4 Théorie des jeux

Extrait du programme officiel :

Notions	Commentaires
Jeux d'accessibilité à deux joueurs sur un graphe. Stratégie. Stratégie gagnante. Position gagnante. Détermination des positions gagnantes par le calcul des attracteurs. Construction de stratégies gagnantes.	On considère des jeux à deux joueurs (J_1 et J_2) modélisés par des graphes bipartis (l'ensemble des états contrôlés par J_1 et l'ensemble des états contrôlés par J_2). Il y a trois types d'états finals : les états gagnants pour J_1, les états gagnants pour J_2 et les états de match nul. On ne considère que les stratégies sans mémoire.
Notion d'heuristique. Algorithme min-max avec une heuristique.	L'élagage alpha-beta n'est pas au programme.
Mise en œuvre	
La connaissance dans le détail des algorithmes de cette section n'est pas un attendu du programme. Les étudiants acquièrent une familiarité avec les idées sous-jacentes qu'ils peuvent réinvestir dans des situations où les modélisations et les recommandations d'implémentation sont guidées, notamment dans leurs aspects arborescents.	

Table des matières

I Jeux sur un graphe

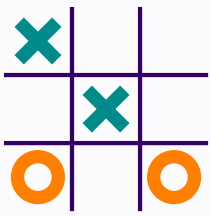
On s'intéresse à la modélisation de certains jeux au moyen d'un graphe.

Nous supposons ici que

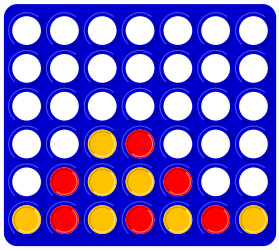
1. il y a deux joueurs, Adam et Ève, qui jouent **alternativement** ;
2. une partie se termine par une victoire d'Adam, une victoire de Ève ou un match nul ;
3. le jeu est à **information totale** : à tout instant, tout joueur voit tout le jeu (pas de jeu de cartes, par exemple).

Exemples 1

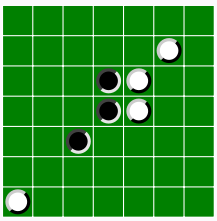
E 1 – Morpion (Tic-Tac-Toe)



E 2 – Puissance 4



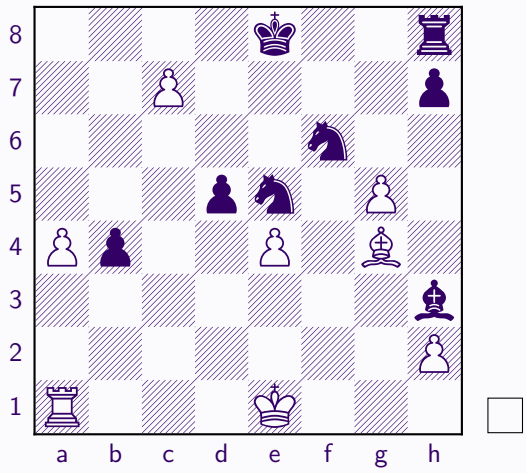
E 3 – Othello (ou Reversi)



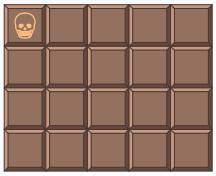
E 4 – Jeu de Nim : chacun son tour, chaque joueur prend 1, 2 ou 3 batonnet·s. Celui qui prend le dernier a perdu.

Exemples 2

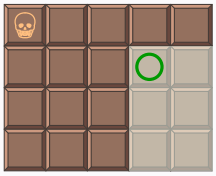
E 1 – Échecs



E 2 – **Jeu de Chomp** : on part d'une tablette de chocolat rectangulaire de format $p \times q$.
La carré supérieur gauche est empoisonné.

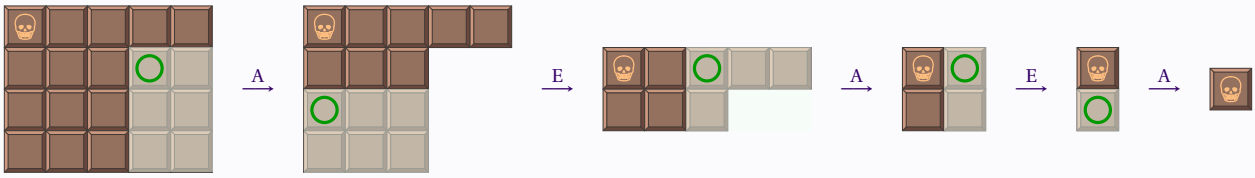


Chaque joueur à son tour choisit un carré qu'il mange, ainsi que tous les carrés se trouvant à sa droite et en dessous.



Celui qui mange le dernier carré a perdu.

Exemple 1 : Une partie de Chomp 4×5



Adam a gagné.

1 Modélisation

Définition 1 : Graphe du jeu

On appelle **graphe du jeu** ou *arène* associé à un tel jeu un graphe $\mathcal{G} = (S, A)$ tel que chaque sommet dans S représente une **position** du jeu (à partir de laquelle soit Adam, soit Ève doit jouer) et chaque arête $a = (s_1, s_2) \in A$ signifie qu'un des deux joueurs peut passer de la position s_1 à la position s_2.

Remarque 1

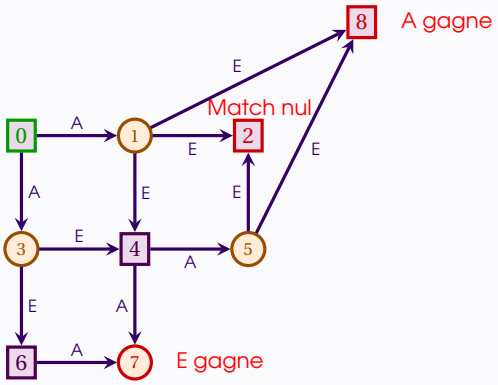
Le jeu peut être dans le même état au moment où soit Adam, soit Ève joue : ce sont deux positions différentes (donc deux sommets différents).

Exemple 2

On joue au jeu comme si on avait un jeton, au départ sur la position initiale (en vert) et on le glisse le long des arêtes à chaque coup d'Adam ou de Ève.

On a supposé ici que les positions 7 (contrôlée par Ève) et 2 et 8 (contrôlées par Adam) correspondent respectivement à une victoire d'Ève, un match nul et une victoire d'Adam.

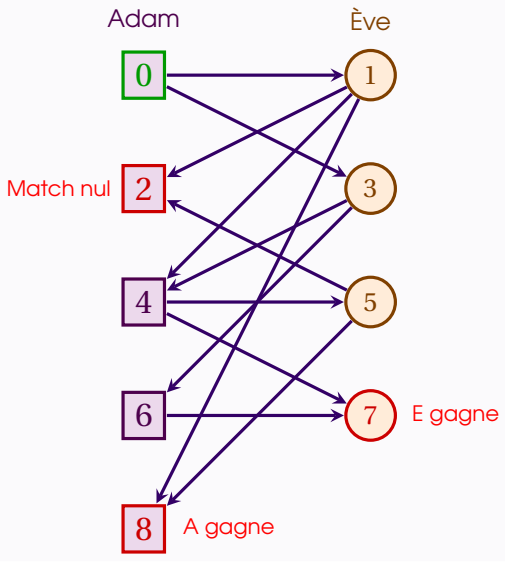
On a précisé ici sur les arête quel joueur contrôle chaque position (qui d'Adam ou d'Ève joue le coup).



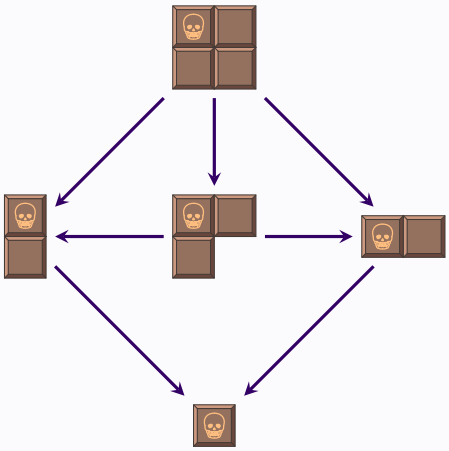
Définition 2 : Graphe biparti

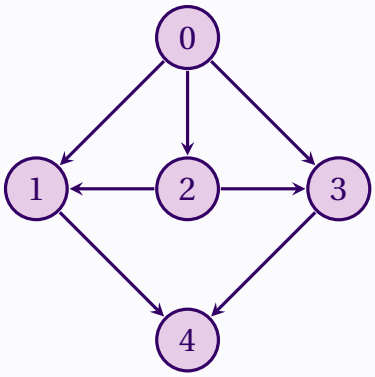
Un graphe $\mathcal{G} = (S, A)$ et dit **biparti** si on peut partitionner S en $S_1 \sqcup S_2$ tel que toute arête $(s, s') \in A$ est telle que s et s' ne sont pas tous deux dans S_1 ou tous deux dans S_2.

Exemple 3 : Sur l'exemple précédent

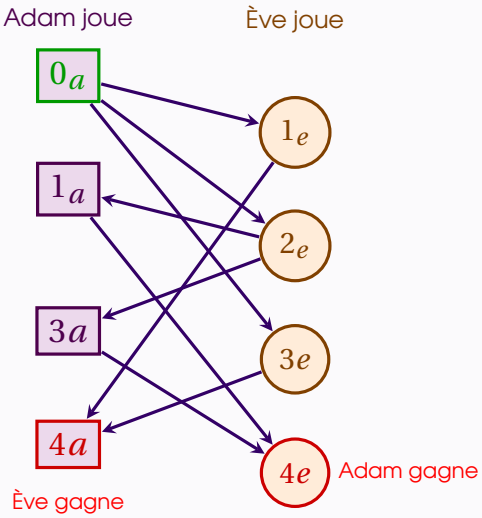


Exemple 4 : pour le jeu de Chomp 2×2





Mais on n'a pas tenu compte de qui joue. Il faut doubler les sommets et le graphe biparti est le suivant :



2 Stratégies et positions gagnantes

On considère une arène $\mathcal{G} = (S = S_a \sqcup S_e, A)$, graphe biparti, les sommets de S_a sont les positions contrôlées par Adam et ceux de S_e sont celles contrôlées par Ève.

On note V_a, V_e et N les ensembles de sommets de victoire d'Adam, de victoire d'Ève et de nul, respectivement.

On définit une fonction Succ tel que pour tout $s \in S$, $\text{Succ}(s) = \{s' \in S,\ (s, s') \in A\}$. Comme le graphe est biparti, on a $\text{Succ}(S_e) \subset S_a$ et $\text{Succ}(S_a) \subset S_e$.

Définition 3 : Partie

Une **partie** est un chemin dans le graphe, c'est-à-dire une suite, finie ou non (s_n) de positions (sommets) tel(le)s que pour tout i, $(s_i, s_{i+1}) \in A$ ie $s_{i+1} \in \text{Succ}(s_i)$.

Définition 4 : Partie gagnante

Une **partie gagnante** pour Adam est une partie finie dont la dernière position appartient à V_a et dont aucune autre n'est dans $V_a \cup V_e$.

Définition analogue pour Ève.

Exemple 5 : pour le jeu de Chomp 2×2

$(0_a, 2_e, 3_a, 4_e)$

Remarque 2

Si la dernière position est dans $S \setminus (V_a \cup V_e)$ et s'il n'existe pas d'arête partant de celle-ci, il y a match nul (c'est une position de N).

Définition 5 : Stratégie

Une **stratégie** (sans mémoire) pour un joueur est la donnée de coups de ce joueur pour certaines de ses positions, sans tenir compte des coups précédents.

Ainsi, une stratégie pour Adam est une fonction $f : S'_a \subset S_a \to S_e$ telle que $\forall\, s_a \in S'_a,\ \ (s_a, f(s_a)) \in A$ ie $f(s_a) \in \mathrm{Succ}(s_a)$. (Idem pour Ève).

On dit que c'est une **stratégie gagnante** pour ce joueur lorsque toutes les parties qu'il joue avec les coups donnés par cette stratégie sont gagnantes pour ce joueur.

Remarque 3

Parfois, on demande que f soit définie sur S_a tout entier, mais seules les images des positions gagnantes sont utilisées. Nous avons fait le choix dans ce cours de ne définir f que sur une partie S'_a de S_a.

Exemple 6 : pour le jeu de Chomp 2×2

La stratégie $f(0) = 2$, $f(1) = f(3) = 4$ est gagnante pour Adam.
Il n'y a pas de stratégie gagnante pour Ève.

Définition 6

Une **position** est dite **gagnante** pour un joueur s'il existe une stratégie gagnante pour ce joueur pour toute partie débutant à cette position. Elle est dite **perdante** si elle est gagnante pour l'autre joueur.

Remarque 4

Ne pas confondre avec les positions de victoire.
Cependant, les positions de V_a et V_e sont toujours gagnantes pour Adam et Ève respectivement.

Exemple 7 : pour le jeu de Chomp 2×2

Les positions $0, 1, 3$ sont gagnantes, $2, 4$ sont perdantes.

Exercice 1 : Montrer que dans le jeu de Chomp $p \times q$, il y a une stratégie gagnante pour le 1er joueur (Adam) si $(p, q) \neq (1, 1)$.

Si $p = 1$ ou $q = 1$, c'est facile.
Sinon, soit la coup en bas à droite est gagnant et c'est terminé, soit il ne l'est pas et donc en jouant ce coup, Éve a une stratégie gagnante sur l'un des autres coups qu'Adam peut jouer directement.

Remarque 5

Ce type de preuve est appelé **vol de stratégie**. Cela ne donne pas de construction d'une stratégie gagnante.

3 Calcul des positions gagnantes

On s'intéresse à la possibilité de gagner en au plus n coups pour Adam à partir d'une position $s \in S$.

- Soit $n = 0$ et il faut que $s \in V_a$.
- Soit $n \geqslant 1$ et on distingue trois cas
  - ⋆ Soit la position était déjà gagnante en au plus $n-1$ coups.
  - ⋆ Soit c'est à Adam de jouer, ie $s = s_a \in S_a$, et il faut trouver un coup menant à $s_e \in S_e$ tel que s_e soit gagnant pour Adam en au plus $n-1$ coups.
  - ⋆ Soit c'est à Ève de jouer ie $s = s_e \in S_e$ et il faut
    - que Ève puisse encore jouer ;
    - que tous les coups d'Ève mènent à des positions gagnantes pour Adam en au plus $n-1$ coups.

Définition 7 : Attracteur

On définit par récurrence

$$\mathscr{A}_a^n = \{\text{positions gagnantes pour Adam en au plus } n \text{ coups}\}$$

par

$$\mathscr{A}_a^0 = V_a$$

et pour tout $n \in \mathbb{N}^*$

$$\begin{aligned}\mathscr{A}_a^n = \mathscr{A}_a^{n-1} \cup \{&s_a \in S_a,\ \exists\, s_e \in \mathscr{A}_a^{n-1},\ \ s_e \in \mathrm{Succ}(s_a)\}\\ \cup \{&s_e \in S_e,\ \mathrm{Succ}(s_e) \neq \varnothing \text{ et } \mathrm{Succ}(s_e) \subset \mathscr{A}_a^{n-1}\}\end{aligned}$$

On appelle **attracteur pour Adam** l'ensemble de toutes les positions gagnantes pour Adam :

$$\mathscr{A}_a = \bigcup_{n\in\mathbb{N}} \mathscr{A}_a^n.$$

On fait de même pour Ève.

Remarques 1

R 1 – La suite $(\mathscr{A}_a^n)_{n\in\mathbb{N}}$ est croissante et les ensembles sont des parties de S qui est fini, donc elle est stationnaire.

R 2 – Voir le sujet de TP pour une implémentation récursive avec mémoïsation : une position est gagnante pour Adam ($s \in \mathscr{A}_a$) lorsque

- s est une position de victoire d'Adam ($s \in V_a$) ;
- c'est à Adam de jouer ($s = s_a \in S_a$) et il existe un coup menant vers une position gagnante

$$\mathrm{Succ}(s) \cap \mathscr{A}_a \neq \varnothing;$$

- c'est à Ève de jouer ($s = s_e \in S_e$), elle peut jouer, et tous ses coups mènent à des positions gagnantes pour Adam

$$\mathrm{Succ}(s) \neq \varnothing \text{ et } \mathrm{Succ}(s) \subset \mathscr{A}_a.$$

Exemple 8 : Pour le jeu de Chomp 2×2

Par Adam :

- $\mathscr{A}_a^0 = \{4_e\}$.
- $\mathscr{A}_a^1 = \{4_e, 1_a, 3_a\}$.
- $\mathscr{A}_a^2 = \{4_e, 1_a, 3_a, 2_e\}$.
- $\mathscr{A}_a^3 = \{4_e, 1_a, 3_a, 2_e, 0_a\}$ qui ne bouge plus.

Par Ève :

- $\mathscr{A}_e^0 = \{4_a\}$.
- $\mathscr{A}_a^1 = \{4_a, 1_e, 3_e\}$ qui ne bouge plus.

4 Calcul d'une stratégie gagnante

- Les positions gagnantes pour Adam sont celle de $\mathscr{A}_a$.
- Les positions gagnantes pour Ève sont celle de $\mathscr{A}_e$.
- Les autres positions conduisent à un match nul (si les deux joueurs jouent de manière optimale).

Méthode : Comment calculer une stratégie pour Adam, qui sera gagnante si la position de départ est dans $\mathscr{A}_a$?

Soit $s_a \in \mathscr{A}_a$ une position gagnante contrôlée par Adam.
Alors il existe $n \in \mathbb{N}$ tel que $s_a \in \mathscr{A}_a^{(n)}$. Prenons-le minimal (on l'appelle alors **rang** de s_a).
- Si $n = 0$, $s_a \in V_a$ et il n'y a rien à faire.
- Sinon, $n \geqslant 1$ et il existe $s_e \in \mathscr{A}_a^{n-1}$ tel que $(s_a, s_e) \in A$ (on a une arête $s_a \to s_e$). On pose alors $f(s_a) = s_e$.

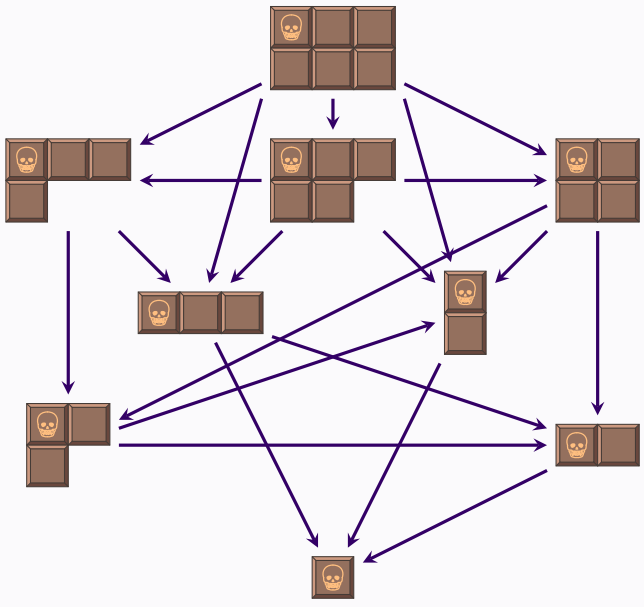
Exemple 9 : Pour le jeu de Chomp 2×2

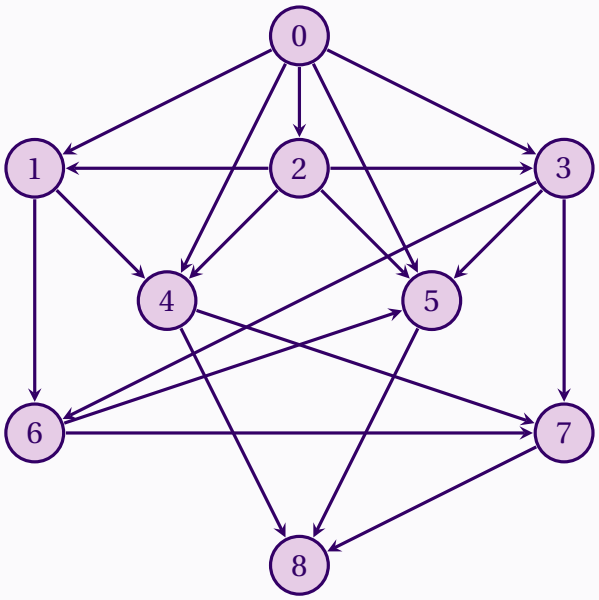
Il n'ya pas de stratégie gagnante pour Ève car la position de départ n'est pas une position gagnante.
Il y en a une pour Adam car $0_a \in \mathscr{A}_a$. $4e$ est une position de victoire. Puis
- $1_a \in \mathscr{A}_a^1$: $f(1_a) = 4_e$ car $1_a \to 4e$ et $4e \in \mathscr{A}_a^0$.
- $3_a \in \mathscr{A}_a^2$: $f(3_a) = 4_e$ car $3_a \to 4e$ et $4e \in \mathscr{A}_a^1$.
- $0_a \in \mathscr{A}_a^3$: $f(0_a) = 2_e$ car $0_a \to 2e$ et $2e \in \mathscr{A}_a^2$.

Il est inutile de préciser des coups à jouer sur des positions non gagnantes.

Exercice 2 : Jeu de Chomp 2×3

Déterminer les attracteurs et d'éventuelles stratégies gagnantes pour le jeu de Chomp 2×3.
On donne le graphe du jeu (sans tenir compte de qui contrôle chaque position).





II Algorithme du min-max

Les calculs de l'ensemble des positions gagnantes (attracteur) et d'une stratégie gagnante ne peuvent être envisagés qu'en cas de jeu ayant un graphe suffisamment petit.

C'est le cas de jeux comme le jeu de Nim, le jeu de Chomp (de taille raisonnable), le tic-tac-toe 3×3 (pas de stratégie gagnante pour chacun des joueurs, mais des stratégies conduisant au nul), mais inenvisageable pour des jeux plus complexes, ne serait-ce que le Puissance 4 (bien que ce jeu ait été résolu en 1988 : il existe une stratégie gagnante pour le premier joueur, qui nécessite que le premier coup soit joué dans la colonne centrale).

Par exemple, pour les dames, le nombre de sommet du graphe est de l'ordre de 10^{32}, pour les échecs, entre 10^{43} et 10^{50}, pour le jeu de Go, de l'ordre de 10^{100}. Il est impensable dans ce cas de pouvoir appliquer les résultats de la première partie.

Nous allons donc voir comment envisager de jouer en approchant un « bon coup » plutôt qu'en jouant de manière optimale.

Pour cette partie, nous demandons que

- le jeu soit à **information totale** ;
- les deux joueurs jouent **alternativement** ;
- le jeu soit **à coups en nombres finis** (pas de partie infinie) ;
- le jeu soit **à somme nulle** : le gain d'un joueur correspond la perte de l'autre.

Le gain peut être un nombre dans $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$.
Pour un jeu comme les jeux cités précédemment, le gain prendra trois valeurs :

- $+\infty$: le joueur gagne ;
- 0 : match nul ;
- $-\infty$: le joueur perd.

Enfin, dans cette partie, notre premier joueur se prénommera Maxence (abrégé en **Max**) et le deuxième sera Minerve (abrégé en **Min**).

Le but de Max est alors de maximiser son gain, et celui de Min est de minimiser le gain de Max.

1 Heuristique

Nous allons utiliser une fonction permettant d'évaluer numériquement la qualité d'une position : plus cette position est favorable à Max, plus sa valeur sera grande, et plus elle sera favorable à Min, plus sa valeur sera petite.

Définition 8 : Heuristique

On appelle **heuristique** du jeu dont l'ensemble des positions est (encore) noté S une fonction

$$H : \left| \begin{array}{ccc} S & \longrightarrow & \mathbb{R} \cup \{-\infty, +\infty\} \\ s & \longmapsto & H(s) \end{array} \right.$$

Remarques 2

R 1 – Une heuristique est en général construite de manière expérimentale. De sa pertinence dépend la qualité de jeu d'une IA l'utilisant.

R 2 – On associe $+\infty$ aux positions de victoire de Max et $-\infty$ à celles de Min.

R 3 – Il suffit de changer H en $-H$ pour adopter le point de vue de l'autre joueur.

Exemple 10 : Puissance 4

Pour le jeu de Puissance 4,

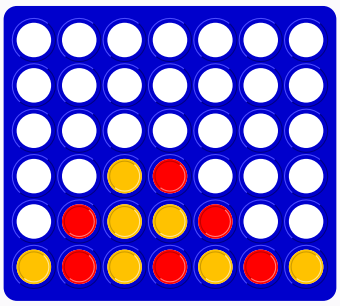


Figure 1 – Une position du Puissance 4

on peut par exemple choisir une heuristique consistant à compter le nombre d'alignements de 4 jetons possibles à un emplacement donné

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

Figure 2 – Chaque valeur correspond au nombre d'alignements de 4 jetons possibles passant par la case

et à sommer les résultats de toutes les positions occupées, positivement pour Max et négativement pour Min (sauf en cas de position gagnante pour Max ou Min, valant $+\infty$ et $-\infty$ respectivement).

Ainsi, si Max joue rouge et Min jaune, la valeur de la position de la figure 1 est

$$13+6+8+4+7+4-11-8-10-3-5-5-3=-3$$

Exemple 11 : Échecs

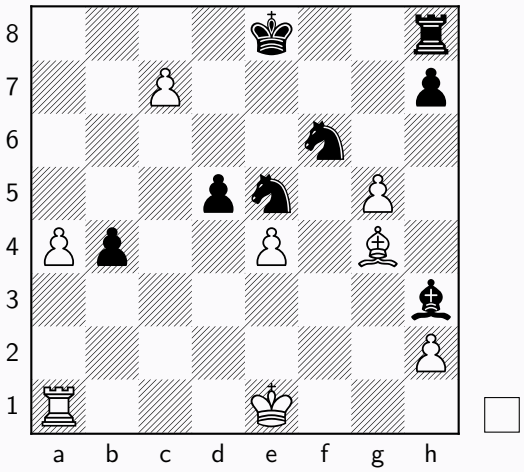


Figure 3 – Une partie d'échecs, trait aux blanc.

Une heuristique possible aux échecs consiste
- à donner la valeur $+\infty$ s'il y a échec et mat en faveur des blancs ;
- à donner la valeur $-\infty$ s'il y a échec et mat en faveur des noirs ;
- sinon, à attribuer des points aux pièces
  - 9 pour la dame ;
  - 5 pour chaque tour ;
  - 3 pour chaque fou et cavalier ;
  - 1 pour chaque pion ;

  à compter positivement pour les pièces blanches et négativement pour les pièces noires.

Ainsi, la valeur du jeu sur la figure 3 est

$$5+3+1+1+1+1+1-5-3-3-3-1-1-1=-4$$

2 Min-Max

Lorsqu'un des deux joueurs doit jouer, plusieurs coups sont possibles (entre un et sept au puissance 4). On peut calculer l'heuristique correspondante à chacune des positions résultantes et choisir le coup qui donne une heuristique maximale pour Maxence ou minimale pour Minerve.

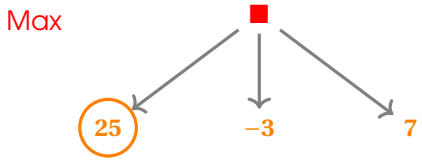


FIGURE 4 – Maxence cherche à maximiser son gain.

Si Maxence veut jouer avec deux coups d'avance, il va supposer que Minerve joue le mieux possible, et donc qu'elle cherche à minimiser le gain de Max au coup suivant.

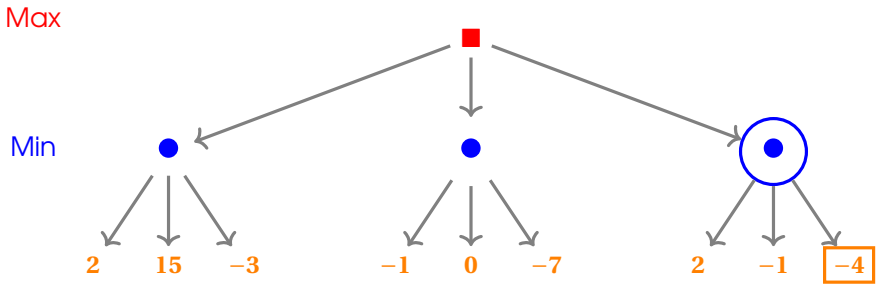


FIGURE 5 – Maxence cherche à maximiser le gain que Minerve va chercher à minimiser.

Et ainsi de suite : on peut regarder les positions à un nombre quelconque de coups et maximiser le gain lorsque c'est à Maxence de jouer et le minimiser lorsque c'est à Minerve de jouer.

Compléter l'exemple suivant en supposant d'abord que c'est à Max de jouer, puis en supposant ensuite que c'est à Min.

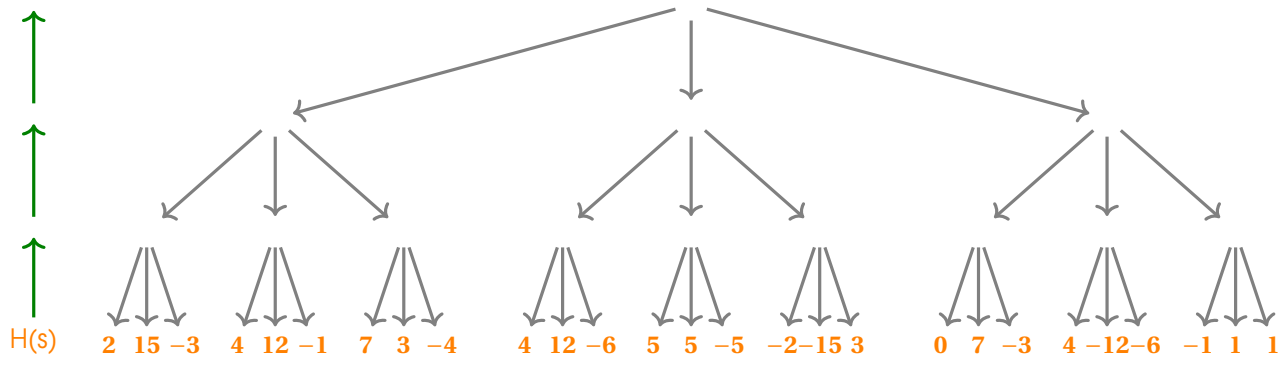


FIGURE 6 – Anticipation à 3 coups. Les valeurs sur les feuilles de l'arbre sont données par l'heuristique. Traiter deux cas : celui où c'est Max qui prévoit son coup, et celui c'est Min.

Solution dans le cas où c'est Max qui calcule son coup :

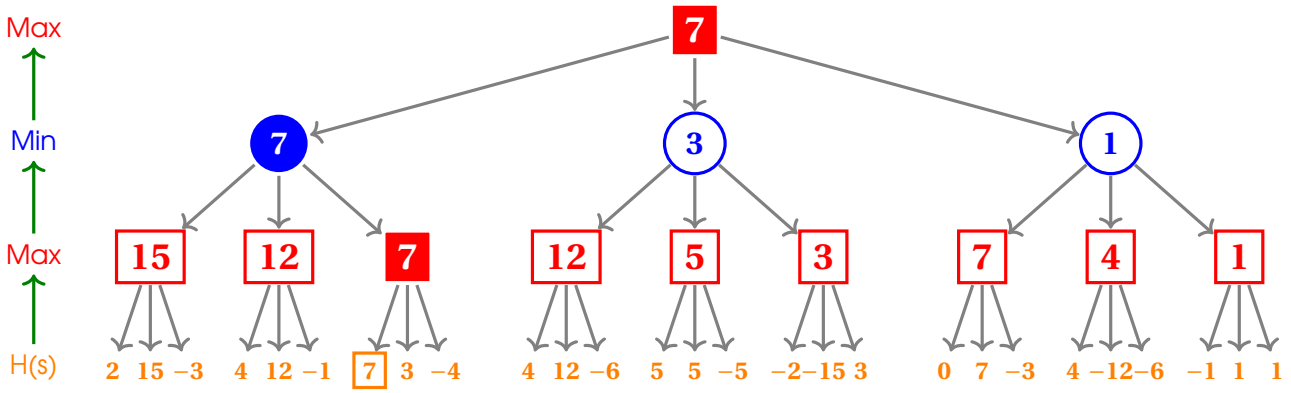


Figure 7 – Max a intérêt à jouer le coup de gauche.

On peut reprendre le même exemple, mais en supposant que c'est à Minerve de jouer :

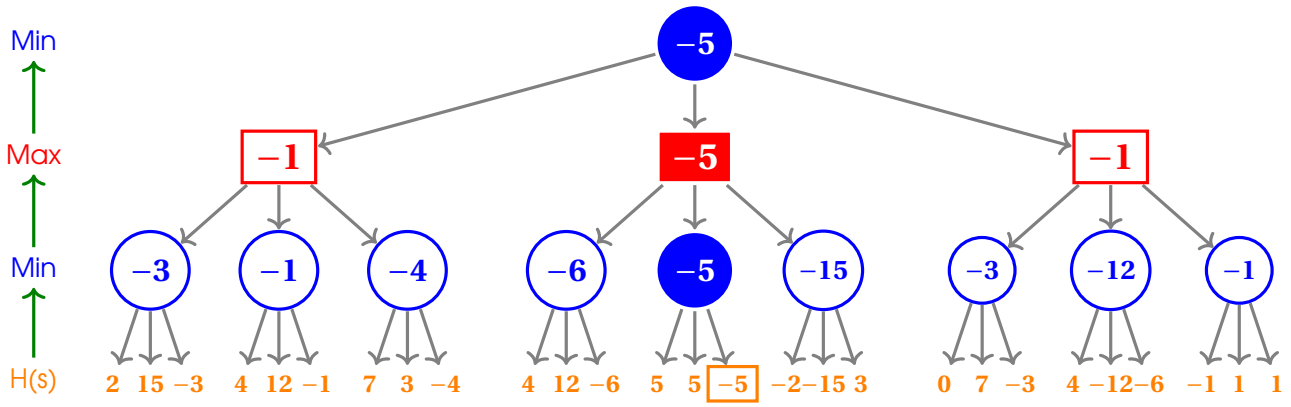


Figure 8 – Min a intérêt à jouer le coup au centre.

Un inconvénient, tout de même : le nombre de positions à étudier croît exponentiellement !

3 Implémentation

On peut implémenter l'algorithme en écrivant deux fonctions mutuellement récursives (qui s'appellent l'une l'autre) :

- `maximin(position, nb, H)`, destinée à Maxence, qui va chercher à maximiser l'heuristique après `nb` coups en partant de la position `position`, en supposant que son adversaire joue au mieux ;
- `minimax(postion, nb, H)`, destinée à Minerve, qui va chercher à minimiser l'heuristique après `nb` coups en partant de la position `postion`, en supposant que son adversaire joue au mieux.

On suppose disposer d'une fonction `successeur(sommet)` renvoyant la liste de toutes les positions atteignables en un coup à partir de la position `sommet`.

Python

```python
def maximin(position, nb, H):
    succ = successeurs(position) # liste des successeurs de position
    if nb == 0 or succ == []: # position est une feuille de l'arbre
        return H(position)
    maxi = -float('inf') # on cherche à maximiser la valeur des successeurs
    for position1 in succ:
        valeur = minimax(position1, nb - 1,  H) # valeur du successeur position1
        maxi = max(maxi, v)
    return maxi
```

Python
```python
def minimax(position, nb, H):
    succ = successeurs(position) # liste des successeurs de position
    if nb == 0 or succ == []: # position est une feuille de l'arbre
        return H(position)
    mini = float('inf') # on cherche à minimiser la valeur des successeurs
    for position1 in succ:
        valeur = maximin(position1, nb - 1, H) # valeur du successeur position1
        mini = min(mini, v)
    return mini
```

Remarque 6

Pour faire jouer le joueur, il faudra aussi renvoyer un coup réalisant le maximum/minimum.

Une petite amélioration du code est possible en remarquant que $\min(a,b) = -\max(-a,-b)$. Autrement dit, un arbre pour anticiper un coup joué par Min est un arbre pour anticiper un coup joué par Max dans lequel on a multiplié toutes les valeurs par -1.

Elle porte le nom de **negamax**.

Python
```python
def negamax(position, nb, H, signe):
    succ = successeurs(position) # liste des successeurs de position
    if nb == 0 or succ == []: # position est une feuille de l'arbre
        return signe * H(position)
    # on cherche à maximiser la valeur des successeurs au tour de Max
    # celle de leur opposé au tour de Min.
    maxi = -float('inf')
    for position1 in succ:
        valeur = negamax(position1, nb - 1,  H, -signe)
        maxi = max(maxi, -valeur)
    return maxi
```

Lorsque l'on cherche un coup à jouer par Max, on appellera `negamax(position, nb, H, 1)`, et lorsque c'est à Min de jouer, on appellera `negamax(position, nb, H, -1)` (l'heuristique étant donnée pour Max, donc à multiplier par -1 pour avoir cette de Min).

4 Cas des jeux de petite taille

$$H(s) = \begin{cases} +\infty & \text{si } s \in V_a \\ -\infty & \text{si } s \in V_e \\ 0 & \text{sinon} \end{cases}$$

L'algorithme de Min-Max avec profondeur $|S|$ donne une stratégie gagnante pour Adam si le gain résultant vaut $+\infty$, une stratégie menant au nul s'il vaut 0.

Exemple 12 : Chomp 2×2

Algorithme min-max complet pour Adam, le numéros de positions sont celles de la première partie du chapitre. On utilise la notation « position : valeur ».

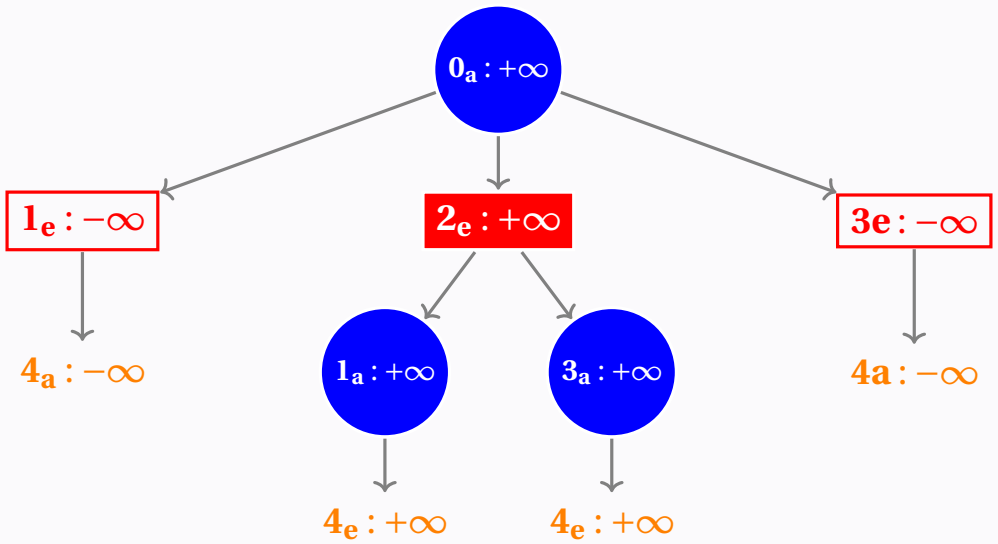


Figure 9 – Une stratégie gagnante pour Adam grâce à l'algorithme du min-max.

Remarque 7

On voit dans ce cas d'éventuels appels récursifs redondants : un mémoïsation est alors envisageable.

5 Complément Hors-Programme : l'élagage α–β

L'algorithme min-max est de complexité exponentielle en le nombre `nb` de coups prévus à l'avance (appelé profondeur de l'arbre) : si pour chaque position, il y a un nombre de coup majoré par a, on obtient une complexité en $\mathcal{O}(a^n)$. C'est trop lent pour des jeux comme les échecs ou le go si on veut une profondeur intéressante (au moins 20).

L'idée, pour améliorer cela, et d'éviter d'explorer certaines branches dans l'arbre lorsque l'on n'est certain que cela n'intéressera pas Max ou Min.

L'élagage α–β consiste à éviter l'évaluation d'un grand nombre de positions en coupant des branches de l'arbre lorsque l'on est certain qu'elles ne participeront pas à la stratégie optimale.

Coupure α Dans la situation de la figure 10, on a entouré les positions déjà explorées.

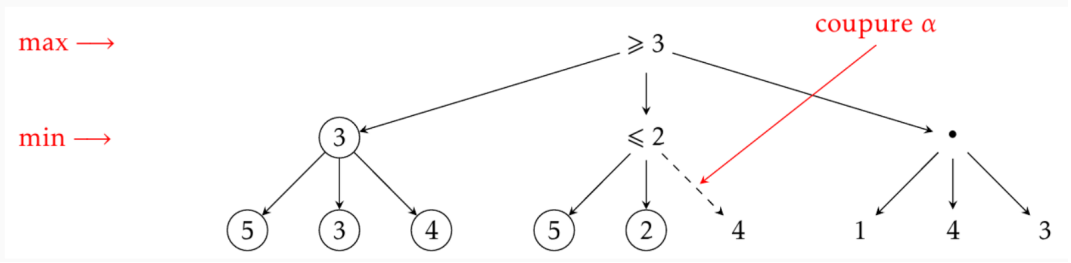


Figure 10 – Une coupure α

Vu la valeur 3 trouvée pour le premier coup envisagé par Max, et vu que dans le deuxième coup, la valeur (min) ne pourra dépasser 2, il n'est inutile de continuer à explorer la branche en pointillés (ainsi que les suivantes dans ce coup envisagé par Max.)

Ignorer une telle branche s'appelle faire une **coupure α**.

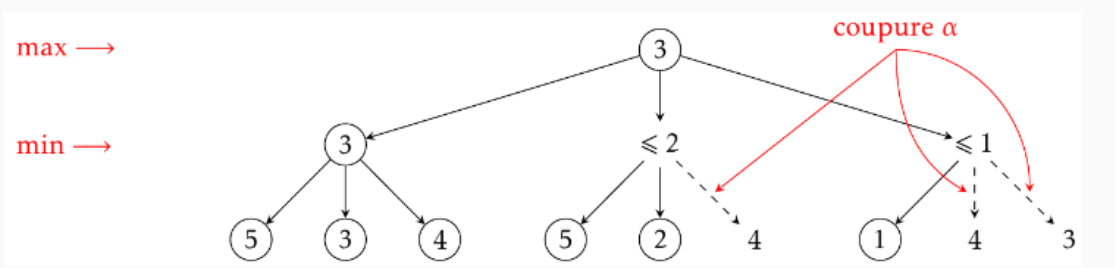


Figure 11 – D'autres coupures α

En répétant ce procédé, on se rend compte que l'on peut obtenir la valeur de la racine de l'arbre (3) en ayant économisé le parcours de 3 branches.

Coupure β Symétriquement, si la racine de l'arbre est une position Min, on peut éviter l'exploration de certaines branches appelé **coupure β**.

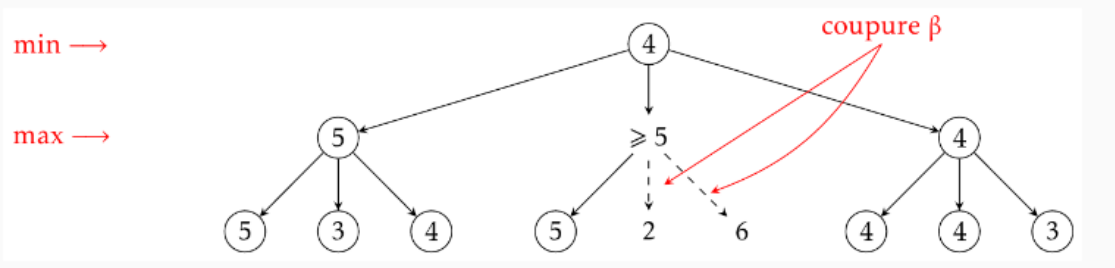


Figure 12 – Coupures β

On en déduit une mise à jour de nos fonctions `minimax` et `maximin` en ajoutant deux arguments `alpha` et `beta` représentant respectivement un minorant et un majorant de la valeur d'un nœud (initialement à $-\infty$ et $+\infty$ c'est-à-dire `-float('inf')` et `float('inf')`.)

Python
```python
def maximin(position, nb, H, alpha, beta):
    succ = successeurs(position) # liste des successeurs de position
    if nb == 0 or succ == []: # position est une feuille de l'arbre
        return H(position)
    maxi = -float('inf') # on cherche à maximiser la valeur des successeurs
    for position1 in succ:
        valeur = minimax(position1, nb - 1,  H, alpha, beta) # valeur du successeur position1
        maxi = max(maxi, valeur)
        if maxi >= beta: # coupure beta
            return maxi
        alpha = max(alpha, maxi) # mise à jour du minorant alpha
    return maxi
```

Python
```python
def minimax(position, nb, H, alpha, beta):
    succ = successeurs(position) # liste des successeurs de position
    if nb == 0 or succ == []: # position est une feuille de l'arbre
        return H(position)
    mini = float('inf') # on cherche à minimiser la valeur des successeurs
    for position1 in succ:
        valeur = maximin(position1, nb - 1, H, alpha, beta) # valeur du successeur position1
        mini = min(mini, valeur)
        if mini <= alpha: # coupure alpha
            return mini
        beta = min(beta, mini) # mise à jour du majorant beta
    return mini
```

Remarque 8

C'est cet algorithme qui est utilisé pour jouer aux échecs (avec environ 20 coups d'avance) ou au jeu de go.

Exercice 3

Écrire une version `negamax` avec élagage α–β.
Indication : α et β seront aussi à multiplier par -1...