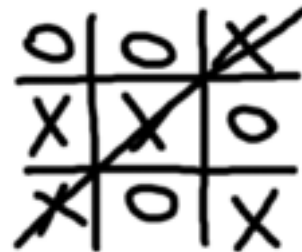


ECL – 2A - S7

2021-2022

Stratégies de Résolution de Problèmes

BE 2 : Morpion (Tic-Tac-Toe)

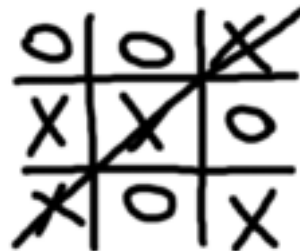


I LE JEU MORPION (TIC-TAC-TOE)

Le principe du jeu en quelques mots :

- Un plateau (grille, échiquier) carré de taille > 2 (variable **Dimension**).
- "Dimension" pions Noirs pour le joueur 1, autant de Blancs pour 2
- Départ : plateau vide
- **But** : placer tous ses pions sur une même ligne, colonne ou diagonale.

Pour Dimension = 3 : une grille 3 x 3, deux joueurs chacun disposant de 3 pions de couleur noire (ou blanche).



I-A Déroulement du jeu

- Quand il a placé 3 pions, **une joueur permute un de ses pions avec une case vide.**
- Le jeux va de cette manière se dérouler jusqu'à trouver un **gagnant**.

	1	2	3
1	●	○	
2	○	○	●
3		●	

figure 1

- **Jeu déclaré nul** : après k tours (par exemple, $k=3$) pour chacun, si la grille se retrouve dans un état précédent, le résultat est déclaré nul et on recommence.

Mais si on veut, on peut calculer des "points" en fonction des positions de chacun (par exemple le centre valant plus que les autres cases) pour déclarer un gagnant.

I-B Aspects pratiques

- Une numérotation de la grille par le couple (L, C) (Ligne, Colonne).
- Le couple (L, C) peut être représenté par une *paire* (L, C) ou par un vecteur à 2 éléments.
- Voir le sujet.

1/1	1/2	1/3
2/1	2/2	2/3
3/1	3/2	3/3

figure2

Une variante du jeu : version de débutant et pour **Dimension** > 3 :

On accepte la fin d'une partie (état de gain) si l'un des joueurs a placé *Dimension* -1 pions dans les mêmes dispositions que ci-dessus.

➔ Par exemple, pour une Dimension=4, un joueur aura gagné s'il a placé 3 pions correctement.

I-C L'algorithme de principe du jeu

Procédure jouer :

Entrées : Grille G ; // Pas de sortie, c'est une procédure = action

Début

Choisir une couleur **Col** \leftarrow parmi {B, N} ;

bool Fini=faux ;

Répéter jusqu'à (Finis = vrai)

Col place un pion de sa couleur sur la grille G (voir la Procédure *placer*)

Finis \leftarrow gagnant(E, **Col**) ;

Si (Finis = Faux)

Alors **Col** \leftarrow l'autre_couleur(Col) ; // C-à-d. si (Col=B) alors Col \leftarrow N sinon Col \leftarrow B

Finsi ;

Fin Répéter

Col est le gagnant

Fin jouer

Fonction gagnant : booléen

Entrées : Grille G, Couleur Col // Col est représenté un des 2 joueurs N,B

Sortie : un booléen : vrai = la couleur Col gagne dans la Grille G,
faux = la couleur Col ne gagne pas

Début

S'il y a une ligne/colonne/diagonale de couleur Col sur la grille G

Alors renvoyer Vrai // succès = gagnant

Sinon renvoyer Faux // échec = non gagnant

Fin si

Fin gagnant

L'action de placer un pion de couleur **Co1** sur la Grille G :

Procédure placer :

Entrées : Grille G , Couleur **Co1** ;

Sortie : rien


S'il y a déjà 3 pions de couleur **Co1** sur la grille G

Alors retirer un pion de couleur **Co1** de la Grille G

Fin si

// Dans tous les cas, on se retrouve avec au moins un pion de couleur **Co1** en main !

Occuper une case vide par un pion de couleur **Co1** sur la grille G

 // **Essayer** de ne pas réoccuper la même case si on vient de retirer un pion (voir cas de "boucle")

Fin placer

Occupation d'une case vide sur la grille :

Dans la version de base (la machine joue aléatoirement), on fait un tirage aléatoire pour choisir une case vide puis on l'occupe.

Procédure Occuper_une_case_vide :

Entrées : Grille G , Couleur Col ;

Sortie : rien

Faire un tirage aléatoire parmi les cases vides de la grille G

Et y placer un pion de couleur Col

Fin Occuper_une_case_vide

☞ Le cas où l'humain (vous) doit jouer : pas de tirage aléatoire ?

→ Un test sur Col sera peut-être nécessaire !

☞ Cette procédure peut avoir un paramètre supplémentaire qui sera **la case à ne pas occuper** ! Car si on vient de libérer une case, mieux vaut ne pas la ré-occuper sous peine de tourner en rond.

I-D Remarques

Remarque 1 : si on décide d'éviter de "tourner en rond" (de "boucler"), on peut utiliser la version suivante de la fonction **gagant** (changements en **rouge**) :

Fonction gagnant_bis : booléen

Entrées : Grille G, Couleur Col // Col est représente un des 2 joueurs N,B

Sortie : un booléen : vrai = la couleur Col gagne dans la Grille G,
faux = la couleur Col ne gagne pas

Début

S'il y une ligne/colonne/diagonale de couleur Col sur la grille G

Alors renvoyer Vrai // succès = gagnant

Sinon **Si l'état de la grille a été rencontré il y a 3 tours**

Alors finir jeu // terminer brutalement !

Sinon renvoyer Faux // échec = non gagnant

Fin si

Fin si

Fin gagnant

Remarque 2 : on peut modifier la procédure **placer** et introduire la notion de **permutation** d'un pion avec une case vide sur la grille. Ce qui donnera :

Procédure placer_bis :

Entrées : Grille G , Couleur Col ;

Sortie : rien

S'il y a déjà 3 pions de couleur Col sur la grille G

Alors **Permuter** un pion de couleur Col avec une case vide sur la Grille G

Sinon **Occuper une case vide** par un pion de couleur Col sur la grille G

Fin si

Fin placer_bis

Permutation : ../..

Permutation d'un pion déjà placé de couleur Col avec une case vide :

La permutation = libération d'une case occupée suivie d'un nouveau placement dans une des (3+1) cases vides.

Procédure permuter_avec_une_case_vide :

Entrées : Grille G , Couleur Col ;

// Pour Dimension=3, on a 3 cases vides et 3 cases de couleur Col

// Dans cette version de base, on utilise le hasard (introduit un non-déterminisme)

Ind = random(1..3) // Choix d'une des cases occupées par la couleur Col, à libérer

Libérer la Ind^e case // Qui est forcément de la couleur Col

Occuper une case vide par un pion de couleur Col sur la grille G

Fin permuter_avec_une_case_vide

II REPRÉSENTATION DE LA GRILLE

- L'efficacité de ces algorithmes dépend des structures de données choisies.
 - un élément important : *le choix de la prochaine case à jouer.*
 - des appels fréquents à la procédure **gagnant**. Elle mérite d'être optimisée.

- 3 vecteurs (ou listes) :
 - VB : vecteur des blancs bien placés ($\text{Blanc}=B$)
 - VN : vecteur des noirs (N) bien placés
 - VV : vecteur des cases vides (cases non occupées)
- Supposons que la **Dimension**=3.
 - Au départ, $VB = VN = \text{vide}$
 - Le vecteur VV contient les 9 cases de la grille = $[1/1, \dots, 3/3]$.

1/1	1/2	1/3
2/1	2/2	2/3
3/1	3/2	3/3

figure2

- Si le joueur B (Blanc) doit jouer, il prendra une des cases vides de VV qu'il insère dans LB .
- Le joueur N fera de même.

- Lorsque chaque joueur aura placé ses 3 pions, les 3 vecteurs VB, VN et VV contiennent chacun 3 éléments.

- Par exemple, pour la figure 2 (ci-contre), on a :

$VN=[1/1, 2/3, 3/2]$ le vecteur VN pour les Noirs

$VB=[1/2, 2/1, 2/2]$ pour les Blancs et

$VV=[1/3, 3/1, 3/3]$ pour les cases vides.

	1	2	3
1	●	○	
2	○	○	●
3		●	

figure 1

- **A partir de ce moment dans le jeu**, chaque joueur **permutera** un des éléments de son vecteur (VB pour les Blancs, VN pour les Noirs) avec un des éléments de VV.
- Le non déterminisme du jeu viendra des choix des pions ou des cases à permuter/libérer/placer.

II-A A propos de l'état gagnant pour la couleur X

Pour tester si la grille représentée par les 3 vecteurs correspond au gain des X (par exemple, les B), il suffit de tester :

$VB = [(L,), (L,), (L,)];$ // une ligne L gagnante (* veut dire n'importe)

$VB = [(, C), (, C), (, C)];$ // une colonne C gagnante (* veut dire n'importe)

$VB = [(L_1, C_1), (L_2, C_2), ((L_3, C_3))];$ // première diagonale gagnante (3 couples avec $L_i = C_i, i = 1..3$)

$VB = [(L_1, C_1), (L_2, C_2), ((L_3, C_3))]$ // 2e diagonale gagnante (voir ci-dessous)

Attention à l'ordre ... dans les tests.

III TRAVAIL-1 DEMANDÉ POUR CETTE PARTIE

- Concevoir et coder d'abord une version toute simple.

Réaliser le programme qui permet de jouer

- humain contre humain
 - ordinateur contre humain
- Pour jouer contre la machine, faites la jouer bêtement (d'abord) !
 - Voir plus loin (et le sujet) pour l'introduction d'une heuristique.
 - **Déterminisme et le Hasard :**
introduire un degré de hasard pour initialiser VV (pas toujours à $[1/1, 1/2, \dots, 3/3]$)
→ (9 valeurs, 9 ! possibilités).
 - **Complexité du problème :** $O(\text{Dimension}^2 !)$

IV OPTIMISATION : CARRÉS MAGIQUES GAGNANTS

Ci-contre, un carré magique pour la Dimension= 3

- la somme de chaque ... =15.
- Voir le sujet pour dimensions > 3

2	7	6
9	5	1
4	3	8

figure 3

Somme pour quelques carrés magiques

- Pour Dimension=3, la somme magique est 15,
- pour la dimension 4, elle est 34,
- et somme=65 pour Dimensions=5.
- et somme=111 pour Dimensions=6.

IV-A Quelques carrés magiques

— Pour Dimension = 3 :

2 7 6

9 5 1

4 3 8

— Pour Dimension = 4 :

1 2 15 16

12 14 3 5

13 7 10 4

8 11 6 9

— Pour Dimension = 5 :

1	2	13	24	25
3	23	17	6	16
20	21	11	8	5
22	4	14	18	7
19	15	10	9	12

— Pour Dimension = 6 :

1	2	3	34	35	36
4	18	28	29	5	27
10	26	30	8	23	14
31	25	13	21	15	6
32	16	20	12	22	9
33	24	17	7	11	19

👉 il faut noter que pour une taille > 3 , les tests simples sur les carrés magiques **ne suffisent pas**. ➔ Voir le sujet.

V HEURISTIQUE (IA)

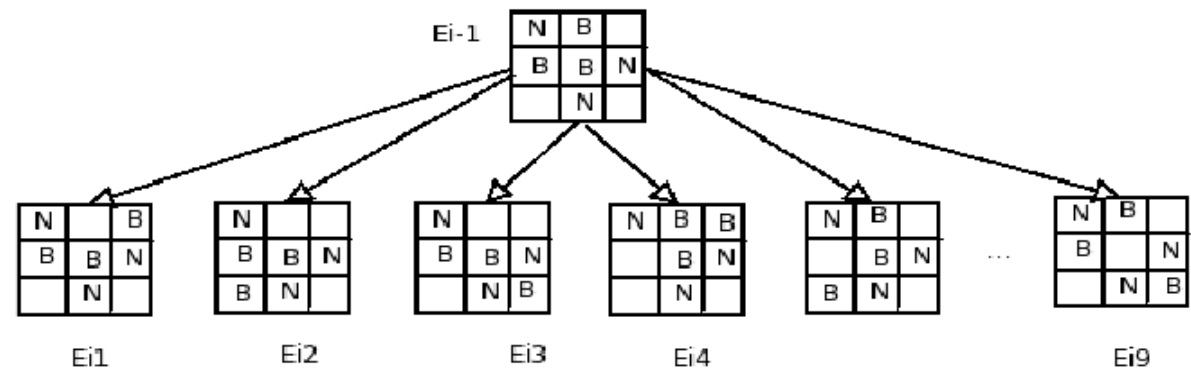
- L'algorithme du jeu de Morpion est un algorithme **irrévocable**
- Au lieu d'un choix aléatoire à chaque étape, **on peut sélectionner la case la plus prometteuse.**
- Peut-on prévoir tous les coups et jouer le meilleur ?
- *L'espace d'états est développé en largeur (par niveau) et à chaque niveau, le meilleur état est choisi pour être à son tour développé.*

- Si on développe tout l'arbre :

◦ Complexité : $|D|^{|V|} = 9^6$

→ 9 cases, 6 pions

→ Trop compliqué !



V-A Une fonction d'évaluation basique

- On décide de développer l'arbre **pour un seul niveau** :
 - Quel est le meilleur prochain coup ? **Comment évaluer la situation** ?
- **Stratégie la plus utilisée** : stratégie du type **Best-First**
 - S'inscrit dans le cadre général **Branch and Bound** (*développer et évaluer*).
- **La clé du succès d'une telle stratégie** :
 - le choix d'une **bonne** fonction d'évaluation d'un état.
- Supposons que la machine = les 'Blancs' :

On aide la machine par une IA, les 'N' = l'humain a son propre intelligence !
- **Exemple de fonction d'évaluation triviale** :

f(nbr. de 2 B (sans aucun N) sur une ligne / colonne / diagonale)

V-B Fonction d'évaluation : une amélioration

Soit à évaluer la valeur de la case (L, C) pour la couleur X . On note :

NL1 : nombre de pions de couleur X sur la ligne L ,

NL2 : ceux de l'adversaire ;

NC1 : nombre de pions de couleur X sur la colonne C ,

NC2 : ceux de l'adversaire ;

ND11 : nombre de pions de couleur X sur la 1ère diagonale passant par (L, C) ,

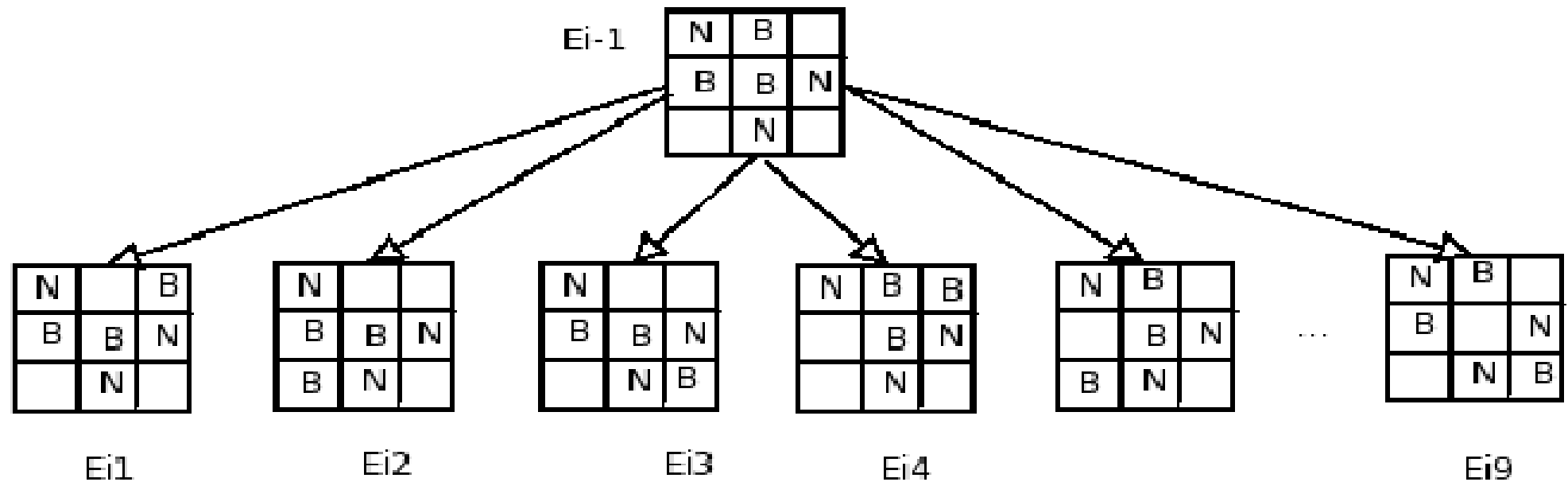
ND12 : pour l'adversaire.

ND21 : nombre de pions de couleur X sur la 2nde diagonale passant par (L, C) ,

ND22 : pour l'adversaire.

La valeur de la case (L, C) calculée par la fonction

$$g = (NL_1 - NL_2)^3 + (NC_1 - NC_2)^3 + (ND1_1 - ND1_2)^3 + (ND2_1 - ND2_2)^3.$$



V-C Une meilleure fonction d'évaluation

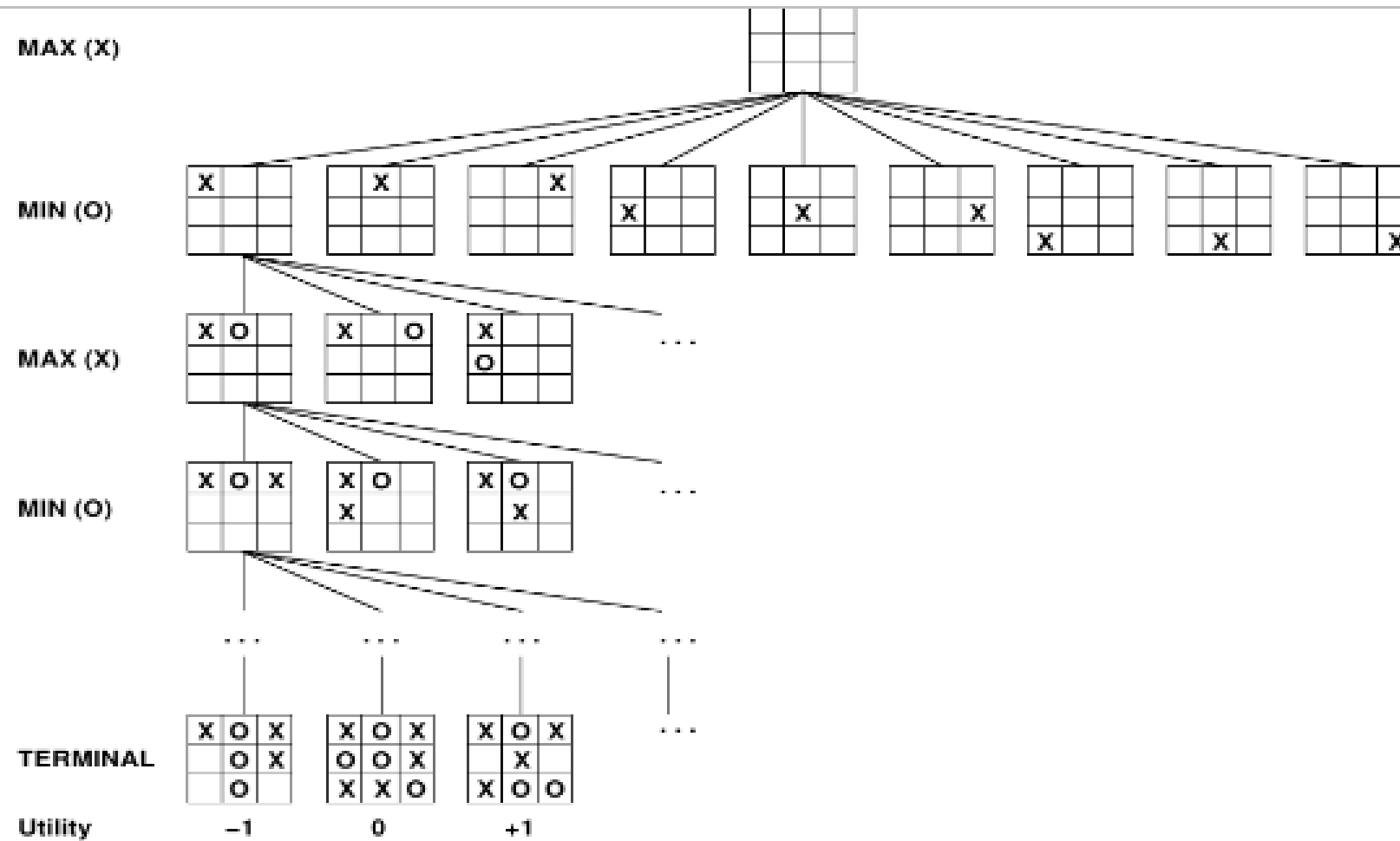
☞ Voir le sujet pour des propositions.

☞ Voir le sujet : **MinMax**

VI TRAVAIL-2 DEMANDÉ POUR CETTE PARTIE

- **Bonus** : Transformer la réalisation du Travail-1 en une **version graphique** (avec *tkinter*).
 - Compléter votre réalisation en intégrant une fonction d'évaluation.
Réaliser le programme qui permet de jouer
 - humain contre humain
 - ordinateur contre humain
 - **Rappel : déterminisme et le Hasard** :
introduire un degré de hasard pour initialiser VV (pas toujours à $[1/1, 1/2, \dots, 3/3]$)
→ (9 valeurs, 9 ! possibilités).
- 👉 **Un fichier de création d'interface Tk est fourni.**

VII HEURISTIQUE MINMAX



VII-A Développement partiel de l'arbre Min MinMax

- Exemple de fonction d'évaluation :

Nous pourrions utiliser les fonctions coût plus haut (linéaire, quadratique, etc.).

Pour changer, **on décide cette fois** d'utiliser la fonction d'évaluation suivante :

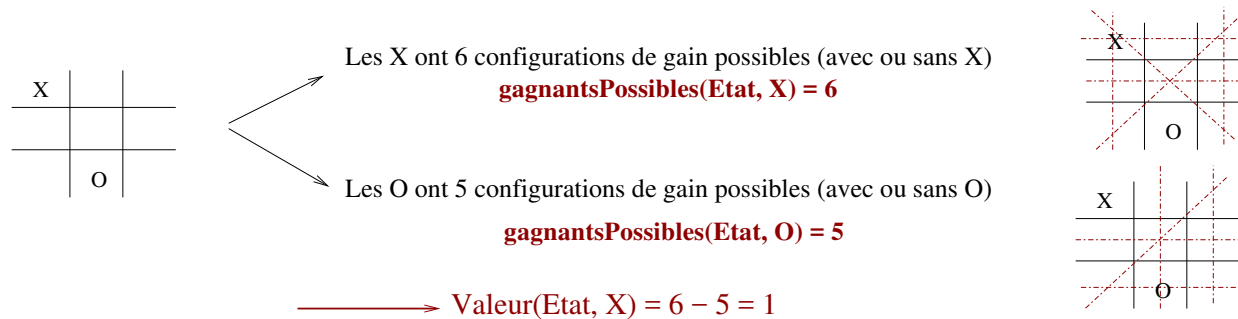
Valeur(Color) = nombre d'états gagnants possibles pour Color -
nombre d'états gagnants possibles pour l'adversaire.

Valeur(Color) = $+\infty$ si Color gagne ;

Valeur(Color) = $-\infty$ si adversaire(Color) gagne !

VII-B Évaluation d'un état

Nombre d'états gagnants possibles pour une couleur dans un État (plateau) donné :



En fait, il suffit de calculer d'abord

- la somme des lignes, colonnes et diagonales bloquées par la couleur Color,
 - faire la même somme pour adversaire(Color) ;
- puis de soustraire la 2e somme de la première.

VII-C Application de MinMax

Munie de la fonction Valeur(Etat, Color), si les X commencent, on développera l'arbre suivant (pour une profondeur=2) :

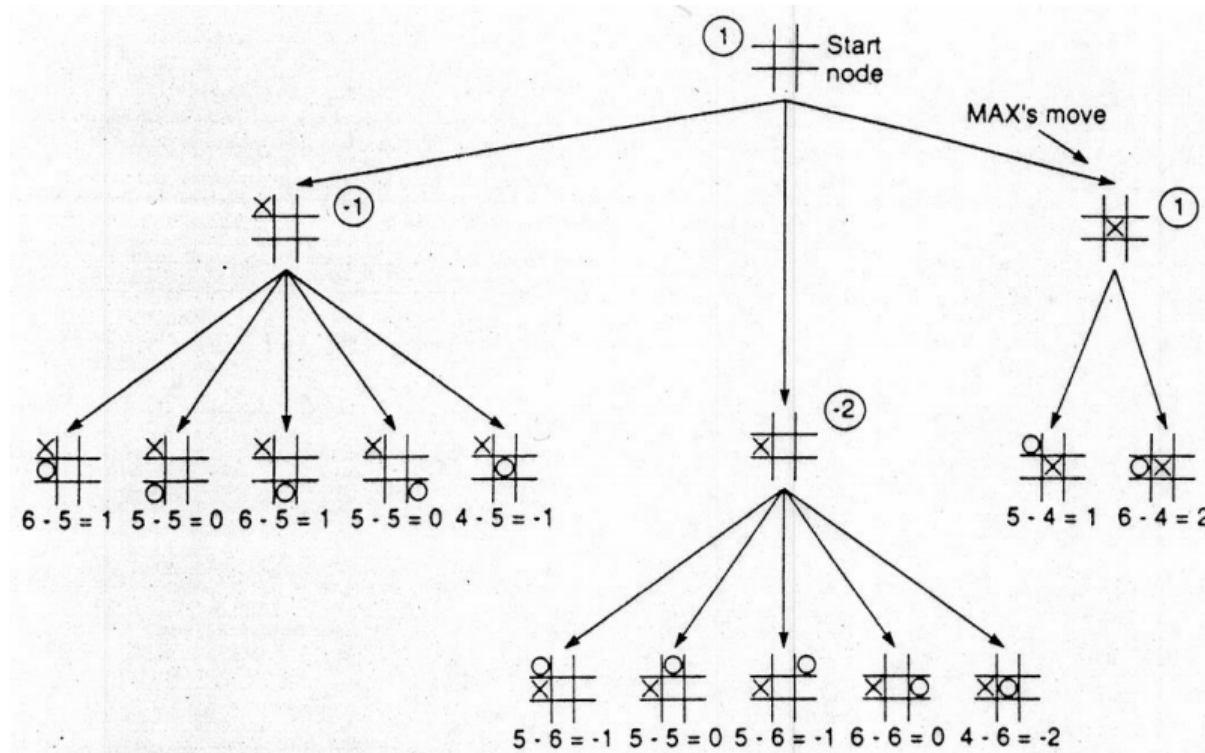


FIGURE I.1 – Choix du 1er coup pour X (en (1, 1))

→ X joue en (2, 2) et O en (1, 2).

VII-D X joue son 2e coup

☞ La "racine" de l'arbre se trouve à droite où X n'a pas encore joué son 2e coup.

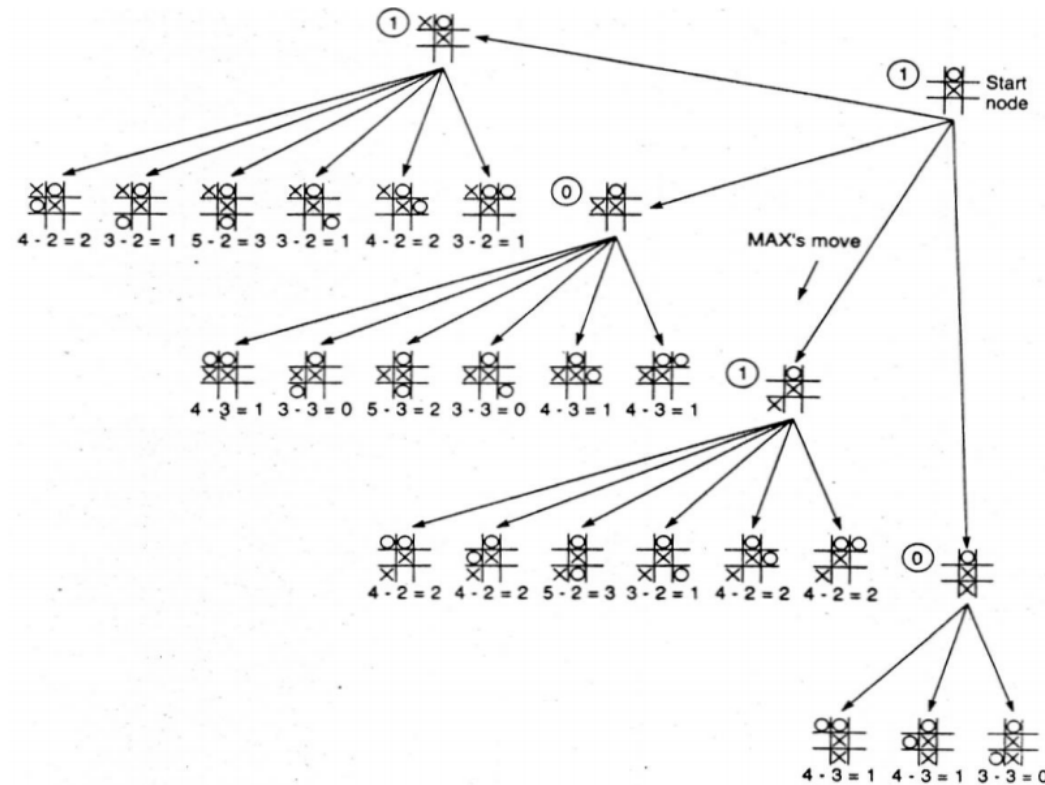


FIGURE I.2 – Choix du 2e coup pour X (en (3, 1))

→ X jouera donc en (3, 1) et O en (1, 3).

VII-E X joue son 3e coup

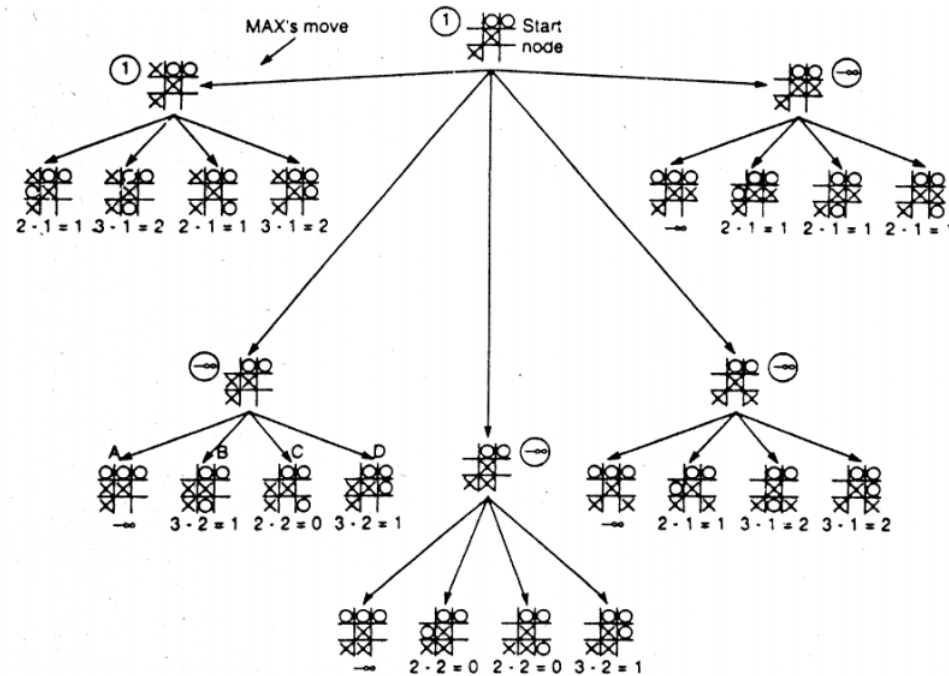


FIGURE I.3 – Choix du 3e coup pour X (en (1, 1)). Toutes les branches sont développées ici.

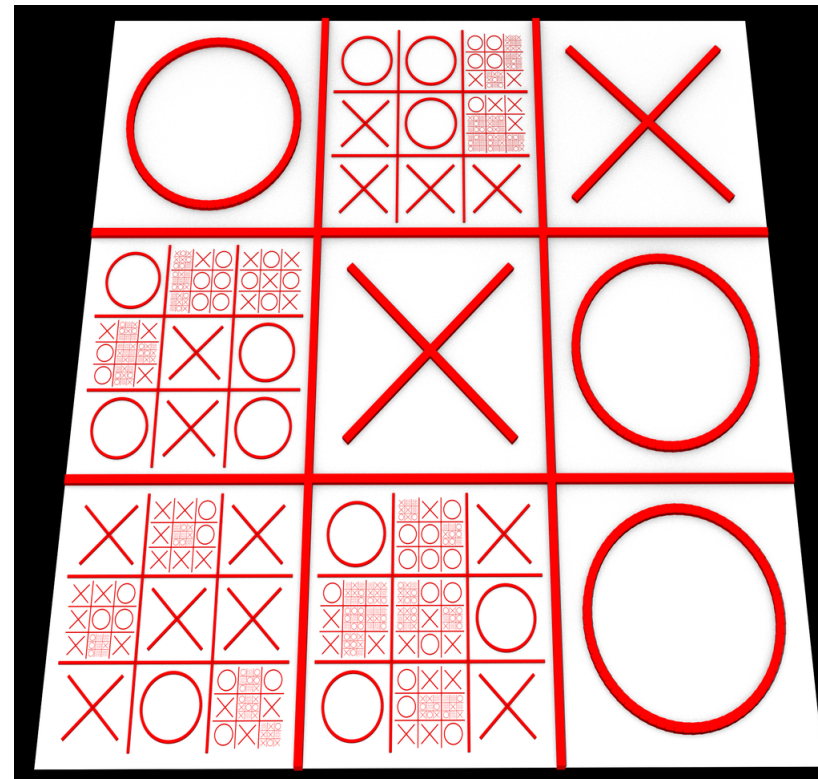
➔ X jouera en (1, 1) et O jouera parmi {(2, 1), (3, 2), (3, 3), (2, 3)}.

Et ainsi de suite....

☞ **On n'est pas certain** de gagner puisqu'on ne développe que 2 niveaux (voire, un nombre limité de successeurs).

VIII DIVERS

- Voir divers plateaux (le sujet).
- En particulier :



IX TRAVAIL-3 DEMANDÉ POUR CETTE PARTIE

- Transformer la réalisation pour intégrer la stratégie **MinMax**.

X BARÈME

- Travail 1 : Jeu de base (jeu aléatoire, Machine / humain vs. Machine) version console : 7 points
- Travail 2 (Bonus) : Ajout de l'interface graphique (tkinter) : +6 points
- Travail 3 : Heuristique Best-First pour Machine vs. Humain en version console : +3 points
- Travail 4 : Comme Travail-3 mais avec l'heuristique Min-Max : +10 points
- Travail 5 (Bonus) : Comme Travail-3 mais avec l'heuristique alpha-Beta : Bonus de 4 points

☞ L'ajout de l'interface graphique & la Travail 5 sont donc des bonus.

☞ Il est possible, à partir du Travail-4, de changer de jeu et de choisir (p.ex.) **Nimes**.