

TP 4 – Jeux d’accessibilité

Gaëtan Staquet

2026

Ce TP est découpé en deux parties. La première se concentre sur le **morpion** et sa modélisation sous forme de jeu d’accessibilité. La seconde porte sur l’implémentation de l’algorithme de l’attracteur. À la fin, vous aurez une implémentation à la complexité optimale et produirez une preuve assistée par ordinateur d’un théorème.

Contrairement aux autres TP, vous allez repartir d’un projet vide. Cependant, les autres consignes sont toujours d’application :

- Le langage à utiliser pendant les TP est Python.
- Vous êtes libre d’utiliser les outils que vous voulez, y inclus des LLM et autres IA génératives.
- Veuillez limiter votre usage de modules externes. En particulier, les bibliothèques de graphes (comme **NetworkX** ou **graph-tool**) sont interdites, étant donné que le but est d’implémenter les algorithmes par vous-même.

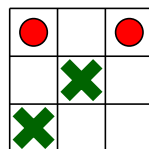
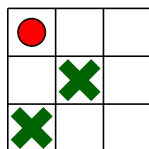
Partie I — Morpion

Le **morpion** (*tic-tac-toe*, *noughts and crosses* ou *Xs and Os* en anglais) est un jeu de stratégie à deux joueurs sur une grille de 3 par 3 cases. Le but est d’aligner trois fois le même symbole.

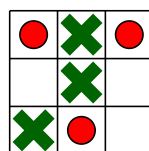
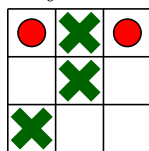
Règles du jeu. Dans le cadre de ce TP, on suppose qu’Ève possède le symbole \times (en vert sur les figures) et Adam \bigcirc (en rouge). Ève est toujours **la première à jouer**. Initialement, la grille **est vide**.

À tour de rôle, Ève et Adam remplissent une case de la grille avec leur symbole. Il ne peut pas y avoir deux symboles par case, *i. e.*, une case ne peut être remplie que si elle est vide. La Figure 1 donne un exemple d’évolution de la configuration du jeu. On suppose qu’Ève a déjà placé deux symboles et Adam un seul (Figure 1a). C’est donc au tour d’Adam de jouer. Il décide de placer un rond dans la case en haut à droite (Figure 1b). Ève continue en mettant une croix dans la case en haut au milieu (Figure 1c). Par la suite, Adam prend la case en bas au milieu (Figure 1d).

Le jeu continue ensuite de cette manière, jusqu’à ce qu’une des trois conditions suivantes soit remplie :



- (a) Configuration de départ pour cet exemple. Ève a posé deux jetons et Adam un. (b) Adam met un symbole dans la case en haut à droite.

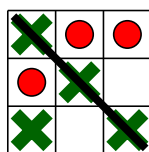


- (c) Ève met un symbole dans la case en haut au milieu. (d) Adam met un symbole dans la case en bas au milieu.

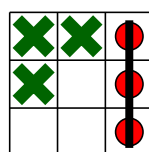
FIGURE 1 – Évolution de la configuration du morpion.

- Ève parvient à aligner trois croix, auquel cas **Ève gagne**,
- Adam parvient à aligner trois ronds, auquel cas **Adam gagne**,
- la grille est complète et il n'y a pas trois symboles identiques alignés, auquel cas **il y a égalité**.

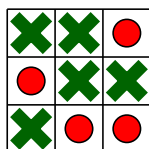
La Figure 2 donne quelques exemples de configurations de fin de jeu.



- (a) Ève gagne.



- (b) Adam gagne.



- (c) Égalité : personne ne gagne.

FIGURE 2 – Exemples de fins de partie pour le morpion.

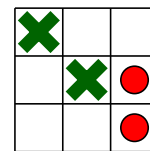
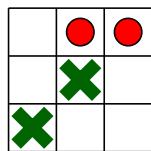
On peut assez naturellement étendre ce jeu pour devoir aligner k symboles sur une grille de $n \times m$ cases.

I.1. But de ce TP

À la fin de ce TP, vous produirez une preuve par ordinateur du théorème suivant.

Théorème I.1.1. *En supposant qu'Adam joue parfaitement, Ève peut au mieux forcer une égalité au morpion.*

Étant donné le nombre important de configurations possibles, énumérer toutes les parties possibles à la main est infaisable. On pourrait approcher la preuve en remarquant que les symétries et les rotations n'ont pas d'importance. Par exemple, les deux grilles suivantes représentent fondamentalement la même situation : il suffit de tourner la grille de 90° .



Cependant, nous allons exploiter une autre possibilité de preuve : en utilisant la puissance de calcul des ordinateurs, on peut explicitement explorer l'entière des configurations possibles et vérifier si Ève a une stratégie qui lui permet de gagner ou, au mieux, de forcer l'égalité, ou si elle n'a aucun espoir, vu qu'Adam peut gagner.

I.2. Arène pour le morpion

Pour rappel, une **arène** est un tuple $\mathcal{A} = (V, E, V_E, V_A)$ où

- (V, E) est un graphe fini orienté tel que, pour tout $v \in V$, $\delta^{\rightarrow}(v) \geq 1$;
- V_E et V_A forment une partition de V , *i. e.*, $V_E \cup V_A = V$ et $V_E \cap V_A = \emptyset$.

On peut construire un jeu d'accessibilité dont l'arène est l'ensemble des configurations atteignables depuis la grille vide :

- Le sommet de départ correspond à la grille vide et appartient à Ève. Pour chaque case, on calcule la configuration obtenue en remplissant la case avec le symbole d'Ève. Cette configuration est un nouveau sommet qui appartient à Adam.
- On réapplique la même idée depuis les sommets d'Adam : à chaque fois, on regarde chaque case et on calcule la configuration dans laquelle on arrive, ce qui donne autant de sommets qui appartiennent à Ève qu'il y a de cases vides.
- On répète ceci jusqu'à ce qu'Ève ou Adam gagne, ou jusqu'à ce que toutes les cases soient remplies. Pour avoir une arène comme définie en cours, on rajoute un arc (v, v) pour chaque sommet v correspondant à une fin de jeu.

Vous allez implémenter cette idée en deux temps. D'abord, vous construirez un arbre des configurations atteignables. Ensuite, vous exploiterez le fait qu'il est possible d'atteindre une configuration de jeu de plusieurs manières. Ainsi, un graphe orienté permet de réduire le nombre de sommets par rapport à un arbre.

Avant de travailler là-dessus, il est important de justifier l'intérêt de le faire.

Exercice I.2.1. Reformulez le Théorème I.1.1 en utilisant des termes de la théorie des jeux (sur graphe).

Comment calculer la région gagnante pour Ève ? Comment calculer l'ensemble des configurations depuis lesquels Ève peut garantir une égalité et pas de victoire ?

Exercice I.2.2. Décrivez (au moins) deux manières d'implémenter la partition des sommets en V_E et V_A telle que tester si un sommet appartient à V_E a une complexité en temps constant (en moyenne).

Laquelle vous paraît être la plus simple à implémenter ? Laquelle vous paraît être la plus efficace pendant l'exécution ?

Afin de simplifier l'implémentation, vous trouverez sur la page Hippocampus les scripts Python suivants :

- `arena.py` qui contient une implémentation d'une arène, avec ses sommets et ses arcs. Comment est-ce que ce code détermine si un sommet appartient à Ève ?
- `configuration.py` qui contient une implémentation d'une configuration du jeu du morpion. Étant donné les signatures des méthodes, comment pourriez-vous jouer un coup (*i.e.*, poser un symbole pour Ève ou Adam) ?

Remarquez que la taille de la grille et le nombre de symboles à devoir aligner sont dictés par les constantes `N_ROWS: int`, `N_COLUMNS: int` et `N_TOKENS_IN_LINE: int`.

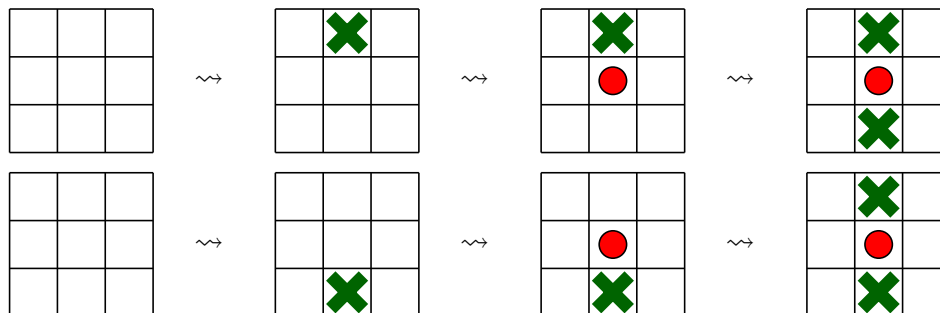
Exercice I.2.3 (Version arbre). Implémentez une fonction qui génère une arène dont le graphe est arbre orienté des parties possibles du morpion. La racine de l'arbre doit être la grille vide et appartenir à Ève.

Quel est le nombre de sommets pour une grille de 2×2 cases, si on suppose qu'il faut aligner **deux symboles identiques** ? Quel est le nombre de sommets pour une grille de 3×3 avec la même condition de victoire ?

Quel est le nombre de sommets pour une grille de 3×3 dans laquelle il faut aligner **trois symboles identiques** ?

Serait-il raisonnable de construire l'arbre pour une grille de 4×4 ?

Comme vous pouvez le constater, l'arène contient beaucoup de sommets et prend du temps à être construite. On peut exploiter le fait que différentes parties finissent dans la même configuration, comme illustré par la figure suivante.



Peu importe le début de la partie, à partir du moment où on retombe sur la même configuration, les coups possibles sont les mêmes. Autrement dit, notre arbre contient beaucoup de configurations qui sont dupliquées. Ainsi, un graphe orienté serait plus approprié.

Exercice I.2.4 (Version graphe orienté). Implémentez une fonction qui génère une arène dont le graphe est l'ensemble des configurations du morpion. Il ne peut pas y avoir deux sommets différents qui correspondent à la même configuration.

Quel est le nombre de sommets pour une grille de 2×2 cases, si on suppose qu'il faut aligner **deux symboles identiques** ? Quel est le nombre de sommets pour une grille de 3×3 avec la même condition de victoire ?

Quel est le nombre de sommets pour une grille de 3×3 dans laquelle il faut aligner **trois symboles identiques** ?

Serait-il raisonnable de construire le graphe pour une grille de 4×4 ?

Exercice I.2.5. Implémentez les fonctions suivantes :

- `get_vertices_eve_winning(arena: Arena)` qui retourne l'ensemble des sommets de l'arène qui sont des configurations gagnantes pour Ève,
- `get_vertices_adam_winning(arena: Arena)` qui retourne l'ensemble des sommets de l'arène qui sont des configurations gagnantes pour Adam,
- `get_vertices_draw(arena: Arena)` qui retourne l'ensemble des sommets de l'arène qui sont des configurations d'égalité (*i.e.*, une fin de partie sans victoire).

Votre code doit pouvoir être appliqué sur une arène pour le morpion, peu importe si elle est en forme d'arbre ou de graphe orienté.

Partie II — Attracteur

Maintenant que nous avons des arènes, il nous faut un algorithme pour déterminer la région gagnante d'Ève. Comme vu en cours, une manière d'y arriver est l'attracteur.

II.1. Implémentation optimale en temps

On va ici implémenter la version optimale de l'algorithme en temps (en sacrifiant un peu de mémoire). Un pseudo-code est donné dans l'Algorithme 1.

Exercice II.1.1. Expliquez avec vos mots le fonctionnement de cet algorithme. Justifiez en quelques mots pourquoi il calcule bien l'ensemble des sommets pour lesquels Ève a une stratégie qui garantit d'atteindre T .

Qu'est-ce qu'il faut modifier dans la fonction pour calculer la région pour Adam ?

Exercice II.1.2. Implémentez l'Algorithme 1. Votre fonction doit permettre de choisir si on se place du coup d'Ève ou d'Adam.

Algorithme 1. Algorithme de l'attracteur

Garantit : \mathcal{A} est une arène et T est un ensemble de sommets

```
1 : procédure ATTRACTEUROPTIMAL( $\mathcal{A} = (V, E, V_E, V_A), T$ )
2 :    $count : V \rightarrow \mathbb{N}^{>0}$  initialement vide
3 :   pour chaque  $v \in V$  faire
4 :     si  $v \in T$  alors
5 :        $count(v) \leftarrow 0$ 
6 :     sinon si  $v \in V_E$  alors
7 :        $count(v) \leftarrow 1$ 
8 :     sinon
9 :        $count(v) \leftarrow \delta^{\rightarrow}(v)$ 
10 :    $W \leftarrow \emptyset$ 
11 :    $file \leftarrow$  une file initialisée avec  $T$ 
12 :   tant que  $file$  n'est pas vide faire
13 :      $v \leftarrow file.RETIRER()$ 
14 :      $W \leftarrow W \cup \{v\}$ 
15 :     pour chaque prédécesseur  $p$  de  $v$  tel que  $count(p) > 0$  faire
16 :        $count(p) \leftarrow count(p) - 1$ 
17 :       si  $count(p) = 0$  alors
18 :          $file.AJOUTER(p)$ 
19 :   retourner  $W$ 
```

II.2. Application au morpion

Exercice II.2.1. Pour une grille de morpion de 2×2 , quel est l'ensemble des sommets à partir desquels Ève peut garantir de gagner ?

Quel est l'ensemble des sommets depuis lesquels Adam peut forcer de gagner ?

Quel est l'ensemble des sommets à partir desquels Ève peut garantir de soit gagner, soit avoir une égalité ? Comment en déterminer l'ensemble des sommets depuis lesquels Ève peut forcer une **égalité sans gagner** ?

Dans quel ensemble se trouve la grille vide ?

Exercice II.2.2. Pour une grille de morpion de 3×3 , dans quel ensemble se trouve la grille vide ?

II.3. Conclusion

Exercice II.3.1. En utilisant tout ce qui a été fait dans ce TP, comment prouver le Théorème I.1.1 en utilisant un ordinateur ?