THÉORIE DES JEUX
JEUX A DEUX JOUEURS
ALGORITHME MINMAX

Informatique Tronc Commun

E. CLERMONT

ALGORITHME MINMAX

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

Exemples : jeu de Nim, le jeu de Chomp (de taille raisonnable), le tic-tac-toe 3*x*3 (pas de stratégie gagnante pour chacun des joueurs, mais des stratégies conduisant au nul),

- mais **inenvisageable pour des jeux plus complexes.**

Exemples :
- pour le **jeu de dames** : le nombre de sommets 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}.

⇒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.

Il devient donc nécessaire de s'appuyer non plus sur une évaluation exacte de la position, mais sur une **estimation de la valeur de la position atteinte**.

ALGORITHME MINMAX

- Pour cette partie, il faut 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.

ALGORITHME MINMAX

- Heuristique

Dans la suite de cette section, nous supposerons posséder une **fonction h** qui a toute position possible p du jeu associe une valeur dans IR, de sorte que :

- plus h(p) est grand, meilleure est la position pour Adam;
- plus h(p) est petit, meilleure est la position pour Eve.

Une telle fonction est appelée **une heuristique**.

Remarques

- Une heuristique est en général construite de manière expérimentale.

De sa pertinence dépend la qualité de jeu d'une IA l'utilisant.

- On associe +∞ aux positions de victoire de Adam et -∞ à celles de Eve.
- Il suffit de changer *h* en -*h* pour adopter le point de vue de l'autre joueur.

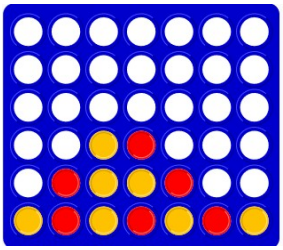
ALGORITHME MINMAX

Exemple d'heuristique

- **Pour le jeu du Puissance 4**
- Pour le jeu de 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

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 Adam et négativement pour Eve.
(sauf en cas de position gagnante pour Adam ou Eve, valant +∞ et -∞ respectivement).
Ainsi, si Max joue avec les pions rouges et Min avec les pions jaunes, la valeur de la position ci-dessus est : 13+6+8+4+7+4-11-8-10-3-5-5-3 = 3

ALGORITHME MINMAX

Exemple d'heuristique

- **Pour les échecs**

Une heuristique possible aux échecs consiste :

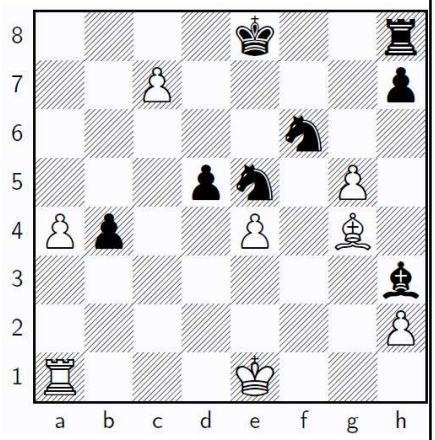
- à donner la valeur +∞ s'il y a échec et mat en faveur des blancs;
- à donner la valeur -∞ s'il y a échec et mat en faveur des noirs;
- sinon, à attribuer des points aux pièces :

⋆ 9 pour la dame;

⋆ 5 pour chaque tour;

⋆ 3 pour chaque fou et cavalier;

⋆ 1 pour chaque pion;



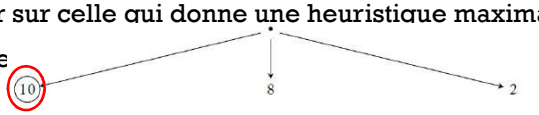
à compter positivement pour les pièces blanches et négativement pour les pièces noires.

- Ainsi, la valeur du jeu de la situation ci-dessus est :

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

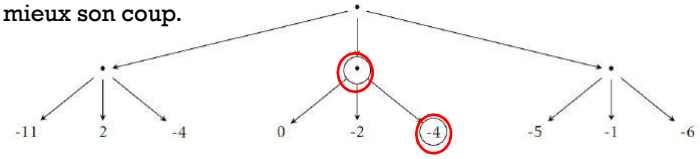
ALGORITHME MINMAX

- On peut calculer l'heuristique correspondante à chacune des positions atteignables et choisir de jouer sur celle qui donne une heuristique maximale pour Adam ou minimale pour Eve.
- Exemple

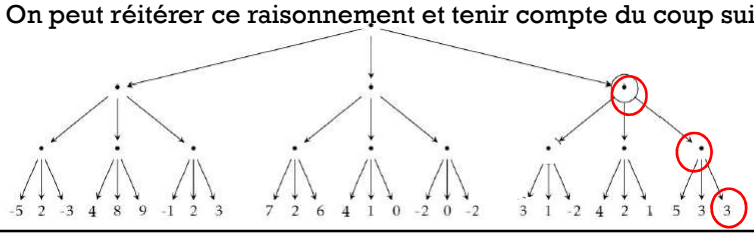


3 configurations possibles pour Adam

Adam choisira le coup le plus a gauche, correspondant à une évaluation égale a 10 (maximum de 10,8 et 2).
Mais Adam peut aussi tenir compte du coup que va jouer Eve ensuite, et donc calculer l'heuristique de chacune des positions qu' Eve pourra atteindre.
Ci-dessous, on constate qu'il vaut mieux pour Adam jouer le coup central, en partant du principe qu' Eve joue au mieux son coup.



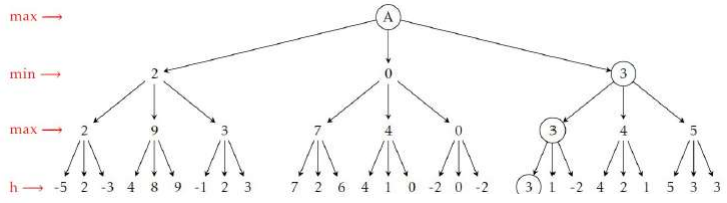
- On peut réitérer ce raisonnement et tenir compte du coup suivant, joue cette fois par Adam.



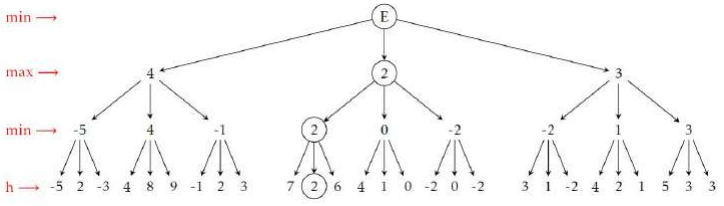
En tenant compte des 2 coups suivants, Adam a en fait intérêt à jouer le coup le plus a droite.

ALGORITHME MINMAX

- Pour calculer le meilleur coup d'Adam, il faut donc commencer par calculer la valeur de l'heuristique de toutes les positions atteignables en n coups (les feuilles de l'arbre).
- si n est impair le père de chacune de ces feuilles se verra attribuer la valeur maximale de ses fils (car Adam aura joué en dernier). Ainsi, de proche en proche, chaque position de l'arbre se verra attribuer une valeur
- Si n est pair, Eve aura joué le dernier coup : le père de chacune de ces feuilles se verra donc attribuer le minimum des valeurs de ses fils



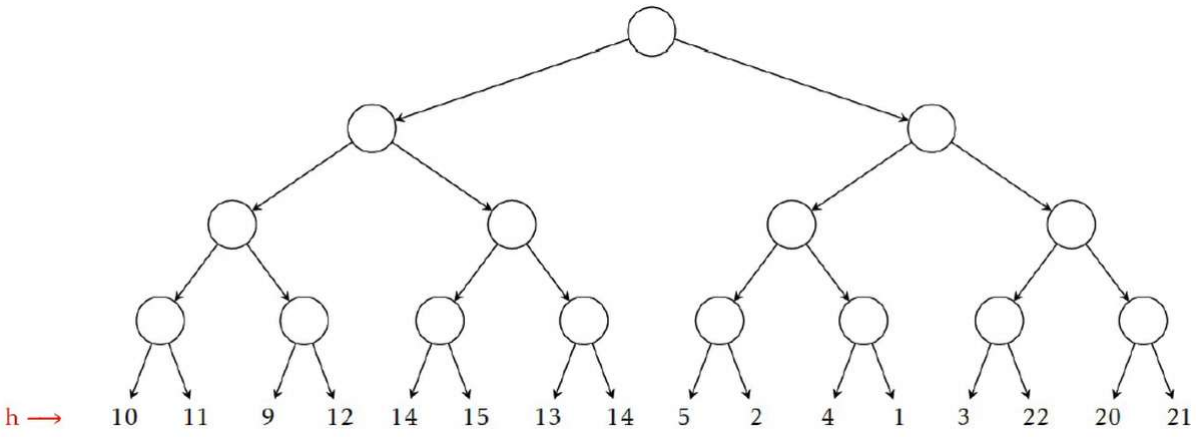
Adam joue



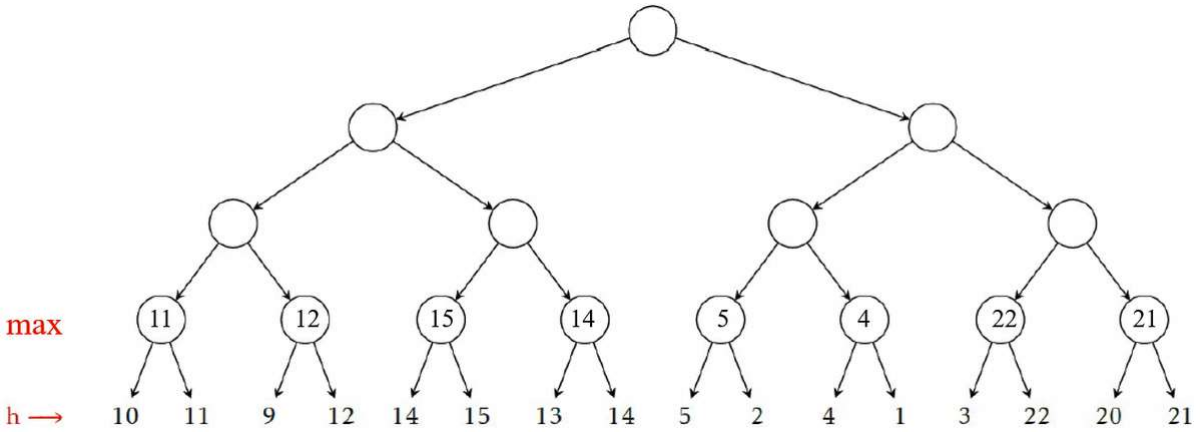
Eve joue

EXERCICE

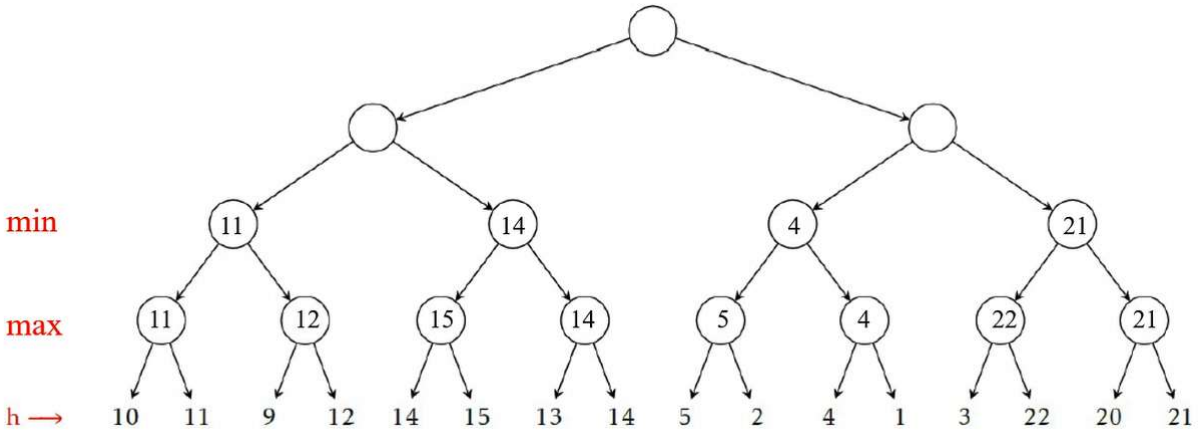
Adam joue



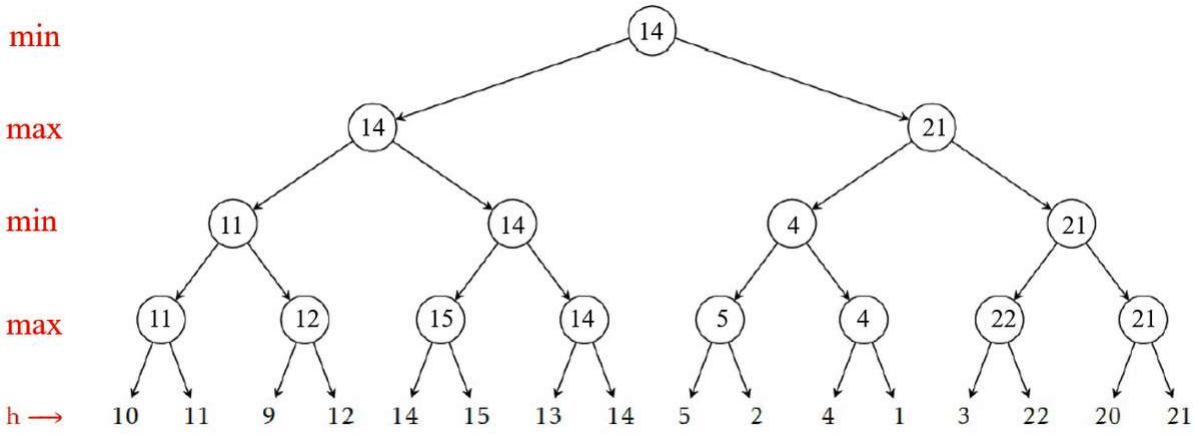
EXERCICE



EXERCICE



EXERCICE



ALGORITHME MINMAX

- On peut implémenter l'algorithme en écrivant 2 fonctions mutuellement récursives (qui s'appellent l'une l'autre) :
- ■ maximin(position, nb, h), destinée à Adam, 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 Eve, 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.

ALGORITHME MINMAX

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

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

ALGORITHME MINMAX

- 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 Eve 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**.

```
def negamax(position, nb, h, signe):
    lesSuccesseurs = successeurs(position) # liste des successeurs de position

    if nb == 0 or succ == []: # position = feuille de l'arbre
        return signe * h(position)

     # on cherche à maximiser la valeur des successeurs au tour d'Adam
    # celle de leur opposé au tour d'Eve.
    maxi = -float('inf')
    for position1 in lesSuccesseurs:
        valeur = negamax( position1, nb - 1, h, -signe )
        maxi = max(maxi, -valeur)
    return maxi
```

THÉORIE DES JEUX
JEUX A DEUX JOUEURS
ALGORITHME MINMAX

Informatique Tronc Commun

E. CLERMONT