeu
                4         sortie : lesCasesLibres, liste des tuples des cases libres
                5     """
                6     pass
                7
                8 #print(cases_libres(grille1))
            9 #[(0, 1), (0, 2), (1, 0), (1, 2), (2, 1)]
    
```

2- Écrire une fonction `joueur(grille)` renvoyant le numéro du joueur qui doit jouer le prochain coup sur la grille passée en paramètre.
On pourra utiliser le nombre de cases libres.

```

Entrée[ ]: ▶ 1 def joueur(grille):
                2     """ casesLibres( grille : list ) ->list
                3         entrees : grille, liste, representant l'etat du jeu
                4         sortie : entier, numéro du joueur qui doit jouer
                5     """
                6     pass
                7
                8 print(joueur(grille1)) # 1
            9 print(joueur(grille2)) # 2
    
```

Calcul d'attracteurs

Pour le calcul des attracteurs, nous n'allons pas passer par le graphe comme dans le cours, mais directement calculer les positions accessibles à partir d'une position donnée.

3- Écrire une fonction `successeurs(grille)` renvoyant la liste des grilles que l'on peut obtenir à partir de `grille` en jouant un coup (autorisé).

On rappelle qu'on a fourni une fonction permettant de réaliser une copie de grille.

Entrée[98]:

```
1 def successeurs(grille):
2     """ successeurs( grille : list ) ->list
3         entrees : grille, liste, representant l'etat du jeu
4         sortie : lesSuccesseurs, liste de listes, correspondant aux positions ac
5     """
6     pass
7
8 afficher(grille1)
9 print("Successeurs :")
10 for g in successeurs(grille1):
11     afficher(g)
```

Pour la suite, nous allons avoir besoin d'utiliser un dictionnaire dont les clés seront des grilles de jeu.

4- Rappeler pourquoi il n'est pas possible d'utiliser une grille telle quelle comme clé dans un dictionnaire.

La solution sera la conversion en tuples de nos grilles, via les fonctions fournies.

Calcul des positions gagnantes (attracteur)

Comme vu en cours, une grille `g` est une position gagnante pour le joueur 1 si on a l'un des cas suivants :

- le joueur 1 est gagnant sur la grille `g` (position de victoire) ;
- c'est au joueur 1 de jouer et il existe au moins un successeur de `g` qui soit une position gagnante pour le joueur 1 ;
- c'est au joueur 2 de jouer, qu'il peut jouer et que tous les successeurs de `g` sont des positions gagnantes pour le joueur 1.

Contrairement à la formalisation du cours, on va construire ici l'attracteur récursivement, par mémoïsation (on ne tient donc pas compte du nombre de coups pour gagner).

Q5- La fonction `attracteur(grille)` renvoie la liste des grilles qui sont des positions gagnantes pour le joueur 1, sachant que `grille` correspond à la grille initiale.

Elle fait appel à la fonction récursive `estGagnante(g,dico)` à compléter. On dispose des fonctions `any(L)` et `all(L)` qui, à partir d'une liste (ou d'un itérable plus généralement) `L` de booléens, renvoient respectivement `True` si l'un des éléments de `L` est vrai pour le premier et `True` si tous les éléments de `L` le sont pour le deuxième ; `False` dans le cas contraire.

Assurez-vous que vous seriez capables d'écrire de telles fonctions !

Entrée[]: ▶

```
1 def estGagnante(g,dico):
2     """ estGagnante(g : list,dico : dict) : bool
3         fonction récursive
4         entrees : g, liste, representant l'etat du jeu
5             : dico, dictionnaire au format(tuple de la grille, booleen)- Moc
6         sortie : booleen, à True si g est une position gagnante, False sinon
7     """
8     t = grilleVersTuple(g)
9     if t not in dico: #memoisation: pour éviter de retester une grille de jeu de
10        # traiter les 3 cas et ajouter dans dico la clé t avec la valeur True ou
11
12
13
14
15
16     return dico[t]
17
18 def attracteur(grille):
19     """ attracteur(grille: list) : list
20     fonction récursive
21     entrees : grille, liste, representant l'etat du jeu
22     sortie : liste des tuples qui correspondent à des grilles de positions g
23     """
24     dico = {}
25     # dico[grilleVersTuple(g)] = True si la grille g est une position gagnante p
26
27     est_gagnante(grille,dico)
28
29     return [t for t in dico if dico[t]]
30     # liste des tuples des positions gagnantes.
31
32
33 #afficher(grille1)
34 A=attracteur(grille1)
35 print(A)
36 """
37 [(1, 1, 2, 2, 1, 1, 2, 1, 2), (1, 1, 2, 2, 1, 1, 2, 0, 2),
38 (1, 1, 2, 0, 1, 0, 2, 1, 2), (1, 1, 2, 0, 1, 0, 2, 0, 2),
39 (1, 1, 1, 2, 1, 0, 2, 0, 2), (1, 1, 0, 2, 1, 0, 2, 1, 2),
40 (1, 1, 0, 2, 1, 0, 2, 0, 2), (1, 1, 1, 0, 1, 2, 2, 0, 2),
41 (1, 1, 0, 0, 1, 2, 2, 1, 2), (1, 1, 0, 0, 1, 2, 2, 0, 2),
42 (1, 1, 1, 2, 1, 2, 2, 1, 2), (1, 0, 1, 2, 1, 2, 2, 1, 2),
43 (1, 0, 1, 2, 1, 0, 2, 0, 2), (1, 0, 1, 0, 1, 2, 2, 0, 2),
44 (1, 2, 0, 1, 1, 1, 2, 0, 2), (1, 2, 2, 1, 1, 1, 2, 1, 2),
45 (1, 2, 2, 1, 1, 0, 2, 1, 2), (1, 2, 0, 1, 1, 0, 2, 0, 2),
46 (1, 0, 2, 1, 1, 1, 2, 0, 2), (1, 0, 2, 1, 1, 0, 2, 0, 2),
47 (1, 2, 2, 0, 1, 1, 2, 1, 2), (1, 2, 0, 0, 1, 1, 2, 0, 2),
48 (1, 0, 2, 2, 1, 1, 2, 1, 2), (1, 0, 2, 0, 1, 1, 2, 1, 2),
49 (1, 0, 2, 0, 1, 1, 2, 0, 2), (1, 0, 2, 0, 1, 0, 2, 1, 2),
50 (1, 0, 0, 2, 1, 0, 2, 1, 2), (1, 0, 0, 0, 1, 2, 2, 1, 2)]
51 """
52 for t in A:
53     afficher(tupleVersGrille(t))
54
```

Stratégie gagnante

6- Le joueur 1 a-t-il une stratégie gagnante à partir de la grille grille1 ?

1

Calcul d'une stratégie gagnante à partir de grille

Si `g` est une grille accessible depuis grille et `t` le tuple associé,

- si le joueur 1 a gagné sur la grille `g`, si `g` n'est pas une position gagnante, ou si le joueur 1 ne contrôle pas la position, `t` n'est pas une clé du dictionnaire `dico` ;
- sinon, `dico[t]` correspond à un coup possible pour le joueur 1 à partir de `g` vers une position gagnante.

7- Écrire une fonction `strategie(grille)` renvoyant un dictionnaire `dico` contenant une stratégie gagnante pour le joueur 1 à partir de la grille `grille` .

Entrée[]:

```
1 def strategie(grille):
2     """
3         entree : grille, liste de listes, representant l'etat du jeu
4         sortie : dictionnaire qui établit pour chaque position gagnante,
5                 le coup suivant à jouer (faisant partie des positions gagnantes )
6                 au format(tuple de la position, liste de la position suivante a
7     """
8     # détermination des positions gagnantes
9
10    # creation d'un dictionnaire
11
12    #parcours de chacune des positions gagnantes pour déterminer le coup suivant
13
14    # retour du dictionnaire
15
16    print(strategie(grille1))
17    """
18    {(1, 1, 2, 2, 1, 1, 2, 0, 2): [[1, 1, 2], [2, 1, 1], [2, 1, 2]],
19     (1, 1, 2, 0, 1, 0, 2, 0, 2): [[1, 1, 2], [0, 1, 0], [2, 1, 2]],
20     (1, 1, 0, 2, 1, 0, 2, 0, 2): [[1, 1, 1], [2, 1, 0], [2, 0, 2]],
21     (1, 1, 0, 0, 1, 2, 2, 0, 2): [[1, 1, 1], [0, 1, 2], [2, 0, 2]],
22     (1, 0, 1, 2, 1, 2, 2, 1, 2): [[1, 1, 1], [2, 1, 2], [2, 1, 2]],
23     (1, 0, 1, 2, 1, 0, 2, 0, 2): [[1, 1, 1], [2, 1, 0], [2, 0, 2]],
24     (1, 0, 1, 0, 1, 2, 2, 0, 2): [[1, 1, 1], [0, 1, 2], [2, 0, 2]],
25     (1, 2, 2, 1, 1, 0, 2, 1, 2): [[1, 2, 2], [1, 1, 1], [2, 1, 2]],
26     (1, 2, 0, 1, 1, 0, 2, 0, 2): [[1, 2, 0], [1, 1, 1], [2, 0, 2]],
27     (1, 0, 2, 1, 1, 0, 2, 0, 2): [[1, 0, 2], [1, 1, 1], [2, 0, 2]],
28     (1, 2, 2, 0, 1, 1, 2, 1, 2): [[1, 2, 2], [1, 1, 1], [2, 1, 2]],
29     (1, 2, 0, 0, 1, 1, 2, 0, 2): [[1, 2, 0], [1, 1, 1], [2, 0, 2]],
30     (1, 0, 2, 2, 1, 1, 2, 1, 2): [[1, 1, 2], [2, 1, 1], [2, 1, 2]],
31     (1, 0, 2, 0, 1, 1, 2, 0, 2): [[1, 0, 2], [1, 1, 1], [2, 0, 2]],
32     (1, 0, 2, 0, 1, 0, 2, 1, 2): [[1, 1, 2], [0, 1, 0], [2, 1, 2]],
33     (1, 0, 0, 2, 1, 0, 2, 1, 2): [[1, 1, 0], [2, 1, 0], [2, 1, 2]],
34     (1, 0, 0, 0, 1, 2, 2, 1, 2): [[1, 1, 0], [0, 1, 2], [2, 1, 2]]}
35    """
```

8- Comprendre la fonction `jeu(grille)` fournie ci-après , puis essayer de jouer contre l'ordinateur (vous êtes le joueur 2).

Entrée[]: ▶

```
1 def jeu(grille):
2     dico = strategie(grille)
3     while gagnant(grille) == 0 and len(casesLibres(grille)) > 0:
4         t = grilleVersTuple(grille)
5         if joueur(grille) == 2:
6             afficher(grille)
7             i, j = -1, -1
8             while i not in (1, 2, 3) or j not in (1, 2, 3) or grille[i - 1][j -
9                 i, j = map(int, list(input("Coordonnées de votre coup de la forme
10                 grille[i - 1][j - 1] = 2
11         else:
12             if t not in dico:
13                 print("Pas de stratégie gagnante")
14                 return
15                 grille = dico[t]
16         gagne = gagnant(grille)
17         afficher(grille)
18         if gagne != 0:
19             print(f"Le joueur {gagne} a gagné !")
20         else:
21             print("Match nul.")
22
23 grille_vider = [[0, 0, 0] for _ in range(3)]
24 jeu(grille_vider)
```