

# Chapitre 11

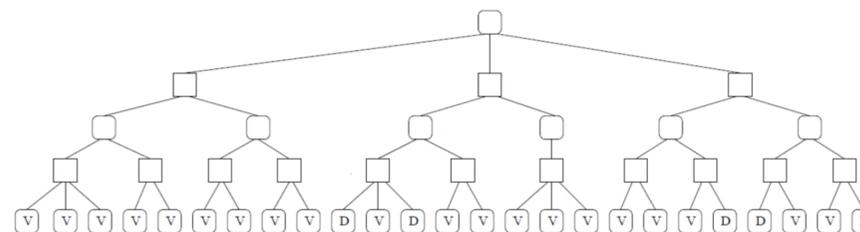
## Jeux d'accessibilité : algorithme minimax

On poursuit avec la recherche de stratégie gagnante dans un jeu d'accessibilité :

- à deux joueurs et à coups asynchrones : le joueur A débute, puis c'est au tour du joueur B, et ainsi de suite,
- à information complète et parfaite : chaque joueur connaît tous les coups que lui et son adversaires peuvent entreprendre et ont préalablement entrepris ainsi que les gains qui en résultent,
- sans hasard,
- à somme nulle : la somme des gains des deux joueurs est nulle : ce que l'un gagne, l'autre le perd. Si l'un gagne, l'autre perd. Il est possible d'aboutir à un match nul, ou à une partie sans fin.

Nous avons vu comment déterminer toutes les positions gagnantes pour un joueur en calculant le bassin attracteur par un algorithme  $O(|S| + |A|)$ . Cette approche n'est possible que pour des jeux dont la taille du graphe  $G = (S, A)$  est très limitée; par exemple le nombre de sommets (configurations de jeux) est estimé à  $10^{32}$  pour le jeu de dame,  $10^{50}$  au jeu d'échec et  $10^{100}$  au jeu de Go contre 42 pour le jeu de Nim. Nous allons voir comment élaborer une méthode efficace d'élaboration de stratégie pour les jeux plus complexes à l'aide de l'algorithme minimax (dû à John Von-Neumann, 1927).

On ne considère plus que des jeux à parties finies : cela implique que le graphe de jeu est sans cycle ; c'est à dire que c'est un arbre ; on le représente alors sous forme d'un arbre enraciné, biparti selon les états contrôlés par le premier joueur  $J_0$  et le second  $J_1$  :



- la racine est l'état initial du jeu,
- chaque noeud est une configuration du jeu, contrôlé par le premier joueur, ou par le second,
- chaque fils représente un coup possible à partir d'un noeud,
- chaque feuille (sommets final) représente une fin de partie,
- chaque branche représente une séquence de coups,
- une partie est un chemin débutant à la racine et s'achevant en une feuille : puisque dans un arbre un chemin entre deux sommets est unique, la partie est entièrement caractérisée par la feuille où elle s'achève.

### 1 Stratégie optimale par minimax

Le jeu étant à somme nulle, les gains des joueurs en chaque sommet final sont des nombres opposés. Appelons les deux joueurs Max et Min, plutôt que  $J_0$  et  $J_1$ , où Max désigne celui qui débute la partie, et notons les gains de Max positivement, et ceux de Min négativement ; lorsque le gain ne consiste qu'en une victoire, une défaite ou un match nul, on les considérera comme des gains  $+1$ ,  $-1$  ou  $0$ . La stratégie pour Max (respectivement Min) consiste à achever la partie sur un sommet final de gain maximal (resp. minimal).

#### 1.1 Évaluation des feuilles et des noeuds

Chaque sommet final  $f \in S$  (=feuille) est évalué par un nombre dans  $\overline{\mathbb{R}} = \mathbb{R} \cup \{+\infty, -\infty\}$  noté  $E(f)$ , représentant le gain du joueur Max (mais aussi celui du joueur Min compté négativement).

On étend alors l'évaluation  $E$  sur les noeuds restants par induction (récurrence), de la façon suivante :

**Définition 1.1**

L'évaluation  $E$  en un noeud  $s \in S$  de l'arbre biparti du jeu est défini par :

–  $E(s) = E(f)$  si  $s$  est une feuille  $s = f$ , est le gain du joueur Max (ainsi que celui de Min mais compté négativement) pour la partie correspondant à cet état final.

– Pour un noeud  $s_0$  controlé par Max,  $E(s_0)$  est le maximum des évaluations sur les fils de  $s_0$  :

$$E(s_0) = \max\{E(s_1) \mid (s_0, s_1) \in A\}$$

– Pour un noeud  $s_1$  controlé par Min,  $E(s_1)$  est le minimum des évaluations sur les fils de  $s_1$  :

$$E(s_1) = \min\{E(s_2) \mid (s_1, s_2) \in A\}$$

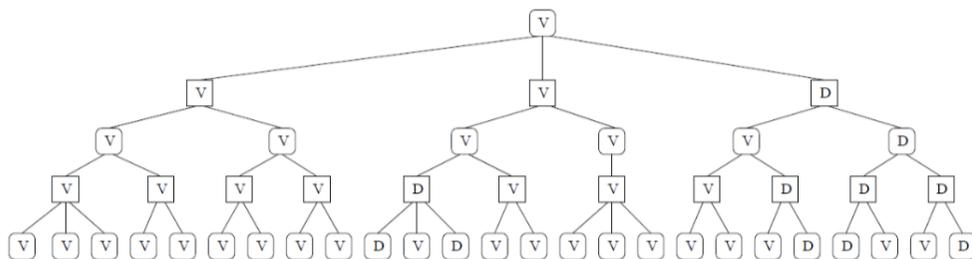
Il est facile de vérifier par récurrence sur la hauteur du noeud qu'au sommet  $s_0$  (resp.  $s_1$ ) le gain final escompté de Max sera  $\geq E(s_0)$  (resp.  $\leq E(s_1)$ ). Il en découle immédiatement :

**Théorème 1.1**

Si le joueur Max (respectivement Min) joue en choisissant au noeud  $s$  le coup qui maximise (resp. minimise)  $E(s)$ , alors pour la partie le gain de Max sera égal à l'évaluation  $E(r)$  du noeud racine  $r$ .

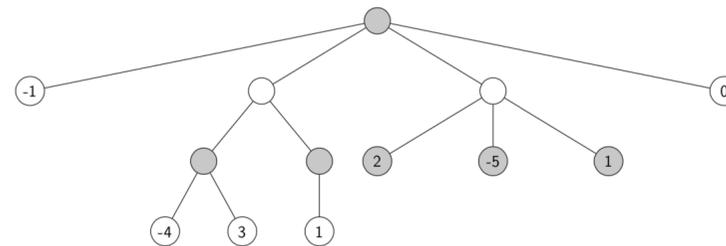
Ainsi on considère que les deux joueurs ont un comportement rationnel, et vont chacun essayer de maximiser leur gain, c'est à dire vont suivre l'évaluation maximale (resp. minimale) pour Max (res. pour Min). Le nom de la stratégie minimax provient de : "minimiser la perte maximale" ou encore "minimiser le gain maximal de l'adversaire".

• **Exemple.** Pour l'arbre de jeu donné ci-dessus il existe une stratégie gagnante pour le joueur Max :



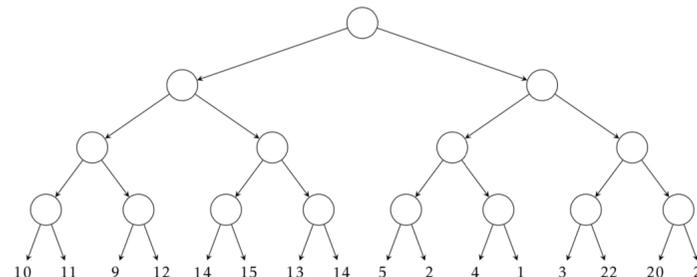
par exemple jouer toujours le premier coup (de gauche).

**Exercice 1.** Dans le jeu représenté par cet arbre, on a donné le gain sur chaque feuille. Quel sera le gain du joueur qui débute si chaque joueur joue optimalement ?



**Exercice 2.** Dans le jeu représenté par cet arbre, on a donné le gain sur chaque feuille. Quel sera le gain pour Max si chaque joueur joue optimalement ?

- a) Lorsque c'est Max qui débute.
- b) Lorsque c'est Min qui débute.
- c) Est-il préférable au début pour le joueur Max de prendre ou de laisser la main ?



**1.2 Évaluation heuristique**

En principe il n'est pas possible dès que le jeu est un peu intéressant de calculer la fonction d'évaluation, et d'en déduire la stratégie minimale. En effet, par exemple, si pour un arbre régulier le degré sortant (nombre de coups possibles) pour chaque noeud est égal à  $d$ , et la profondeur du graphe (nombre de coup d'une partie) est égal  $p$ , il y a :

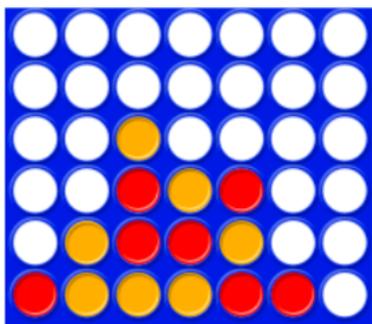
$$\sum_{k=0}^p d^k = \frac{1 - d^{p+1}}{1 - d} = O(d^{p+1})$$

noeuds, et la complexité du calcul de l'évaluation est exponentielle en fonction de la profondeur de l'arbre (nombre de coups durant une partie), de l'ordre de  $O(d^{p+2})$ .

Pour établir une intelligence artificielle capable de jouer intelligemment, il est nécessaire de :

- Évaluer à l'aide d'une heuristique  $H$ , une bonne approximation de l'évaluation d'un sommet qui n'est pas une feuille ;  $E(s)$  sera évalué par  $H(s)$ , afin de :
- Limiter la profondeur de l'arbre étudié aux  $p$  coups suivants, pour  $p$  pas trop grand (c'est l'horizon de l'IA) ; l'évaluation des noeuds à profondeur  $p$  (les feuilles du sous-arbre étudié) sera alors obtenu à l'aide de l'heuristique  $H$  ; celle des noeuds aïeux, par le calcul récursif de  $E$ .

Exemple d'heuristique : le jeu Puissance 4 consiste à aligner 4 pions de même couleur sur une grille de six rangées par sept colonnes. Les joueurs placent alternativement un jeton de leur couleur dans une colonne, qui glisse jusqu'à la position libre la plus basse sur la colonne. Le gagnant est le premier joueur à avoir aligné quatre jetons de sa couleur (en ligne colonne ou diagonale) ; si à la fin du jeu (lorsqu'aucun emplacement ne reste disponible dans la grille) aucun joueur n'y est parvenu, la partie est déclarée nulle.



Une heuristique simple peut s'obtenir en affectant à chaque position une valeur égale au nombre au nombre de pions pouvant potentiellement être alignés lorsqu'on possède un pion sur cette position, puis à sommer la somme des valeurs occupées, positivement pour Max, négativement pour Min, sauf pour une position gagnante pour Max (resp. Min) qui sera évaluée à  $+\infty$  (resp.  $-\infty$ ).

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

Par exemple, pour la configuration ci-dessus, si la couleur de Max est le jaune, la valeur heuristique est égale à :

$$-3 + 4 + 6 + 5 - 8 - 11 + 11 + 7 - 10 + 13 - 5 + 8 - 11 + 4 = 2$$

Autre exemple : Pour le jeu d'échec, on peut sommer la valeur des pièces en sa possession : 9 pour la reine, 5 pour une tour, 3.25 pour un fou ou un cavalier, et 1 pour un pion.

L'algorithme récursif d'évaluation peut s'écrire alors :

**Minimax**( $s$ ,  $p$ ,  $E$ ,  $H$ , evalMax)

Entrée : un noeud  $s$  d'un arbre de jeu

un entier  $p \geq 0$  : horizon de l'IA

la fonction d'évaluation  $E$  pour les feuilles de l'arbre,

une heuristique  $H$  pour les autres noeuds,

un booléen evalMax Vrai lorsque c'est au tour de Max.

Sortie : Une estimation par minimax de l'évaluation  $E(s)$  du noeud  $s$ .

SI  $s$  est une feuille ALORS :

REVOYER  $E(s)$

SI  $p = 0$  ALORS :

REVOYER  $H(s)$

SI evalMax est Vrai ALORS :

REVOYER  $\max\{ \text{Minimax}(s', p-1, E, H, \text{Faux}) \mid s' \text{ fils de } s \}$

SINON :

REVOYER  $\min\{ \text{Minimax}(s', p-1, E, H, \text{Vrai}) \mid s' \text{ fils de } s \}$

ce qui donnerait en code Python, donnée une fonction `files(s)` qui renvoie la liste des fils de `s` dans l'arbre, et les fonctions  $E$  et  $H$  (définies au préalable, sur les feuilles pour  $E$ , sur tous les sommets pour  $H$ ) :

```
def minimax(s, p, evalMax=True):
    if files(s) == []:
        return E(s)
    if p == 0:
        return H(s)
    if evalMax:
        maxi = -float('inf')
        for s1 in files(s):
            eval = minimax(s1, p-1, False)
            if maxi < eval:
                maxi = eval
```

```

    return maxi
else:
    mini = float('inf')
    for s1 in fils(s):
        eval = minimax(s1,p-1,True)
        if eval < mini:
            mini = eval
    return mini

```

La stratégie s'obtient en construisant en outre, une liste des fils où le max/min a été atteint à chaque appel récursif :

```

def rec_minimax(s,p,evalMax=True, L):
    if fils(s) == []:
        return E(s), []
    if p == 0:
        return H(s), []
    if evalMax:
        maxi = -float('inf')
        for s1 in fils(s):
            eval, L1 = rec_minimax(s1,p-1,False,L)
            if maxi < eval:
                maxi = eval
                nmax = s1
        L1.append(nmax)
        return maxi, L1
    else:
        mini = float('inf')
        for s1 in fils(s):
            eval, L1 = rec_minimax(s1,p-1,True, L)
            if eval < mini:
                mini = eval
                nmin = s1
        L1.append(nmin)
        return mini, L1

def minimax(s,p,evalMax=True):
    evaluation, strategie = rec_minimax(s,p,evalMax=True, [])
    return evaluation, strategie[::-1]

```