

# Fondement de l'Intelligence Artificielle



Abir CHaabani

[abir.chaabani@gmail.com](mailto:abir.chaabani@gmail.com)

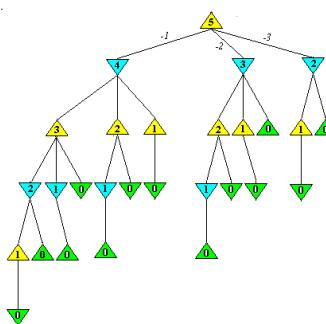
[abir.chaabani@enicar.ucar.tn](mailto:abir.chaabani@enicar.ucar.tn)

Université de Carthage - ENICARTHAGE

2 GINF 2024/2025

1

## Chapitre 4



## Les algorithmes de jeux

2

# Plan Chapitre 4

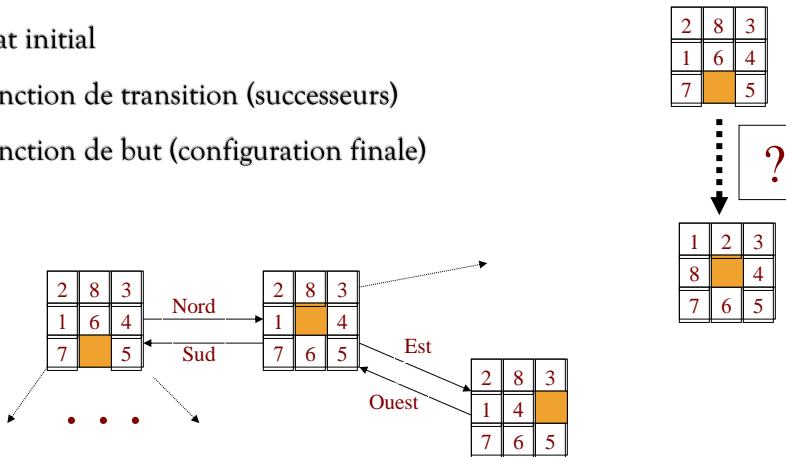
Se familiariser avec les sujets suivants :

- Jeux entre deux adversaires.
- Algorithme minimax.
- Élagage alpha-beta.
- Décisions imparfaites en temps réelle

3

# Rappel

- ❖ Notion d'état (configuration)
- ❖ État initial
- ❖ Fonction de transition (successeurs)
- ❖ Fonction de but (configuration finale)



4

# Rappel

## ❖ Environnement

- Complètement observable (vs. partiellement observable).
- Déterministe (vs. Stochastique)
- Épisodique (vs. séquentiel)
- Statique (vs. Dynamique)
- Discret (vs. Continu)
- Agent unique (vs. multi-agent)

5

# Vers les jeux avec adversité ...

- ❖ Q : Est-il possible d'utiliser A\* pour des jeux entre deux adversaires ?
  - Q : Comment définir un état pour le jeu d'échecs ?
  - Q : Quelle est la fonction de but ?
  - Q : Quelle est la fonction de transition ?
- ❖ R : Non. Pas directement.
- ❖ Q : Quelle hypothèse est violée dans les jeux ?
- ❖ R : Dans les jeux, l'environnement est multi-agent. Le joueur adverse peut modifier l'environnement.
- ❖ Q : Comment peut-on résoudre ce problème ?
- ❖ Les jeux sont des environnements multi-agents. L'adversaire agit sur l'environnement, ce qui le rend : dynamique (il peut changer pendant la réflexion), et parfois non-déterministe (si l'adversaire joue de manière imprévisible).
- ❖ R : Supposer que l'adversaire joue de façon rationnelle...

6

# Particularité des jeux avec adversaires

- ❖ Plusieurs acteurs qui modifient l'environnement (les configurations/états du jeu).
- ❖ Les coups des adversaires sont “imprévisibles”.
- ❖ Le temps de réaction à un coup de l'adversaire est limité.

7

# Relation entre les joueurs

- ❖ Dans un jeu, des joueurs peuvent être :
- ❖ **Coopératifs**
  - ❖ ils veulent atteindre le même but
- ❖ **Des adversaires en compétition**
  - ❖ un gain pour les uns est une perte pour les autres
  - ❖ cas particulier : les jeux à somme nulle (*zero-sum games*)
  - ❖ jeux d'échecs, de dame, tic-tac-toe, Connect 4, etc.
- ❖ **Mixte**
  - ❖ Il y a tout un spectre entre les jeux purement coopératifs et les jeux avec adversaires. Exemple : jeux d'alliances.

8

# Particularité des jeux à tour de rôle

- ❖ Dépendamment des jeux, certains joueurs peuvent être:
  - ❖ Coopératifs
  - ❖ Rivals
- ❖ Les joueurs peuvent avoir une connaissance totale ou partielle de l'état du jeu.
- ❖ Ici nous considérons d'abord **les jeux entre deux adversaires, à somme nulle, à tour de rôle, avec une connaissance parfaite de l'état du jeu, avec des actions déterministes.**
- ❖ Nous aborderons brièvement les généralisations à plusieurs joueurs et avec des actions aléatoires.

9

# Hypothèses

- ❖ Dans ce cours, nous aborderons les :
  - ❖ jeux à **deux adversaires**
  - ❖ jeux à **tour de rôle**
  - ❖ jeux à **somme nulle**
  - ❖ jeux avec **complètement observés**
  - ❖ jeux **déterministes** (sans hasard ou incertitude)
- ❖ Brièvement, nous allons explorer une généralisation à plusieurs joueurs et avec des actions aléatoires (par exemple, jeux dans lesquels on jette un dé pour choisir une action).

10

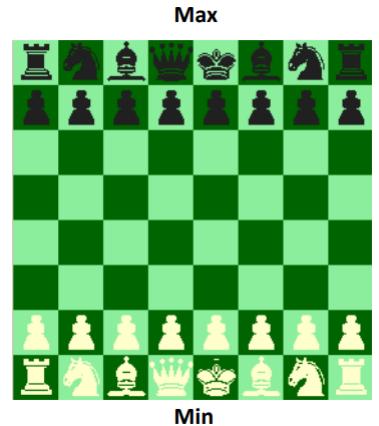
# Jeux vs. problèmes de recherche

- ❖ Jouer c'est rechercher le mouvement permettant de gagner
- ❖ Les jeux de plateau contiennent les notions de:
  - état initial et état gagnant (but)
  - opérateurs de transition (règles du jeu, déplacements de pièces)
- ❖ Une fonction heuristique permet d'estimer s'il y a gain, perte ou match nul

11

# Jeux entre deux adversaires

- ❖ Noms des joueurs : Max vs. Min
- ❖ Max est le premier à jouer (notre joueur):
  - ➔ «premier» au sens «prochain» dans la situation courante.
- ❖ Min est son adversaire
- ❖ Max cherche à maximiser son gain.
- ❖ Min, son adversaire, cherche à minimiser le gain de Max (ou maximiser sa propre perte, ce qui revient au même).
- ❖ Toute récompense positive pour Max est une perte équivalente pour Min, et vice versa.



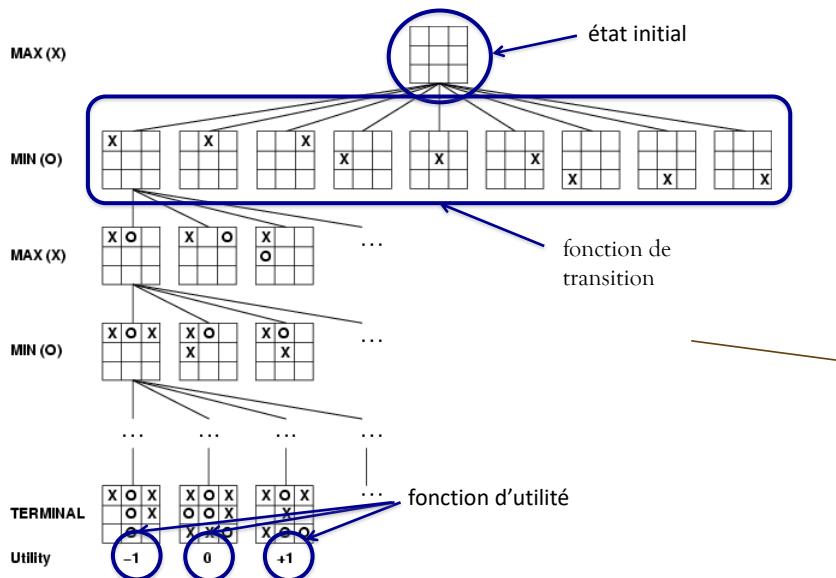
12

# Arbre de recherche

- Comme pour les problèmes que A\* peut résoudre, on commence par déterminer la structure de notre espace de recherche
- Un problème de jeu peut être vu comme un problème de recherche dans un arbre :
  - **Un noeud (état) initial :** configuration initiale du jeu
  - **Une fonction de transition :**
    - ⑥ retournant un ensemble de paires (action, noeud successeur)
    - ✓ action possible (légale)
    - ✓ noeud (état) résultant de l'exécution de cette action
  - **Un test de terminaison**
    - ⑥ indique si le jeu est terminé
  - **Une fonction d'utilité** pour les états finaux (c'est la récompense reçue)

13

## Arbre de recherche tic-tac-toe



- Max (X) joue en premier et cherche à maximiser son score.
- Min (O) joue ensuite et cherche à minimiser le score de Max.
- La partie s'arrête lorsqu'un joueur gagne ou lorsqu'il y a égalité (plateau plein).
- Une évaluation simple est utilisée :
  - +1 si Max (X) gagne.
  - -1 si Min (O) gagne.
  - 0 si c'est un match nul.

# MiniMax



15

Deep Blue est le **superordinateur d'IBM** qui a battu **Garry Kasparov** en 1997 aux échecs. Son fonctionnement repose principalement sur l'algorithme **Minimax**

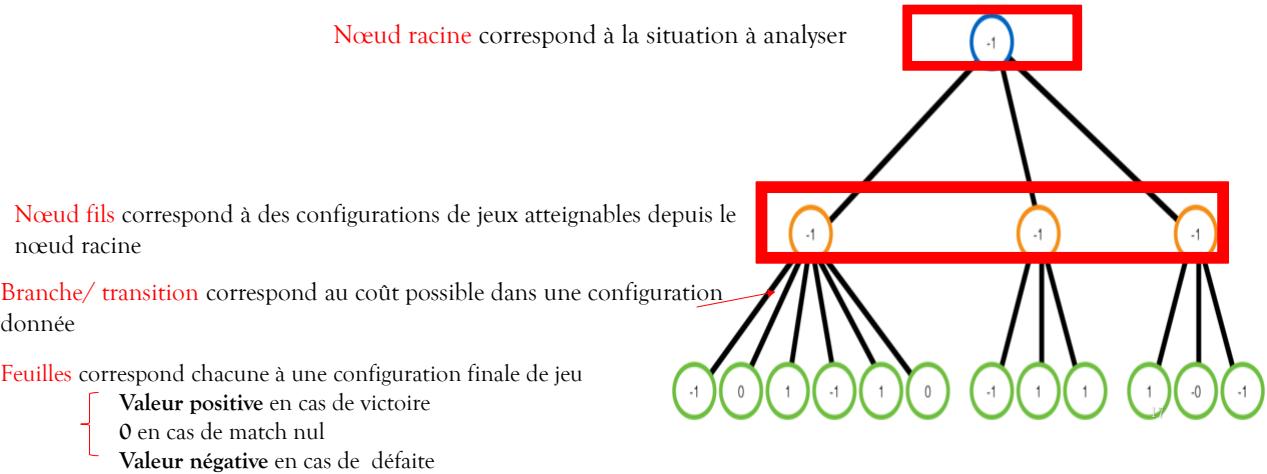
## Algorithme minimax

- L'**algorithme Minimax** est utilisé pour prendre des décisions optimales **dans les jeux à somme nulle** comme le Tic-Tac-Toe, les échecs,...
- S'applique dans **les jeux à information complète**: les deux joueurs ont à tous moments toute l'information sur l'état du jeu, c.à.d les possibilités de jeu de son adversaire, les gains résultants de ses actions, ...
- **Objectif:** Trouver le coup parfait pour un jeu déterministe à information parfaite
- **Principe du Minimax**
  - ✓ Max cherche à maximiser son score (+1 si gagne).
  - ✓ Min (l'adversaire) cherche à minimiser le score de Max (-1 si gagne).
  - ✓ Le jeu est représenté sous forme **d'un arbre de recherche**, où chaque noeud correspond à un **état du plateau**.

Les gains réalisés par un joueur sont des pertes pour l'autre, donc il y a un gagnant, un perdant ou un match nul

# Algorithme minimax

1) L'algorithme MiniMax est associé à un arbre de jeu



## Algorithme minimax: Principe

- Maximiser la valeur d'utilité minimax pour Max avec l'hypothèse que **Min joue parfaitement pour la minimiser**; Ceci revient à choisir le coup qui mène vers l'état qui a la meilleure valeur possible contre le meilleur jeu de l'adversaire.
- 1) L'algorithme MiniMax est associé à un arbre de jeu
  - 2) Calculer la valeur de la fonction de gain pour chaque nœud terminal
  - 3) Propager ces valeurs aux nœuds non-terminaux
    - la valeur minimum (adversaire) aux nœuds MIN
    - la valeur maximum (joueur) aux nœuds MAX
- A. Si le nœud est terminal : Retourner directement la valeur de la fonction d'utilité **UTILITY(n)**.
- B. Si c'est un nœud Max (le joueur Max joue) : **Choisir le maximum** des valeurs minimax de ses successeurs.
- C. Si c'est un nœud Min (l'adversaire Min joue) : **Choisir le minimum** des valeurs minimax de ses successeurs.

# Algorithme minimax: Principe

- Proposer un algorithme qui en tenant compte de l'intérêt opposé de l'adversaire, attribue la meilleure note possible à toutes les situations de l'arbre de jeu
- Mini-Max:** la fonction que nous cherchons à calculer par cet algorithme qui prend en paramètre la situation à évaluer ainsi l'information qui doit jouer dans cette situation.

## Fonction MINIMAX(nœud, profondeur, estMax)

Si nœud est terminal ou profondeur = 0

Retourner la valeur d'utilité du nœud

Si estMax (tour de MAX)

MeilleureValeur =  $-\infty$

Pour chaque enfant du nœud

Valeur = MINIMAX(enfant, profondeur - 1, FAUX)

MeilleureValeur =  $\max(\text{MeilleureValeur}, \text{Valeur})$

Retourner MeilleureValeur

Sinon (tour de MIN)

MeilleureValeur =  $+\infty$

Pour chaque enfant du nœud

Valeur = MINIMAX(enfant, profondeur - 1, VRAI)

MeilleureValeur =  $\min(\text{MeilleureValeur}, \text{Valeur})$

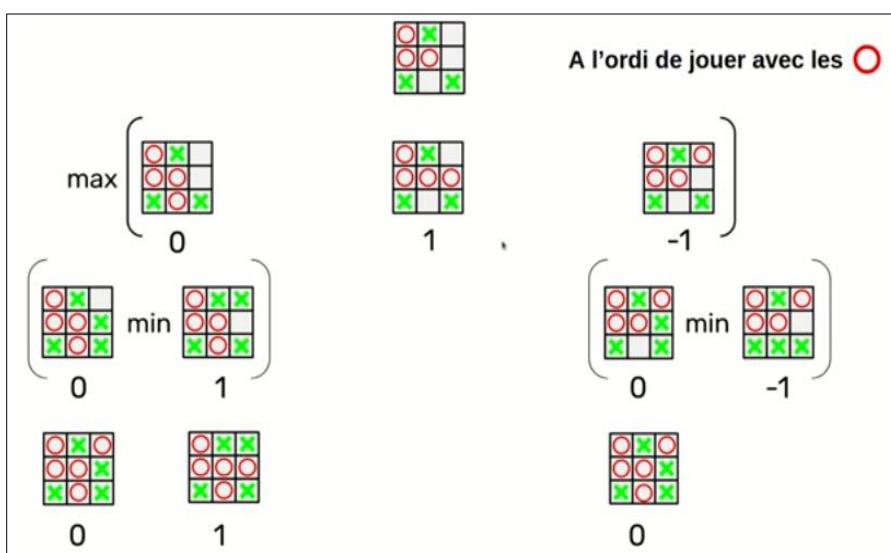
Retourner MeilleureValeur

Explorer tous les nœud c.à.d toutes les configurations à partir d'une situation

19

# Algorithme minimax: Exemple

Exemple à partir d'une configuration pour le jeu de morphin (tic-tac-toe)

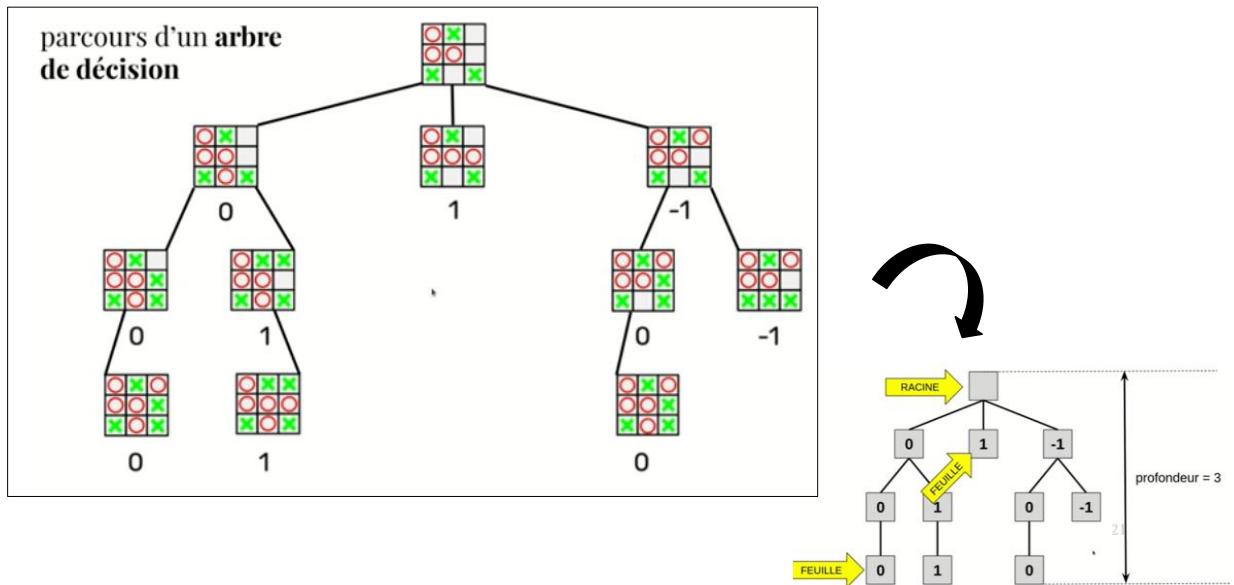


L'algorithme Minimax est utilisé pour :

- Trouver le meilleur coup possible pour un joueur (appelé Max),
- En supposant que l'adversaire (Min) joue de manière parfaite pour minimiser le score de Max

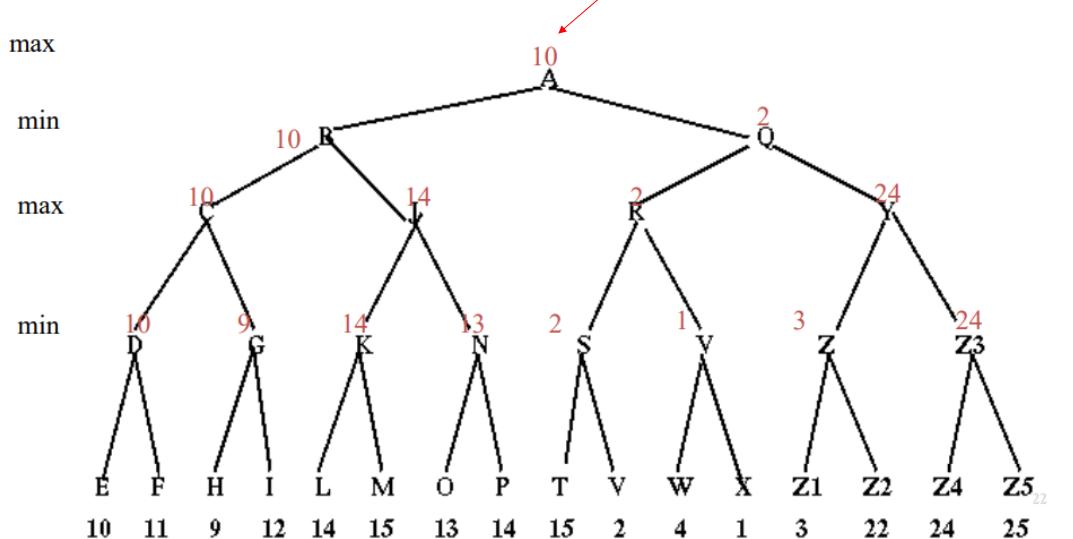
20

# Algorithme minimax: Exemple



# Algorithme minimax: Exemple

- Exemples



# Algorithme minimax: Propriétés

## Complet?

Oui (si l'arbre est fini)

## Optimal?

Oui (contre un adversaire qui joue optimalement)

## Complexité en temps?

$O(b^m)$ :

$b$ : le nombre maximum d'actions/coups légaux à chaque étape

$m$ : nombre maximum de coup dans un jeu (profondeur maximale de l'arbre),

## Complexité en espace?

$O(bm)$ , parce que l'algorithme effectue une recherche en profondeur.

Pour le jeu d'échec:  $b \approx 35$  et  $m \approx 100$  pour un jeu « raisonnable »

Il n'est pas réaliste d'espérer, **un nombre plus grand que le nombre d'atomes dans l'univers!!!!!!**

23

# Comment accélérer la recherche

## Deux approches

la première maintient l'exactitude de la solution

la deuxième introduit une approximation

### 1. Élagage alpha-bêta (alpha-beta pruning)

idée : identifier des chemins dans l'arbre qui sont explorés inutilement

### 2. Couper la recherche et remplacer l'utilité par une fonction d'évaluation heuristique

idée : faire une recherche la plus profonde possible en fonction du temps à notre disposition et tenter de prédire le résultat de la partie si on n'arrive pas à la fin

24

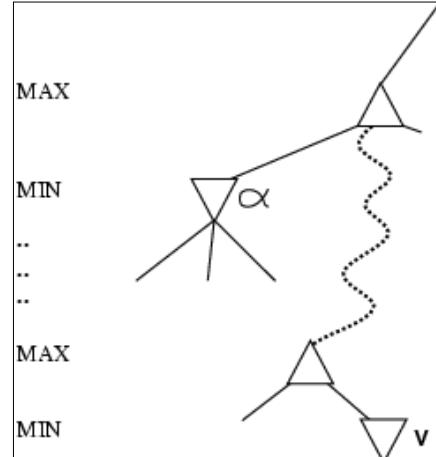
# Alpha-beta Pruning

L'algorithme **alpha-beta** tire son nom des paramètres  $\alpha$  et  $\beta$  décrivant les bornes des valeurs d'utilité enregistrées durant le parcours.

-  $\alpha$  est la valeur du meilleur choix pour Max (c.-à-d., plus grande valeur) trouvée jusqu'ici:

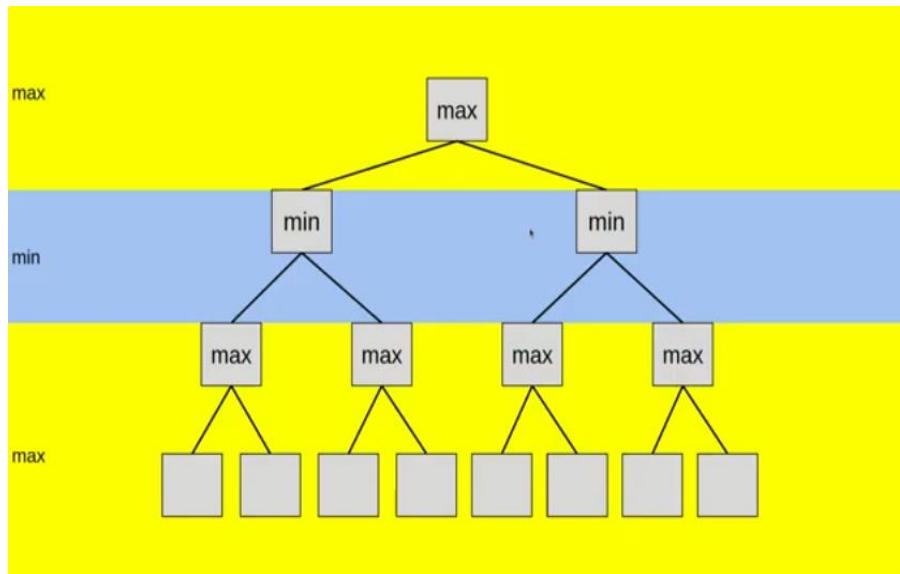
- Si le nœud  $v$  a une valeur pire que  $\alpha$ , Max n'ira jamais à  $v$ : couper la branche

-  $\beta$  est défini de manière analogue pour Min.



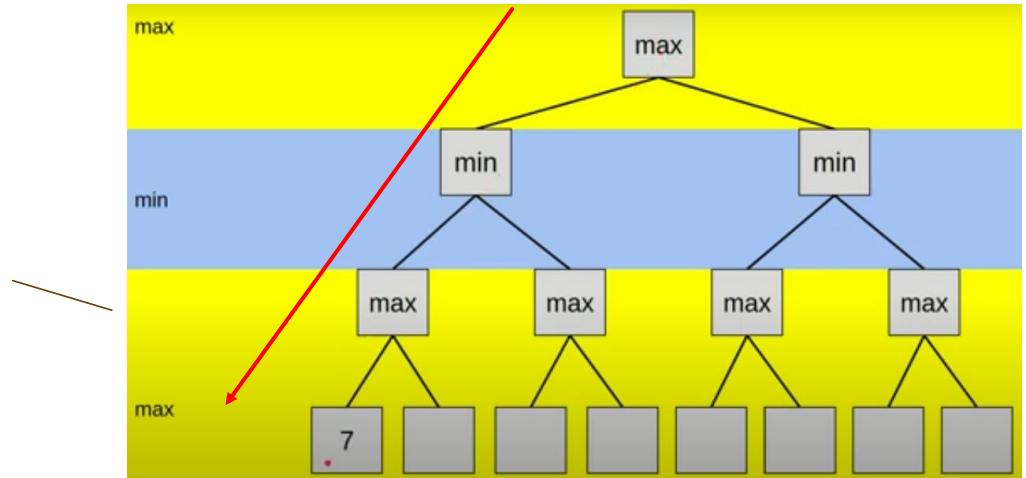
25

## Alpha-beta Pruning : Principe



# Alpha-beta Pruning : Principe

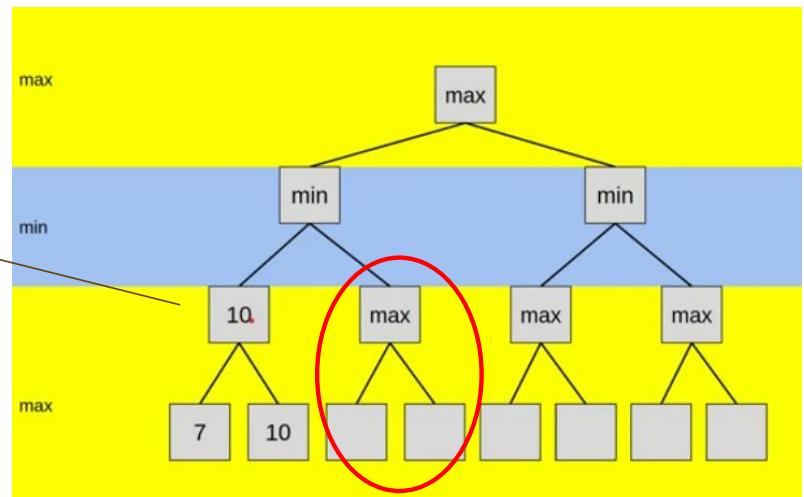
- Alors on va commencer par dérouler l'algorithme Minimax en parcourant en profondeur jusqu'à une feuille.
- Evaluer cette première feuille



# Alpha-beta Pruning : Principe

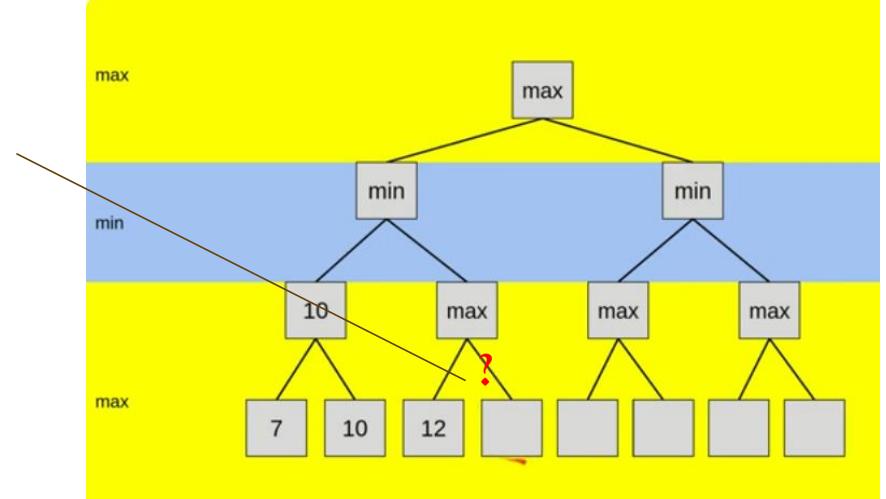
- On reprend alors la valeur maximale qui est 10.

- Par la suite cette valeur va remonter mais il faut tout d'abord évaluer le sous arbre droit pour pouvoir comparer les deux et prendre que le minimum



# Alpha-beta Pruning : Principe

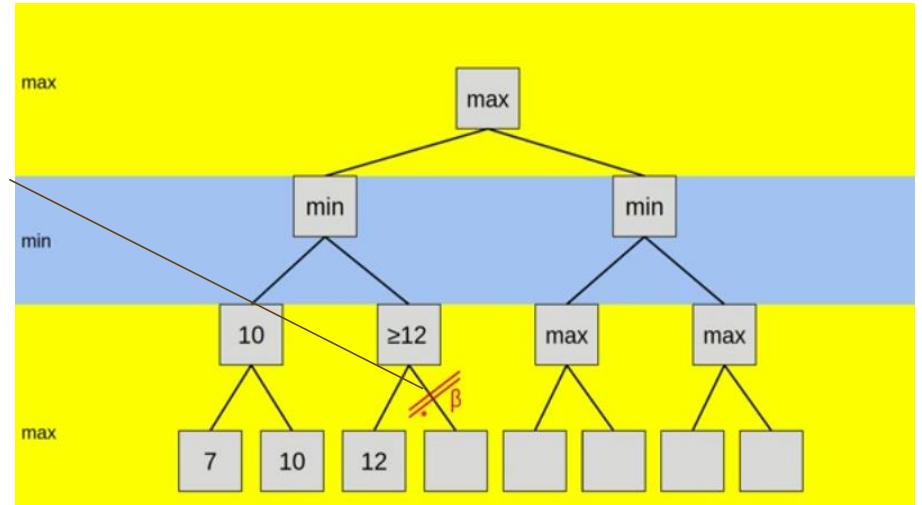
- On va se demander de l'utilité d'évaluer cette partie car:
  - ✓ Si la valeur trouvée x est < 12 et bien 12 qui monte car c'est un nœud max
  - ✓ Si la valeur trouvé x est > 12 alors c'est x qui va remonter



# Alpha-beta Pruning : Principe

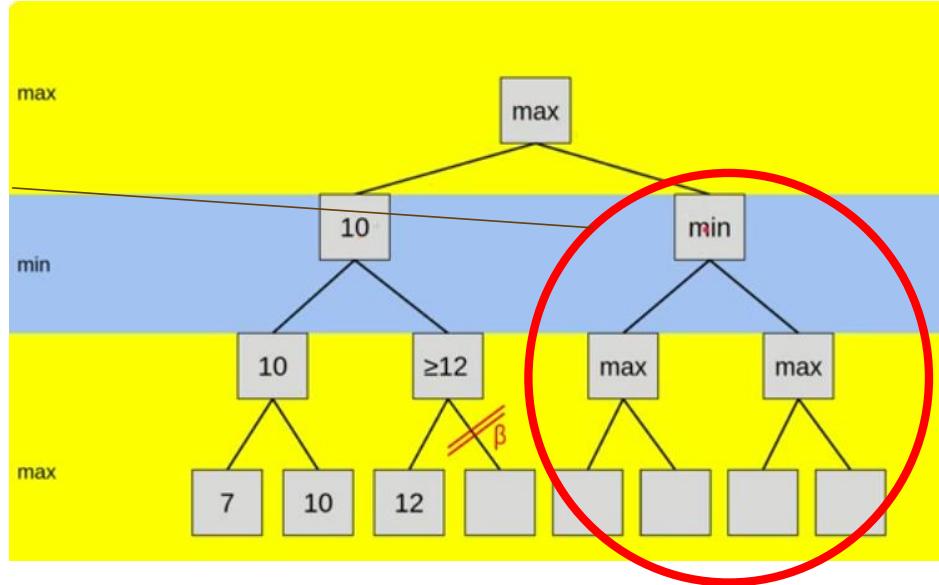
Le nœud qui va être comparé à cet valeur x vaut 10 donc quelque que soit la valeur de cette cellule toujours 10 qui va remonter vers le nœud min .

$\Rightarrow$  On va pratiquer un élagage  $\beta$



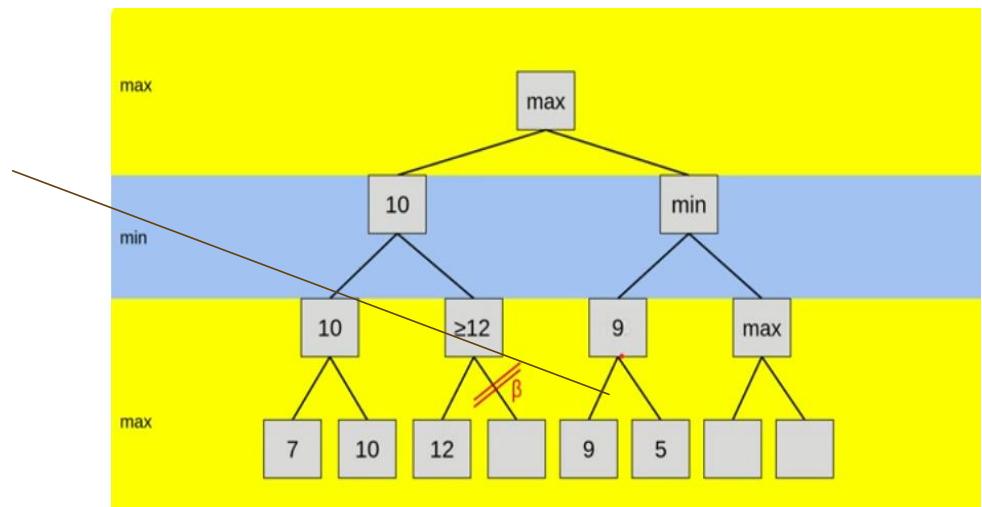
# Alpha-beta Pruning : Principe

On va maintenant évaluer le sous arbre droit pour trouver la valeur qu'on va la comparer avec 10



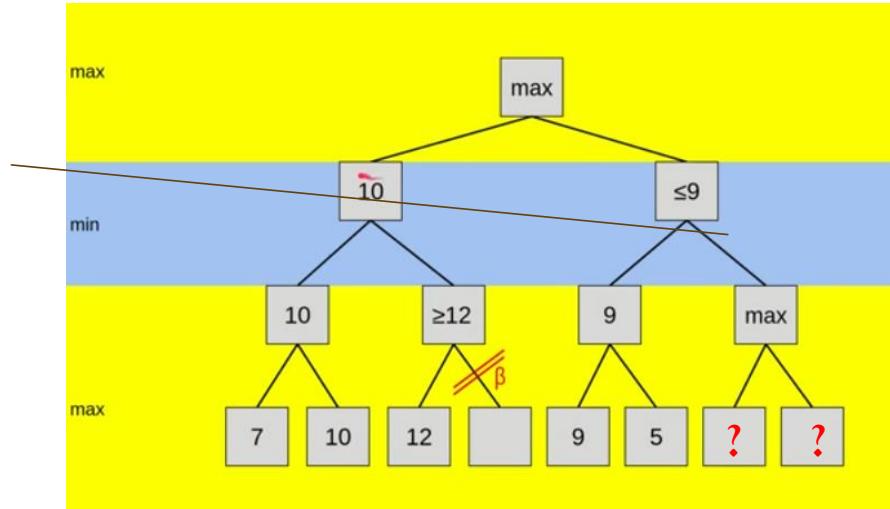
# Alpha-beta Pruning : Principe

On descend dans l'arbre de jeu jusqu'à trouver une feuille et on l'évalue. Puis on va remonter 9 puisque c'est un nœud max



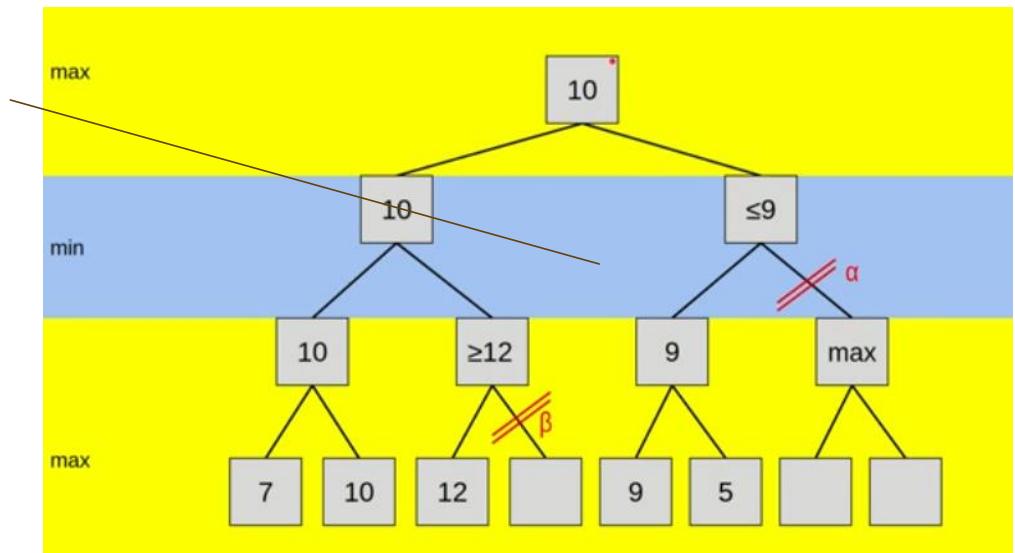
# Alpha-beta Pruning : Principe

On va s'interroger de nouveau: quelque soit les deux valeurs de la sous branche droite, la valeur qui va remonter est toujours  $\leq 9$  puisqu'on va la remonter vers un nœud père min



# Alpha-beta Pruning : Principe

$\Rightarrow$  On va pratiquer **un élagage  $\alpha$**



# Alpha-beta Pruning : Algorithme

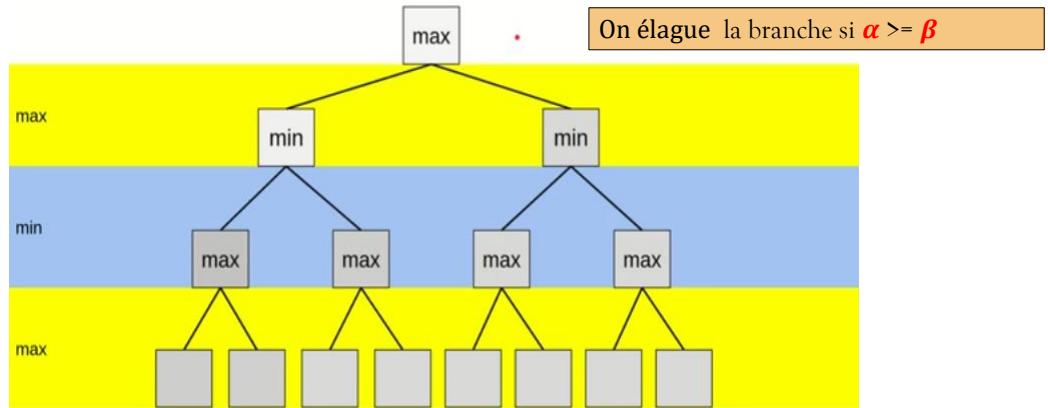
**Règle générale:** On va introduire ces bornes  $\alpha$  et  $\beta$  que l'on va transmettre de père en fils et les modifier en fonction des évaluations qui sont faites

$\alpha$  est la meilleure alternative pour un nœud max.

$\alpha$  est initialisé à  $-\infty$  et il va augmenter d'une évaluation à une autre, il est mis à jour sur un nœud max.

$\beta$  est la meilleure alternative pour un nœud min.

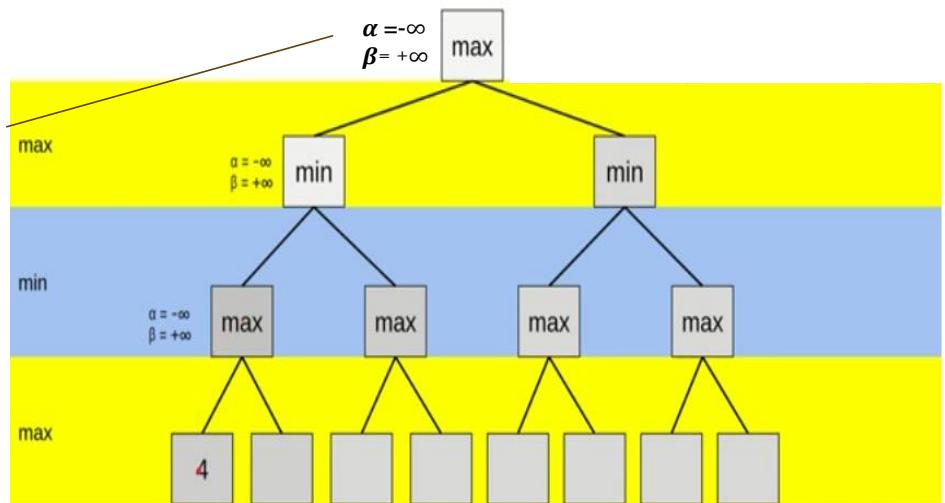
$\beta$  est initialisé à  $+\infty$  et il va diminuer d'une évaluation à une autre, il est mis à jour sur un nœud min.



# Alpha-beta Pruning : Algorithme

## Exemple

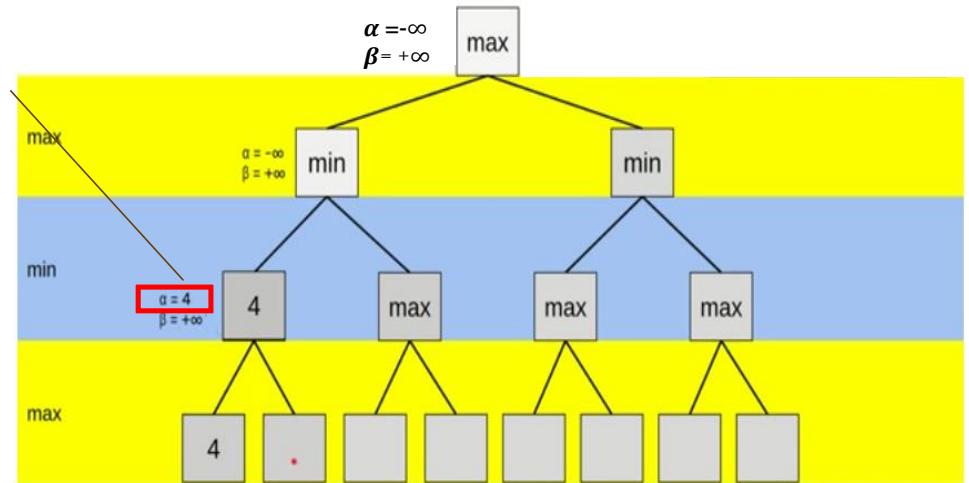
$\alpha$  et  $\beta$  sont initialisés et vont être transmis d'un nœud à un autre jusqu'à arriver à une feuille, et on l'évalue



# Alpha-beta Pruning : Algorithme

## Exemple

4 remonte au nœud père et on s'aperçoit que 4 est meilleur maximum que celui qu'on a jusqu'à présent ( $-\infty$ ) donc on change la valeur de alpha



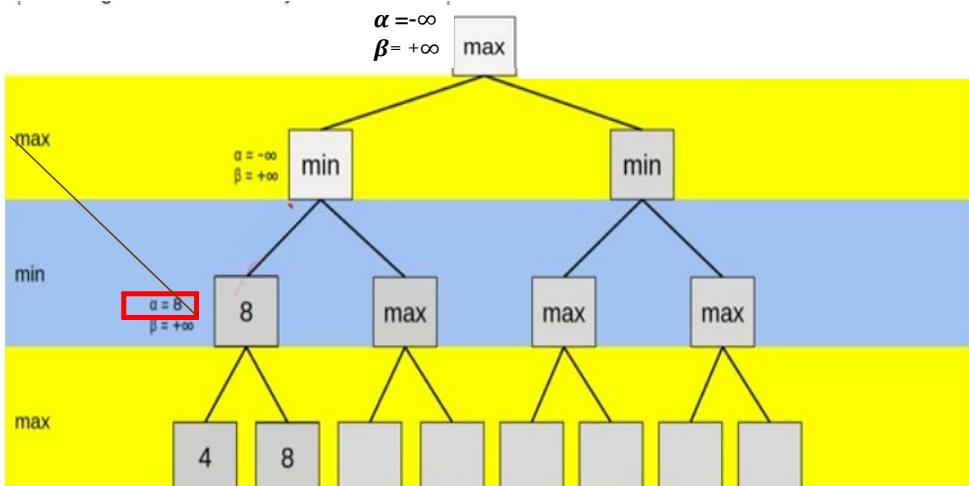
# Alpha-beta Pruning : Algorithme

## Exemple

On évalue la deuxième feuille, supposant on a trouvé 8.

Cette valeur remonte car  $8 > 4$  et on est dans un nœud max et aussi il est un meilleur maximum donc on met à jour  $\alpha$

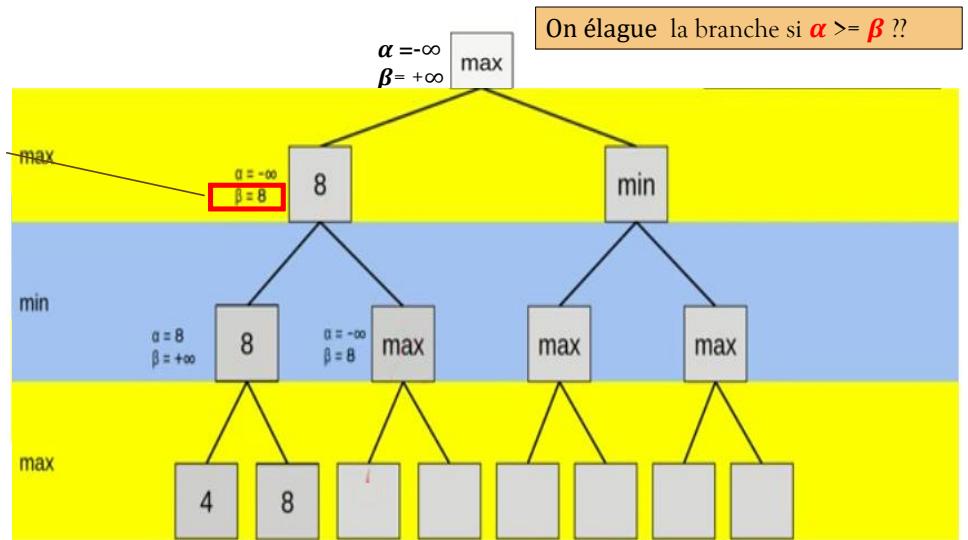
On élague la branche si  $\alpha \geq \beta$  ??



# Alpha-beta Pruning : Algorithme

## Exemple

La valeur 8 remonte vers le père (en attendant d'évaluer la sous arbre droit)  
 8 est un meilleur minimum <  $+\infty$ , donc on met à jour  $\beta$ .  
 Les valeurs  $\alpha$  et  $\beta$  sont transmis au nœud fils et on passe à évaluer la feuille de gauche

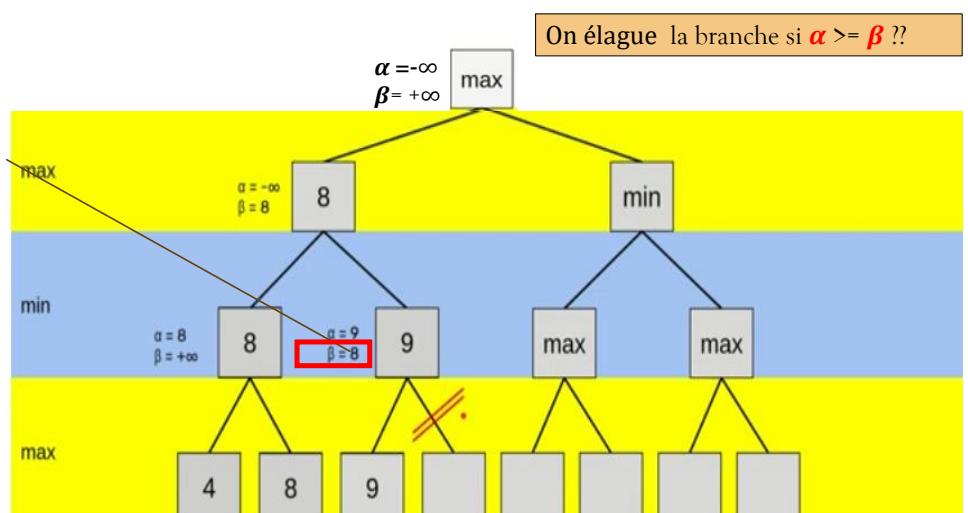


# Alpha-beta Pruning : Algorithme

## Exemple

La feuille prend la valeur 9 qui plus grande que le meilleur maximum donc on met à jour  $\alpha$   
 •  $\alpha > \beta$ , on élague la branche

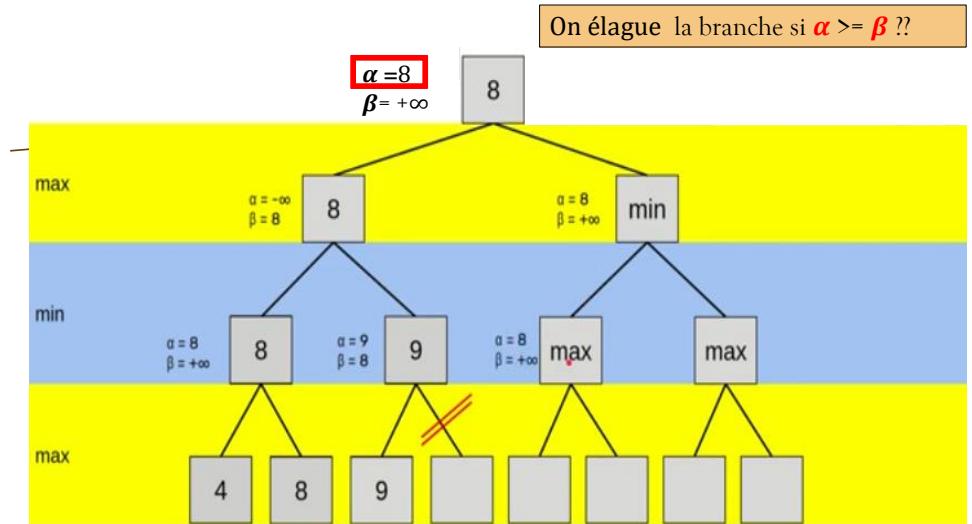
La valeur 8 remonte vers le nœud père



# Alpha-beta Pruning : Algorithme

## Exemple

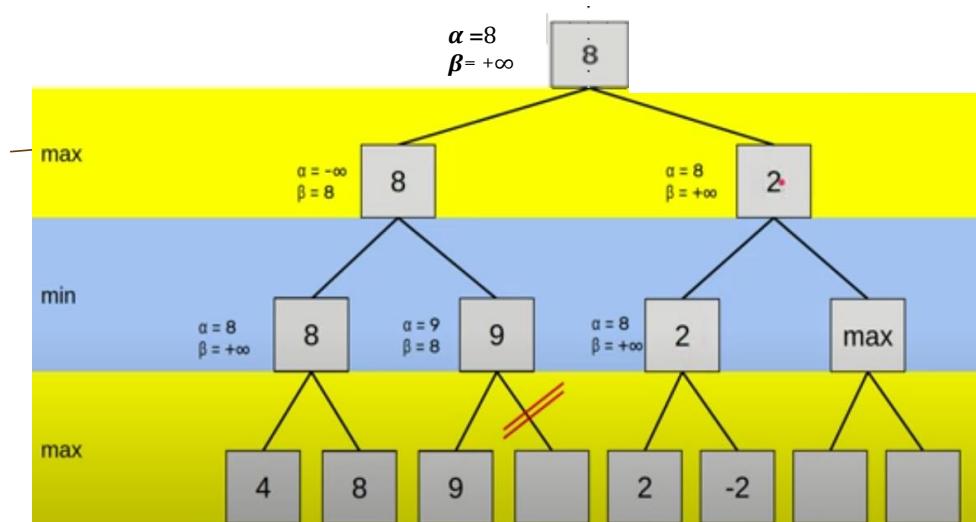
8 est un meilleur maximum que celui du nœud père donc  $\alpha = 8$  que l'on le transmet au nœuds fils et on évalue la feuille



# Alpha-beta Pruning : Algorithme

## Exemple

- 2 remonte vers le père et on va se demander si 2 est un meilleur maximum que la valeur courante? La réponse est non
- On évalue la seconde feuille (-2) et donc 2 qui remonte vers le père min

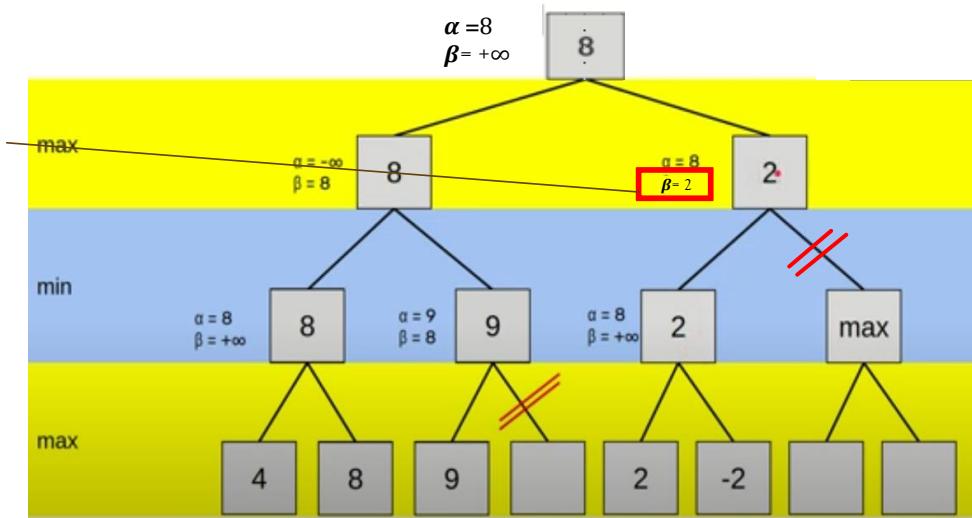


# Alpha-beta Pruning : Algorithme

Exemple

On élague la branche si  $\alpha \geq \beta$  ??

- 2 est il plus petit que le meilleur minimum jusqu'à ici?
- Oui donc  $\beta$  est mis à jour
- **$8 > 2$ , donc élagage**
- Et à la fin on choisie (8, 2), on remonte 8



# Alpha-beta Pruning : Pseudocode

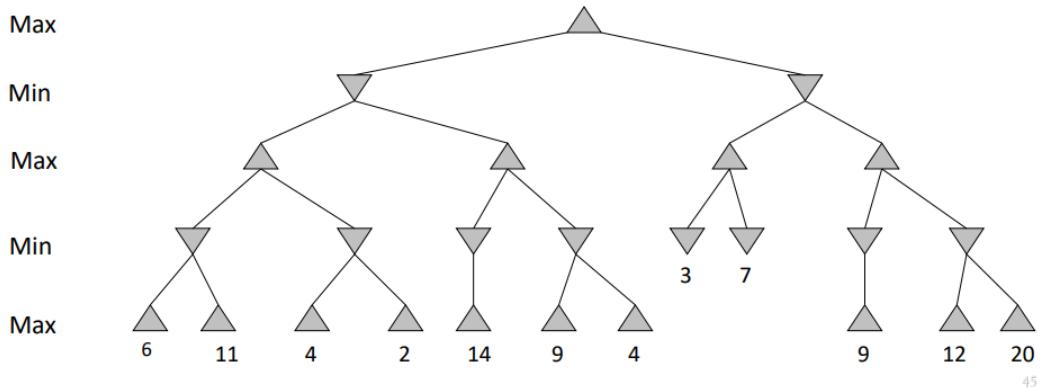
```

fonction minimax(situation, profondeur, α, β)
    si la situation est finale ou profondeur = 3 faire
        renvoyer évaluation de la situation
    sinon faire
        si le joueur est l'ordinateur faire
            valeur ← -∞
            pour toutes les situations atteignables par le joueur humain faire
                valeur ← max(valeur,minimax(situation suivante,profondeur+1,α,β))
            si valeur > β faire
                renvoyer valeur
            fin si
            α ← max(α,valeur) ← mise à jour de α
        fin pour
        sinon faire
            valeur ← +∞
            pour toutes les situations atteignables par l'ordinateur faire
                valeur ← min(valeur,minimax(situation suivante,profondeur+1,α,β))
            si valeur < α faire
                renvoyer valeur
            fin si
            β ← min(β,valeur) ← mise à jour de β
        fin pour
    fin si
    renvoyer valeur
fin si

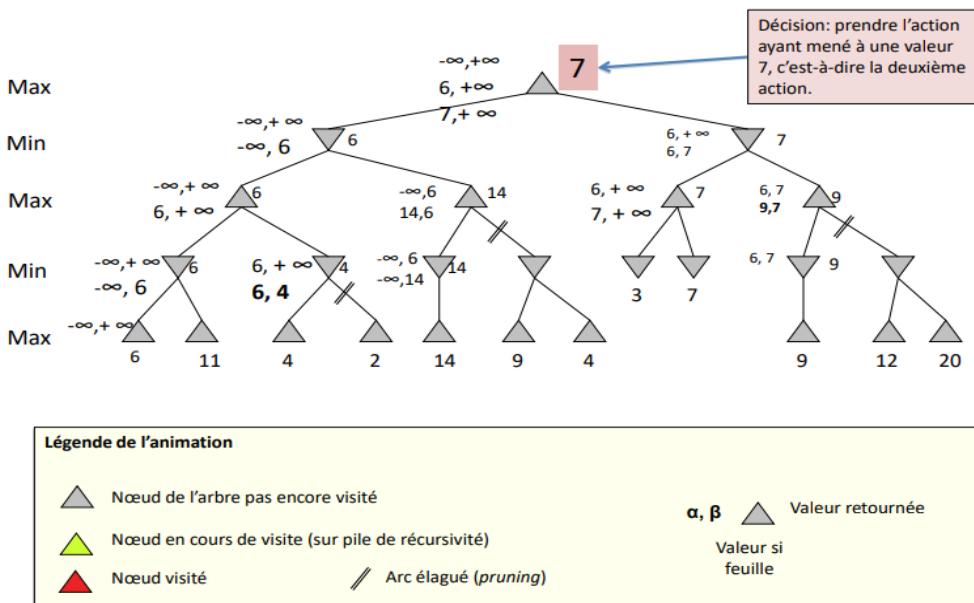
```

## Exemple d'Alpha-beta pruning

## Exercice



## Exemple d'Alpha-beta pruning



# Propriétés de alpha-beta pruning

- L'élagage n'affecte pas le résultat final de *minimax*.
- Dans le pire des cas, alpha-beta *pruning* ne fait aucun élagage; il examine  $b^m$  noeuds terminaux comme l'algorithme *minimax*:
  - ✓  $b$ : le nombre maximum d'actions/coups légaux à chaque étape
  - ✓  $m$ : nombre maximum de coup dans un jeu (profondeur maximale de l'arbre).
- Un bon ordonnancement des actions à chaque noeud améliore l'efficacité.
  - ✓ Dans le meilleur des cas (ordonnancement parfait), la complexité en temps est de  $O(b^{m/2})$
  - ✓ On peut faire une recherche deux fois plus profondément comparé à *minimax*!

47

# Decisions en temps réel

L'algorithme **Minimax** explore toutes les possibilités jusqu'aux feuilles de l'arbre<sup>1</sup> jeu pour prendre la meilleure décision.

**Problème :** Si l'arbre de jeu est trop grand, Minimax prend trop de temps.

**Exemple :**

- Au jeu d'échecs, il y a des millions de possibilités après seulement quelques coups.
- Impossible d'explorer tout l'arbre en temps réel.

## Solution 1 : Limiter la recherche

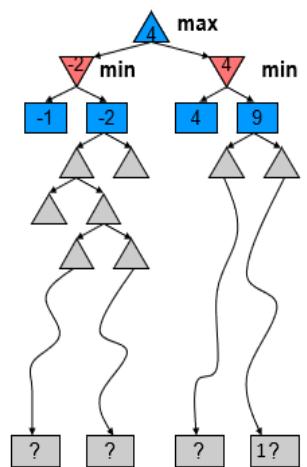
**Idée :** Couper la recherche avant d'atteindre les feuilles.

• On fixe une profondeur max (par exemple 3 ou 4 niveaux). On n'évalue pas l'arbre entier, mais seulement une partie.

• À la fin de la profondeur, on estime la qualité des positions avec une fonction heuristique.

## Solution 2 : Utiliser une fonction d'évaluation heuristique

- Estimation de l'utilité qui aurait été obtenue en faisant une recherche complète.
- On peut voir ça comme une estimation de la « chance » qu'une configuration mènera à une victoire. La solution optimale n'est plus garantie



48

# Exemple de fonction d'évaluation

- Pour le jeu d'échec, une fonction d'évaluation typique est une somme (linéaire) pondérée de "métriques" estimant la qualité de la configuration:

$$\bullet \quad Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

- Par exemple:

- ✓  $w_i$  = poids du pion,
- ✓  $f_i(s)$  = (nombre d'occurrence d'un type de pion d'un joueur) - (nombre de pions de l'adversaire)

On peut ajouter la position des pièces (stratégie) :

1. Un cavalier bien placé (au centre) peut valoir +0,5 supplémentaire.
2. Un roi en sécurité (roqué) peut ajouter +0,3.
3. Un pion isolé (sans protection) peut pénaliser de -0,2.

49

# Exemple de fonction d'évaluation

- Pour le tic-tac-toe, supposons que Max joue avec les X.

$Eval(s) =$

(nombre de ligne, colonnes et diagonales disponibles pour Max) moins (nombre de ligne, colonnes et diagonales disponibles pour Min)

X		O

- Max (X) a 6 possibilités de gagner (2 lignes, 2 colonnes, 2 diagonales).
- Min (O) a 4 possibilités de gagner (2 lignes, 2 colonnes).
- $Eval(s) = 6 - 4 = 2$  (état favorable pour max)

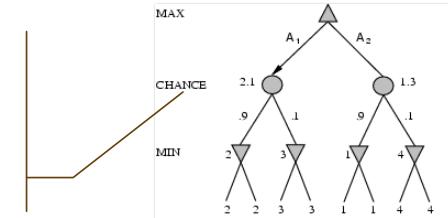
O	X	X
	O	

- Max (X) a 4 possibilités de gagner (2 lignes, 1 colonne, 1 diagonale).
- Min (O) a 3 possibilités de gagner (2 lignes, 1 colonne).
- $Eval(s) = 4 - 3 = 1$

50

# Généralisation aux actions aléatoires

Dans les jeux avec un élément aléatoire (comme lancer un dé ou les actions des fantômes dans Pac-Man), il y a une incertitude sur l'état futur du jeu. Cela nécessite l'ajout d'un autre type de nœud dans l'arbre de décision, appelé un nœud Chance.



- ❖ Exemples :
- ❖ Jeux où on lance un dé pour déterminer la prochaine action
- ❖ Actions des fantômes dans Pacman

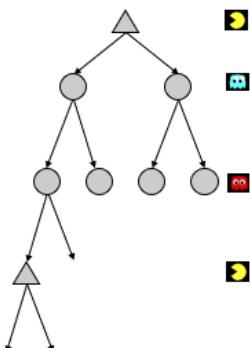
- ❖ Solution : On ajoute des noeuds chance, en plus des nœuds Max et Min
- ❖ L'utilité d'un nœud chance est l'utilité espérée, c.à.d. au lieu de choisir le meilleur ou le pire coup, on fait une moyenne pondérée des résultats possibles.

51

# Algorithme Expectimax

L'algorithme **Expectimax** est une extension de Minimax qui prend en compte le hasard dans les jeux en ajoutant des **nœuds chance**. Il est souvent utilisé dans des jeux comme Pac-Man, où certaines actions sont aléatoires.

- Un modèle probabiliste des comportements de l'opposant:
  - ✓ Le modèle peut être **une simple distribution de probabilités**
  - ✓ Le modèle peut être plus sophistiqué, demandant des **inférences/calculs élaborés**
  - ✓ Le modèle peut représenter des actions stochastiques/incontrôlables (à cause de l'opposant, l'environnement)
  - ✓ Le modèle pourrait signifier que des actions de l'adversaire sont probables
- Pour cette leçon, supposer que (de façon magique) nous avons une distribution de probabilités à associer aux actions de l'adversaire/environnement



*Avoir une croyance probabiliste sur les actions d'un agent ne signifie pas que l'agent lance effectivement un dé!*

# Algorithme Expectimax: Principe

## Principe de l'algorithme Expectimax

1. **Nœud Max** (tour du joueur) → choisit le **meilleur score** parmi ses enfants.
2. **Nœud Min** (tour de l'adversaire) → choisit le **pire score** pour le joueur.
3. **Nœud Chance** (événement aléatoire, comme un lancer de dé ou un mouvement aléatoire d'un fantôme) → **calcule une moyenne pondérée** des scores des enfants.

53

# Quelques succès et défis

Jeu de dames: En 1994, Chinook a mis fin aux 40 ans de règne du champion du monde Marion Tinsley.

Jeu d'échecs: En 1997, Deep Blue a battu le champion du monde Garry Kasparov dans un match de six jeux.

Othello: les champions humains refusent la compétition contre des ordinateurs, parce que ces derniers sont trop bons!

Go: les champions humains refusent la compétition contre des ordinateurs, parce que ces derniers sont trop mauvais!

54