TP #7 — Stratégie gagnante pour le jeu du Morpion

Adapté d'un TP de Jérémy Larochette

Le but de ce TP est d'implémenter le calcul des attracteurs « de haut en bas » pour le jeu du Morpion, et d'utiliser ceux-ci pour implémenter une « intelligence artificielle » qui ne peut jamais perdre.
Commencez par télécharger le fichier morpion_base.py qui définit certaines fonctions auxiliaires utiles.

Règles du jeu

Dans le jeu de tic-tac-toe (ou morpion), deux 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

On représente une grille par une `list` de `list` 3×3 contenant 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)

On donne dans le fichier Python des fonctions :

- `afficher(grille:list[list[int]])` qui réalise un affichage textuel de la grille `grille`
- `gagnant(grille:list[list[int]]) -> int` qui renvoie le numéro du joueur gagnant dans `grille` s'il en existe un, et `0` sinon
- `grille2tuple(grille:list[list[int]]) -> tuple[int]` qui renvoie un `tuple` de taille 9 dont les éléments sont ceux de `grille` énumérés par ligne
- `tuple2grille(grille:tuple[int]) -> list[list[int]]` qui renvoie une grille (sous forme de `list[list[int]]`) dont les éléments sont ceux du `tuple` `grille` de taille 9
- `copie(grille:list[list[int]]) -> list[list[int]]` qui crée et renvoie une copie **profonde** de `grille`, sans la modifier.

ainsi qu'un exemple de grille `grille1`.

```
>>> afficher(grille1)
 _ _ _
|X| | |
| |X| |
|O| |O|
 - - -
>>> gagnant(grille1)
0
>>> grille2 = [[1, 0, 2], [2, 1, 0], [0, 0, 1]]
>>> afficher(grille2)
 _ _ _
|X| |O|
|O|X| |
| | |X|
 - - -
>>> gagnant(grille2)
1
>>> grille2tuple(grille1)
(1, 0, 0, 0, 1, 0, 2, 0, 2)
>>> tuple2grille((1, 0, 0, 0, 1, 0, 2, 0, 2))
[[1, 0, 0], [0, 1, 0], [2, 0, 2]]
```

1. Écrivez une fonction `cases_libres(grille)` renvoyant la liste des positions `(i, j)` des cases vides de `grille` (c'est-à-dire telles que `grille[i][j]` vaut 0).

   ```
   >>> cases_libres(grille1)
   [(0, 1), (0, 2), (1, 0), (1, 2), (2, 1)]
   ```

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

   ```
   >>> joueur(grille1)
   1
   ```

Calcul d'attracteurs

Nous allons d'abord définir une fonction auxiliaire pratique.

3. Écrivez une fonction `successeurs(grille)` renvoyant la liste des grilles que l'on peut obtenir à partir de `grille` en jouant *un* (unique) coup (légal). *Conseils* :

— on rappelle qu'on a fourni une fonction permettant faire une copie de grille
— faites attention à bien prendre en compte l'identité du prochain joueur dont c'est le tour, et les cas où l'un des joueurs a (déjà) gagné

```
>>> successeurs(grille1)
[[[1, 1, 0], [0, 1, 0], [2, 0, 2]],
 [[1, 0, 1], [0, 1, 0], [2, 0, 2]],
 [[1, 0, 0], [1, 1, 0], [2, 0, 2]],
 [[1, 0, 0], [0, 1, 1], [2, 0, 2]],
 [[1, 0, 0], [0, 1, 0], [2, 1, 2]]]
```

Dans un calcul récursif d'attracteur de haut-en-bas, il peut arriver qu'on effectue un appel récursif de multiples fois pour la même grille. Pour éviter tout recalcul inutile, on propose d'utiliser un dictionnaire (de type `dict`) pour mémoïser les résultats.

4. Pourquoi ne peut-on pas utiliser une grille (représentée comme liste) comme clef d'un dictionnaire ? Comment peut on résoudre ce problème, par exemple en utilisant des fonctions déjà fournies ?

On commence par calculer un attracteur pour le joueur 1. Pour cela on va définir une fonction de victoire pour ce joueur en utilisant la récursion suivante :

— Cas de base :
  — une grille gagnante pour le joueur 1 est une position de victoire (pour le joueur 1)
  — une grille gagnante pour le joueur 2 n'est pas une position de victoire
  — une grille gagnante pour aucun joueur telle qu'aucun joueur ne peut jouer (autrement dit, une grille complète de match nul) n'est pas une position de victoire

— Cas récursifs :
  — si c'est au joueur 1 de jouer, une grille g est une position de victoire (pour le joueur 1) ss.'il **existe** une grille successeure de g qui est une position de victoire (pour le joueur 1)
  — si c'est au joueur 2 de jouer, une grille g est une position de victoire (pour le joueur 1) ssi. **toutes** les grilles successeures de g sont des positions de victoire (pour le joueur 1)

5. Écrivez une fonction récursive mémoïsante :

```
estPosVictoire1(grille:list[list[int]], gmem:dict)
-> bool
```

qui prend en entrée une grille `grille`, un dictionnaire `gmem` utilisé pour la mémoïsation et qui :

— modifie au besoin `gmem` de façon à ce qu'après que l'appel `estPosVictoire1(grille, gmem)` a retourné, on a `gmem[gt]` égal à `True` si la grille représentée par `grille` est une position de victoire pour le joueur 1 (comme définie ci-dessus) et `False` sinon, où `gt` est une clef bien choisie pour représenter `grille` dans le dictionnaire
— renvoie `True` si la grille représentée par `grille` est une position de victoire pour le joueur 1, et `False` sinon

```
>>> estPosVictoire1(grille1, {})
False
```

Conseil : bien que ce ne soit pas nécessaire pour `estPosVictoire1` elle-même, il est très utile (pour répondre à la question suivante) de l'implémenter de façon *non paresseuse* : même s'il est parfois possible de décider si une position est une position de victoire sans explorer tous ses successeurs, mieux vaut néanmoins **tous** les explorer.

6. Utilisez votre fonction `estPosVictoire1` pour écrire une fonction `attracteur1(grille:list[list[int]])` qui prend en entrée une grille `grille`, calcule et renvoie un dictionnaire `wp1` tel que la clef `gt` est présente dans `wp1` (avec une valeur associée quelconque ; par exemple `True`) ssi. `gt` est une clef représentant une grille qui :

— est accessible depuis la `grille` par une suite de mouvements légaux
— et est une position de victoire pour le joueur 1 (dans le même sens qu'à la question précédente)

```
>>> wp1 = attracteur1(grille1)
>>> (1, 0, 0, 2, 1, 0, 2, 1, 2) in wp1
True
>>> # victoire non garantie
>>> # en cas de jeu optimal par J2
>>> (1, 0, 0, 0, 1, 0, 2, 1, 2) in wp1
False
>>> # victoire garantie mais pas une
>>> # position légalement atteignable
>>> (1, 0, 0, 2, 1, 0, 2, 0, 0) in wp1
False
>>> len(wp1)
28
```

7. Écrivez des fonctions `estPosVictoire2` et `attracteur2` similaires à `estPosVictoire1` et `attracteur1`, mais pour le joueur 2.

```
>>> wp2 = attracteur2(grille1)
>>> len(wp2)
29
```

8. Complétez la fonction `jeu` dont le squelette est donné dans le fichier fourni de façon à ce que le joueur 1 (contrôlé par la fonction) gagne toujours quand cela est possible, et fasse sinon match nul.
Il suffit pour cela de calculer les attracteurs `wp1` et `wp2` des deux joueurs et :

— jouer vers une position dans `wp1` quand cela est possible
— ne jamais jouer vers une position dans `wp2` (pour le jeu du morpion, c'est toujours une possible)

L'interface fournie par la fonction `jeu` est un peu fruste. N'hésitez pas si besoin à demander de l'aide pour l'utiliser