

Théorie des jeux : Algorithme Minimax

- 1 Introduction
- 2 Algorithme MinMax
- 3 Elagage Alpha-Beta

Introduction

Classification des jeux

On peut classer les jeux existants selon plusieurs catégories :

- jeux **coopératifs** vs **compétitions**
- à **somme nulle** (ce qui est gagné par l'un est perdu par l'autre, et réciproquement) vs à somme **non nulle**
- coups **synchrones** vs coups **asynchrones** (alternatifs)
- information **complète** vs information **incomplète**
- **déterministe** vs avec du **hasard**
- **unique** vs **répété**

Stratégie du minimax

Théorème du minimax (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 du minimax

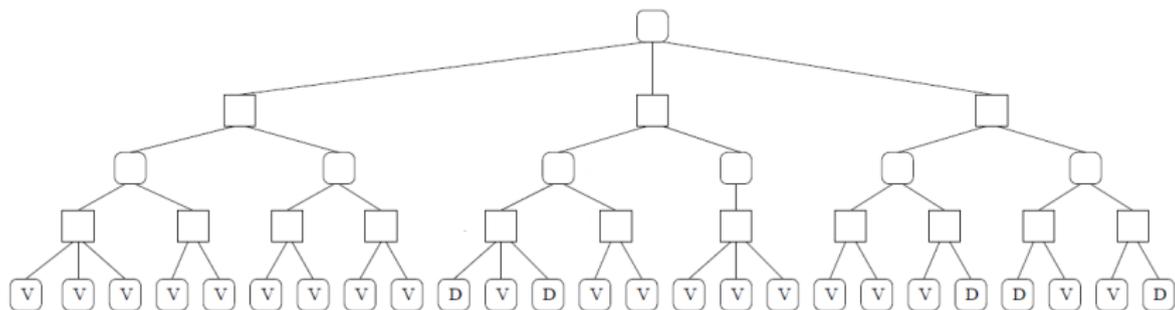
- Le joueur qui commence (joueur MAX) :
 - cherche à trouver, parmi toutes les situations à sa disposition, une situation qui lui permet de **maximiser** ses gains
- L'autre joueur (joueur MIN) :
 - doit trouver, à partir de toutes les situations qui conduisent à la victoire du premier joueur, la situation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

Minimax = minimiser la perte maximum

Scénario idéal

L'arbre complet

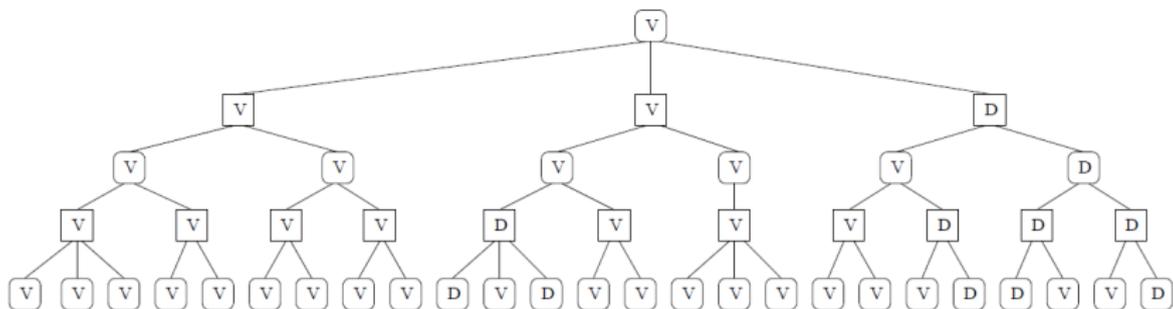
A partir d'une position donnée, il existe une arborescence de coups jusqu'à la victoire (V), au nul (N) ou à la défaite (D). A titre d'exemple, que penser de cette situation ? (cercle représente le joueur MAX, carré représente le joueur MIN)



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)



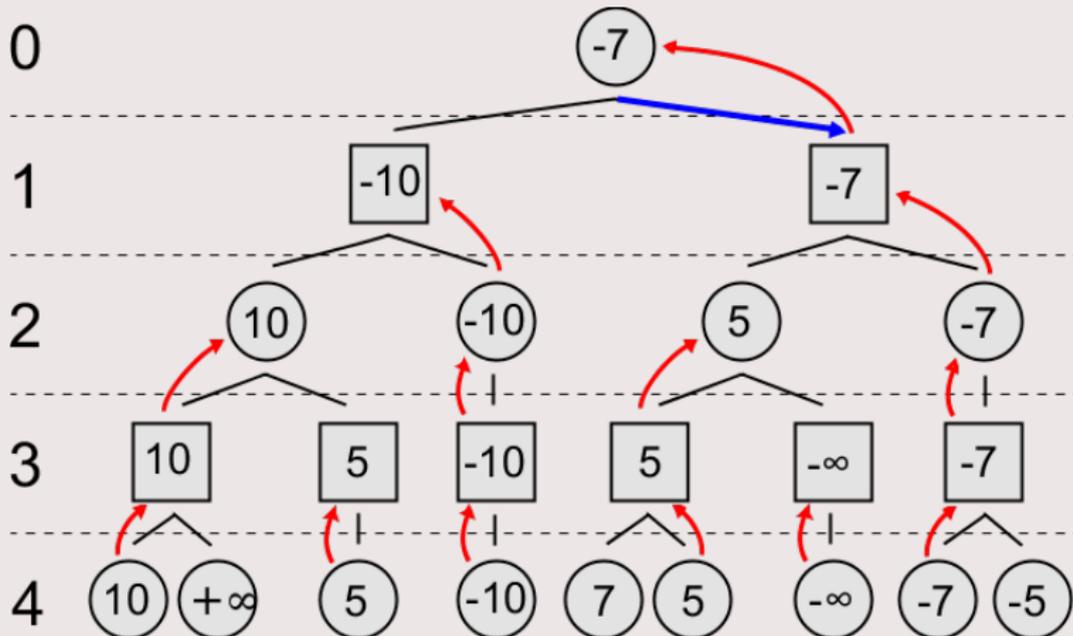
Scénario réel

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 $\max_{f \in \text{fils}}(\text{MinMax}(f, \text{profondeur} - 1, \text{faux}))$

Sinon {on évalue le joueur adverse}

retourner $\min_{f \in \text{fils}}(\text{MinMax}(f, \text{profondeur} - 1, \text{vrai}))$

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
```

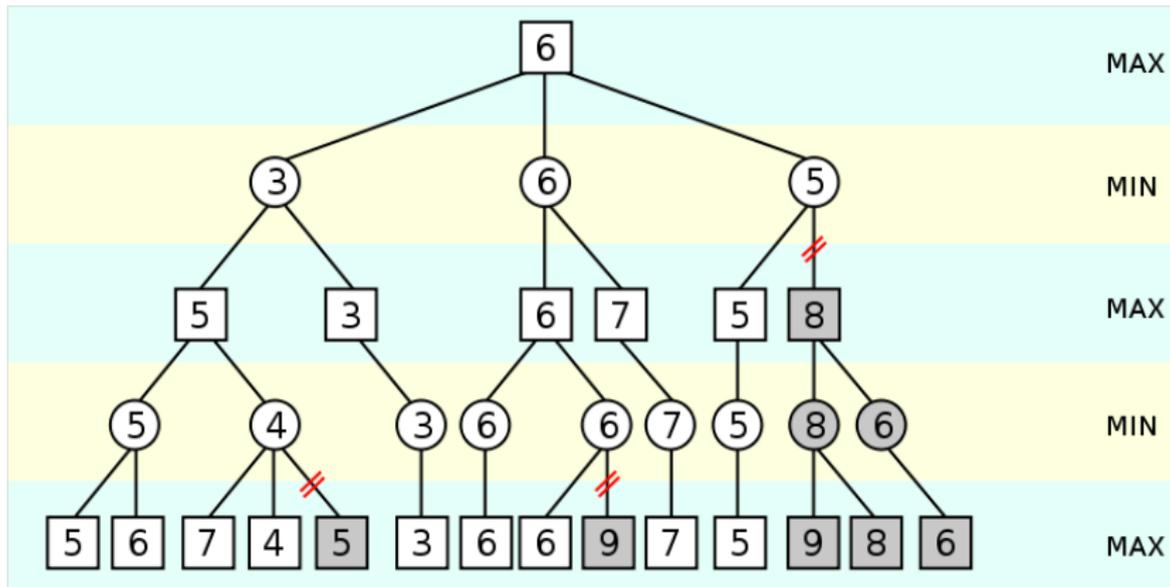
Algorithme de MinMax

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

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

Optimisation de MinMax

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



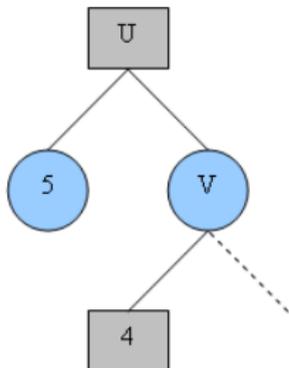
Optimisation : elagage Alpha-Beta

Principes

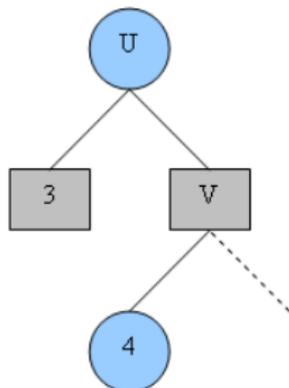
- Associer à chaque nœud, en plus de sa valeur, 2 autres quantités :
 - alpha : approximation par défaut = score minimum du joueur MAX.
 - beta : approximation par excès = score maximum du joueur MIN
- Le couple (alpha,beta) avec $\alpha < \beta$ est appelé *fenêtre de valeur*
 - au début, alpha est initialisée à $-\infty$, beta à $+\infty$
 - quand $\alpha \geq \beta \Rightarrow$ élaguer le nœud correspond

Optimisation : elagage Alpha-Beta

Coups



Coupe Alpha



Coupe Beta

Algorithme AlphaBeta

AlphaBeta

```
1  Fonction AlphaBeta(noeud : Plateau , profondeur : Entier ,  
    alpha : Entier , beta : Entier , evalMax : Booleen) :  
    Entier  
2  Variables score : Entier  
3  Debut  
4    Si profondeur = 0 ou victoire(noeud) ou defaite(noeud) ou  
    nul(noeud) Alors  
5      retourner evaluation(noeud)  
6  Sinon  
7    Si evalMax Alors //joueur MAX  
8      Pour chaque coup de coupJouables(noeud)  
9        score = AlphaBeta(applique(coup , noeud) , profondeur -1,  
        alpha , beta , faux)  
10       Si score > alpha Alors alpha = score FinSi  
11       // on a trouvé un meilleur coup  
12       Si alpha  $\geq$  beta Alors retourner alpha FinSi  
13       // coupe beta  
14     FinPour  
15     retourner alpha // c'est le meilleur coup
```

Algorithme AlphaBeta

AlphaBeta

```
1  ...
2  Sinon // joueur MIN
3    Pour chaque coup de coupJouables(noeud)
4      score = AlphaBeta(applique(coup, noeud), profondeur - 1,
5                          alpha, beta, vrai)
6      Si score < beta Alors beta = score FinSi
7      // l'adversaire a trouvé un pire coup
8      Si alpha ≥ beta Alors retourner beta FinSi
9      // coupe alpha
10     FinPour
11     retourner beta // meilleur coup pour l'adversaire
12   FinSi
13 Fin
```

Optimisation : elagage Alpha-Beta

- Choix de coup à la racine :
AlphaBeta(applique(coup,racine), profondeurMax, $-\infty$, $+\infty$, faux)
- Améliorations :
 - trier les coups en ordre optimal (meilleurs coups sont cherchés au premier),
 - ...