TP Morpion - Positions gagnantes et stratégies

Eléments de correction

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.

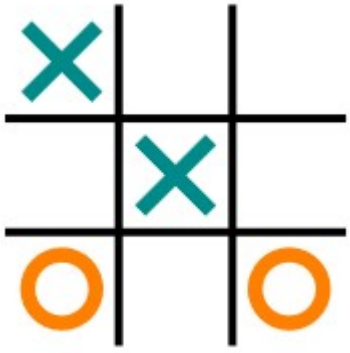
Le joueur 1 commence.

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]:

```
grille1 = [[1, 0, 0],
           [0, 1, 0],
           [2, 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[2]:

```python
grille1 = [[1, 0, 0], [0, 1, 0], [2, 0, 2]]

def afficher(grille):
    """ afficher( grille : list )
        affiche la grille
        entrees : grille, liste, representant l'etat du jeu
    """
    #conversion = {0:' ', 1:'✕', 2:'●'}
    conversion = {0:' ', 1:'X', 2:'O'}
    print(' _ _ _')
    for ligne in grille:
        print('|', end='') #end='' pour ne pas revenir a la ligne
        for elem in ligne:
            print(conversion[elem], end='|')
        print()
    print(' ¯ ¯ ¯')

def gagnant(grille):
    """ gagnant( grille : list ) ->int
        entrees : grille, liste, representant l'etat du jeu
        sortie : entier qui vaut 0, 1, 2 selon si personne, le joueur 1 ou le jo
    """
    for k in (0, 1, 2): # Analyse des lignes et des colonnes
        if grille[k][0] == grille[k][1] == grille[k][2]:
            return grille[k][0]
        if grille[0][k] == grille[1][k] == grille[2][k]:
            return grille[0][k]
    for k in (0, 2) : # Analyse des diagonales
        if grille[k][0] == grille[1][1] == grille[2 - k][2]:
            return grille[1][1]
    return 0

def grilleVersTuple(grille):
    """ grilleVersTuple( grille : list ) ->tuple
        convertit une grille 3x3 en tuple
        entrees : grille, liste, representant l'etat du jeu
        sortie : tuple qui correspond à grille
    """
    return tuple(grille[0] + grille[1] + grille[2])

def tupleVersGrille(tuple):
    """ tupleVersGrille( tuple : tuple ) ->list
        convertit un tuple en grille 3x3
        entrée : tuple qui correspond à grille
        sortie : grille, liste, representant au tuple sous forme de liste de lis
    """
    return [[tuple[3 * i+j] for j in range(3)] for i in range(3)]

def copie(grille):
    """ copie( grille : list ) ->list
        renvoie une copie de la grille
        entrees : grille, liste, representant l'etat du jeu
        sortie : list qui correspond à une copie de grille
    """
    return [ligne.copy() for ligne in grille]
    # alternative :
    # return tupleVersgrille(grille2tuple(grille))

```

Entrée[3]:

```
afficher(grille1)

print ( gagnant(grille1) )
# affiche  0

grille2 = [[1, 0, 2], [2, 1, 0], [0, 0, 1]]
afficher(grille2)

print( gagnant(grille2) )
# 1
t = grilleVersTuple(grille1)
print(t)
#(1, 0, 0, 0, 1, 0, 2, 0, 2)
g = tupleVersGrille(t)
print(g) #[[1, 0, 0], [0, 1, 0], [2, 0, 2]]
```

```
 _ _ _
|X| | |
| |X| |
|O|_|O|
 ‾ ‾ ‾

0

 _ _ _
|X| |O|
|O|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 paramaètre 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[4]:

```
def casesLibres(grille):
    """ casesLibres( grille : list ) ->list
        entrees : grille, liste, representant l'etat du jeu
        sortie : lesCasesLibres, liste des tuples des cases libres
    """
    pass

#print(cases_libres(grille1))
#[(0, 1), (0, 2), (1, 0), (1, 2), (2, 1)]
```

Entrée[5]:

```
def casesLibres(grille):
    """ casesLibres( grille : list ) ->list
        entrees : grille, liste, representant l'etat du jeu
        sortie : lesCasesLibres, liste des tuples des cases libres
    """
    lesCasesLibres = []
    for i in range(len(grille)):
        for j in range(len(grille[0])):

            if grille[i][j]==0:
                lesCasesLibres.append((i,j))
    return lesCasesLibres
print(cases_libres(grille1))
```

```
Traceback (most recent call last):
  File "<input>", line 13, in <module>
NameError: name 'cases_libres' is not defined
```

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[6]:

```
def joueur(grille):
    """ casesLibres( grille : list ) ->list
        entrees : grille, liste, representant l'etat du jeu
        sortie : entier, numéro du joueur qui doit jouer
    """
    pass

print(joueur(grille1)) # 1
print(joueur(grille2)) # 2
```

```
None
None
```

Entrée[7]:

```
def joueur(grille):
    """ casesLibres( grille : list ) ->list
        entrees : grille, liste, representant l'etat du jeu
        sortie : entier, numéro du joueur qui doit jouer
    """
    if (len(casesLibres(grille)) % 2==0):
        return 2
    else :
        return 1
def joueur(grille):
    return 2- (len(casesLibres(grille)) % 2)

print(joueur(grille1)) # 1
print(joueur(grille2)) # 2
```

```
1
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.

Attention à bien tenir compte des cas où l'un des joueurs a gagné

Entrée[8]:

```
def successeurs(grille):
    """ successeurs( grille : list ) ->list
        entrees : grille, liste, representant l'etat du jeu
        sortie : lesSuccesseurs, liste de listes, correspondant aux positions ac
    """
    pass

afficher(grille1)
print("Successeurs :")
for g in successeurs(grille1):
    afficher(g)
```

```
 _ _ _
|X| | |
| |X| |
|O|_|O|
 ‾ ‾ ‾

Successeurs :

Traceback (most recent call last):
  File "<input>", line 10, in <module>
TypeError: 'NoneType' object is not iterable
```

Entrée[22]:

```
def successeurs(grille):
    """ successeurs( grille : list ) ->list
        entrees : grille, liste, representant l'etat du jeu
        sortie : lesSuccesseurs, liste de listes, correspondant aux positions ac
    """
    lesSuccesseurs = []
    if gagnant(grille) != 0: # S'il y a un gagnant
        return lesSuccesseurs

    numJoueur = joueur(grille) #recup du numéro de joueur
    lesCasesLibres = casesLibres(grille) #recup de l'ensemble des positions des
    for i, j in lesCasesLibres:
        g = copie(grille)
        g[i][j] = numJoueur
        lesSuccesseurs.append(g)
    return lesSuccesseurs


afficher(grille1)
print("Successeurs :")
for g in successeurs(grille1):
    afficher(g)

```

```
 _ _ _
|X| | |
| |X| |
|O|_|O|
 ‾ ‾ ‾
Successeurs :
 _ _ _
|X|X| |
| |X| |
|O|_|O|
 ‾ ‾ ‾

 _ _ _
|X| |X|
| |X| |
|O|_|O|
 ‾ ‾ ‾

 _ _ _
|X| | |
|X|X| |
|O|_|O|
 ‾ ‾ ‾

 _ _ _
|X| | |
| |X|X|
|O|_|O|
 ‾ ‾ ‾

 _ _ _
|X| | |
| |X| |
|O|X|O|
 ‾ ‾ ‾
```

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.

```
```

```
Correction : Un type mutable (comme une liste) ne peut pas faire office de 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[23]:

```
def estGagnante(g,dico):
    """ estGagnante(g : list,dico : dict) : bool
        fonction récursive
        entrees : g, liste, representant l'etat du jeu
                : dico, dictionnaire au format(tuple de la grille, booleen)- Mod
        sortie : booleen, à True si g est une position gagnante, False sinon
    """
    t = grilleVersTuple(g)
    if t not in dico: #memoisation: pour éviter de retester une grille de jeu de
        # traiter les 3 cas et ajouter dans dico la clé t avec la valeur True ou









    return dico[t]

def attracteur(grille):
    """ attracteur(grille: list) : list
    fonction récursive
        entrees : grille, liste, representant l'etat du jeu
        sortie : liste des tuples qui correspondent à des grilles de positions g
    """
    dico = {}
    # dico[grilleVersTuple(g)] = True si la grille g est une position gagnante p

    est_gagnante(grille,dico)

    return [t for t in dico if dico[t]]
    # liste des tuples des positions gagnantes.


#afficher(grille1)
A=attracteur(grille1)
print(A)
"""
[(1, 1, 2, 2, 1, 1, 2, 1, 2), (1, 1, 2, 2, 1, 1, 2, 0, 2),
(1, 1, 2, 0, 1, 0, 2, 1, 2), (1, 1, 2, 0, 1, 0, 2, 0, 2),
(1, 1, 1, 2, 1, 0, 2, 0, 2), (1, 1, 0, 2, 1, 0, 2, 1, 2),
(1, 1, 0, 2, 1, 0, 2, 0, 2), (1, 1, 1, 0, 1, 2, 2, 0, 2),
(1, 1, 0, 0, 1, 2, 2, 1, 2), (1, 1, 0, 0, 1, 2, 2, 0, 2),
(1, 1, 1, 2, 1, 2, 2, 1, 2), (1, 0, 1, 2, 1, 2, 2, 1, 2),
(1, 0, 1, 2, 1, 0, 2, 0, 2), (1, 0, 1, 0, 1, 2, 2, 0, 2),
(1, 2, 0, 1, 1, 1, 2, 0, 2), (1, 2, 2, 1, 1, 1, 2, 1, 2),
(1, 2, 2, 1, 1, 0, 2, 1, 2), (1, 2, 0, 1, 1, 0, 2, 0, 2),
(1, 0, 2, 1, 1, 1, 2, 0, 2), (1, 0, 2, 1, 1, 0, 2, 0, 2),
(1, 2, 2, 0, 1, 1, 2, 1, 2), (1, 2, 0, 0, 1, 1, 2, 0, 2),
(1, 0, 2, 2, 1, 1, 2, 1, 2), (1, 0, 2, 0, 1, 1, 2, 1, 2),
(1, 0, 2, 0, 1, 1, 2, 0, 2), (1, 0, 2, 0, 1, 0, 2, 1, 2),
(1, 0, 0, 2, 1, 0, 2, 1, 2), (1, 0, 0, 0, 1, 2, 2, 1, 2)]
"""
for t in A:
    afficher(tupleVersGrille(t))

```

```
  File "<input>", line 16
    return dico[t]
    ^^^^^^
IndentationError: expected an indented block after 'if' statement on line 9
```

Entrée[24]:

```
def estGagnante(g,dico):
    t = grilleVersTuple(g)
    if t not in dico:
        if gagnant(g) == 1: # Le joueur 1 a gagné
            dico[t] = True
        elif joueur(g) == 1: # Le joueur 1 joue vers une position gagnante
            succ = successeurs(g)
            L= [estGagnante(g1,dico) for g1 in succ]
            # dico[t] = any(est_gagnante(g1,dico) for g1 in successeurs(g))
            dico[t] = any(L)
        else: # Le joueur 2 ne peut jouer que des positions gagnantes pour 1
            succ = successeurs(g)
            L= [estGagnante(g1,dico) for g1 in succ]
            dico[t] = succ != [] and all(L)
    return dico[t]

def attracteur(grille):
    dico = {}
    # dico[grilleVersTuple(g)] = True si la grille g est une position gagnante p

    estGagnante(grille,dico)

    return [t for t in dico if dico[t]]
    # liste des tuples des positions gagnantes.

afficher(grille1)
A=attracteur(grille1)
print(A)
"""
[(1, 1, 2, 2, 1, 1, 2, 1, 2), (1, 1, 2, 2, 1, 1, 2, 0, 2),
(1, 1, 2, 0, 1, 0, 2, 1, 2), (1, 1, 2, 0, 1, 0, 2, 0, 2),
(1, 1, 1, 2, 1, 0, 2, 0, 2), (1, 1, 0, 2, 1, 0, 2, 1, 2),
(1, 1, 0, 2, 1, 0, 2, 0, 2), (1, 1, 1, 0, 1, 2, 2, 0, 2),
(1, 1, 0, 0, 1, 2, 2, 1, 2), (1, 1, 0, 0, 1, 2, 2, 0, 2),
(1, 1, 1, 2, 1, 2, 2, 1, 2), (1, 0, 1, 2, 1, 2, 2, 1, 2),
(1, 0, 1, 2, 1, 0, 2, 0, 2), (1, 0, 1, 0, 1, 2, 2, 0, 2),
(1, 2, 0, 1, 1, 1, 2, 0, 2), (1, 2, 2, 1, 1, 1, 2, 1, 2),
(1, 2, 2, 1, 1, 0, 2, 1, 2), (1, 2, 0, 1, 1, 0, 2, 0, 2),
(1, 0, 2, 1, 1, 1, 2, 0, 2), (1, 0, 2, 1, 1, 0, 2, 0, 2),
(1, 2, 2, 0, 1, 1, 2, 1, 2), (1, 2, 0, 0, 1, 1, 2, 0, 2),
(1, 0, 2, 2, 1, 1, 2, 1, 2), (1, 0, 2, 0, 1, 1, 2, 1, 2),
(1, 0, 2, 0, 1, 1, 2, 0, 2), (1, 0, 2, 0, 1, 0, 2, 1, 2),
(1, 0, 0, 2, 1, 0, 2, 1, 2), (1, 0, 0, 0, 1, 2, 2, 1, 2)]
"""
for t in A:
    afficher(tupleVersGrille(t))

```

```
 _ _ _
|X| | |
| |X| |
|O|_|O|
 ‾ ‾ ‾
```

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

```
 ‾ ‾ ‾
|X|X|O|
```

```
|o|x|x|
|o|x|o|
```

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[25]:

```
def strategie(grille):
    """
        entree : grille, liste de listes, representant l'etat du jeu
        sortie : dictionnaire qui établit pour chaque position gagnante,
          le coup suivant à jouer (faisant partie des positions gagnantes )
                au format(tuple de la position, liste de la position suivante a
    """
    # détermination des positions gagnantes

    # creation d'un dictionnaire

    #parcours de chacune des positions gagnantes pour déterminer le coup suivant

    # retour du dictionnaire

print(strategie(grille1))
"""
{(1, 1, 2, 2, 1, 1, 2, 0, 2): [[1, 1, 2], [2, 1, 1], [2, 1, 2]],
(1, 1, 2, 0, 1, 0, 2, 0, 2): [[1, 1, 2], [0, 1, 0], [2, 1, 2]],
(1, 1, 0, 2, 1, 0, 2, 0, 2): [[1, 1, 1], [2, 1, 0], [2, 0, 2]],
(1, 1, 0, 0, 1, 2, 2, 0, 2): [[1, 1, 1], [0, 1, 2], [2, 0, 2]],
(1, 0, 1, 2, 1, 2, 2, 1, 2): [[1, 1, 1], [2, 1, 2], [2, 1, 2]],
(1, 0, 1, 2, 1, 0, 2, 0, 2): [[1, 1, 1], [2, 1, 0], [2, 0, 2]],
(1, 0, 1, 0, 1, 2, 2, 0, 2): [[1, 1, 1], [0, 1, 2], [2, 0, 2]],
(1, 2, 2, 1, 1, 0, 2, 1, 2): [[1, 2, 2], [1, 1, 1], [2, 1, 2]],
(1, 2, 0, 1, 1, 0, 2, 0, 2): [[1, 2, 0], [1, 1, 1], [2, 0, 2]],
(1, 0, 2, 1, 1, 0, 2, 0, 2): [[1, 0, 2], [1, 1, 1], [2, 0, 2]],
(1, 2, 2, 0, 1, 1, 2, 1, 2): [[1, 2, 2], [1, 1, 1], [2, 1, 2]],
(1, 2, 0, 0, 1, 1, 2, 0, 2): [[1, 2, 0], [1, 1, 1], [2, 0, 2]],
(1, 0, 2, 2, 1, 1, 2, 1, 2): [[1, 1, 2], [2, 1, 1], [2, 1, 2]],
(1, 0, 2, 0, 1, 1, 2, 0, 2): [[1, 0, 2], [1, 1, 1], [2, 0, 2]],
(1, 0, 2, 0, 1, 0, 2, 1, 2): [[1, 1, 2], [0, 1, 0], [2, 1, 2]],
(1, 0, 0, 2, 1, 0, 2, 1, 2): [[1, 1, 0], [2, 1, 0], [2, 1, 2]],
(1, 0, 0, 0, 1, 2, 2, 1, 2): [[1, 1, 0], [0, 1, 2], [2, 1, 2]]}
"""
```

None

Sortie[25]:

```
'\n{(1, 1, 2, 2, 1, 1, 2, 0, 2): [[1, 1, 2], [2, 1, 1], [2, 1, 2]], \n(1, 1, 2, 0,
1, 0, 2, 0, 2): [[1, 1, 2], [0, 1, 0], [2, 1, 2]], \n(1, 1, 0, 2, 1, 0, 2, 0, 2):
[[1, 1, 1], [2, 1, 0], [2, 0, 2]], \n(1, 1, 0, 0, 1, 2, 2, 0, 2): [[1, 1, 1], [0,
1, 2], [2, 0, 2]], \n(1, 0, 1, 2, 1, 2, 2, 1, 2): [[1, 1, 1], [2, 1, 2], [2, 1,
2]], \n(1, 0, 1, 2, 1, 0, 2, 0, 2): [[1, 1, 1], [2, 1, 0], [2, 0, 2]], \n(1, 0, 1,
0, 1, 2, 2, 0, 2): [[1, 1, 1], [0, 1, 2], [2, 0, 2]], \n(1, 2, 2, 1, 1, 0, 2, 1,
2): [[1, 2, 2], [1, 1, 1], [2, 1, 2]], \n(1, 2, 0, 1, 1, 0, 2, 0, 2): [[1, 2, 0],
[1, 1, 1], [2, 0, 2]], \n(1, 0, 2, 1, 1, 0, 2, 0, 2): [[1, 0, 2], [1, 1, 1], [2, 0,
2]], \n(1, 2, 2, 0, 1, 1, 2, 1, 2): [[1, 2, 2], [1, 1, 1], [2, 1, 2]], \n(1, 2, 0,
0, 1, 1, 2, 0, 2): [[1, 2, 0], [1, 1, 1], [2, 0, 2]], \n(1, 0, 2, 2, 1, 1, 2, 1,
2): [[1, 1, 2], [2, 1, 1], [2, 1, 2]], \n(1, 0, 2, 0, 1, 1, 2, 0, 2): [[1, 0, 2],
[1, 1, 1], [2, 0, 2]], \n(1, 0, 2, 0, 1, 0, 2, 1, 2): [[1, 1, 2], [0, 1, 0], [2, 1,
2]], \n(1, 0, 0, 2, 1, 0, 2, 1, 2): [[1, 1, 0], [2, 1, 0], [2, 1, 2]], \n(1, 0, 0,
0, 1, 2, 2, 1, 2): [[1, 1, 0], [0, 1, 2], [2, 1, 2]]}\n'
```

Entrée[26]:

```
def coup_gagnant(g,dico,positions_gagnantes):
    "Mémoïse un coup gagnant à partir de g lorsqu'il existe."
    t = grilleVersTuple(g)
    if t not in dico and gagnant(g) == 0 and joueur(g) == 1 and t in positions_g
            # Le joueur 1 joue vers une position gagnante
            for g1 in successeurs(g): # chercher un coup gagnant
                if grilleVersTuple(g1) in positions_gagnantes:
                    dico[t] = g1 # le coup à jouer dans cette stratégie
                    return
def strategie(grille):
    """
        entree : grille, liste de listes, representant l'etat du jeu
        sortie : dictionnaire qui établit pour chaque position gagnante,
          le coup suivant à jouer (faisant partie des positions gagnantes )
                au format(tuple de la position, liste de la position suivante a
    """
    positions_gagnantes = attracteur(grille)
    dico = {}

    for pos in positions_gagnantes:
        g = tupleVersGrille(pos)
        coup_gagnant( g, dico,positions_gagnantes )

    return dico
print(strategie(grille1))
"""
{(1, 1, 2, 2, 1, 1, 2, 0, 2): [[1, 1, 2], [2, 1, 1], [2, 1, 2]],
(1, 1, 2, 0, 1, 0, 2, 0, 2): [[1, 1, 2], [0, 1, 0], [2, 1, 2]],
(1, 1, 0, 2, 1, 0, 2, 0, 2): [[1, 1, 1], [2, 1, 0], [2, 0, 2]],
(1, 1, 0, 0, 1, 2, 2, 0, 2): [[1, 1, 1], [0, 1, 2], [2, 0, 2]],
(1, 0, 1, 2, 1, 2, 2, 1, 2): [[1, 1, 1], [2, 1, 2], [2, 1, 2]],
(1, 0, 1, 2, 1, 0, 2, 0, 2): [[1, 1, 1], [2, 1, 0], [2, 0, 2]],
(1, 0, 1, 0, 1, 2, 2, 0, 2): [[1, 1, 1], [0, 1, 2], [2, 0, 2]],
(1, 2, 2, 1, 1, 0, 2, 1, 2): [[1, 2, 2], [1, 1, 1], [2, 1, 2]],
(1, 2, 0, 1, 1, 0, 2, 0, 2): [[1, 2, 0], [1, 1, 1], [2, 0, 2]],
(1, 0, 2, 1, 1, 0, 2, 0, 2): [[1, 0, 2], [1, 1, 1], [2, 0, 2]],
(1, 2, 2, 0, 1, 1, 2, 1, 2): [[1, 2, 2], [1, 1, 1], [2, 1, 2]],
(1, 2, 0, 0, 1, 1, 2, 0, 2): [[1, 2, 0], [1, 1, 1], [2, 0, 2]],
(1, 0, 2, 2, 1, 1, 2, 1, 2): [[1, 1, 2], [2, 1, 1], [2, 1, 2]],
(1, 0, 2, 0, 1, 1, 2, 0, 2): [[1, 0, 2], [1, 1, 1], [2, 0, 2]],
(1, 0, 2, 0, 1, 0, 2, 1, 2): [[1, 1, 2], [0, 1, 0], [2, 1, 2]],
(1, 0, 0, 2, 1, 0, 2, 1, 2): [[1, 1, 0], [2, 1, 0], [2, 1, 2]],
(1, 0, 0, 0, 1, 2, 2, 1, 2): [[1, 1, 0], [0, 1, 2], [2, 1, 2]]}
"""
```

```
{(1, 1, 2, 2, 1, 1, 2, 0, 2): [[1, 1, 2], [2, 1, 1], [2, 1, 2]], (1, 1, 2, 0, 1, 0,
2, 0, 2): [[1, 1, 2], [0, 1, 0], [2, 1, 2]], (1, 1, 0, 2, 1, 0, 2, 0, 2): [[1, 1,
1], [2, 1, 0], [2, 0, 2]], (1, 1, 0, 0, 1, 2, 2, 0, 2): [[1, 1, 1], [0, 1, 2], [2,
0, 2]], (1, 0, 1, 2, 1, 2, 2, 1, 2): [[1, 1, 1], [2, 1, 2], [2, 1, 2]], (1, 0, 1,
2, 1, 0, 2, 0, 2): [[1, 1, 1], [2, 1, 0], [2, 0, 2]], (1, 0, 1, 0, 1, 2, 2, 0, 2):
[[1, 1, 1], [0, 1, 2], [2, 0, 2]], (1, 2, 2, 1, 1, 0, 2, 1, 2): [[1, 2, 2], [1, 1,
1], [2, 1, 2]], (1, 2, 0, 1, 1, 0, 2, 0, 2): [[1, 2, 0], [1, 1, 1], [2, 0, 2]], (1,
0, 2, 1, 1, 0, 2, 0, 2): [[1, 0, 2], [1, 1, 1], [2, 0, 2]], (1, 2, 2, 0, 1, 1, 2,
1, 2): [[1, 2, 2], [1, 1, 1], [2, 1, 2]], (1, 2, 0, 0, 1, 1, 2, 0, 2): [[1, 2, 0],
[1, 1, 1], [2, 0, 2]], (1, 0, 2, 2, 1, 1, 2, 1, 2): [[1, 1, 2], [2, 1, 1], [2, 1,
2]], (1, 0, 2, 0, 1, 1, 2, 0, 2): [[1, 0, 2], [1, 1, 1], [2, 0, 2]], (1, 0, 2, 0,
1, 0, 2, 1, 2): [[1, 1, 2], [0, 1, 0], [2, 1, 2]], (1, 0, 0, 2, 1, 0, 2, 1, 2):
[[1, 1, 0], [2, 1, 0], [2, 1, 2]], (1, 0, 0, 0, 1, 2, 2, 1, 2): [[1, 1, 0], [0, 1,
2], [2, 1, 2]]}
```

Sortie[26]:

```
'\n{(1, 1, 2, 2, 1, 1, 2, 0, 2): [[1, 1, 2], [2, 1, 1], [2, 1, 2]], \n(1, 1, 2, 0,
1, 0, 2, 0, 2): [[1, 1, 2], [0, 1, 0], [2, 1, 2]], \n(1, 1, 0, 2, 1, 0, 2, 0, 2):
[[1, 1, 1], [2, 1, 0], [2, 0, 2]], \n(1, 1, 0, 0, 1, 2, 2, 0, 2): [[1, 1, 1], [0,
1, 2], [2, 0, 2]], \n(1, 0, 1, 2, 1, 2, 2, 1, 2): [[1, 1, 1], [2, 1, 2], [2, 1,
2]], \n(1, 0, 1, 2, 1, 0, 2, 0, 2): [[1, 1, 1], [2, 1, 0], [2, 0, 2]], \n(1, 0, 1,
```

```
0, 1, 2, 2, 0, 2): [[1, 1, 1], [0, 1, 2], [2, 0, 2]], \n(1, 2, 2, 1, 1, 0, 2, 1,
2): [[1, 2, 2], [1, 1, 1], [2, 1, 2]], \n(1, 2, 0, 1, 1, 0, 2, 0, 2): [[1, 2, 0],
[1, 1, 1], [2, 0, 2]], \n(1, 0, 2, 1, 1, 0, 2, 0, 2): [[1, 0, 2], [1, 1, 1], [2, 0,
2]], \n(1, 2, 2, 0, 1, 1, 2, 1, 2): [[1, 2, 2], [1, 1, 1], [2, 1, 2]], \n(1, 2, 0,
0, 1, 1, 2, 0, 2): [[1, 2, 0], [1, 1, 1], [2, 0, 2]], \n(1, 0, 2, 2, 1, 1, 2, 1,
2): [[1, 1, 2], [2, 1, 1], [2, 1, 2]], \n(1, 0, 2, 0, 1, 1, 2, 0, 2): [[1, 0, 2],
[1, 1, 1], [2, 0, 2]], \n(1, 0, 2, 0, 1, 0, 2, 1, 2): [[1, 1, 2], [0, 1, 0], [2, 1,
2]], \n(1, 0, 0, 2, 1, 0, 2, 1, 2): [[1, 1, 0], [2, 1, 0], [2, 1, 2]], \n(1, 0, 0,
0, 1, 2, 2, 1, 2): [[1, 1, 0], [0, 1, 2], [2, 1, 2]]}\n'
```

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

Entrée[27]:

```
def jeu(grille):
    dico = strategie(grille)
    while gagnant(grille) == 0 and len(casesLibres(grille)) > 0:
        t = grilleVersTuple(grille)
        if joueur(grille) == 2:
            afficher(grille)
            i, j = -1, -1
            while i not in (1, 2, 3) or j not in (1, 2, 3) or grille[i - 1][j -
                i, j = map(int, list(input("Coordonnées de votre coup de la forr
            grille[i - 1][j - 1] = 2
        else:
            if t not in dico:
                print("Pas de stratégie gagnante")
                return
            grille = dico[t]
    gagne = gagnant(grille)
    afficher(grille)
    if gagne != 0:
        print(f"Le joueur {gagne} a gagné !")
    else:
        print("Match nul.")

grille_vide = [[0, 0, 0] for _ in range(3)]
jeu(grille_vide)

```

```
Pas de stratégie gagnante
```

Entrée[28]:

```
def jeu(grille):
    dico = strategie(grille)
    while gagnant(grille) == 0 and len(casesLibres(grille)) > 0:
        t = grilleVersTuple(grille)
        if joueur(grille) == 2:
            afficher(grille)
            i, j = -1, -1
            while i not in (1, 2, 3) or j not in (1, 2, 3) or grille[i - 1][j -
                i, j = map(int, list(input("Coordonnées de votre coup de la forr
            grille[i - 1][j - 1] = 2
        else:
            if t not in dico:
                print("Pas de stratégie gagnante")
                #===== ajout dans le cas ou pas de stratégie gagnante
                lesG= successeurs(grille)
                nbSucc= len(lesG)
                if nbSucc>0 :
                    grille = lesG[0] #on prend par defaut le 1er successeur (on
                # choix de la 1ere position des successeurs
                 #return
                #===== ajout dans le cas ou pas de stratégie gagnante
            else :
                grille = dico[t]
    gagne = gagnant(grille)
    afficher(grille)
    if gagne != 0:
        print(f"Le joueur {gagne} a gagné !")
    else:
        print("Match nul.")

grille_vide = [[0, 0, 0] for _ in range(3)]
jeu(grille_vide)

```

```
Pas de stratégie gagnante
 _ _ _
|X| | |
| | | |
|_|_|_|

Coordonnées de votre coup de la forme LigneColonne : None

Traceback (most recent call last):
  File "<input>", line 32, in <module>
  File "<input>", line 9, in jeu
TypeError: 'NoneType' object is not iterable
```

Entrée[ ]:

```

```

Entrée[ ]:

```

```