

# Digital Tech Year

## Intelligence Artificielle et jeux: recherche adversariale

Centrale-Supélec



## Un épisode marquant !!!

En 1997, Deep Blue bat Kasparov, le champion du monde d'échecs : un évènement dans le monde de l'IA et des jeux.



## Un autre épisode marquant !!!

En 1990, Goliath est une IA qui a le niveau d'un joueur débutant au jeu de GO.  
En 2016, Alpha Go bat Lee Sedol, un des meilleurs joueurs mondiaux.



# Pourquoi étudier les jeux ?

## Une preuve d'intelligence ?

- ▶ Presque tous les hommes jouent.
- ▶ Résultats faciles à vulgariser par les chercheurs.

## Une vitrine de l'IA

Il est facile de mesurer expérimentalement les progrès dans un jeu à deux joueurs.

- ▶ Les jeux sont populaires pour démontrer les nouvelles idées en IA
- ▶ Les techniques découvertes doivent servir à d'autres domaines de l'IA

# Historique des jeux : un article fondateur de l'IA

Philosophical Magazine, Ser. 7, Vol. 41, No. 314 - March 1950.

**XXII. Programming a Computer for Playing Chess<sup>1</sup>**

By CLAUDE E. SHANNON

Bell Telephone Laboratories, Inc., Murray Hill, N.J.<sup>2</sup>  
 [Received November 8, 1949]

Le premier article d'I.A. est un article sur les jeux (Shannon, 1950).

Philosophical Magazine, Ser. 7, Vol. 41, No. 314 - March 1950.

## XXII. Programming a Computer for Playing Chess<sup>1</sup>

By CLAUDE E. SHANNON

Bell Telephone Laboratories, Inc., Murray Hill, N.J.<sup>2</sup>  
 [Received November 8, 1949]

### I. INTRODUCTION

This paper is concerned with the problem of constructing a computing routine or "program" for a modern general purpose computer which will enable it to play chess. Although perhaps of no practical importance, the question is of theoretical interest, and it is hoped that a satisfactory solution of this problem will act as a wedge in attacking other problems of a similar nature and of greater significance. Some possibilities of this direction are:

- (1) Machines for designing filters, equations, etc.
- (2) Machines for designing relay and switching circuits.
- (3) Machines which will handle routing of telephone calls based on the individual circumstances rather than by fixed patterns.
- (4) Machines for performing symbolic (non-numerical) mathematical operations.
- (5) Machines capable of translating from one language to another.
- (6) Machines for making strategic decisions in simplified military operations.
- (7) Machines capable of educating a melody.
- (8) Machines capable of logical deduction.

It is believed that all of these and many other devices of a similar nature are possible developments in the immediate future. The techniques developed for modern electronic and relay type computers make them not only theoretical possibilities, but in several cases worthy of serious consideration from the economic point of view.

Machines of this general type are an extension over the ordinary use of numerical inverters in several ways. First, the entities dealt with are not necessarily numbers, but other chess positions, circuits, mathematical expressions, words, etc. Second, the proper procedure involves general principles, something of the nature of judgment, and considerable trial and error, rather than a strict, ruledriven computing process. Finally, the solutions of these problems are not merely right or wrong but have a continuous range of "quality" from the best down to the worst. We might be satisfied with a machine that designed good filters even though they were not always the best possible.

<sup>1</sup> First presented at the National IRE Convention, March 9, 1949, New York, U.S.A.  
<sup>2</sup> Communicated by the Author

# Quelques dates

- ▶ 1945 : Turing présente les échecs comme ce que peuvent faire les ordinateurs.
- ▶ 1946 : Turing parle d'intelligence des machines, en relation avec les échecs.
- ▶ 1950 : Turing écrit le premier programme de jeux.
- ▶ 1950 : Shannon écrit le premier article sur les jeux.
- ▶ 1957 : Prédiction de Simon : dans 10 ans, le champion du monde d'échecs sera un ordinateur.
- ▶ 1963 : Le programme de jeu de dames de Samuel gagne contre le champion du monde
- ▶ 1989 : Création du World Man Machine Championship.

# Et l'humain ?

## Les jeux, le domaine des machines ?

- ▶ Jeux pour lesquels les ordinateurs sont bien supérieurs à l'humain :
  - ▶ Dames
  - ▶ Othello
  - ▶ Scrabble
- ▶ Jeux pour lesquels les ordinateurs sont de niveau mondial :
  - ▶ Echecs
  - ▶ Backgammon
  - ▶ Go (c.f. Mogo, INRIA  
<http://www.lri.fr/~teytaud/taiwanopen2009.html>, <http://www.inria.fr/saclay/ressources/culture-scientifique/mogo>)
- ▶ Les challenges :
  - ▶ Bridge

# Jeux que l'on va étudier

## Caractéristiques

- ▶ Les Règles du jeu sont formalisables et précises
- ▶ Deux adversaires
- ▶ Jeu à somme zéro (ce que l'un gagne l'autre le perd)
- ▶ Jeu ouvert

# Problèmes de jeux

## Caractéristiques

- ▶ Problème de recherche en présence d'un adversaire (**recherche adversariale**)
  - ▶ Introduction d'une incertitude
  - ▶ Tous les mouvements ne sont pas contrôlés par l'ordinateur.
- ▶ Les programmes de jeux doivent faire face à l'imprévu.
- ▶ Complexité : les jeux sont souvent trop complexes pour envisager une solution de recherche exhaustive (exemple : arbre de recherche des échecs)
- ▶ Le but de la recherche est de choisir à chaque pas le meilleur prochain coup à jouer

# Type de jeux

## Classification des jeux

	Déterministe	Avec hasard
Information complète	échecs, jeu de dames, othello, go	monopoly, backgammon
Information incomplète	bataille navale	poker, bridge

# Jeux traités dans ce cours

## Caractéristiques

- ▶ Les deux joueurs jouent **tour à tour**
- ▶ Les deux joueurs ont **toutes les informations possibles sur la partie** : environnement déterministe et totalement observable
- ▶ Les valeurs renvoyées par les fonctions d'utilité des agents à la fin de la partie sont soit nulles soit égales.

## Jeux vidéos

- ▶ Privilégient les qualités graphiques aux qualités de jeux
- ▶ Contraintes temps réel fortes
- ▶ *Triche* pour rendre le jeu réaliste : chocs, ...

## Formulation sous forme d'un problème de recherche

- ▶ **Etat** : position des éléments du jeu et identité du joueur qui doit jouer
- ▶ **Etat initial** : position initiale des éléments du jeu et identité du joueur qui va commencer.
- ▶ **Fonction successeur** : fonction indiquant les coups possibles pour un état donné
- ▶ **Etat terminal** : un des joueurs a gagné ou bien il y a égalité.
- ▶ **Fonction d'utilité** : renvoie une valeur évaluant les états terminaux : +1 (gagnant), -1 (perdant) et 0 (égalité)

# Définitions

## Arbre de jeu

Arbre qui consiste en des noeuds qui représentent les options des joueurs dans un jeu à deux joueurs. Il y a alternance entre les options des deux joueurs.

## Tour ou Couche

Les différents niveaux dans un arbre de jeu s'appellent les tours ou les couches

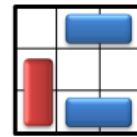
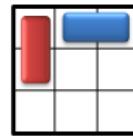
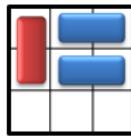
## Noeuds finaux

Ils indiquent la fin du jeu : soit une configuration gagnante, perdante ou un match nul

## Les dominos de couleurs

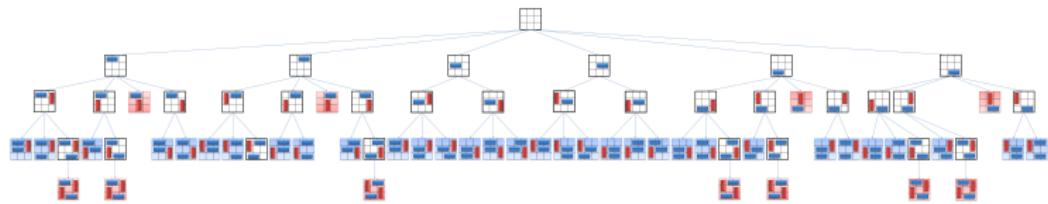
### Principe

Sur une grille  $n \times n$ , il faut être le dernier à poser son domino  $2 \times 1$ . Le joueur 1 joue horizontalement, l'autre verticalement

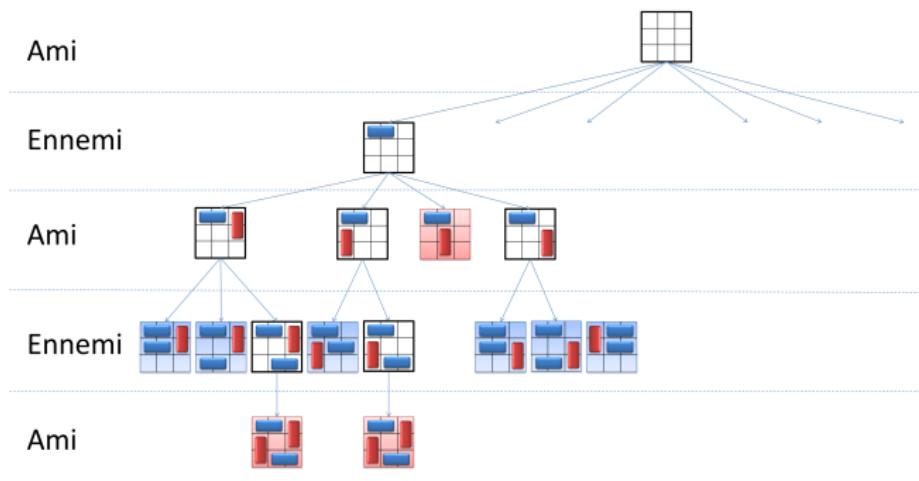


## Exploration de l'arbre de jeu

- ▶ La simulation de tous les coups possibles à partir de l'état initial permet de construire l'arbre de jeu.



# Exploration de l'arbre de jeu

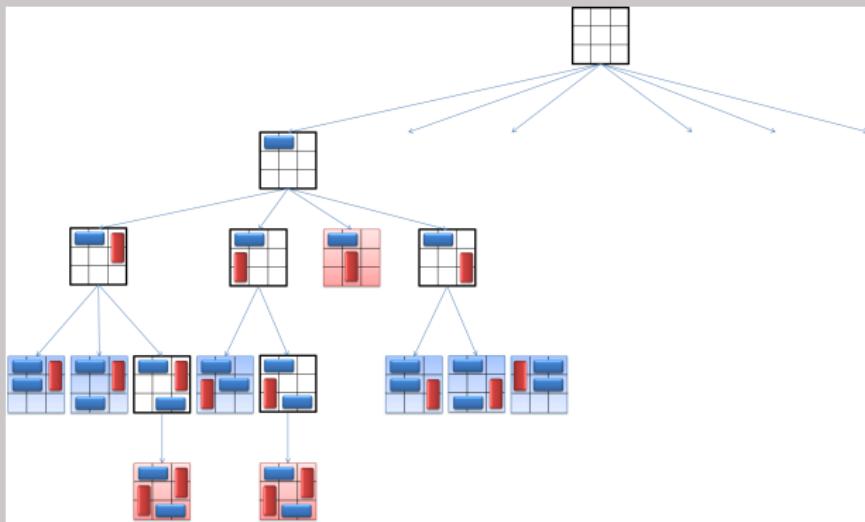


## Exploration de l'arbre de jeu

Parcours en profondeur d'abord dans un arbre décrit en intention.

# Exploration de l'arbre de jeu

## Exploration complète de l'arbre de jeu



Pour une grille de n

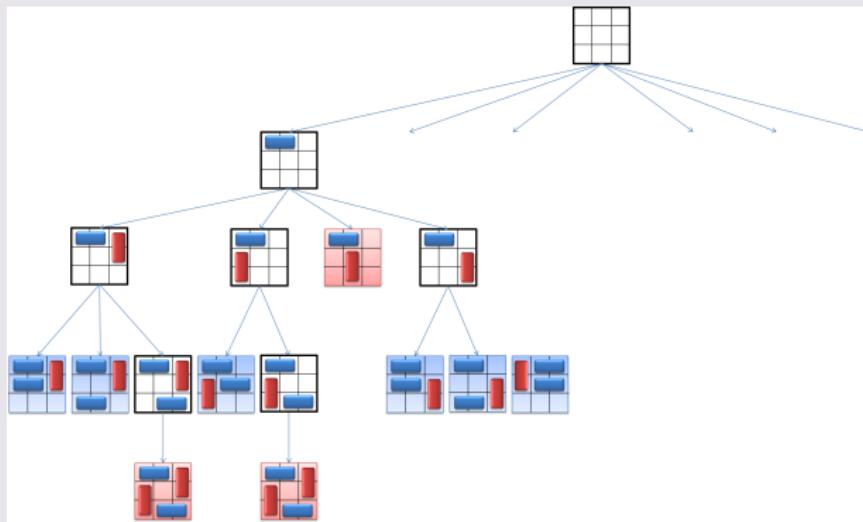
Explosion combinatoire dès que l'on dépasse une grille de 6. Comment améliorer la recherche de solutions ?

# Exploration de l'arbre de jeux : améliorations

## Ne pas ré-exploré les sous graphes communs

- ▶ Idée : Dans certains jeux, un même état peut être atteint par différents endroits. Idée : on ne réexplore pas les sous-arbres déjà vus.
- ▶ On peut aller plus loin en supprimant tous les états symétriques

## Cas des dominos



## Exploration de l'arbre de jeux : améliorations

### Ne pas réexplorer les sous graphes communs : problèmes

- ▶ Il faut introduire un mécanisme pour retrouver les noeuds déjà explorés
- ▶ Il faut garder tous les noeuds en mémoire
- ▶ On passe beaucoup de temps à vérifier si un noeud a déjà été vu.

# Introduction d'heuristiques

## Idée

- ▶ On ne développe l'arbre de recherche que jusqu'à une certaine valeur maximale  $p$ . Les feuilles de l'arbre ne sont plus forcément les positions finales du jeu.
- ▶ On doit donc évaluer les positions. L'information *perdu ou gagné* n'est plus suffisante

## Définition

Les heuristiques de jeux sont des fonctions qui associent un réel aux plateaux de jeux. Dans les jeux avec adversaire, plus l'heuristique est grande plus on est proche de la victoire et inversement plus elle est négative et faible et plus on est proche de la défaite.

# Cas des dominos

## Exemple

- ▶ On développe l'arbre de recherche jusqu'à une profondeur donnée
- ▶ On évalue la position aux feuilles
- ▶ On fait remonter l'évaluation pour trouver le meilleur coup

# Introduction d'heuristiques

## Nouveau problème

- ▶ Trouver la branche permettant de maximiser la valeur heuristique obtenue sur le plateau après les  $p$  prochains coups
- ▶ Les deux joueurs jouent bien : ils suivent les indications d'heuristiques

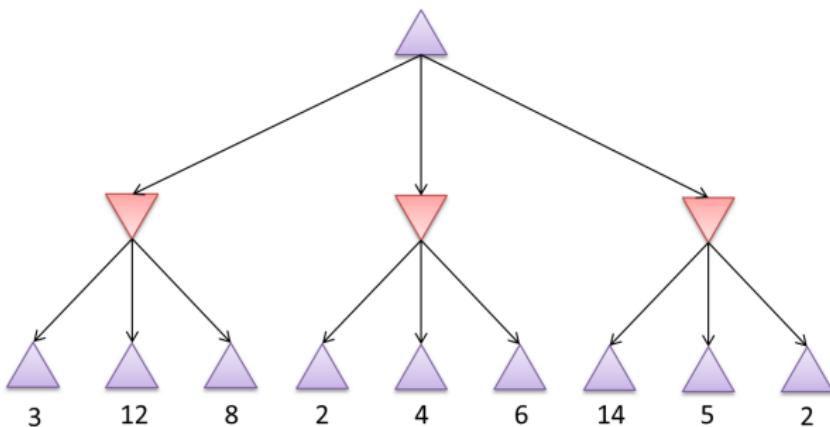
Comment faire remonter la meilleure feuille à la racine ?

# Algorithme MiniMax

## Principe

- ▶ On s'intéresse à un des deux joueurs qu'on appelle Max. Les valeurs d'utilité sont exprimées du point de vue de Max.
- ▶ Le but est de faire gagner Max : trouver la séquence d'actions qui permet d'arriver à un état terminal dont l'utilité est élevée ?
- ▶ **NON** : Max doit tenir compte des actions de Min.
- ▶ Le principe de l'algorithme Minimax est de développer complètement l'arbre de jeux, de noter chaque feuille avec sa valeur et de faire remonter ces valeurs avec l'hypothèse que chaque joueur choisit le meilleur coup pour lui :
  - ▶ valeur minimum aux noeuds MIN
  - ▶ valeur maximum aux noeuds MAX

## Algorithme MiniMax : principe



### Principe

- ▶ A l'état initial Max a le choix entre trois actions.
- ▶ Min a alors le choix entre trois actions dépendantes de l'action de Max au tour précédent.

# Algorithme MiniMax

## Minimax

Valeur d'utilité qu'on aura à la fin de la partie si les deux joueurs jouent de façon optimale (en jouant leur meilleur coup) à partir de l'état courant et jusqu'à la fin de la partie.

## Coup optimal

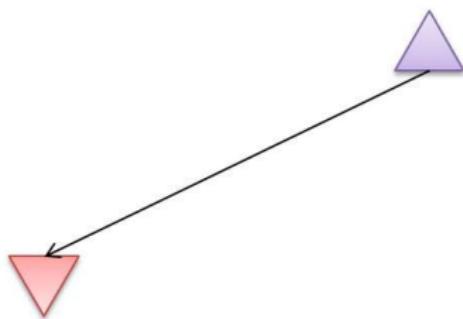
- ▶ Max va aller vers l'état qui a un minimax maximal
- ▶ Min va aller vers l'état qui a un minimax minimal.

En partant des feuilles, on peut calculer le minimax de chaque état.

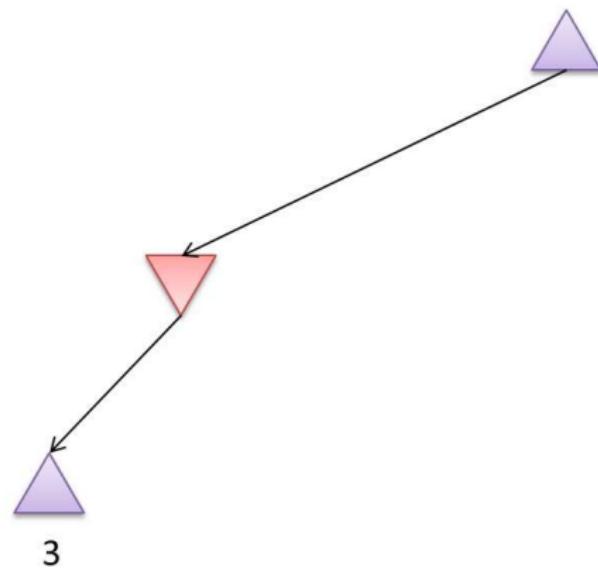
# Algorithme MiniMax : pas à pas



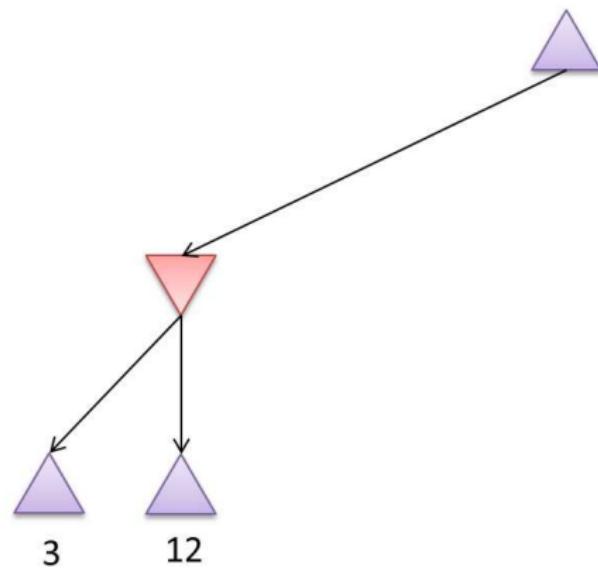
## Algorithme MiniMax : pas à pas



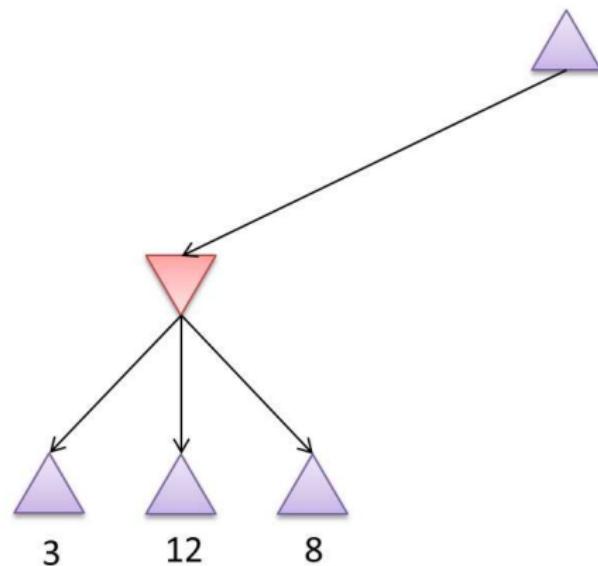
## Algorithme MiniMax : pas à pas



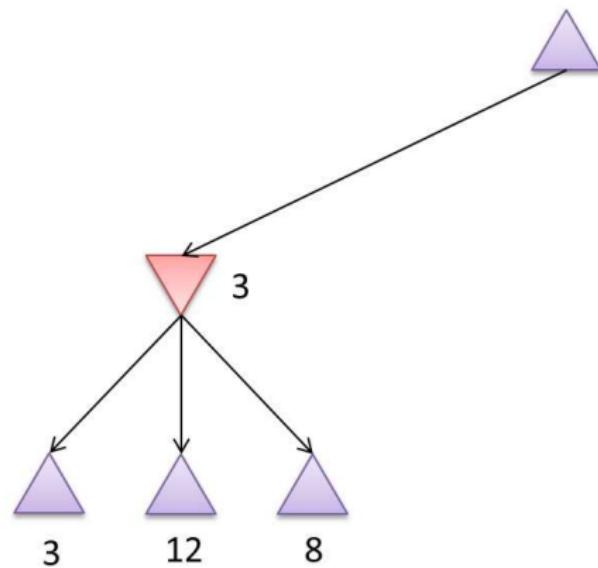
## Algorithme MiniMax : pas à pas



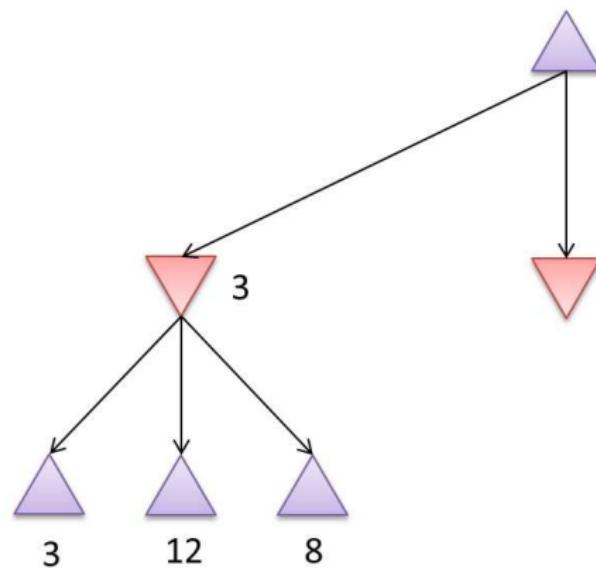
## Algorithme MiniMax : pas à pas



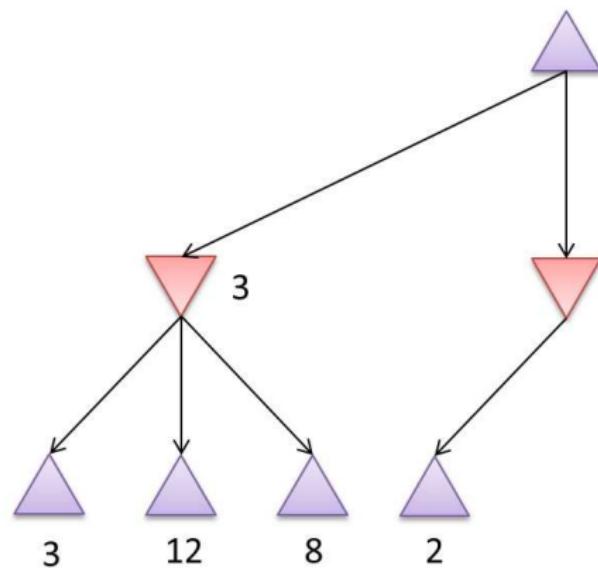
## Algorithme MiniMax : pas à pas



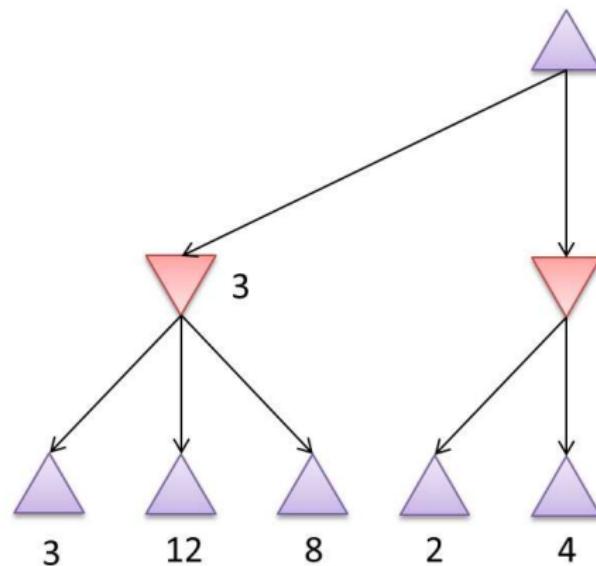
## Algorithme MiniMax : pas à pas



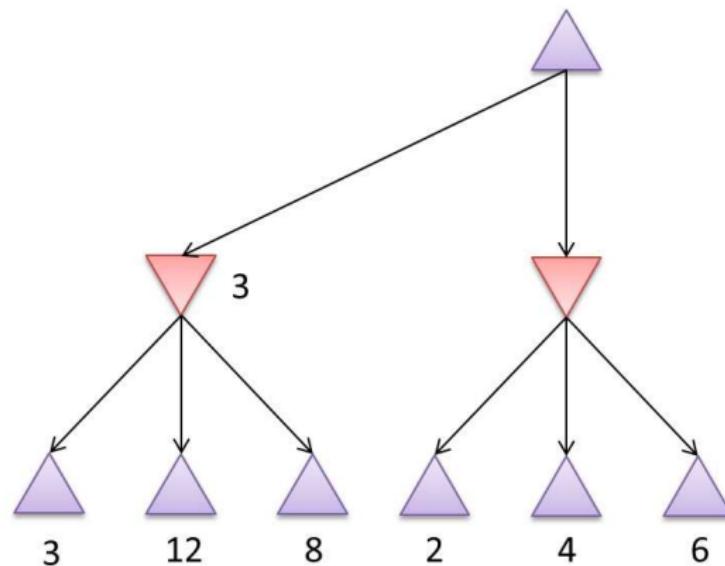
## Algorithme MiniMax : pas à pas



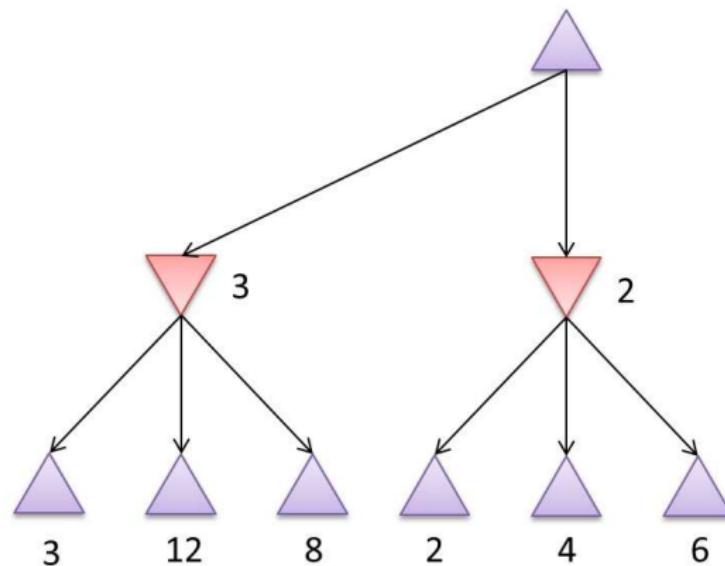
## Algorithme MiniMax : pas à pas



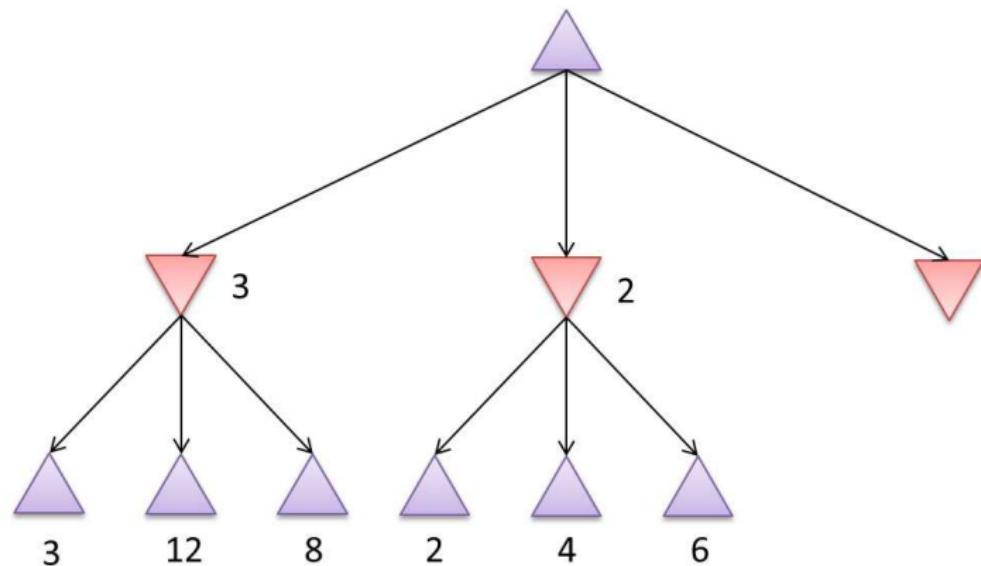
## Algorithme MiniMax : pas à pas



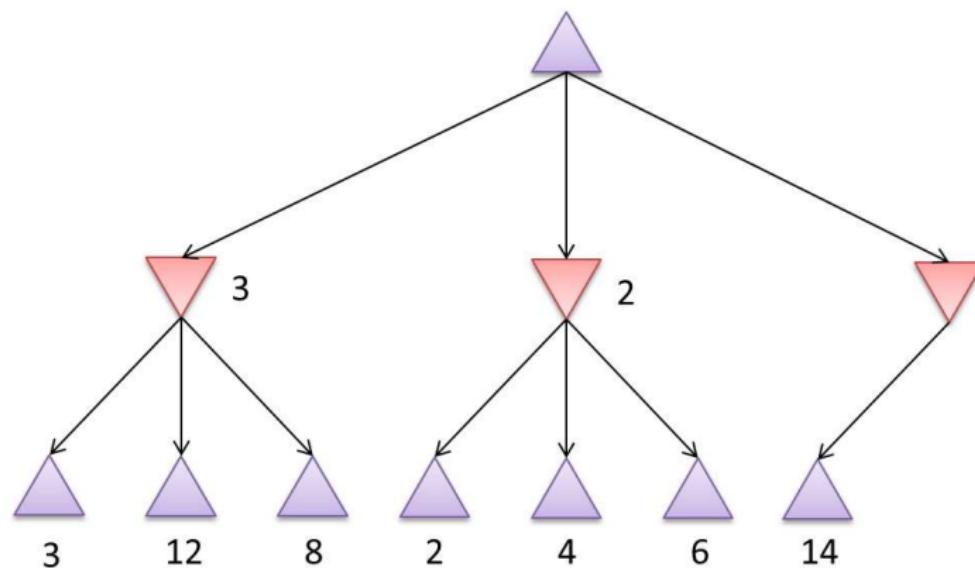
## Algorithme MiniMax : pas à pas



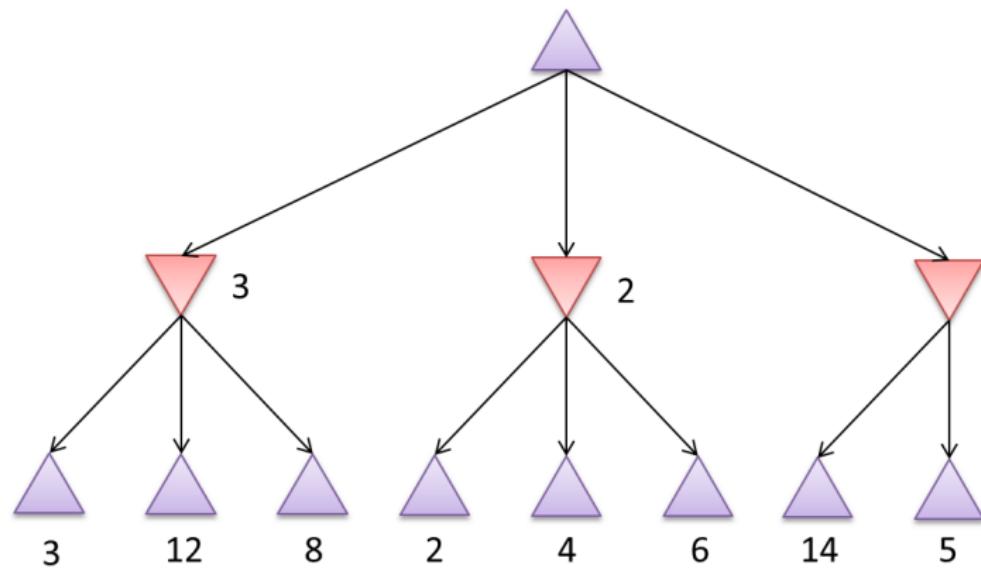
## Algorithme MiniMax : pas à pas



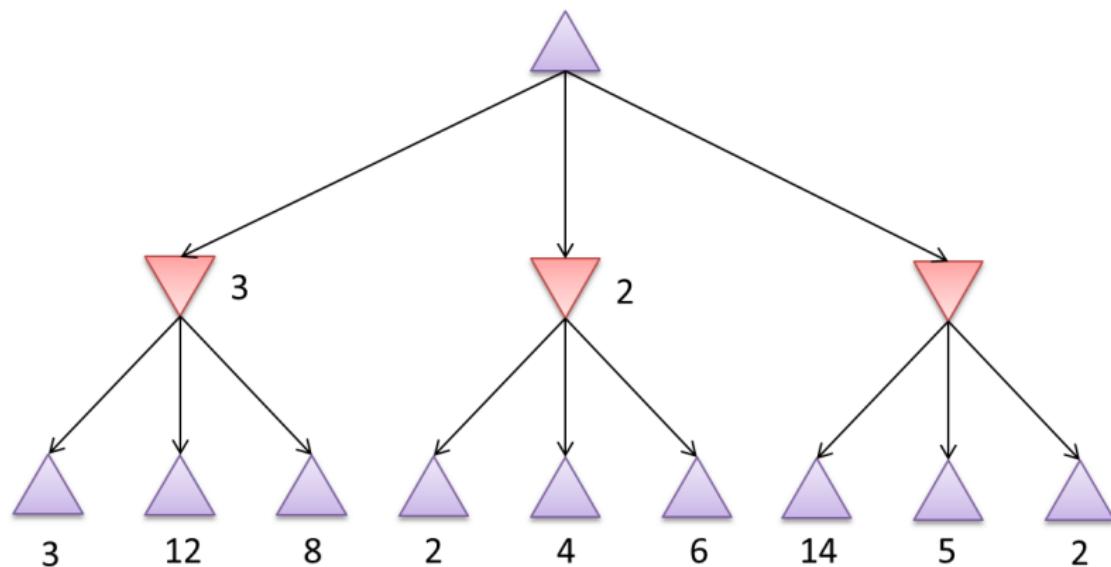
## Algorithme MiniMax : pas à pas



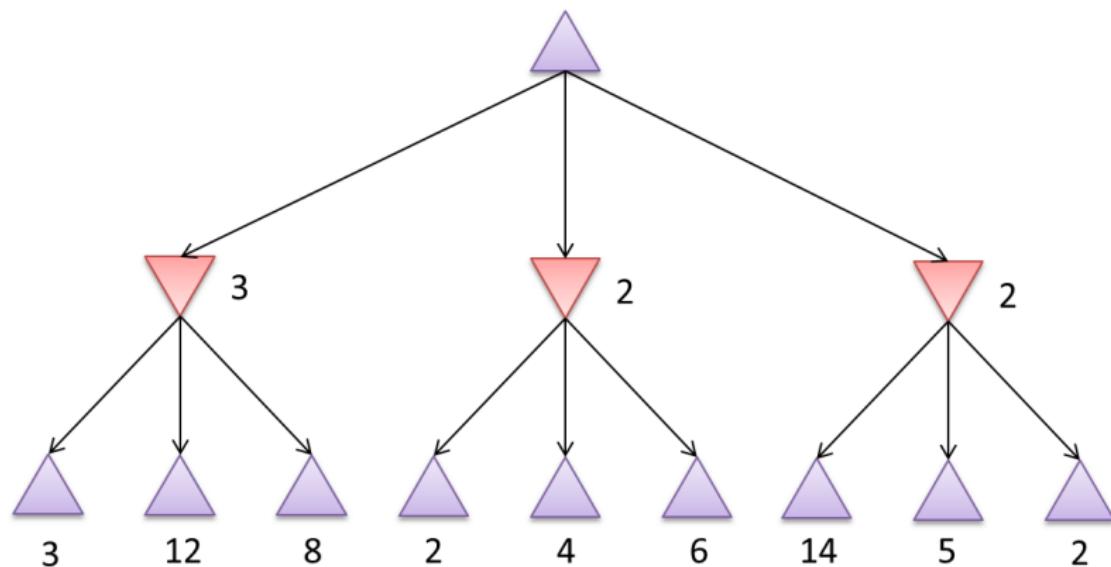
## Algorithme MiniMax : pas à pas



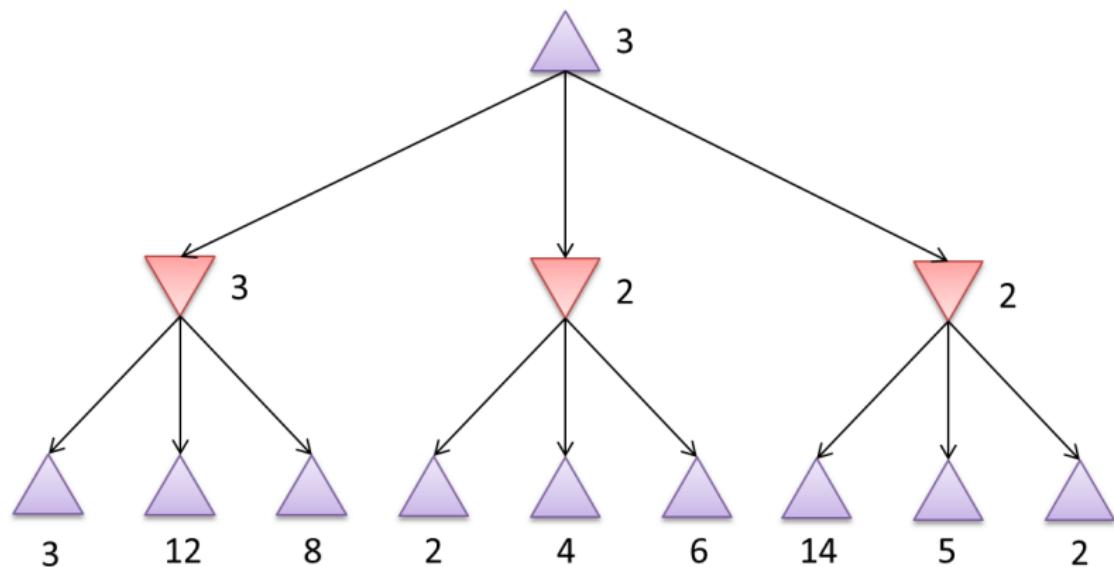
## Algorithme MiniMax : pas à pas



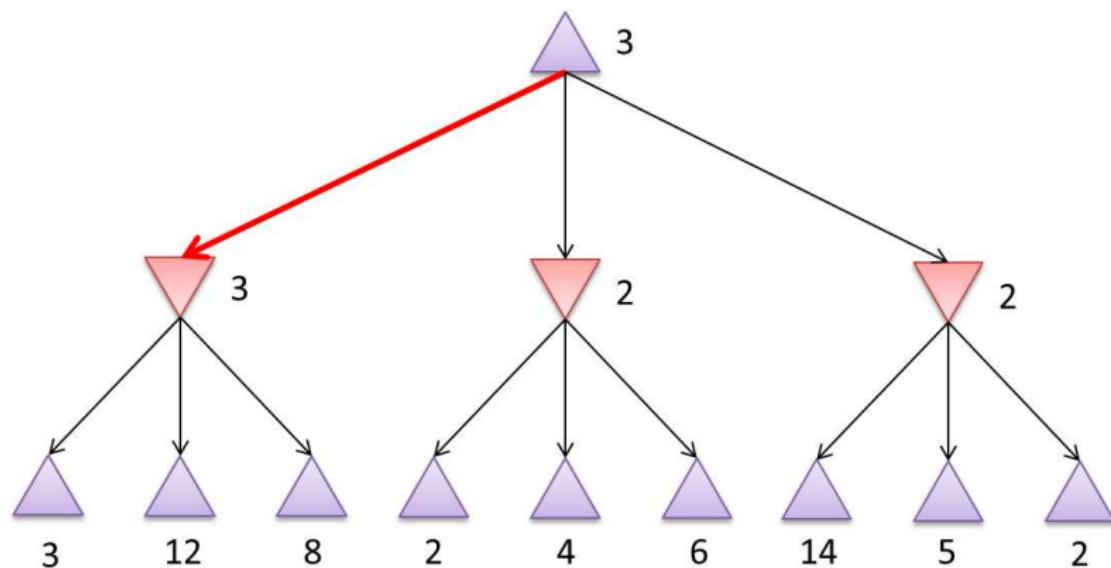
## Algorithme MiniMax : pas à pas



## Algorithme MiniMax : pas à pas



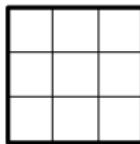
## Algorithme MiniMax : pas à pas



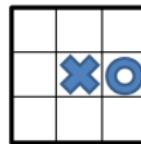
## Fonction d'évaluation pour un état du jeu

- ▶  $e(s) = +\infty$  si  $s$  est une situation de gain pour Max
- ▶  $e(s) = -\infty$  si  $s$  est une situation de gain pour Min
- ▶  $e(s)$  est une mesure de caractère favorable de  $s$  pour Max .

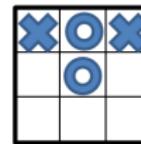
$$e(s) = \# \text{ lignes/cols/diags ouvertes pour Max} - \# \text{ lignes/cols/diags ouvertes pour Min}$$



$$8 - 8 = 0$$

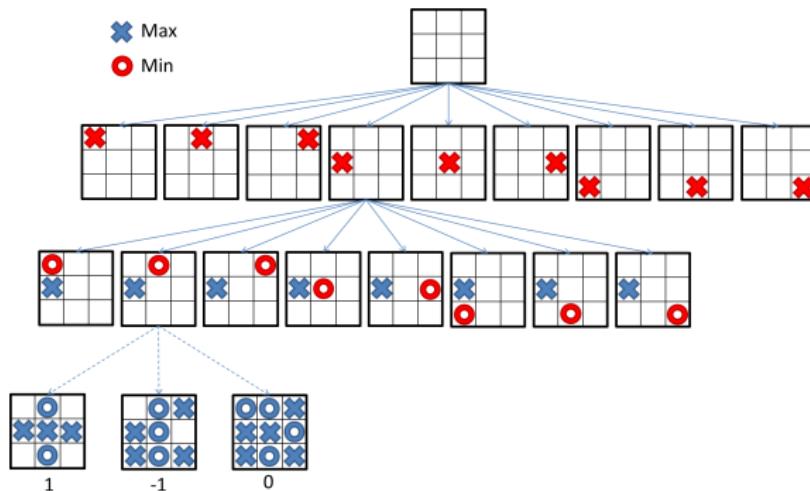


$$6 - 4 = 2$$



$$3 - 3 = 0$$

## Exemple : le morpion (Tic Tac Toe)



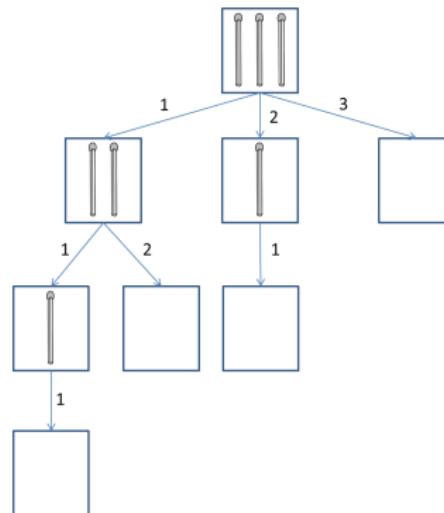
- ▶ Etendre l'arbre complet de jeu.
- ▶ Calculer les valeurs de la fonction de gain aux noeuds terminaux
- ▶ Propager ces valeurs vers la racine selon MiniMax.

# Le jeu d'allumettes

## Principe

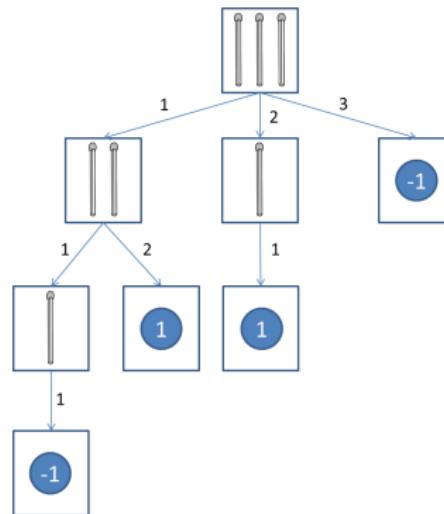
Chaque joueur peut prendre à chaque tour 1 ou 2 ou 3 allumettes. Le joueur qui prend la dernière allumette a perdu.

On commence par développer l'arbre de jeu.



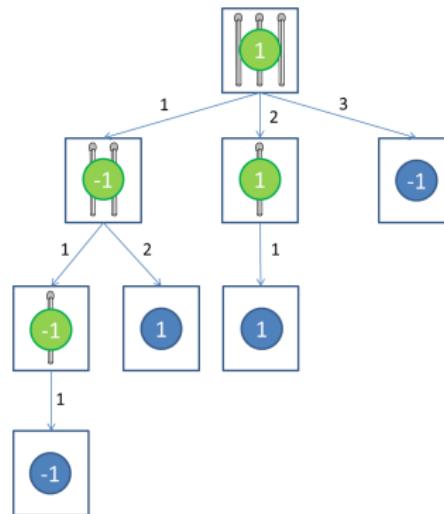
# Le jeu d'allumettes

On étiquette les feuilles selon qu'il s'agit d'une partie gagnante ou perdante.



# Le jeu d'allumettes

On fait remonter les évaluations depuis les feuilles jusqu'à la racine.



# Algo MiniMax

```

function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game
    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v  $\leftarrow -\infty$ 
    for a, s in SUCCESSORS(state) do v  $\leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$ 
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v  $\leftarrow \infty$ 
    for a, s in SUCCESSORS(state) do v  $\leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))$ 
    return v

```

- ▶ MiniMax-Decision renvoie l'action que le joueur Max doit jouer
- ▶ Max-Value (MaxMin) renvoie le minimax d'un état ou c'est à Max de jouer
- ▶ Min-Value (MinMax) renvoie le minimax d'un état ou c'est à Min de jouer.

# Propriétés de l'algorithme

## Propriétés

- ▶ **Complétude**  
Oui si l'arbre de jeu est fini
- ▶ **Complexité en temps**  
 $o(b^m)$  (parcours en profondeur d'abord) b : facteur de branchement. m : horizon de recherche
- ▶ **Espace** :  $o(bm)$
- ▶ **Optimal** : Oui si l'adversaire est aussi optimal

Algorithme inutilisable sur des applications réelles : il n'est en pratique pas possible d'explorer l'arbre en entier.

# Améliorations de l'algorithme MiniMax

## Deux approches

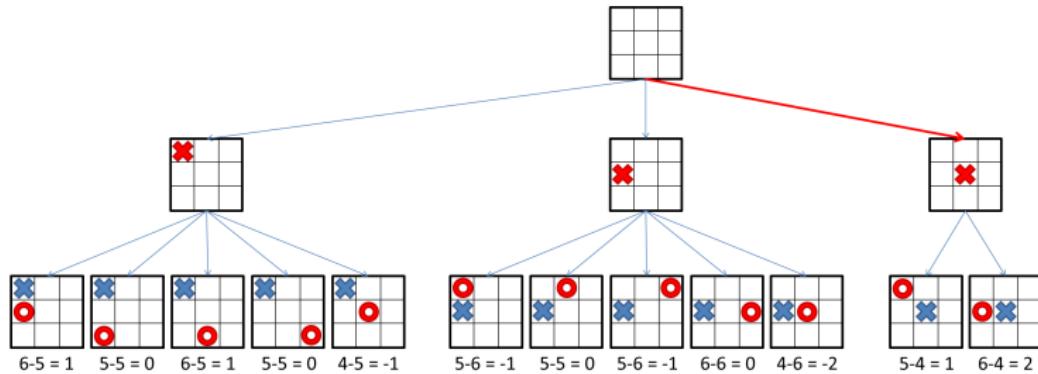
- ▶ Limiter la profondeur de l'exploration : MiniMax avec profondeur limitée
- ▶ Effectuer des coupes dans l'arbre de jeux : Alpha-Beta

# Algorithme MiniMax avec profondeur limitée (ou recherche n-coups)

## Algorithme

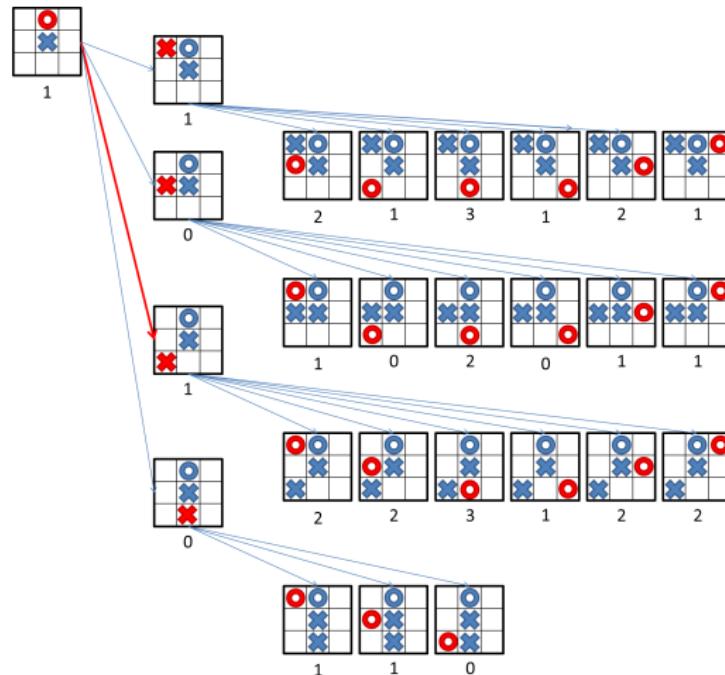
1. Etendre l'arbre de jeu à partir de l'état courant (c'est à Max de jouer) jusqu'à une profondeur **h (horizon)**.
2. Calculer la fonction d'évaluation pour chacune des feuilles de l'arbre.
3. Retro-propager les valeurs des noeuds-feuilles vers la racine de l'arbre
  - 3.1 Un noeud Max reçoit la valeur maximum de l'évaluation de ses successeurs
  - 3.2 Un noeud Min reçoit la valeur minimum de l'évaluation de ses successeurs.
4. Choisir le mouvement vers le noeud Min qui possède la valeur retropropagée la plus élevée.

## Exemple : Tic Tac Toe avec horizon 2



D'après <http://cui.unige.ch/DI/cours/IA>

## Exemple : Tic Tac Toe avec horizon 2



D'après <http://cui.unige.ch/DI/cours/IA>

# Algorithme MiniMax avec profondeur limitée (ou recherche n-coups)

La profondeur maximale est souvent liée à la machine utilisée mais aussi à l'avancement dans la partie. Pour information, le niveau des joueurs artificiels d'échec en fonction de cette profondeur :

- ▶ 4 coups : joueur novice
- ▶ 8 coups : joueur maître
- ▶ 12 coups : Deep Blue

# Algorithme MiniMax

## Importance de la fonction d'évaluation

En général une fonction linéaire pondérant la valeur de chaque pièce du jeu

## Importance de la fonction d'évaluation

$$\begin{aligned}f(P) = & 200(K - K') + 9(Q - Q') + 5(R - R') + 3(B - B' + N - N') + (P - P') \\& - \frac{1}{2}(D - D' + S - S' + I - I') \\& + 0.1(M - M') \dots\end{aligned}$$

avec :

- ▶ K, Q, R, B, N, P représentent le nombre de rois, reines, tours, fous, cavaliers et pions blancs (prime pour les noirs) ;
- ▶ D, S, I représentent les pions blancs qui sont doublés, arriérés, isolés ;
- ▶ M est une mesure de mobilité (par exemple, le nombre de mouvements autorisés que peut faire le joueur blanc).

# Algorithme d'élagage Alpha-Beta

## Elagage admissible

On veut trouver la même valeur d'évaluation finale du noeud racine sans développer tout l'arbre. Il faut élaguer les parties de l'arbre de recherche qui sont sans conséquence sur l'évaluation d'un noeud

## Idée

On considère un noeud  $n$  dans l'arbre de recherche tel que le joueur peut jouer en  $n$ . Si il existe pour le joueur un choix  $m$  meilleur que  $n$  (soit à partir du noeud parent de  $n$  soit plus haut dans l'arbre),  $n$  ne sera jamais joué.

# Algorithme d'élagage Alpha-Beta

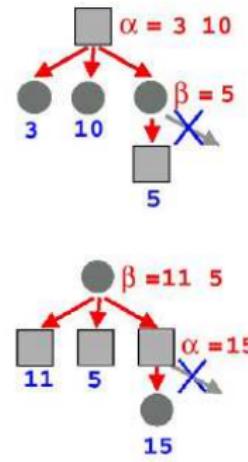
## Principe

- ▶ Etendre l'arbre de jeu jusqu'à une profondeur  $h$ .
- ▶ Ne plus générer les successeurs d'un noeud dès qu'il est évident que ce noeud ne sera pas choisi.
  - ▶ Chaque noeud Max garde la trace d'une  $\alpha$ -valeur, c'est à dire la valeur de son meilleur successeur trouvé jusqu'ici. Le meilleur choix à un instant donné pour Max sur le chemin développé.  $\alpha$ -valeur est croissante
  - ▶ Chaque noeud Min garde la trace d'une  $\beta$ -valeur, c'est à dire la valeur de son meilleur successeur trouvé.  $\beta$ -valeur est décroissante
- ▶ Les coupes ont lieu dès que  $\alpha$ -valeur est supérieure à  $\beta$ -valeur

# Algorithme d'élagage Alpha-Beta

## Deux règles

1. Interrompre la recherche d'un noeud Max si son  $\alpha$ -valeur  $\geq \beta$ -valeur de son noeud parent.
2. Interrompre la recherche d'un noeud Min si sa  $\beta$ -valeur  $\leq \alpha$ -valeur de son noeud parent.



# Algorithme d'élagage Alpha-Beta

## Algorithm 1: Algorithme MaxValue

```
Function .MaxValue (etat,  $\alpha$ ,  $\beta$ ) /* Evaluation niveau AMI */
```

**Data:**

- ▶ *etat* : plateau de jeu courant
- ▶  $\alpha$  : meilleure évaluation courante AMI
- ▶  $\beta$  : meilleure évaluation courante ENNEMIE

**begin**

```

    if EstFeuille(etat) then
        | return evalue(etat) /* Evaluation heuristique */ */
    end
    forall successeur s de etat do
        |  $\alpha \leftarrow \max(\alpha, \text{MinValue}(s, \alpha, \beta))$ 
        | if  $\alpha \geq \beta$  then
            |     | return  $\beta$  /* Coupe  $\beta$  */
        | end
    end
    return  $\alpha$ 
end
```

# Algorithme d'élagage Alpha-Beta

---

## Algorithm 2: Algorithme MinValue

---

```

Function MinValue (etat,  $\alpha$ ,  $\beta$ )/* Evaluation niveau ENNEMI */
```

**Data:**

- ▶ *etat* : plateau de jeu courant
- ▶  $\alpha$  : meilleure évaluation courante AMI
- ▶  $\beta$  : meilleure évaluation courante ENNEMIE

```

begin
    if EstFeuille(etat) then
        | return evaluer(etat)/* Evaluation heuristique */ */
    end
    forall successeur s de etat do
        |  $\beta \leftarrow \min(\beta, \text{MaxValue}(s, \alpha, \beta))$ 
        | if  $\alpha \geq \beta$  then
            |     | return  $\beta$ /* Coupe  $\alpha$  */ */
        | end
    end
    return  $\alpha$ 
end

```

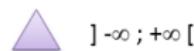
---

# Algorithme d'élagage Alpha-Beta

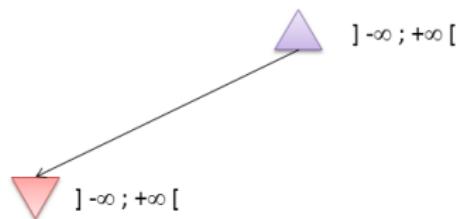
## Principe

- ▶ Pour évaluer un plateau, on appelle *MaxValue* avec le plateau courant.
- ▶ On a initialement  $\alpha = -\infty$  et  $\beta = +\infty$
- ▶  $\alpha$  et  $\beta$  sont locales à chaque fonction

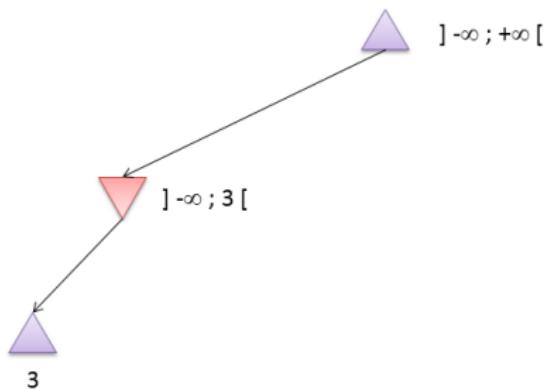
# Exemple 1



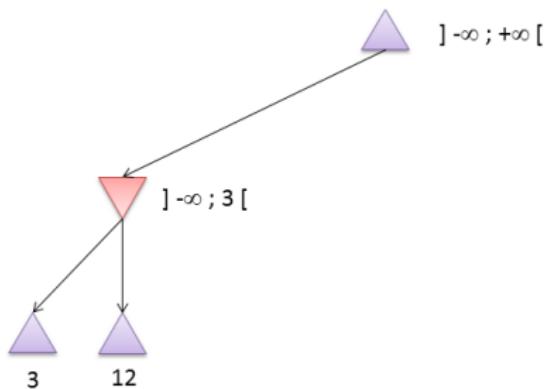
# Exemple 1



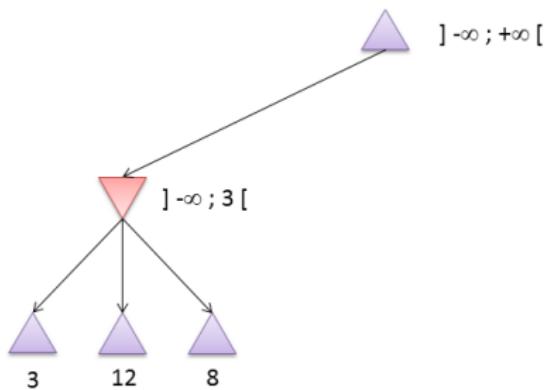
## Exemple 1



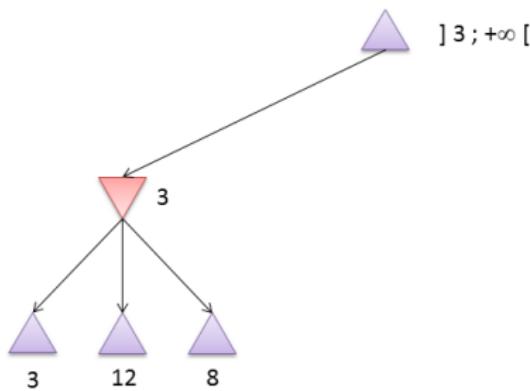
## Exemple 1



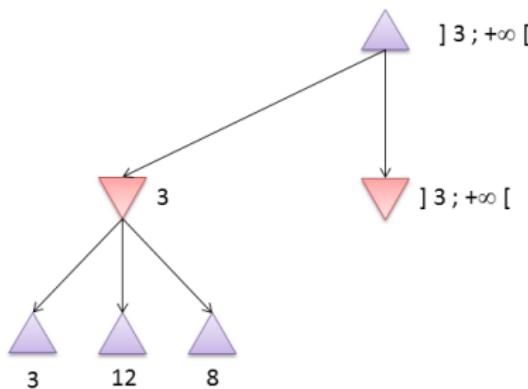
## Exemple 1



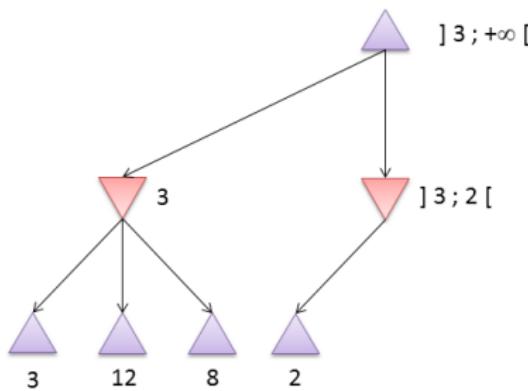
## Exemple 1



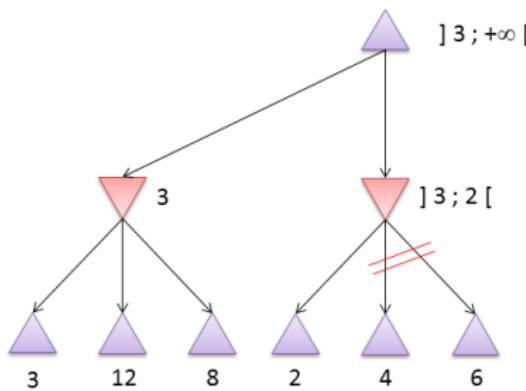
## Exemple 1



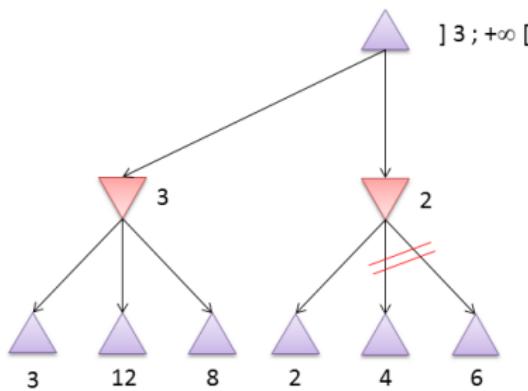
## Exemple 1



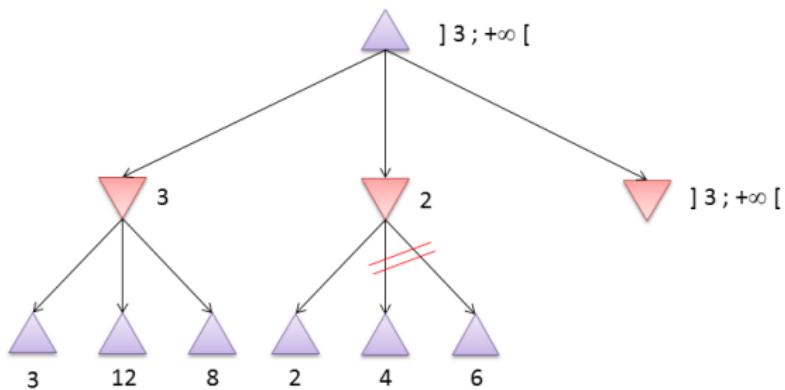
## Exemple 1



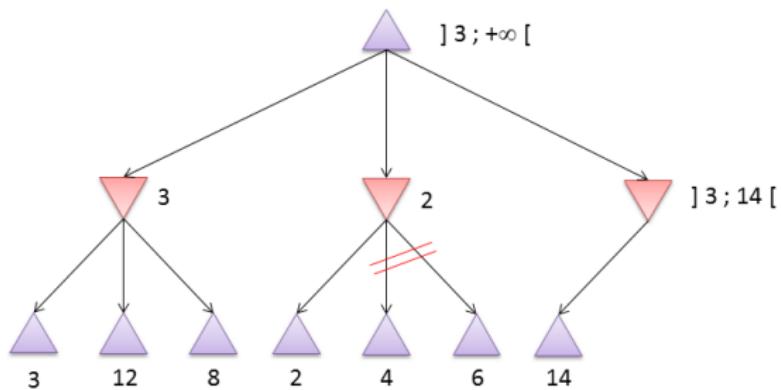
## Exemple 1



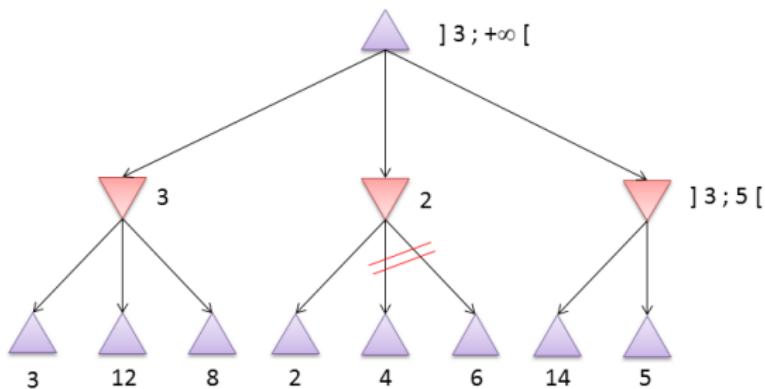
## Exemple 1



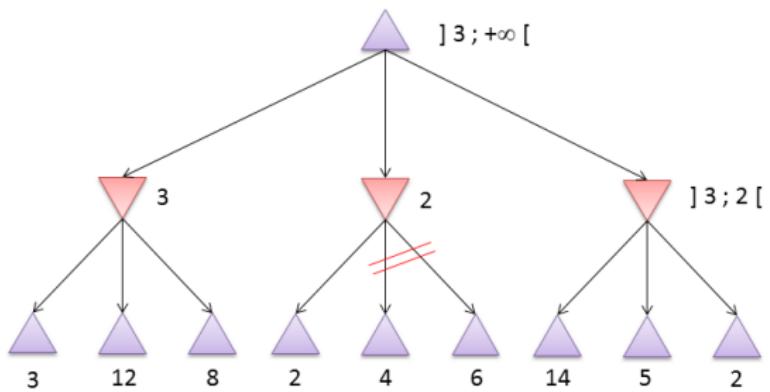
## Exemple 1



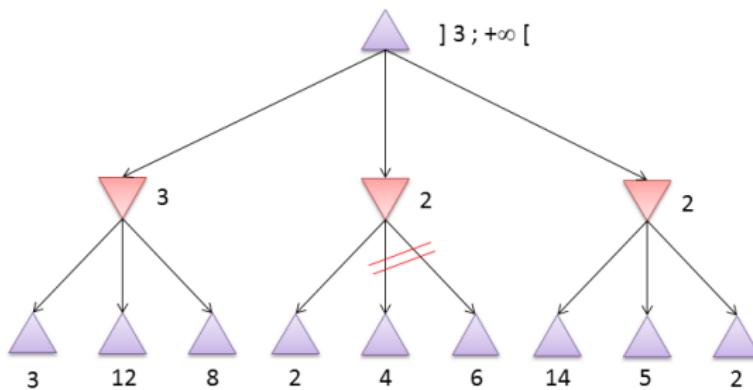
## Exemple 1



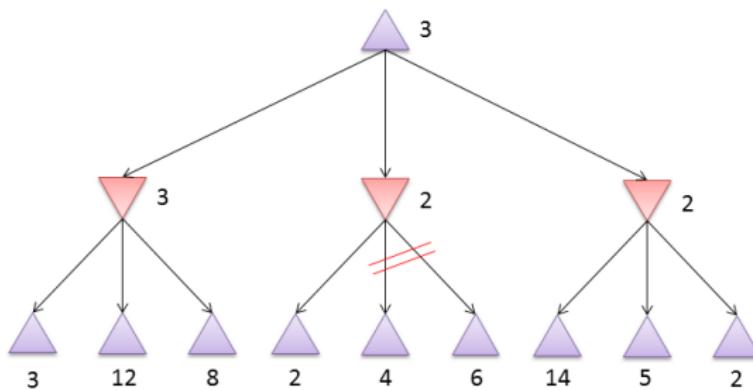
## Exemple 1



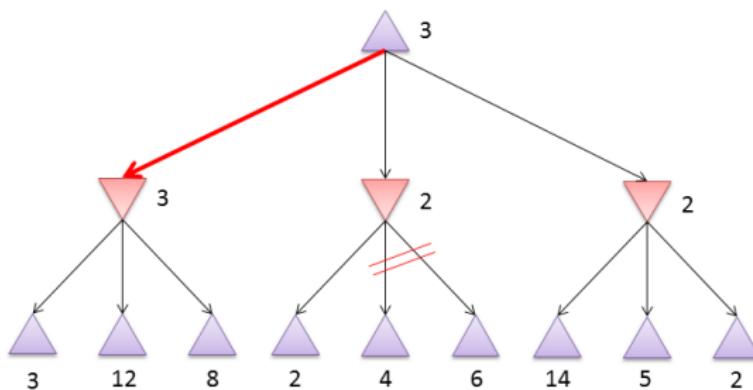
## Exemple 1



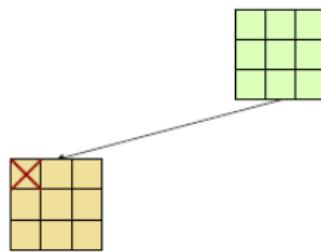
## Exemple 1



## Exemple 1

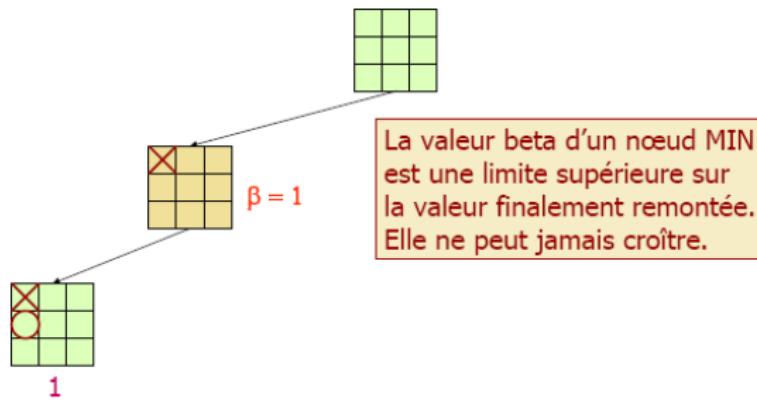


## Exemple 2 : Le Tic Tac toe



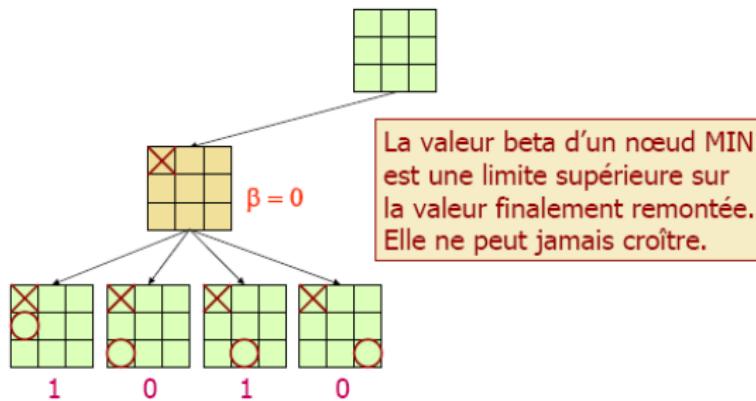
D'après <http://cui.unige.ch/DI/cours/IA>

## Exemple 2 : Le Tic Tac toe



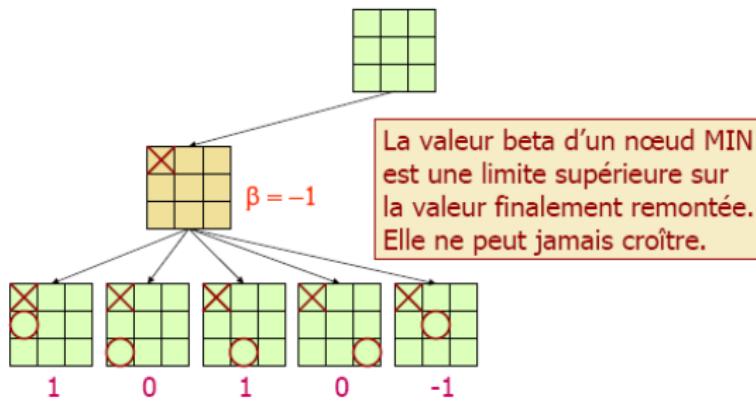
D'après <http://cui.unige.ch/DI/cours/IA>

## Exemple 2 : Le Tic Tac toe



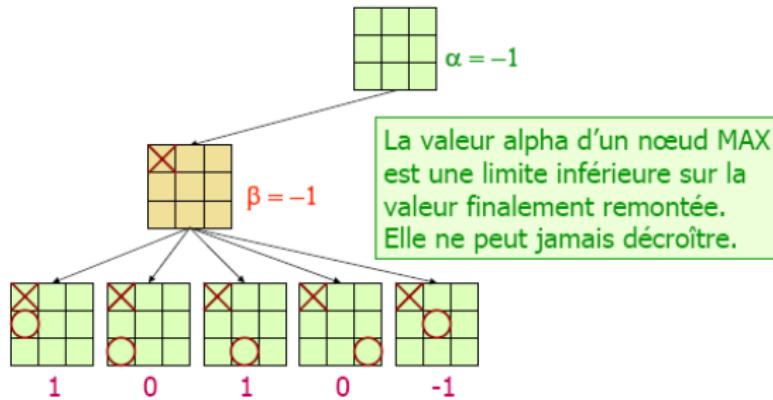
D'après <http://cui.unige.ch/DI/cours/IA>

## Exemple 2 : Le Tic Tac toe



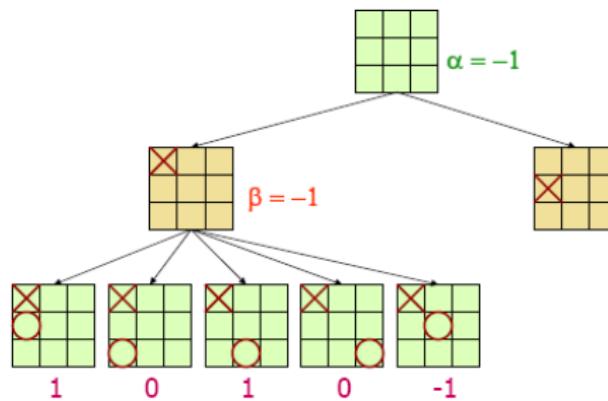
D'après <http://cui.unige.ch/DI/cours/IA>

## Exemple 2 : Le Tic Tac toe



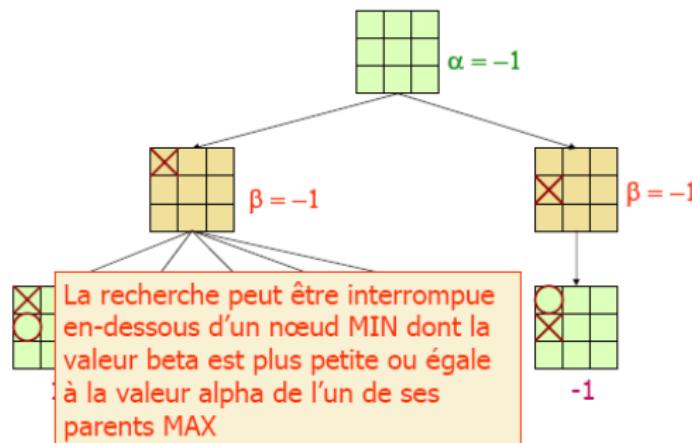
D'après <http://cui.unige.ch/DI/cours/IA>

## Exemple 2 : Le Tic Tac toe



D'après <http://cui.unige.ch/DI/cours/IA>

## Exemple 2 : Le Tic Tac toe



D'après <http://cui.unige.ch/DI/cours/IA>

## Résumé de Alpha-Beta

- ▶ La valeur alpha d'un noeud MAX est une limite inférieure sur la valeur remontée
- ▶ La valeur beta d'un noeud MIN est une limite supérieure sur la valeur remontée
- ▶ Mettre à jour alpha-beta du parent d'un noeud N lorsque toute la recherche en dessous de N est terminée ou a été interrompue.
- ▶ Interrompre la recherche en dessous d'un noeud de type MAX si sa valeur alpha est supérieure à la valeur beta d'un ancêtre de N de type MIN
- ▶ Interrompre la recherche en dessous d'un noeud de type MIN si sa valeur beta est inférieure à la valeur alpha d'un ancêtre de N de type MIN

# Résumé de Alpha-Beta

## Efficacité théorique

On suppose que les fils sont ordonnés idéalement ou presque le coût de l'exploration est de  $O(b^{\frac{d}{2}})$  au lieu de  $O(b^d)$

- ▶ Facteur de branchement théorique  $\sqrt{b}$
- ▶ La recherche peut aller deux fois plus loin dans l'arbre

# Résumé de Alpha-Beta

## Pour aller plus loin

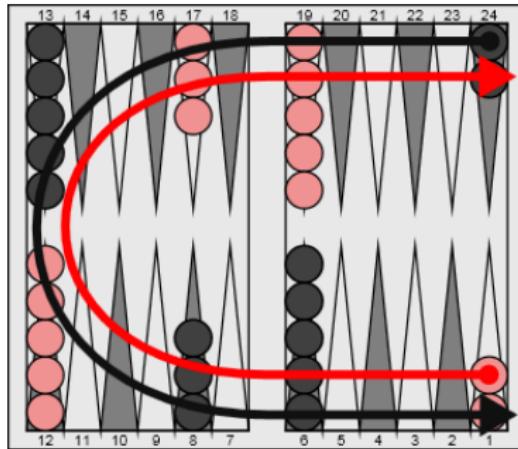
- ▶ Négamax : on n'alterne pas deux fonctions Max/Min mais on se restreint à une seule fonction en utilisant l'opposé du résultat à chaque niveau
- ▶ Recherche  $\alpha\beta$  sur des fenêtres réduites
- ▶ SSS : exploration en largeur

# Comparaison des algorithmes

Nombre de nœuds explorés et temps d'exécution (ramené à un temps sans unité) pour le jeu othello.

$p$	minimax	$\alpha - \beta$	SSS*
1	3 - 0.37	3 - 0.36	3 - 0.36
2	14 - 1.38	13 - 1.4	11 - 1.19
3	61 - 6.0	37 - 3.9	35 - 3.7
4	349 - 32.5	150 - 14.77	95 - 10.0
5	2050 - 185.7	418 - 41.4	292 - 19.2
6	13773 - 1213.5	1830 - 172.9	1617 - 117.3

# Backgammon



## Backgammon

- ▶ But : sortir toutes les dames du plateau
- ▶ Un jeu de stratégie mais dont les déplacements possibles sont déterminés par un lancer de dés
- ▶ On lance deux dés qui indiquent les mouvements possibles

# Backgammon

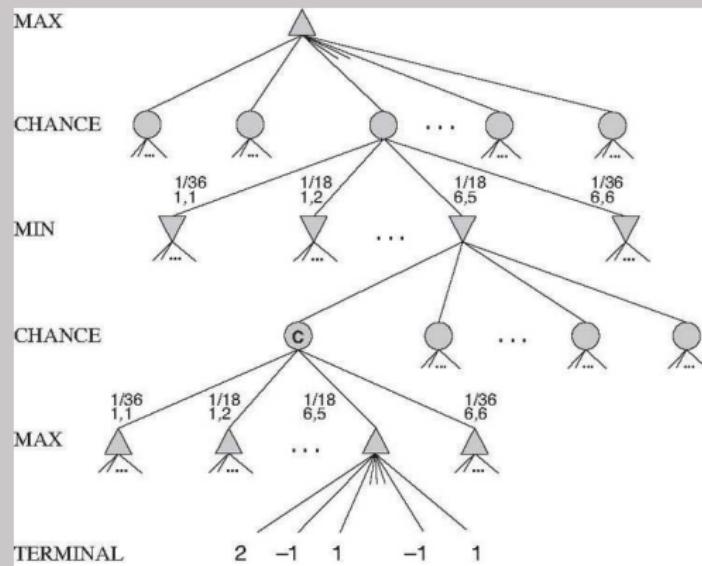
## Problématique

Comment modifier les algorithmes précédents pour tenir compte du lancer de dés ?

# Jeux avec hasard

## Idée

On va non seulement avoir des couches MIN et MAX, mais aussi des couches CHANCE pour représenter le rôle du hasard.



# Jeux avec hasard

## ExpectMinMax

L'algorithme est le même que le MiniMax sauf que la valeur rentrée est la suivante :

EXPECTMINIMAX( $n$ ) =

$$\begin{cases} \text{evaluation}(n) & \text{si } n \text{ est un nœud terminal} \\ \max_{s \in \text{successeurs}(n)} \text{EXPECTMINIMAX}(s) & \text{si } n \text{ est un nœud MAX} \\ \min_{s \in \text{successeurs}(n)} \text{EXPECTMINIMAX}(s) & \text{si } n \text{ est un nœud MIN} \\ \sum_{s \in \text{successeurs}(n)} P(s) \cdot \text{EXPECTMINIMAX}(s) & \text{si } n \text{ est un nœud CHANCE} \end{cases}$$

# De la pratique à la théorie

## Contrôle du temps avec $\alpha - \beta$

Dans la plupart des jeux, le temps d'un coup est limité. Pour cela, généralement, on construit l'arbre jusqu'au niveau 3. On regarde si le temps est dépassé. S'il ne l'est pas, on relance la recherche sur le niveau suivant, et ainsi de suite.

## Table de transposition

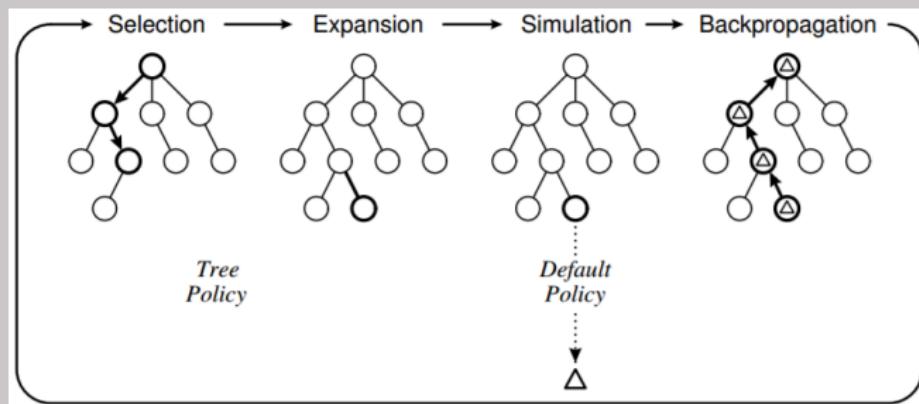
- ▶ Puisque l'algorithme évalue chaque plateau, on peut les stocker dans une table de hachage
- ▶ La fonction de hachage attribue un entier aléatoire à chaque case du tableau (une fois pour toute) et fait le XOR des positions

# Monte-Carlo Tree Search

## Utilisation

Quand l'espace de recherche est trop grand ou que la fonction d'évaluation est complexe (par ex. : le jeu de Go), on utilise une méthode dite de Monte-Carlo.

## Principe



# Monte-Carlo Tree Search

## 4 étapes

- ▶ **Sélection** : à partir de la racine, un choix est fait récursivement pour descendre jusqu'à un nœud qu'il est urgent d'étendre parmi les nœuds extensibles (i.e. pas terminaux et qui possèdent au moins un fils non visité).
- ▶ **Expansion** : un ou plusieurs fils sont ajoutés au nœud sélectionné précédemment en fonction des actions disponibles.
- ▶ **Simulation** : simulations de parties pour produire un score.
- ▶ **Propagation arrière** : on met à jour les statistiques des nœuds de la branche parcourue.

Chaque nœud mémorise :

- ▶ le nombre de fois qu'il a été visité
- ▶ le score obtenu

# Monte-Carlo Tree Search

## Stratégies pour choisir le prochain coup à jouer

On choisit une des actions à partir de la racine selon un critère parmi :

- ▶ **Max** : on choisit l'action qui mène au nœud qui a la récompense la plus grande ;
- ▶ **Robustesse** : on choisit l'action qui mène au nœud le plus visité ;
- ▶ **Max et robustesse** : on choisit l'action qui mène au nœud qui a à la fois la plus grande récompense et le plus visité. Si un tel nœud n'est pas trouvé, on essaie au moins d'atteindre un nombre de visites acceptables et on choisit le nœud qui a la récompense la plus grande.
- ▶ ...

# Méthodes de Monte-Carlo

## Comment choisir le nœud à étendre ?

Dilemme dit d'exploration/exploitation :

- ▶ faire plus de simulations pour des coups qui semblent les meilleurs (pour diminuer l'incertitude de leur évaluation),
- ▶ mais on veut explorer les moins bons dans le cas où ils seraient en fait meilleurs.

# Méthodes de Monte-Carlo

## Critère UCB (Upper Confidence Bound)

Sélectionner l'action qui maximise le critère UCB :

$$\mu_i + C \times \sqrt{\log(p) \div p_i}$$

où

- ▶  $\mu_i$  est la moyenne des scores des parties commençant par ce coup,
- ▶  $p$  est le nombre de parties jouées
- ▶  $p_i$  est le nombre de parties commençant par le coup
- ▶  $C$  est une constante qu'il faut régler pour adapter l'algorithme au problème.

## Comment choisir $C$ en pratique ?

$C$  dépend du jeu.

Une constante élevée favorisera l'exploration alors qu'une petite constante favorisera l'exploitation.

# Alpha GO



Although progress has been steady,  
it will take many decades of research and development  
before world-championship calibre GO programs exist.

Jonathan Schaeffer, 2001

Alpha GO, 2016, bat un joueur de niveau mondial.

# Alpha GO

## Principe

- ▶ basé sur MCTS
- ▶ utilisation de réseaux de neurones pour choisir le nœuds à étendre, pour simuler les parties et pour évaluer un plateau de jeu
- ▶ utilise à la fois des parties jouées par des humains mais également des parties jouées contre lui-même

# Alpha GO

## Sélection des coups (network policy)

Deux réseaux de neurones (profonds) différents + 1 pour initialiser :

- ▶  $P_\pi$  : petit réseau entraîné sur des parties réelles, estime  $P(a|s)$ , et qui sera utilisé pendant les simulations pour connaître le prochain coup à jouer (temps de réponse : 2  $\mu s$ )
- ▶  $P_\sigma$  : réseau plus gros, entraîné sur les parties réelles, estime également  $P(a|s)$
- ▶  $P_\rho$  : réseau de même topologie que  $P_\sigma$ , initialisé comme  $P_\sigma$ , puis entraîné par renforcement sur des parties avec lui-même pour maximiser les victoires (temps de réponse : 3 ms), utilisé dans la phase de sélection

## Estimation du gain (value policy)

Un réseau profond de neurones pour calculer  $V(s)$ , la valeur d'un plateau de jeu, estimé à partir des parties contre lui-même.

# Alpha GO

## En pratique

- ▶ 30 millions de positions jouées par des humains utilisées
- ▶  $V(s)$  comme critère de MCTS permet à elle seule de battre tous les logiciels de GO mais pour vaincre les champions humains il faut utiliser également la valeur issue de la simulation
- ▶ Version normale du logiciel : 40 threads, 48 CPUs, 8GPUs (pour jouer, pas pour apprendre)
- ▶ Version distribuée : 40 threads, 1202 CPUs, 176 GPUs
- ▶ AlphaGO remporte 494/495 parties contre les autres programmes