Informatique PT. 2025 - 2026

Mathématiques & Informatique - F. Gaunard
`http://frederic.gaunard.com`
Lycée Voltaire, Paris 11e.

7 Théorie des jeux

1 Introduction

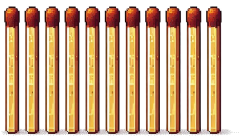
Ce chapitre présente une introduction à la **résolution** des jeux d'**accessiblité**.

Plus précisément; deux joueurs antagonistes (que l'on nomme traditionnellement *Alice* et *Bob)* s'affrontent alternativement dans un jeu.
On ne s'intéresse qu'aux jeux à **information totale** (ou complète) et **sans hasard**; à chaque instant d'une partie, chacun des deux joueurs a une vision complète de l'état du jeu, ce qui exclut notamment les jeux de cartes (on ne connait pas le jeu de l'adversaire ou une partie des cartes n'est pas distribuée...) mais comprend des jeux comme les dames, les échecs, le go, le Tic-Tac-Toe "Morpion", le Puissance 4 ou encore, et ce sera l'exemple conducteur de ce cours, le *jeu de Nim*.

À chaque instant de la partie, une décision de coup prise par l'un des joueurs se fait en fonction de la situation présente et ne dépend pas des configurations passées, ce qui justifie la terminologie de jeu *sans mémoire*.

Exemple 1. **Jeu de Nim**

Le *jeu de Nim*, dans une variante populaire, oppose deux joueurs avec la règle suivante : partant d'un tas de n allumettes, à tour de rôle chaque joueur peut en enlever 1, 2 ou 3; le perdant est celui qui prend la dernière allumette. Il existe des variantes du jeu où chaque joueur ne peut retirer qu'une ou deux allumettes à chaque tour. Nous suivrons dans ce cours la première version énoncée.



À chaque tour de jeu, chaque joueur connait donc le nombre d'allumettes restantes et décide, en fonction de ce qu'il voit et d'une *stratégie*; combien d'allumettes enlever.

On va commencer avec une approche qui fonctionne pour des jeux simples (avec un petit nombre de configurations) : le calcul des *positions gagnantes* ou *attracteurs*. Pour les jeux plus complexes, nous verrons comment bâtir une stratégie à l'aide d'une *heuristique*.

Remarque 1. **Note historique**

Issue des mathématiques dans les années 1920, la théorie des jeux permet de prévoir le comportement des agents économiques, en faisant l'hypothèse que tout agent (aussi appelé *joueur*) effectue toujours un choix rationnel visant à maximiser ses gains et à minimiser ses pertes.
Les fondements mathématiques de la théorie moderne des jeux sont décrits autour des années 1920 par Ernst Zermelo et Emile Borel, puis développés par Oskar Morgenstern, John von Neumann ou encore John Nash qui reçoit en 1994 le prix Nobel d'économie [*Wikipédia*].
Le film américain *Un homme d'exception* (*A beautiful mind*, 2001) est une version hollywoodienne de sa biographie.

2 Modélisation et résolution de jeux simples

2.1 Graphe biparti et stratégie gagnante

Définition 1. **Graphe biparti ou arène**

On modélise le type de jeu à deux joueurs précédemment décrit par un graphe orienté $G = (\mathcal{S}, \mathcal{A})$ **biparti** (aussi appelé *arène*) :

- ✗ L'ensemble des sommets est partitionné en deux sous ensembles : $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2, \quad \mathcal{S}_1 \cap \mathcal{S}_2 = \emptyset$;
  $\mathcal{S}_1$ est l'ensemble des sommets *contrôlés* par Alice, c'est à dire à partir desquels Alice jouera et $\mathcal{S}_2$ ceux contrôlés par Bob;
- ✗ Chaque arc du graphe a une extrémité dans un des deux sous-ensemble et l'autre dans l'autre :
  $$\forall a = (x, y) \in \mathcal{A}, \qquad [x \in \mathcal{S}_1 \text{ et } y \in \mathcal{S}_2] \quad \text{ou} \quad [y \in \mathcal{S}_1 \text{ et } x \in \mathcal{S}_2].$$
  Les arcs représentent les transitions possibles entre les états du jeu, c'est à dire les choix possibles que les joueurs peuvent faire à chacun de leur tour de jeu.
- ✗ Un sommet est dit *initial* s'il n'a pas de prédécesseur et *terminal* s'il n'a pas de successeur.
- ✗ Une *partie* est alors un chemin du graphe entre un sommet initial et un sommet terminal.
- ✗ On suppose que le graphe est *a-cyclique* (*i.e.* ne possède pas de cycle) ce qui garantie que toute partie sera finie. On parle alors de jeu d'**accessibilité**.

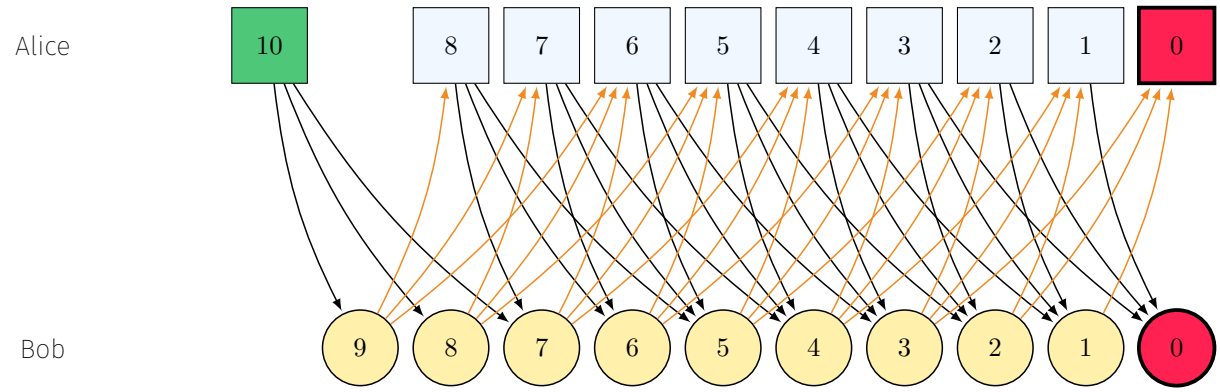
Remarque 2.

L'ensemble F des sommets terminaux se partitionne en trois parties :

- ✗ L'ensemble V^1 des sommets terminaux gagnants pour Alice ;
- ✗ L'ensemble V^2 des sommets terminaux gagnants pour Bob ;
- ✗ L'ensemble N des sommets terminaux représentants un match nul. Selon les jeux, ce dernier ensemble peut être vide.

Exemple 2. **Graphe biparti du jeu de Nim à 10 allumettes**

On représente l'arène du jeu de Nim à 10 allumettes. Chaque sommet représente un nombre d'allumettes avant que le joueur dont c'est le tour prenne une décision. Chaque arc représente un coup possible pour le joueur qui contrôle le sommet dont part cet arc.



Les sommets représentés par un rectangle sont ceux du premier joueur (Alice) et ceux par un cercle du second (Bob). Le seul sommet initial est le sommet 10 contrôlé par Alice (en vert). Les deux joueurs contrôlent un sommet terminal, qui est gagnant, le sommet 0 (en rouge).

Exercice 1.

Implémenter, sous forme d'un dictionnaire, l'arène du jeu de Nim à 10 allumettes.

Définition 2. **Stratégie**

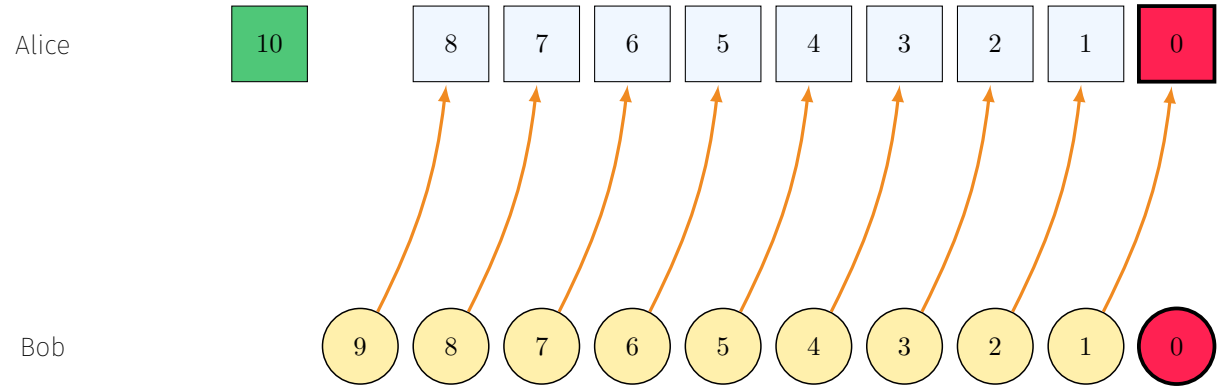
✗ Une *stratégie* pour Alice (resp. Bob) est une fonction φ qui à chaque sommet $s \in \mathcal{S}_1 \setminus F$ (resp. $s \in \mathcal{S}_2 \setminus F$) associe un sommet $s' \in \mathcal{S}_2$ (resp. $s' \in \mathcal{S}_1$) tel que $(s, s') \in \mathcal{A}$.
Suivre une stratégie φ revient à choisir d'aller en $s' = \varphi(s)$ lorsque l'on est en s.

✗ Une stratégie φ est dite *gagnante* (pour un joueur) depuis un sommet s_0 si toute partie jouée (par ce joueur) depuis s_0 en suivant φ mène à sa victoire et ce, peu importe les choix de son adversaire.

✗ Un sommet est appelé *position gagnante* (pour un joueur) s'il existe une stratégie gagnante (pour ce joueur) depuis ce sommet.

Exemple 3. **Une stratégie... absurde**

En reprenant notre fil rouge du jeu de Nim à 10 allumettes, un exemple de stratégie pour Bob serait de systématiquement retirer une seule allumette. Elle n'est clairement pas gagnante.



Si l'on représente chaque sommet du graphe biparti sous la forme (j, i) où $j \in [\![0, 10]\!]$ est un nombre d'allumettes et $i \in [\![0, 1]\!]$ correspond au joueur (1 pour Alice, 2 pour Bob), la stratégie ci-dessus est alors

$$\varphi : (j, 2) \in [\![1, 9]\!] \times \{2\} \longmapsto (j - 1, 1)$$

☞ Résoudre un jeu d'accessibilité revient donc à déterminer les positions gagnantes (de chacun des joueurs) et une stratégie pour chacun définie à partir de chaque position gagnante que le joueur contrôle.
Pour cela, on construit l'*attracteur* de chaque joueur, que l'on construit par récurrence.

2.2 Calcul des attracteurs

Définition 3. **Attracteur (d'Alice)**

Soient $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ avec $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$ un graphe biparti et V^1 l'ensemble des sommets terminaux gagnants (définissant une victoire) pour Alice.
On définit la suite $(\mathscr{A}_i^1)_{i \geq 0}$ de sous-ensembles de sommets comme suit :

✗ $\mathscr{A}_0^1 = V^1$.

✗ Pour $i \geq 0$, $\mathscr{A}_{i+1}^1$ est donc l'ensemble des sommets déjà dans $\mathscr{A}_i^1$, des sommets contrôlés par Alice ayant au moins un successeur dans $\mathscr{A}_i^1$ et des sommets contrôlés par Bob dont tous les successeurs sont dans $\mathscr{A}_i^1$, autrement dit

$$\mathscr{A}_{i+1}^1 = \mathscr{A}_i^1 \cup \{s \in \mathcal{S}_1 : \exists s' \in \mathscr{A}_i^1,\ (s, s') \in \mathcal{A}\} \cup \{s \in \mathcal{S}_2 : \forall s' \in \mathcal{S},\ (s, s') \in \mathcal{A} \Longrightarrow s' \in \mathscr{A}_i^1\}.$$

On appelle **attracteur** d'Alice la réunion croissante $\mathscr{A}^1$ des sous-ensembles suivants

$$\mathscr{A}^1 = \bigcup_{i=0}^{+\infty} \mathscr{A}_i^1.$$

Par construction, $\mathscr{A}_i^1$ représente l'ensemble des positions gagnantes pour Alice lui permettant de gagner en au plus i coups.

Remarque 3. **Attracteur de Bob**

On peut naturellement définir de manière analogue l'attracteur de Bob à partir de l'ensemble V^2 des sommets terminaux gagnants contrôlés par celui-ci :

- ✗ $\mathscr{A}_0^2 = V^2$.
- ✗ Pour $i \geq 0$,

$$\mathscr{A}_{i+1}^2 = \mathscr{A}_i^2 \cup \{s \in \mathcal{S}_2 : \exists s' \in \mathscr{A}_i^2,\ (s,s') \in \mathcal{A}\} \cup \{s \in \mathcal{S}_1 : \forall s' \in \mathcal{S},\ (s,s') \in \mathcal{A} \Longrightarrow s' \in \mathscr{A}_i^2\},$$

puis

$$\mathscr{A}^2 = \bigcup_{i=0}^{+\infty} \mathscr{A}_i^2.$$

☞ On peut montrer que l'attracteur d'Alice est alors exactement l'ensemble des positions gagnantes pour Alice. Si l'on peut faire match nul dans le jeu considéré, il faut calculer indépendamment les attracteurs d'Alice et Bob. Sinon, les deux ensemble sont complémentaires. En particulier, on a le résultat suivant.

Théorème 1.

Il existe une stratégie gagnante pour Alice (resp. Bob) à partir de s si et seulement si $s \in \mathscr{A}^1$ (resp. $s \in \mathscr{A}^2$).

Exercice 2.

Déterminer «à la main» les attracteurs d'Alice et de Bob pour le jeu de Nim à 5 allumettes. Commenter.

Algorithme de calcul des attracteurs

La construction de l'attracteur se fait donc par un parcours du graphe «à l'envers» à partir de V^1 (ou de V^2) et en remontant les arcs, donc par un parcours (en largeur) du graphe *transposé*, et présente donc une complexité en $O(\mathrm{Card}(\mathcal{S}) + \mathrm{Card}(\mathcal{A}))$. Il sera utile de calculer les degrés sortants des sommets de $\mathcal{G}$ (qui seront les degrés entrants dans $\mathcal{G}^\top$) pour gérer la condition «tout successeur d'un sommet de $\mathcal{S}_2$ aboutit à un sommet attracteur».

```
def transpose(G):
    GT ={v : [] for v in G} # initialisation du graphe transposé
    for s in G: # pour chaque sommet de G
        for v in G[s] # parcours des successeurs v de s
            GT[v].append(s) # s est un prédécesseur de v : ajout de s à GT[v]
    return GT
```

```
def degres_sortants (G):
    return {v:len(G[v]) for v in G}
```

On peut alors écrire l'algorithme de calcul de l'attracteur, avec une fonction de parcours. La fonction renvoie l'attracteur d'Alice sous la forme d'un dictionnaire dont les clés sont ses positions gagnantes.

```
def attracteur (G,S1,V1):
    nG = degres_sortants (G) # degrés sortant de G
    GT = transpose (G) # graphe transposé
    A = {} # attracteur : dictionnaire des positions gagnantes
    def parcours(s):
        if s not in A:
            A[s] = True
            for v in GT[s]:
                nG[v] -= 1
                if v in S1 or nG[v] == 0:
                    parcours(v)
    for s in V1:
        parcours(s)
    return A
```

Construction d'une stratégie gagnante

Pour construire, en parallèle du calcul de l'attracteur, une stratégie gagnante pour Alice, il suffit de modifier légèrement l'algorithme précédent ; au moment de lancer `parcours(v)`, on peut poser $\varphi(v) = s$.

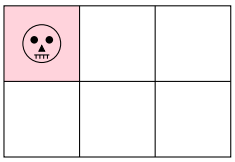
```
def attracteur_avec_strategie(G, S1, V1):
    nG = degres_sortants (G) # degrés sortant de G
    GT = transpose (G) # graphe transposé
    A = {} # attracteur : dictionnaire des positions gagnantes
    phi = dict()

    def parcours(s):
        if s not in A:
            A[s] = True
            for v in GT[s]:
                nG[v] -= 1
                # Si v est contrôlé par Alice ou
                #  tous ses successeurs sont dans l'attracteur
                if v in S1 or nG[v] == 0:
                    # On définit le coup gagnant depuis v vers s
                    if v in S1:
                        phi[v] = s
                    parcours(v)
    for s in V1:
        parcours(s)

    return A, phi
```

Exercice 3. **Jeu de Chomp**

On considère une tablette de chocolat rectangulaire (à n lignes et p colonnes) dont le coin supérieur gauche est empoisonné. Chaque joueur choisit à tour de rôle un carré et le mange, ainsi que tous les morceaux situés à la droite et en dessous du carré choisi. Bien évidemment, le joueur qui n'a plus d'autre choix que de manger le carré empoisonné a perdu.
On se place dans le cas ici de la tablette avec $n = 2$ et $p = 3$.



1. Représenter l'arène de ce jeu. Chaque sommet est une *configuration* de la tablette de chocolat.
2. Méthode de l'attracteur.
   a. Déterminer les positions gagnantes pour Alice.
   b. Existe-t-il une stratégie gagnante pour Alice ? Si oui, en déterminer une.
3. Montrer que, si $(n, p) \neq (1, 1)$, alors il existe toujours une stratégie gagnante pour Alice.
   On distinguera les cas $n = 1, p > 1$ *puis* $n > 1, p = 1$ *puis* $n > 1, p > 1$ *avec, dans ce dernier cas, un raisonnement par l'absurde.*

3 Algorithme min-max

Dans cette section, on considère toujours un jeu modélisé par un graphe biparti $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ où : $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$.

En théorie, le calcul des attracteurs présenté dans la section précédente permet de déterminer les stratégies gagnantes et de jouer de manière parfaite.
En pratique, l'algorithme permettant de les déterminer n'est pas utilisable pour des jeux complexes, c'est-à-dire lorsque le

graphe associé $\mathcal{G}$ est trop gros. Par exemple, on estime que le nombre d'états du jeu est de l'ordre de 10^{32} pour les dames, d'au moins 10^{46} pour les échecs et de 10^{100} pour le go rendant le parcours du graphe inexploitable.

Pour contourner cette difficulté, nous allons présenter l'algorithme min-max qui ne nécessite pas une exploration complète du graphe du jeu. En contre-partie, la stratégie obtenue via cette approche ne sera plus parfaite en général.

3.1 Heuristique

L'algorithme min-max repose sur une fonction permettant d'évaluer la qualité de chaque position du jeu.

Définition 4. **Heuristique**

Une **heuristique** pour le jeu étudié est une fonction $h : \mathcal{S} \to \mathbb{R} \cup \{-\infty, +\infty\}$ de sorte que pour toute position $p \in \mathcal{S}$ du jeu :

- ✗ plus $h(p)$ est grand, meilleure est la position pour Alice ;
- ✗ plus $h(p)$ est petit, meilleure est la position pour Bob.

☞ En pratique :

- ✗ si $s \in \mathcal{S}$ est un sommet final représentant une victoire d'Alice, alors on pose : $h(s) = +\infty$;
- ✗ si $s \in \mathcal{S}$ est un sommet final représentant une victoire de Bob, alors on pose : $h(s) = -\infty$.

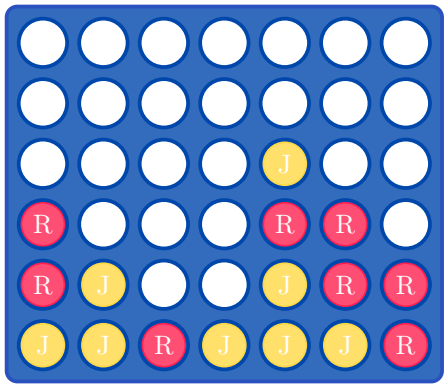
Le mot «heuristique» signifie «qui sert à la découverte» ou «qui est propre à guider une recherche».

☞ Toujours en pratique, une heuristique est souvent construite de manière expérimentale.

Exemple 4. **Puissance 4**

Pour illustrer la notion d'heuristique, on s'intéresse au jeu *Puissance 4*. Le but du jeu est d'aligner une suite de 4 pions de même couleur dans une grille rectangulaire composée de 6 lignes et 7 colonnes.
Chaque joueur dispose de 12 pions d'une même couleur. Les deux joueurs placent tour à tour un pion dans la colonne de leur choix. Le pion coulisse alors jusqu'à la position la plus basse possible dans ladite colonne, à la suite de quoi, c'est à l'adversaire de jouer.
Le vainqueur est le joueur qui réalise en premier un alignement (horizontal, vertical ou diagonal) consécutif d'au moins 4 pions de sa couleur. Si toutes les cases de la grille de jeu sont remplies et qu'aucun des deux joueurs n'a réalisé un tel alignement, la partie est déclarée nulle.
Dans la suite, on suppose qu'Alice joue avec des pions jaunes et que Bob joue avec des pions rouges.

On commence par construire une heuristique pour ce jeu. Pour ce faire, on attribue à chaque case du jeu le nombre d'alignements possibles de 4 jetons contenant cette case. Le valeurs attribuées sont reportées dans le tableau ci-dessous :



3	4	5	7	5	4	3
4	6	8	10	8	6	4
5	8	11	13	11	8	5
5	8	11	13	11	8	5
4	6	8	10	8	6	4
3	4	5	7	5	4	3

Ensuite, pour calculer l'heuristique d'une position (non finale) du jeu, on calcule la somme des valeurs des cases contrôlées par Alice à laquelle on retranche la somme des valeurs des cases contrôlées par Bob.
Par exemple, l'heuristique de la position s représentée ci-dessus est

$$h(s) = (3+4+7+5+4+6+8+11) - (5+3+4+6+4+5+11+8) = 2.$$

3.2 Principe de l'algorithme

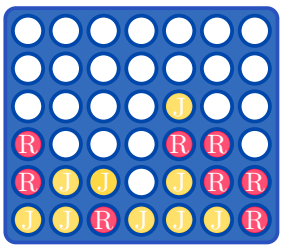
Dans cette sous-section, on suppose que l'on dispose d'une heuristique $h : \mathcal{S} \to \mathbb{R} \cup \{-\infty, +\infty\}$.

Au moment où l'un des deux protagonistes doit jouer, plusieurs possibilités s'offrent à lui (entre une et sept pour le puissance 4). Une solution simple pour choisir le coup à jouer consiste à calculer l'heuristique correspondant à chacune des configurations atteignables et à jouer celle d'heuristique maximale (pour Alice) ou minimale (pour Bob). Il est alors coutume de renommer Alice en *Max* et Bob en *Min*.
Mais Alice peut aussi tenir compte du coup que va jouer Bob ensuite, et donc calculer l'heuristique de chacune des positions que Bob pourra atteindre. On peut répéter ce raisonnement, mais le nombre de configurations à examiner croît exponentiellement, donc il est nécessaire de **limiter la profondeur n de la recherche**, et on obtient alors un algorithme glouton (qui ne donnera pas nécessairement une solution optimale).

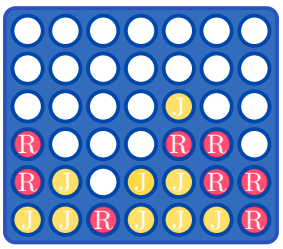
Pour illustrer le principe de l'algorithme min-max, reprenons l'exemple du jeu Puissance 4. On représente les états contrôlés par Alice par des ronds bleus et les états contrôlés par Bob par des carrés rouges.
Afin d'avoir des arbres de taille raisonnable, on limite les coups possibles des deux joueurs à placer un pion dans une des trois colonnes centrales.

On part de la position du jeu Puissance 4 représentée ci-avant: c'est à Alice de jouer.
Une première possibilité pour effectuer le choix du coup à jouer est de calculer l'heuristique de chacune des positions atteignables, puis de rejoindre celle dont l'heuristique est maximale. En effet, Alice cherche à se trouver dans une position avec une forte heuristique.

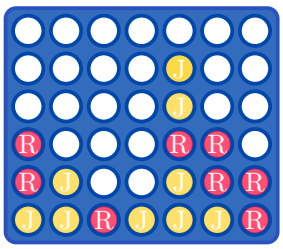
Pour le calcul de l'heuristique, on ajoute à l'heuristique de la position précédente l'heuristique de la case où l'on va placer le pion.
Les trois positions atteignables pour Alice et leur heuristique :



$h(s) = 2 + 8 = 10$



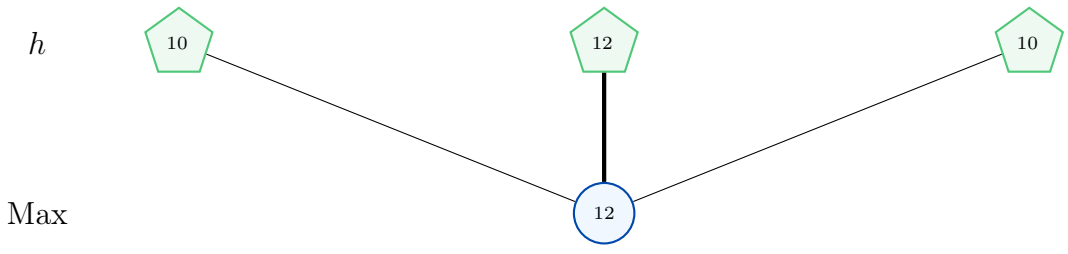
$h(s) = 2 + 10 = 12$



$h(s) = 2 + 8 = 10$

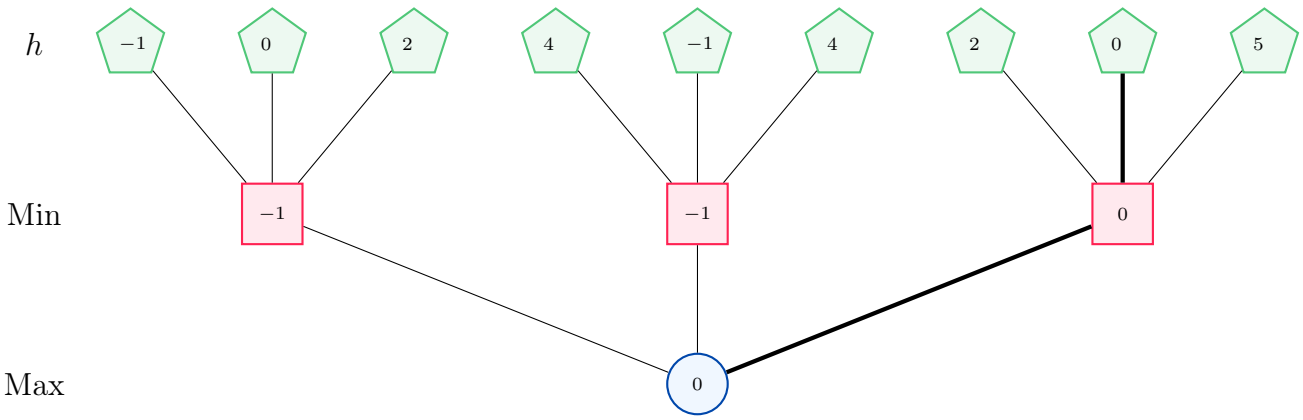
Alice va donc choisir de jouer en colonne centrale. On pouvait représenter la situation à l'aide d'un arbre.

Ce premier arbre a une profondeur de 1.



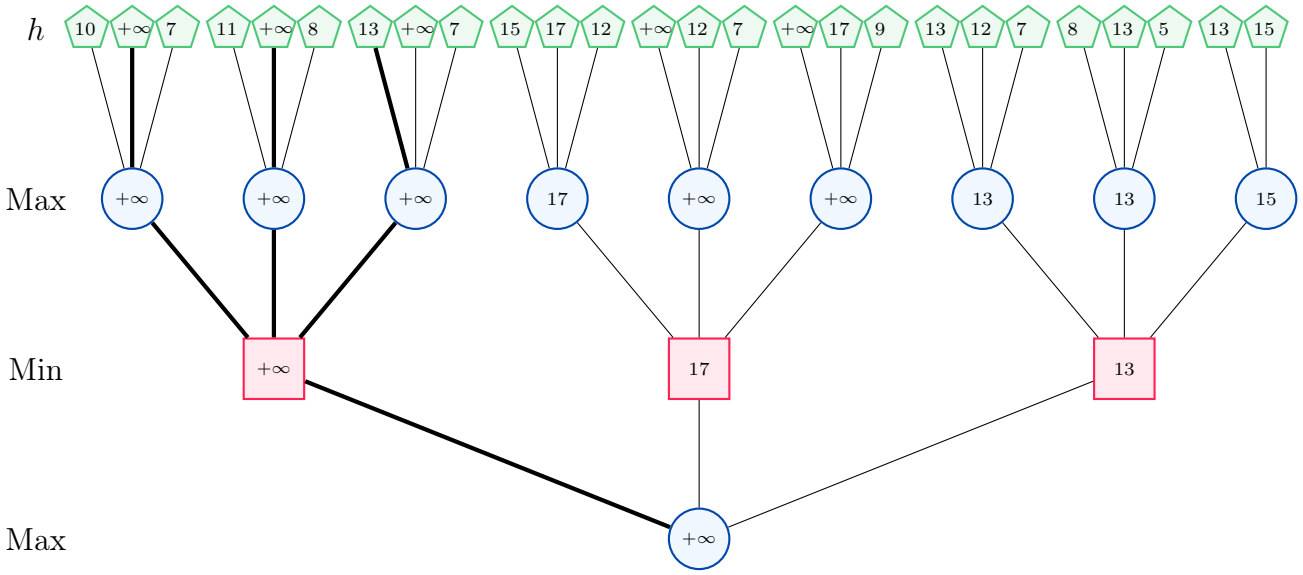
Une seconde possibilité pour effectuer le choix du coup à jouer pour Alice est de tenir en plus compte du coup suivant qui sera joué par Bob : elle peut calculer l'heuristique des positions atteignables en deux coups, puis de choisir le coup lui permettant d'obtenir une heuristique maximale (en supposant que Bob va jouer de sorte à minimiser l'heuristique).

Ci-dessous l'arbre de profondeur 2.



En adoptant cette stratégie, Alice choisit de jouer dans la colonne de droite.

On peut poursuivre. Une troisième possibilité, pour effectuer le coup du choix à effectuer pour Alice, est d'explorer toutes les possibilités pour les trois coups suivants.
On représente alors l'arbre de profondeur 3.



En adoptant cette stratégie, Alice choisit de jouer dans la colonne de gauche (et sera vainqueur en 3 coups).

Pour calculer le meilleur coup d'Alice avec une recherche à la profondeur n, il faut donc commencer par calculer la valeur de l'heuristique de toutes les positions atteignables en n coups. Ensuite, il faut distinguer deux cas :

- ✗ si n est impair, Alice va jouer le dernier coup, donc l'antécédent de chacun des sommets de profondeur n se verra attribuer la valeur maximale de ses successeurs ;
- ✗ si n est pair, Bob va jouer le dernier coup, donc l'antécédent de chacun des sommets de profondeur n se verra attribuer la valeur minimale de ses successeurs.

Pour implémenter cet algorithme, nous allons écrire deux fonctions récursives:

- ✗ `maximin(p,n)` (destinée à Alice) va chercher à maximiser l'heuristique après n coups en partant de la position p, en supposant que son adversaire joue au mieux ;
- ✗ `minimax(p,n)` (destinée à Bob) va chercher à minimiser l'heuristique après n coups en partant de la position p, en supposant que son adversaire joue au mieux.

Ces deux fonctions sont mutuellement récursives : pour calculer `maximin(p,n)` on calcule pour chaque position $p_1, \dots, p_k$ atteignable à partir de p la valeur de l'heuristique des positions `minimax(pi,n-1)` avant de choisir la position conduisant à

la valeur maximale.
De manière symétrique, pour calculer `minimax(p,n)` on calcule pour chaque position $p_1, \dots, p_k$ atteignable à partir de p la valeur de l'heuristique des positions `maximin(pi,n-1)` avant de choisir la position conduisant à la valeur minimale.
Pour la rédaction de l'algorithme, on suppose définies la fonction `h` qui prend pour argument une position du jeu et renvoie la valeur de son heuristique, ainsi que la fonction `successeurs` qui renvoie la liste des positions atteignables à partir de la position p.

```
def maximin(p,n):
    if n==0 or successeurs(p)==[]:
        return h(p)
    maxi=-np.inf
    for pk in successeurs(p):
        s=minimax(pk,n-1)
        if s>maxi:
            maxi=s
    return maxi
```

```
def minimax(p,n):
    if n==0 or successeurs(p)==[]:
        return h(p)
    mini=np.inf
    for pk in successeurs(p):
        s=maximin(pk,n-1)
        if s<mini:
            mini=s
    return mini
```

Remarque 4.

L'algorithme Min-max prend en paramètre une profondeur n, qui limite la recherche à une certaine hauteur dans l'arbre de jeu. Dans le pire des cas, cette profondeur n correspond à la hauteur complète de l'arbre, ce qui signifie que tous les états possibles du jeu sont explorés jusqu'aux feuilles.
La complexité temporelle de l'algorithme est alors donnée par $O(p^n)$, où p est le facteur de branchement, c'est-à-dire le nombre moyen d'actions possibles à chaque étape du jeu, et n est la profondeur maximale explorée.
Ainsi, plus p ou n sont grands, plus le nombre total d'états à explorer augmente de façon exponentielle.
On représente dans le tableau suivant les tailles des arbres de jeu pour différents jeux :

Jeu	Branchement	Profondeur	Taille estimée
Morpion (Tic-Tac-Toe)	3	9	26830
Puissance 4	7	42	10^{14}
Dames	8	50	10^{20}
Échecs	35	80	10^{120} (nombre de Shannon)
Go	250	300+	10^{170}

Exercice 4. **Tic-Tac-Toe**

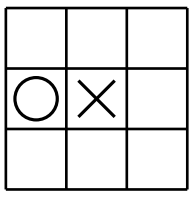
Les deux joueurs qui s'affrontent doivent remplir, chacun à leur tour, une case d'une grille 3×3 avec le symbole qui leur est attribué : ◯ ou ×. Le gagnant est celui qui arrive à aligner trois symboles identiques, horizontalement, verticalement ou en diagonale. Il est coutume de laisser le joueur jouant × effectuer le premier coup de la partie.

Afin de définir l'heuristique de chaque position, on commence par un peu de terminologie. Les lignes/colonnes/diagonales sont dites *gagnables* si elles ne contiennent que des ×, des ◯, ou sont vides. Une ligne est *presque gagnante* lorsqu'elle contient exactement deux × ou deux ◯ et aucune pièce adverse.

Configuration accessible	Score
Lignes/colonnes/diagonales gagnables pour × ou ◯	$+1$
Lignes/colonnes/diagonales presque gagnantes pour ×	$+3$
Lignes/colonnes/diagonales presque gagnantes pour ◯	-3
Centre contrôlé par ×	$+2$
Centre contrôlé par ◯	-2

État terminal	Score
Victoire immédiate pour ×	$+\infty$
Victoire immédiate pour ◯	$-\infty$
Égalité (plateau complet sans gagnant)	0

Appliquer l'algorithme min-max avec une profondeur de 2 pour déterminer le prochain coup d'Alice à partir des différentes positions ci-dessous :

i. 

ii.