Théorie des jeux : Algorithme MinMax

Introduction

Classification des jeux

On peu classer les jeux existants selon plusieurs catégories :

- à **somme nulle** vs à somme **non nulle**
- information **complète** vs information **incomplète**
- **déterministe** vs avec du **hasard**
- coups **synchrones** vs coups **asynchrones**
- **unique** vs **répété**

Stratégie de MinMax

Von Neumann, 1928

Stratégie indépendante du jeu auquel on veut jouer tant que ce jeu est :

- à deux joueurs
- à somme nulle
- à information complète
- à coups asynchrones (nombre fini et limité)
- déterministe

Beaucoup de jeux de plateau correspondent à ces critères : échecs, go, othello, puissance4, morpion, etc.

Représentation de jeux

Arbre de décision

On représente la suite des coups à jouer par un arbre n-aire

- racine : jeu à l'état initial
- nœud : situation de jeu
- fils : possibilité de coup joué par l'autre joueur à partir de la nouvelle position de jeu
- branche : séquence de coups
- feuille : fin de partie ou fin du nombre de coups à anticiper
- degré de l'arbre : nombre maximum de possibilité de coups pour un tour de jeu

Principes

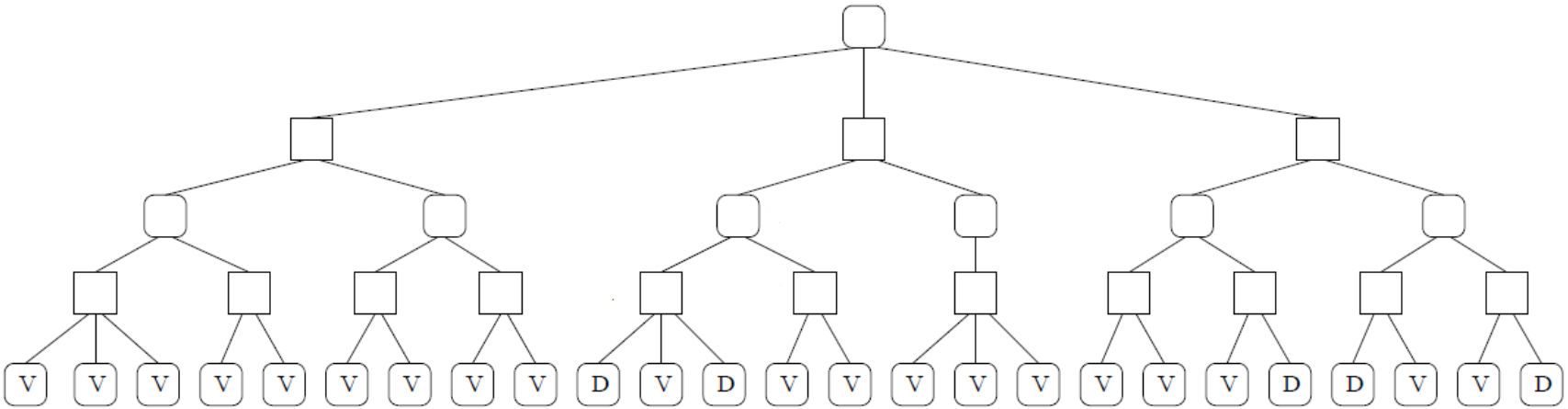
Stratégie de MinMax

- Le joueur qui commence (joueur MAX) :
  - cherche à trouver, parmi toutes les situations à sa disposition, une situation qui lui permet de **maximiser** ses gains
- L'autre joueur (joueur MIN) :
  - doit trouver, à partir de toutes les situations qui conduisent à la victoire du permier joueur, la situation qui **minimise** les gains de ce joueur

Scénario idéal

L'arbre complet

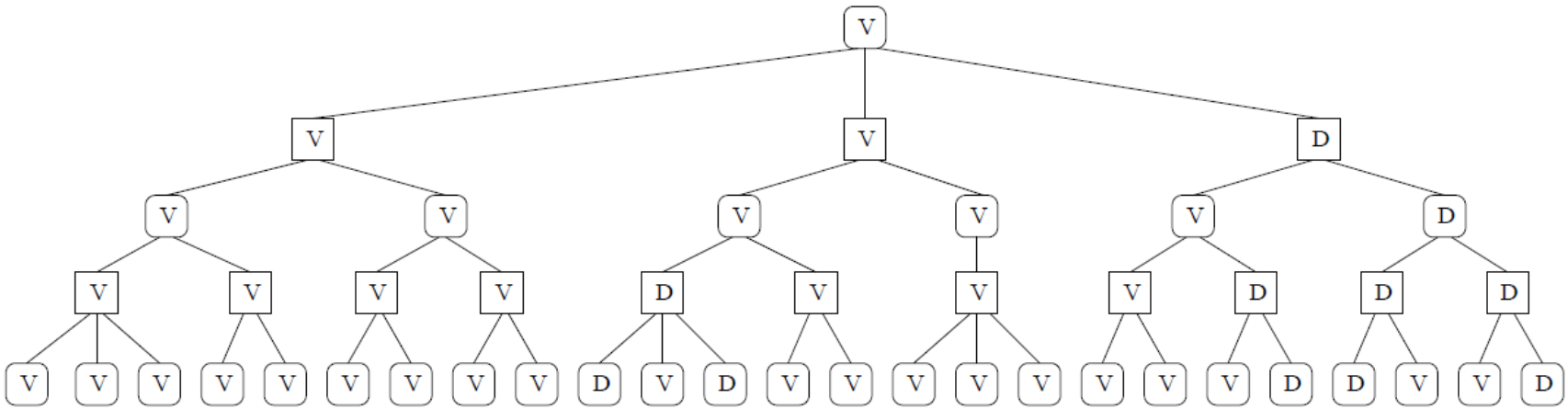
A partir d'une position donnée, il existe une arborescence de coups jusqu'à la victoire (V), au nul (N) ou à la défaite (D). A titre d'exemple, que penser de cette situation ? (cercle représente le joueur MAX, carré représente le joueur MIN)



Scénario idéal

L'arbre complet

A partir d'une position donnée, il existe une arborescence de coups jusqu'à la victoire (V), au nul (N) ou à la défaite (D). A titre d'exemple, que penser de cette situation ? (cercle représente le joueur MAX, carré représente le joueur MIN)



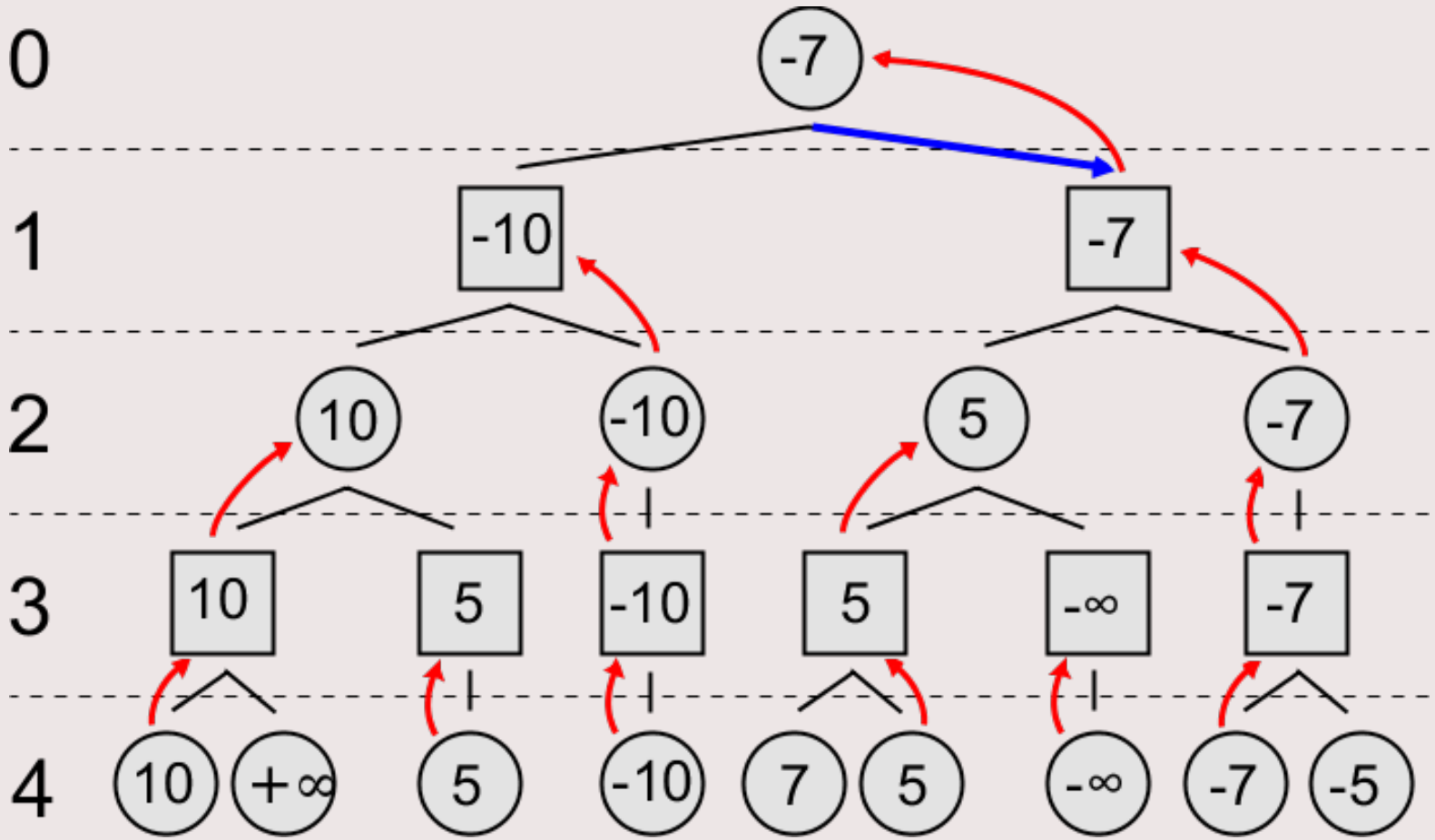
Scénario réel

L'arbre incomplet avec une profondeur limitée

- En réalité, il est impossible de développer entièrement l'arbre du jeu et de dire si une feuille correspond à une position gagnante ou à une position perdante (à cause d'une complexité combinatoire)
- Dans ce cas, il est nécessaire de disposer d'une fonction d'évaluation (heuristique), capable d'estimer le plus précisément possible la qualité d'une position.
  - on définit alors une profondeur de recherche (horizon de l'IA)
  - les feuilles de l'arbre sont associées à une valeur numérique donnée par cette fonction d'évaluation.

Scénario réel

Exemple avec profondeur 4



Algorithme de MinMax

Principe

- Fonction récursive sur la profondeur
  - paramètres :
    - nœud : configuration actuelle du plateau du jeu
    - profondeur : profondeur actuelle
    - evalMax : si vrai alors joueur MAX sinon joueur MIN
  - retour : valeur du nœud
- Conditions d'arrêt
  - fin de jeu (victoire, nul ou défaite)
  - ou profondeur = 0 (on atteint l'horizon d'IA)

Algorithme de MinMax

Pseudo-code : version 1

Fonction MinMax(noeud : Plateau, profondeur : Entier, evalMax : Booleen) : Entier
Début
  Si profondeur $= 0$ **ou** victoire(noeud) **ou** defaite(noeud) **ou** nul(noeud) **Alors**
    Retourner evaluation(noeud)
  Sinon {on est sur un noeud interne}
    Si evalMax **Alors**
      Retourner $max_{f \in fils}(MinMax(f, profondeur - 1, faux))$
    Sinon {on évalue le joueur adversaire}
      Retourner $min_{f \in fils}(MinMax(f, profondeur - 1, vrai))$
    FinSi
  FinSi
Fin

Algorithme de MinMax

Pseudo-code : version 2

```
1  Fonction MinMax(noeud : Plateau, profondeur: Entier, evalMax
       : Booleen): Entier
2  Variables l: Liste d'Entiers
3  Debut
4    Si profondeur = 0 ou victoire(noeud) ou defaite(noeud) ou
         nul(noeud) Alors
5      Retourner evaluation(noeud)
6    Sinon {on est sur un noeud interne}
7      Pour chaque coup de coupsJouables(noeud)
8        l = ajouter(l, MinMax(applique(coup,noeud), profondeur
             -1, non evalMax))
9      FinPour
10     Si evalMax Alors
11       Retourner max(l)
12     Sinon {on évalue le joueur adversaire}
13       Retourner min(l)
14     FinSi
15   FinSi
16 Fin
```

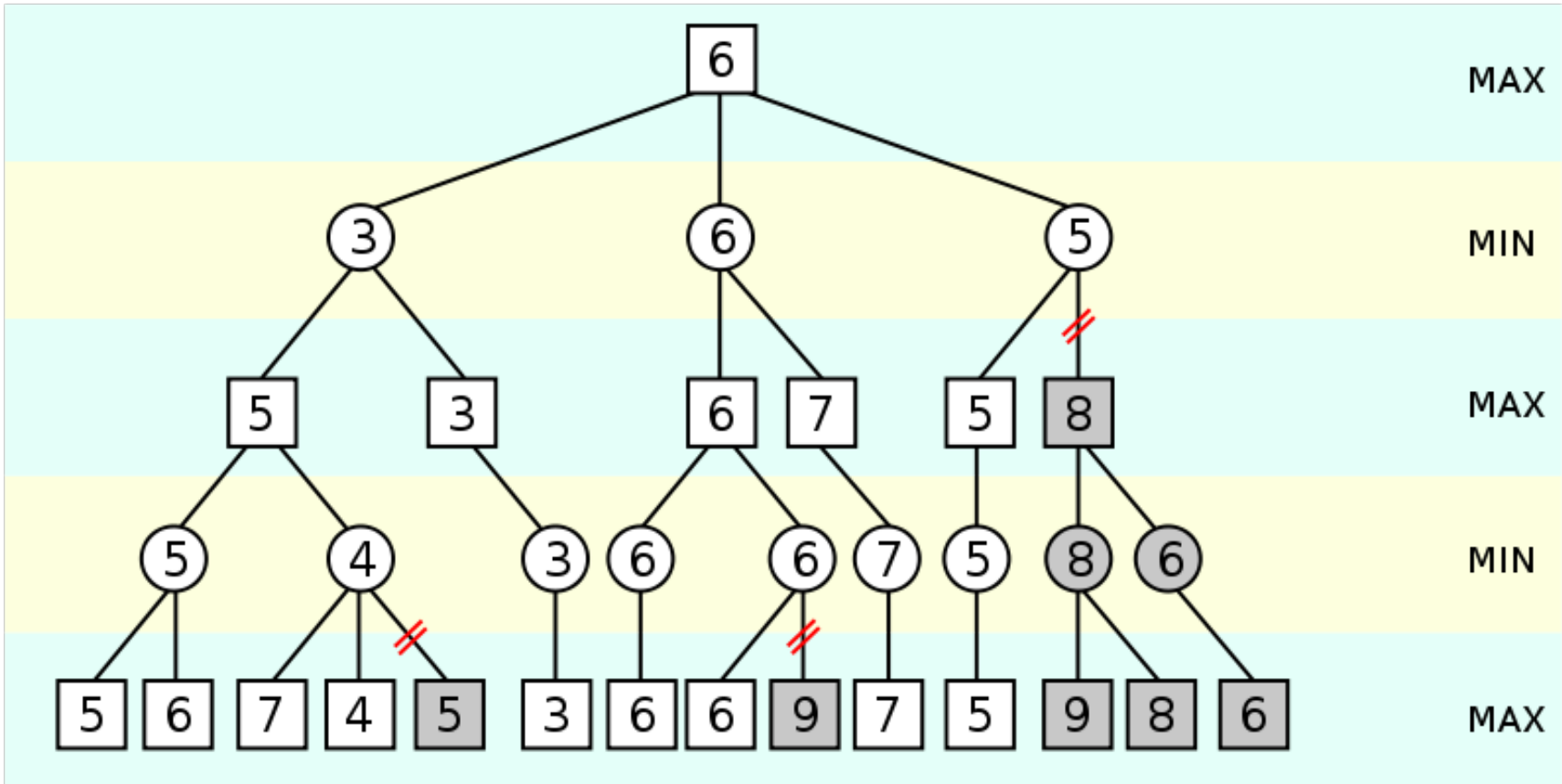
Algorithme de MinMax

Pseudo-code : version 2 - choix du coup à la racine

```
1  Structure EvalCoup{
2    coup : Coup
3    valeur : Entier
4  }
5
6  Fonction jouer(noeud : Plateau): Coup
7  Variables lCoup: Liste d'EvalCoup
8  Debut
9    Pour chaque coup de coupJouables(noeud)
10     lCoup = ajouter(lCoup, creerCoupVal(coup, MinMax(
           applique(coup,noeud),profondeurMax,faux)))
11   FinPour
12   Retourner coupMax(lCoup) // le coup de la liste dont la
         valeur est maximale
13 Fin
```

Optimisation du MinMax

MinMax peut être optimisé en enlevant certaines branches qui, selon le fonctionnement de l'algorithme, n'ont pas à être explorées



Optimisation : elagage Alpha-Beta

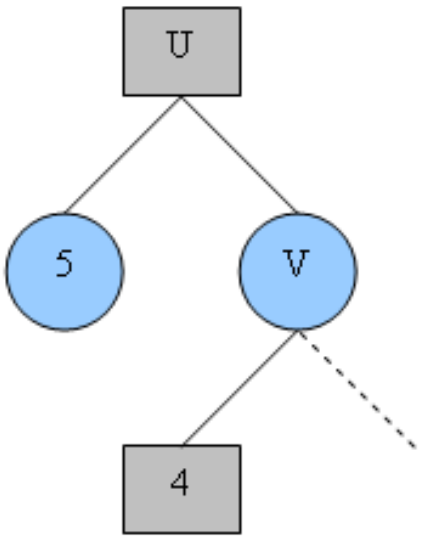
Principes

Associer à chaque nœud, en plus de sa valeur, 2 autres quantités :

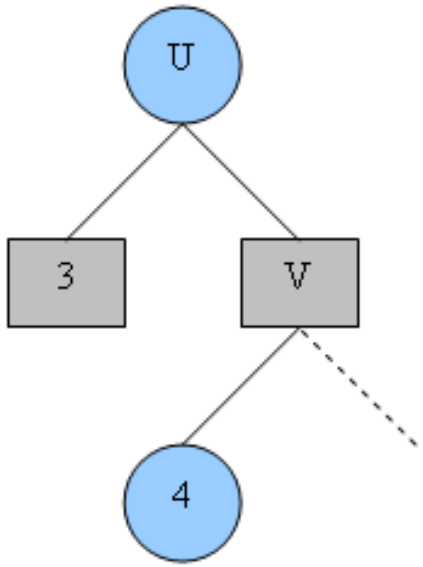
- alpha (initialisée à $-\infty$) : approximation par défaut de la vraie valeur du nœud
- beta (initialisée à $+\infty$) : approximation par excès de la vraie valeur du nœud

Optimisation : elagage Alpha-Beta

Coupures



Coupure Alpha



Coupure Beta

Algorithme

AlphaBeta

```
1  Fonction AlphaBeta(noeud : Plateau, profondeur : Entier,
       alpha : Entier, beta : Entier, evalMax : Booleen) :
       Entier
2  Debut
3    Si profondeur = 0 ou victoire(noeud) ou defaite(noeud) ou
         nul(noeud) Alors
4      Retourner evaluation(noeud)
5    Sinon
6      Si evalMax Alors //joueur MAX
7        alpha = -infini
8        Pour chaque coup de coupJouables(noeud)
9          alpha = max(alpha, AlphaBeta(applique(coup, noeud),
               profondeur-1, alpha, beta, faux))
10         Si alpha ≥ beta Alors
11           Retourner alpha // coupe beta
12         FinSi
13       FinPour
14       Retourner alpha
15       ...
```

Algorithme

AlphaBeta

```
1   ...
2       Sinon // joueur MIN
3         beta = +infini
4         Pour chaque coup de coupJouables(noeud)
5           beta = min(beta, AlphaBeta(applique(coup, noeud),
                profondeur-1, alpha, beta, vrai))
6           Si beta ≤ alpha Alors
7             Retourner beta // coupe alpha
8           FinSi
9         FinPour
10        Retourner beta
11      FinSi
12    FinSi
13  Fin
```