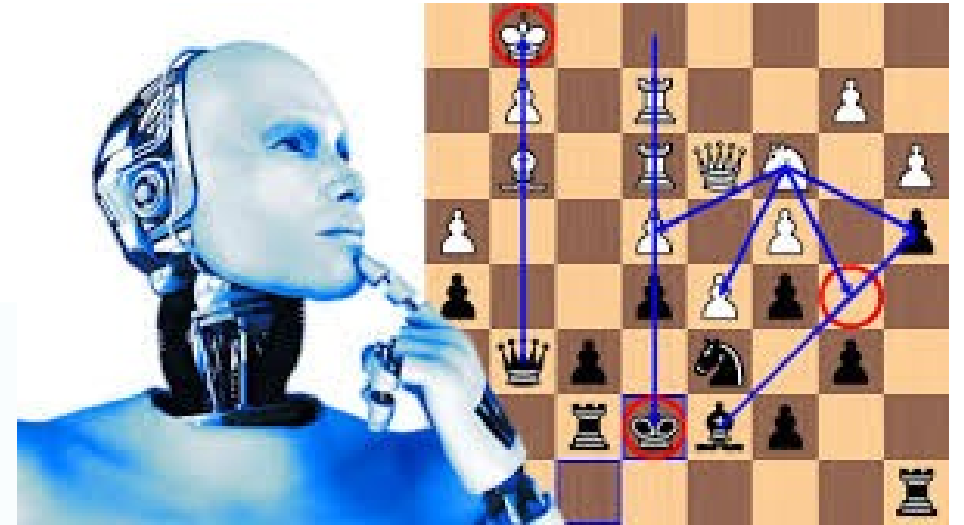


CHAPITRE 4



Exploration en situation d'adversité



« Où l'on voit les problèmes qui surviennent lorsqu'on essaie de bâtir des plans à l'avance dans un monde où d'autres agents établissent des plans opposés aux nôtres »

*Stuart Russel & Peter Norvig
Artificial Intelligence*

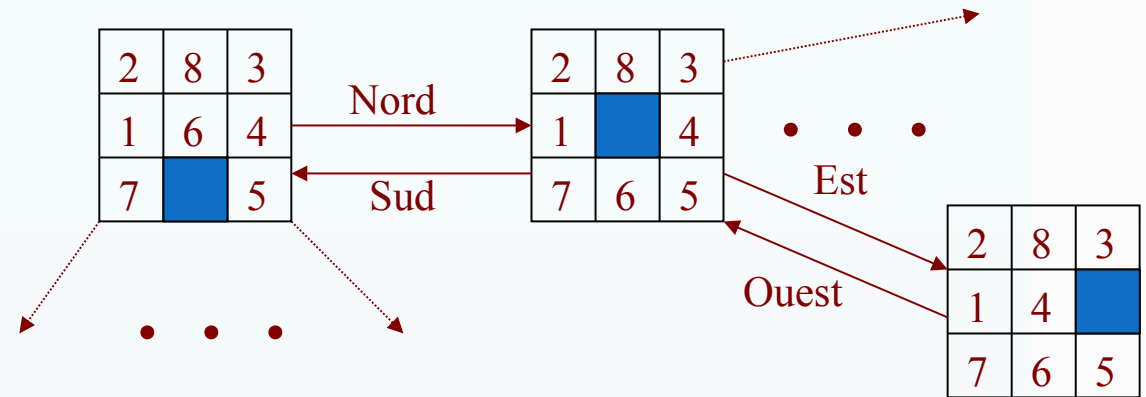
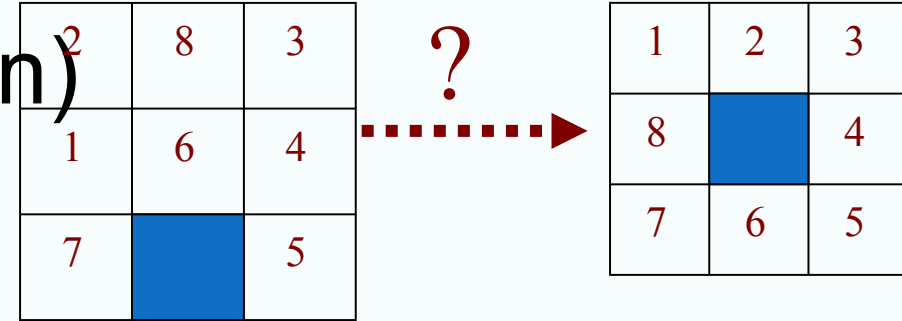
Rappel : A*

- Notion d'état (configuration)

- État initial

- Fonction de transition (successeurs)

- Fonction de but (configuration finale)



Exploration en situation d'adversité : les jeux

- ❑ Un environnement multiagents.
- ❑ Chaque agent doit tenir compte des actions des autres agents.
- ❑ L'imprévisibilité des autres agents crée des contingences dans la résolution du problème.
- ❑ Résolution dans un environnement concurrentiel : les buts des agents sont en conflit (si l'un gagne l'autre perd !).

Essayons A* pour les jeux d'échec quand même ...

- ❑ Comment définir un état du jeu d'échec ?
- ❑ Comment définir un état but ?
- ❑ Quelle est la fonction de transition en présence d'adversité ?
L'adversaire peut modifier l'arbre d'exploration en cours.
- ❑ Et même si nous le faisons :
 - ❑ Le jeu d'échec a un facteur de branchement ≈ 35
 - ❑ Une partie compte souvent jusqu'à 50 coups par joueur
 - ❑ L'arbre d'exploration du jeu d'échec compte $\approx 35^{100}$ nœuds

Programmation de jeux VS Résolution de problèmes

- ❑ Prendre en compte l'imprévisibilité de l'adversaire.
- ❑ Les jeux, comme les problèmes du monde réel nécessitent une prise de décision rapide même si le calcul de la solution optimale est impossible.
- ❑ Plusieurs acteurs qui modifient l'environnement (les configurations/états du jeu).
- ❑ Un algorithme tel que A^* s'avère complètement inefficace.

Solutions étudiées

- ❑ Algorithmes pour définir un coup optimal (algorithme *MiniMax* de Neumann).
- ❑ Elagage *pruning* pour réduire le coût de la recherche (algorithme de *McCarthy*).
- ❑ Fonctions d'évaluation heuristiques pour prendre des décisions en temps réel.

Types de jeux

- ❑ Jeux à information parfaite (actions des adversaires sont observables) :
 - ❑ Déterministes (non stochastique) : échec, go, dames
 - ❑ Non déterministes (hasard) : backgammon (dépend des dés jetés)
- ❑ Jeux à information imparfaite (actions des adversaires ne sont pas toutes observables, exemple le bridge où les joueurs ne voient pas toutes les cartes) :
 - ❑ Déterministes : bridge, poker, scrabble
 - ❑ Non déterministes : Monopoly

Cadre du cours

- ❑ Les joueurs peuvent être
 - ❑ Coopératifs : cherchent à atteindre le même but (des alliances peuvent se former ou se briser à mesure que la partie avance : A et B peuvent s'allier momentanément pour attaquer C qui se trouve en meilleure posture).
 - ❑ Rivalisant : si l'un gagne, l'autre perd.

Définition d'un jeu

- Un jeu peut être défini formellement comme un problème d'exploration par :
 - s_0 : **l'état initial**, qui décrit la configuration initiale du jeu.
 - $\text{JOUEUR}(s)$: définit quel joueur a la main à l'état s .
 - $\text{ACTION}(s)$: renvoie l'ensemble des coups autorisés dans l'état s .
 - $\text{RÉSULTAT}(s,a)$: **modèle de transition**, qui définit l'état résultant de l'action a à partir de l'état s .
 - $\text{TEST-TERMINAL}(s)$: **test de terminaison**, qui retourne vrai si la partie est finie et faux sinon. Les états dans lesquels le jeu s'arrête sont appelés **états terminaux**.
 - $\text{UTILITÉ}(s,p)$: **fonction d'utilité**, appelée aussi fonction objectif ou de gain, définit une valeur numérique pour un jeu qui se termine dans un état terminal s pour un joueur p (récompense reçue).

Jeux à somme nulle

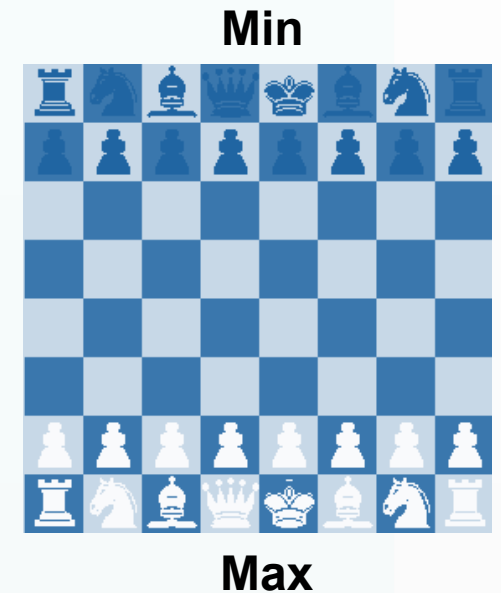
- ❑ Un jeu est dit à somme nulle si la somme des valeurs de la fonction d'utilité obtenues pour tous les joueurs en fin de partie est toujours la même pour toute instance du jeu.
- ❑ En jeu d'échec par exemple, si un joueur gagne, l'autre perd nécessairement : c'est pour cet antagonisme entre les fonctions d'utilité qu'on parle de **situation d'adversité**, ce jeu est à somme nulle :
 - ❑ Victoire : 1
 - ❑ Défaite : 0
 - ❑ Match nul :
- ❑ « Jeu à somme constante » serait plus approprié, mais il paraît que le terme « jeu à somme nulle » a été retenue si on imagine que chaque joueur verse un droit d'entrée de .

Objectifs du cours (pour des jeux à tour de rôle)

- ❑ Jeux à somme nulle, à information parfaite et déterministes :
 - ❑ L'algorithme MiniMAX
 - ❑ L'élagage $\neg \wedge - \vee \wedge$
- ❑ Jeux non déterministes
- ❑ Jeux à information imparfaite en temps réel
- ❑ Conclusion

Jeux à deux joueurs

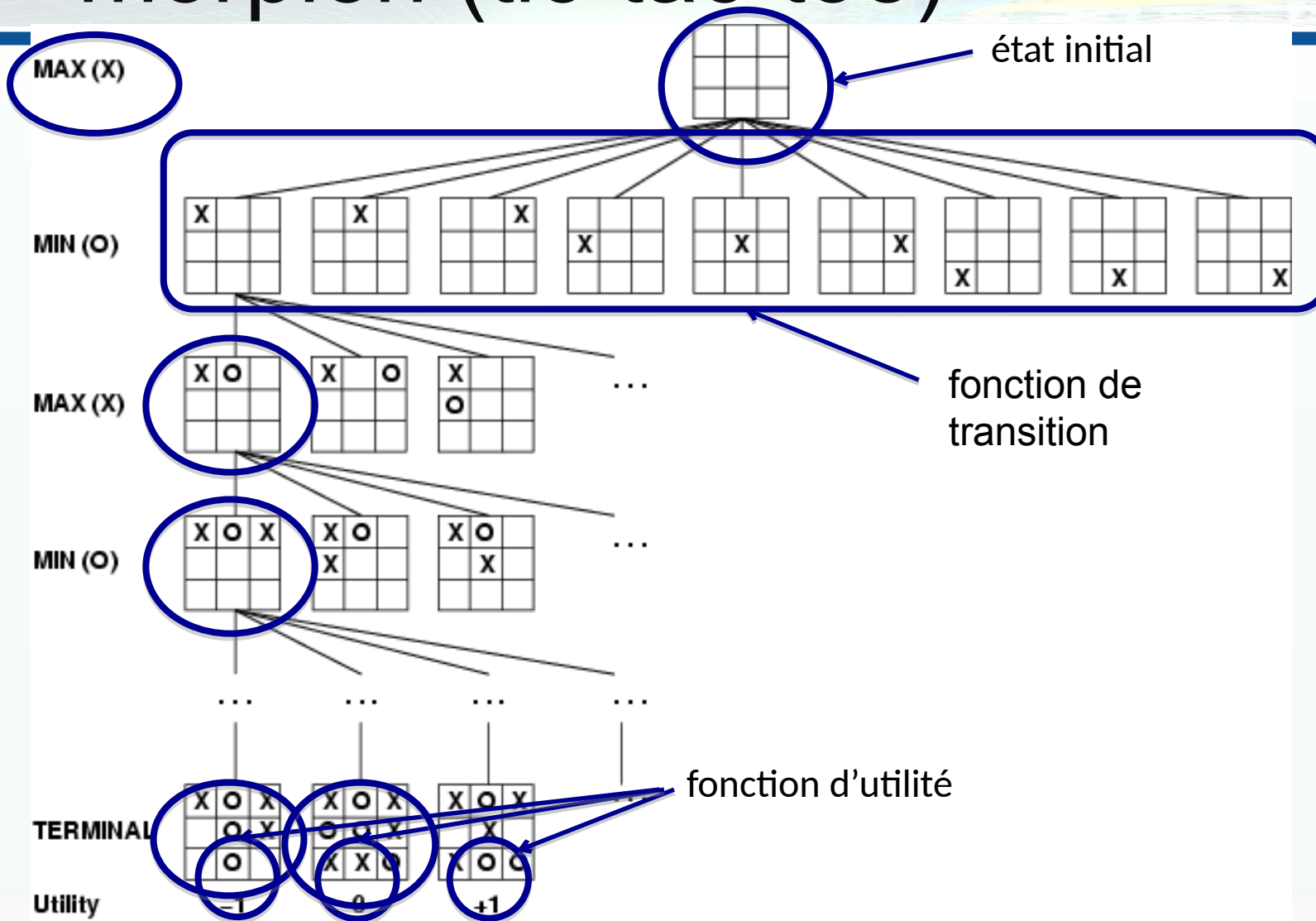
- ❑ Deux joueurs : Max et Min.
- ❑ Max joue en premier.
- ❑ Ensuite les deux joueurs jouent à tour de rôle jusqu'à la fin du jeu.
- ❑ A la fin du jeu, les points sont attribués au vainqueur et des pénalités au perdant : Min reçoit l'opposé de Max.



Arbre de jeu

- L'arbre de jeu pour une partie est défini par :
 - L'état initial : S_0
 - La fonction ACTIONS : coups autorisés.
 - La fonction RÉSULTAT : modèle de transition.
- Les nœuds sont les états du jeu.
- Les arrêtes sont des coups (actions).

Partie de l'arbre de jeu pour le jeu de morpion (tic-tac-toe)

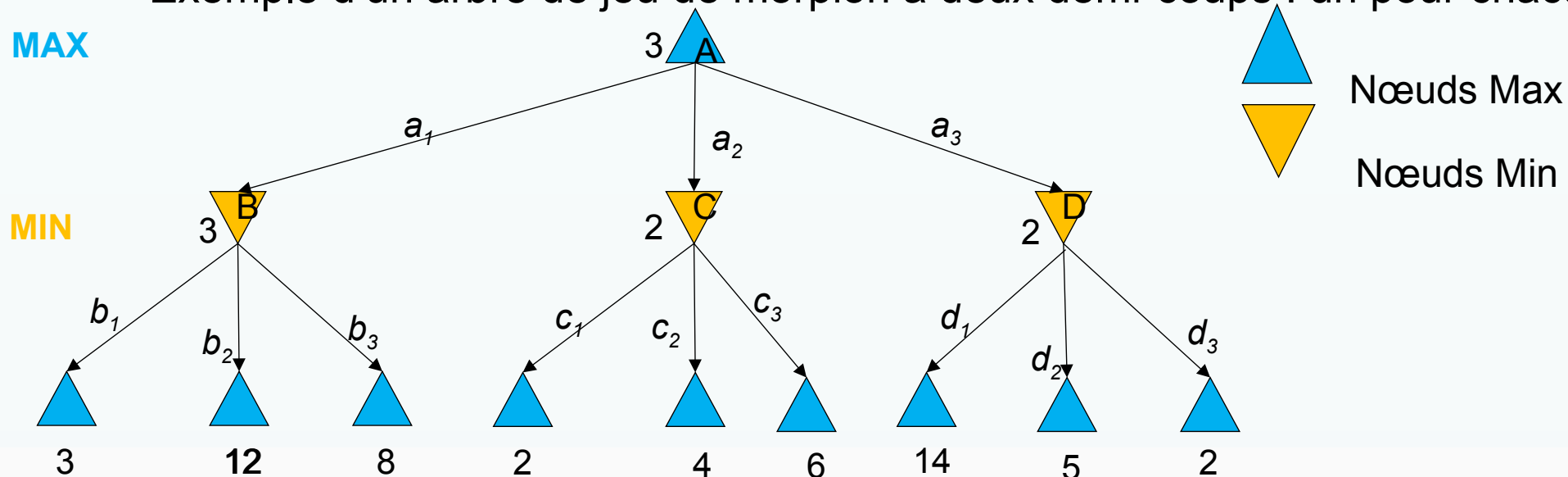


- Max dispose au début de 9 coups possibles
- Max et Min placent alternativement une marque (croix pour Max et rond pour Min)
- Ceci jusqu'à atteindre un nœud feuille (état terminal) : l'un des joueurs a aligné 3 marques ou toutes les cases sont remplies
- Le nombre placé sur chaque feuille indique la valeur d'utilité de l'état terminal : les valeurs élevées sont bonnes pour Max et mauvaises pour Min.

Algorithme MINIMAX

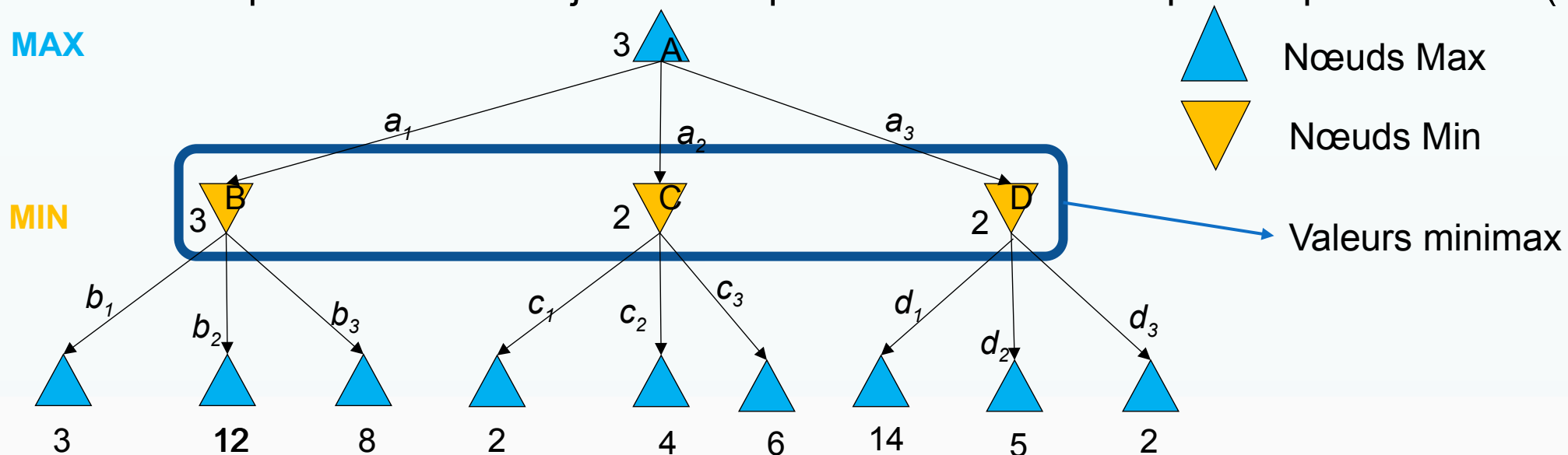
Principe de l'algorithme MINIMAX

- ❑ Il permet de trouver le meilleur coup pour un jeu à information parfaite et déterministe.
- ❑ A chaque tour, choisir l'action menant à l'état qui a une meilleure valeur *minimax* (selon le jeu de l'adversaire).
- ❑ Exemple d'un arbre de jeu de morpion à deux demi-coups : un pour chacun (un tour)



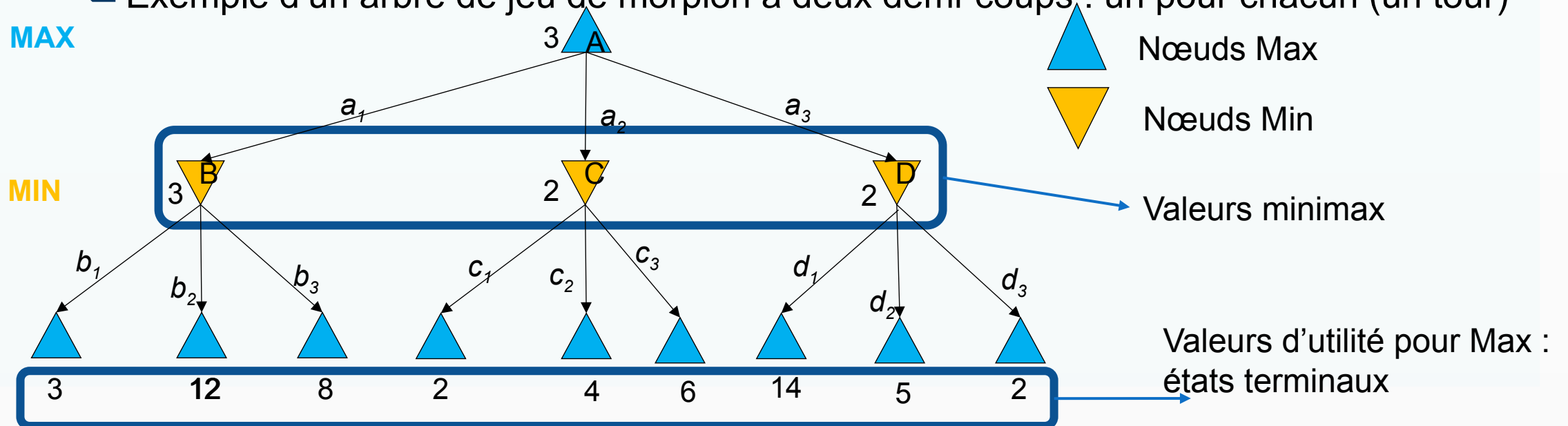
Principe de l'algorithme MINIMAX

- ❑ Il permet de trouver le meilleur coup pour un jeu à information parfaite et déterministe.
- ❑ A chaque tour, choisir l'action menant à l'état qui a une meilleure valeur *minimax* (selon le jeu de l'adversaire).
- ❑ Exemple d'un arbre de jeu de morpion à deux demi-coups : un pour chacun (un tour)



Principe de l'algorithme MINIMAX

- ❑ Il permet de trouver le meilleur coup pour un jeu à information parfaite et déterministe.
- ❑ A chaque tour, choisir l'action menant à l'état qui a une meilleure valeur *minimax* (selon le jeu de l'adversaire).
- ❑ Exemple d'un arbre de jeu de morpion à deux demi-coups : un pour chacun (un tour)



Fonction MINIMAX

- A partir de l'arbre de jeu, on peut déterminer une stratégie optimale à partir de la valeur **minimax** de chaque nœud n , appelons la $\text{MINIMAX}(n)$.

$\text{MINIMAX}(s) =$

$\text{UTILITÉ}(s)$

si TEST-
TERMINAL(s)

si

$\max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RÉSULTAT}(s, a))$

si JOUEUR(s) = M

- C'est un appel récursif qui remonte jusqu'à la racine de l'arbre.

$\min_{a \in \text{Actions}(s)}$

si

Algorithme MINIMAX

fonction DÉCISION-MINIMAX(état) **retourne**
une action

retourner $\operatorname{argmax}_{a \in \text{ACTIONS}(s)} \text{VALEUR-}$
 $\text{MIN}(\text{RÉSULTAT}(\text{état}, a))$

fonction VALEUR-MAX(état) **retourne** *une*
valeur d'utilité

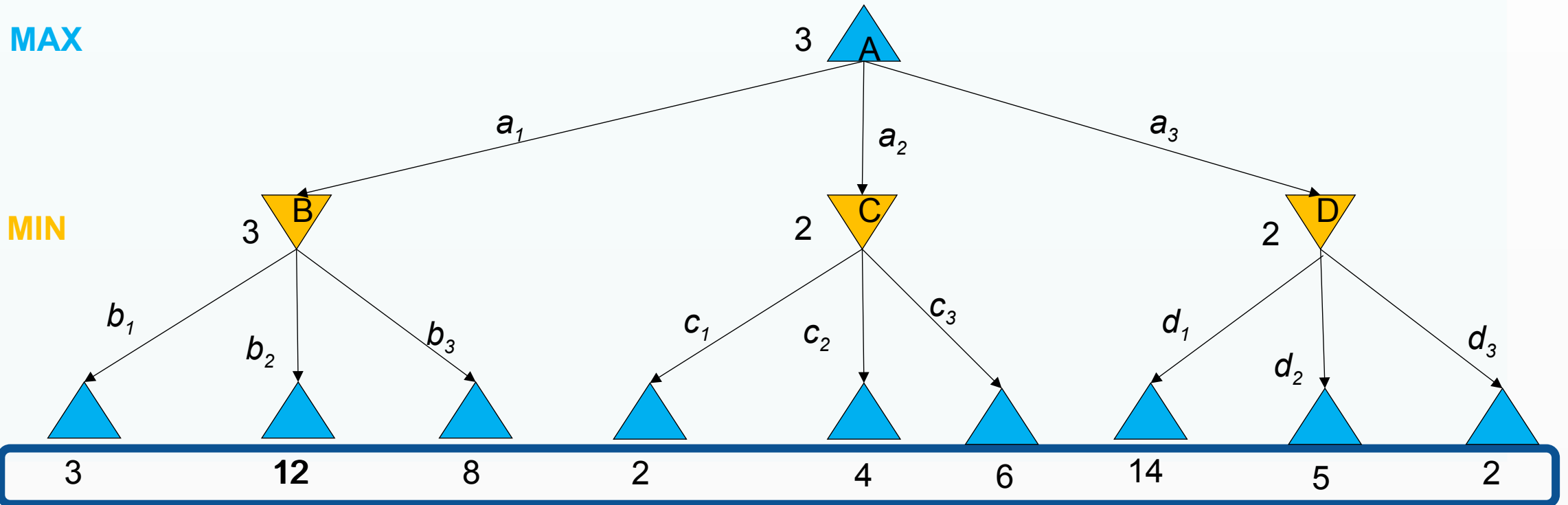
si TEST-TERMINAL(état) **alors retourner**
 $\text{UTILITÉ}(\text{état})$

pour chaque a **dans** $\text{ACTIONS}(\text{état})$ **faire**
 $\text{MAX}(, \text{VALEUR-MIN}(\text{RÉSULTAT}(s, a)))$
retourner

fonction VALEUR-MIN(état) **retourne** *une*
valeur d'utilité

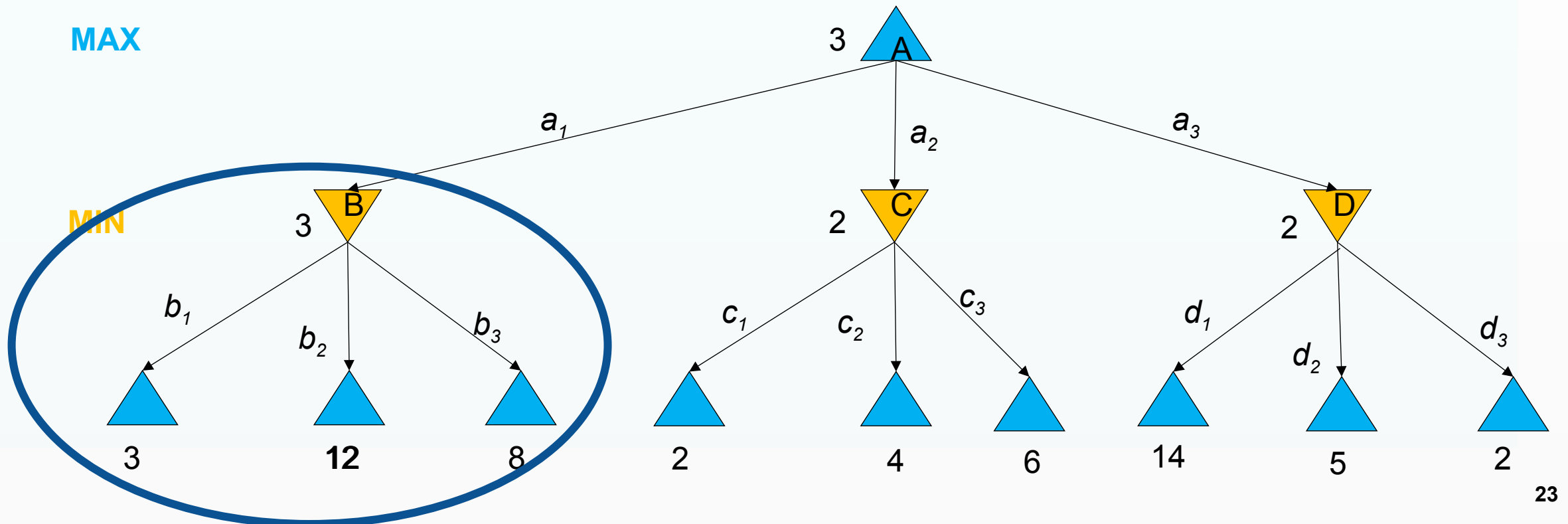
Fonctionnement

- ❑ La fonction d'UTLITE donne les valeurs des nœuds terminaux



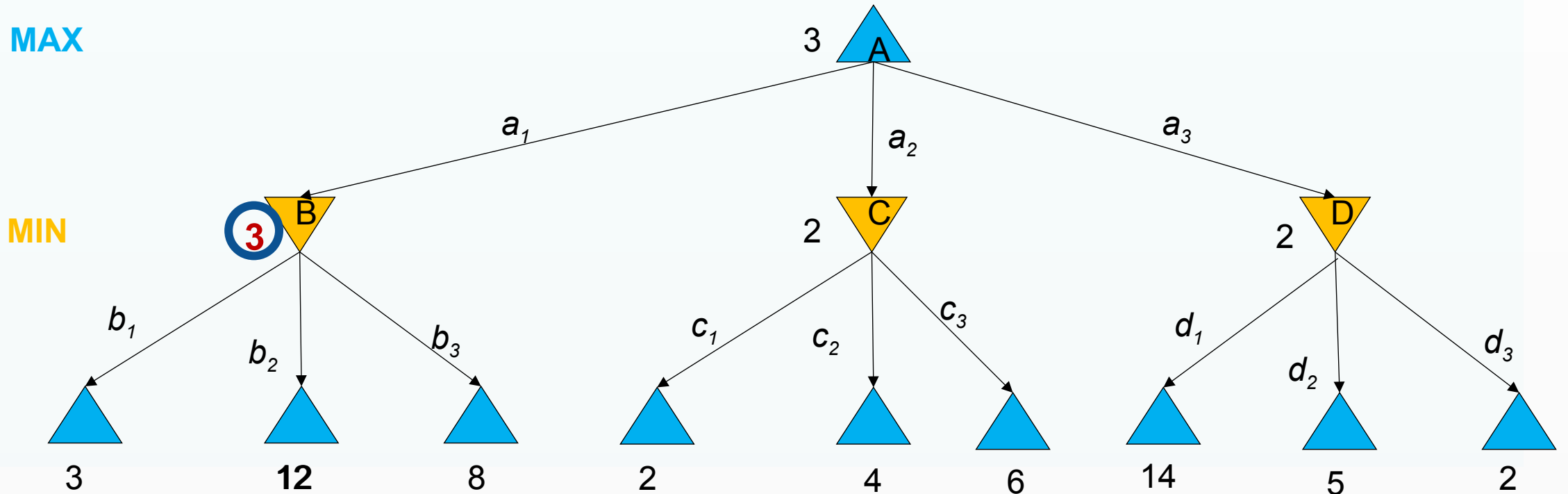
Fonctionnement

- Le premier nœud MIN, B, a trois successeurs de valeurs : 3, 12 et 8.



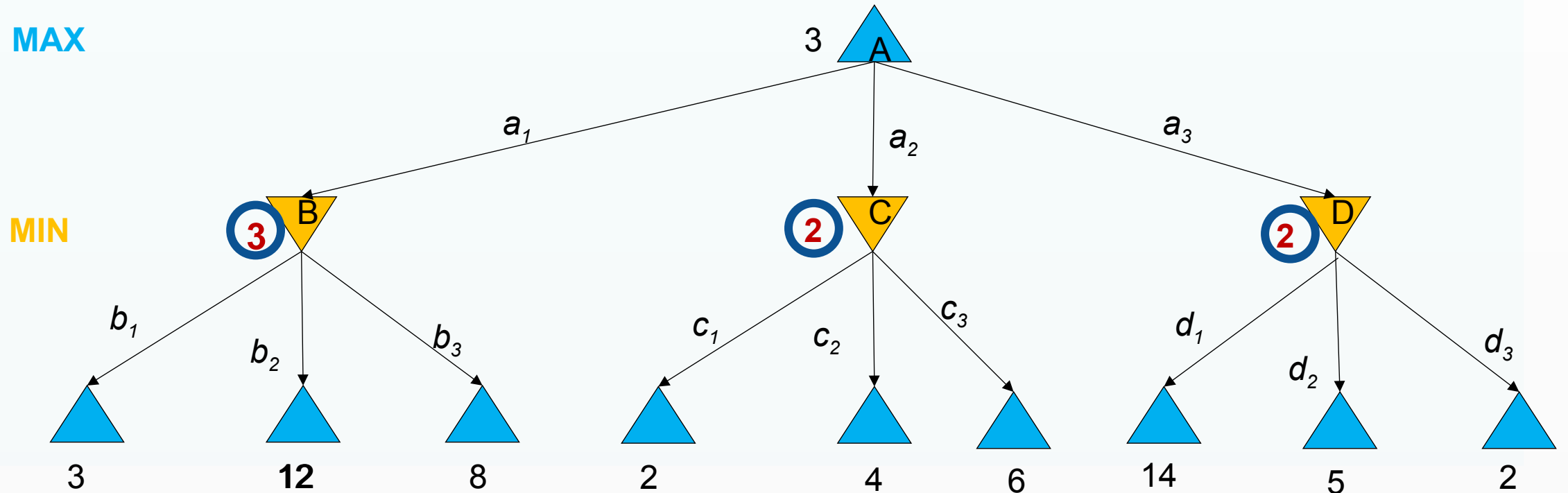
Fonctionnement

□ MIN cherche à minimiser son gain, sa valeur minimax est donc 3



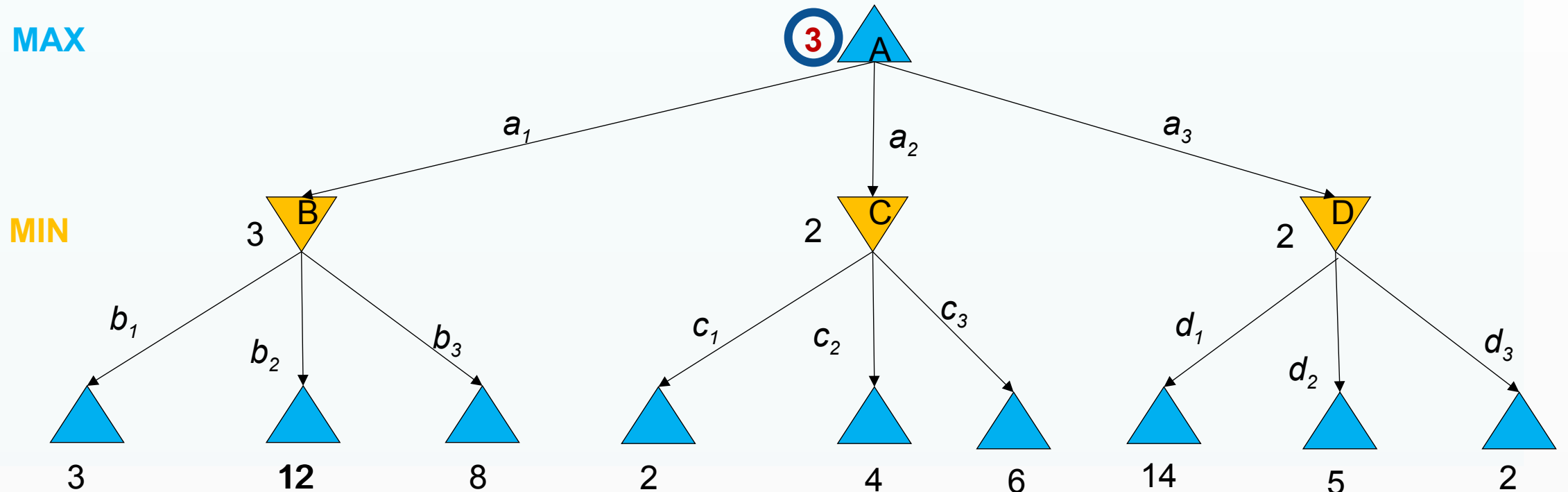
Fonctionnement

□ De même les deux autres nœuds MIN ont une valeur minimax égale à 2.



Fonctionnement

- Le nœud racine (nœud MAX) se trouve avec 3 états successeurs de valeurs minimax : 3, 2, 2, sa valeur minimax est donc 3.

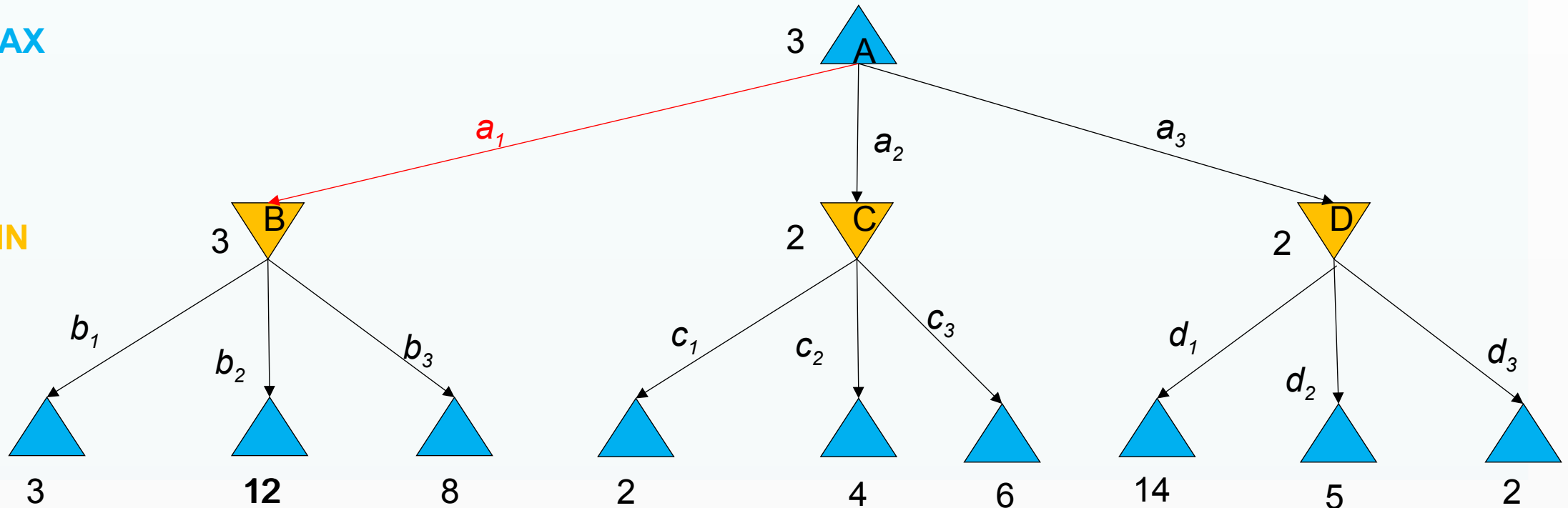


Fonctionnement

- On peut même identifier la **décision minimax** à la racine : l'action a_1 est un choix optimal pour MAX.
- Mais attention !** Le coup optimal pour MAX suppose aussi que MIN joue d'une façon optimale.

MAX

MIN



Propriétés MINIMAX

☐ Complet ?

- ☐ Oui (si l'arbre est fini)

☐ Optimal ?

- ☐ Oui (contre un adversaire qui joue d'une façon optimale)

☐ Complexité en temps ?

- ☐ $O(b^m)$:

- b : le nombre maximum d'actions/coups légales à chaque étape
- m : nombre maximum de coup dans un jeu (profondeur maximale de l'arbre).

Propriétés MINIMAX

❑ Complexité en espace ?

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

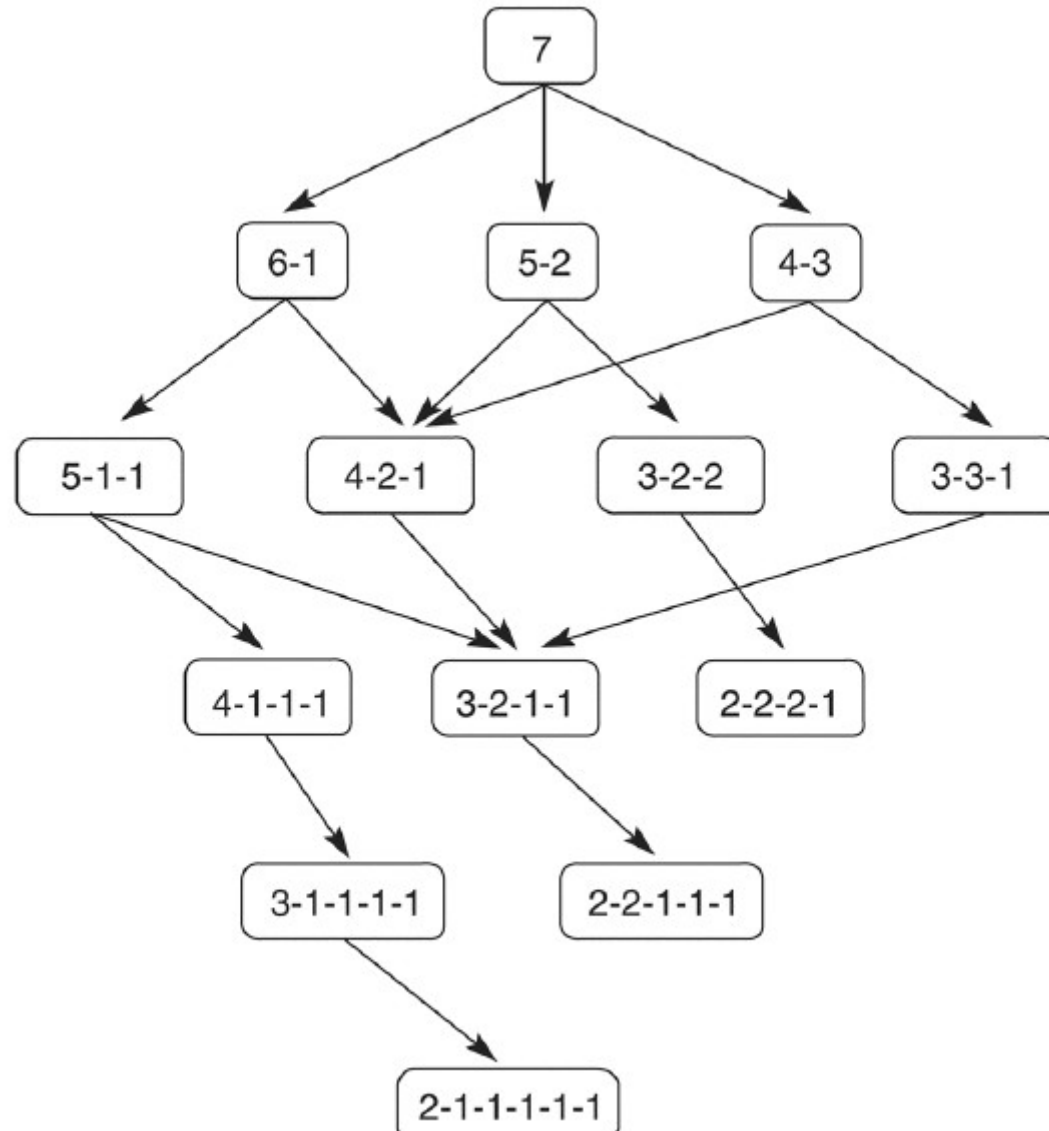
❑ Pour le jeu d'échec: $b \approx 35$ et $m \approx 100$ pour un jeu « raisonnable » :

❑ Il n'est pas réaliste d'espérer trouver une solution exacte en temps réel

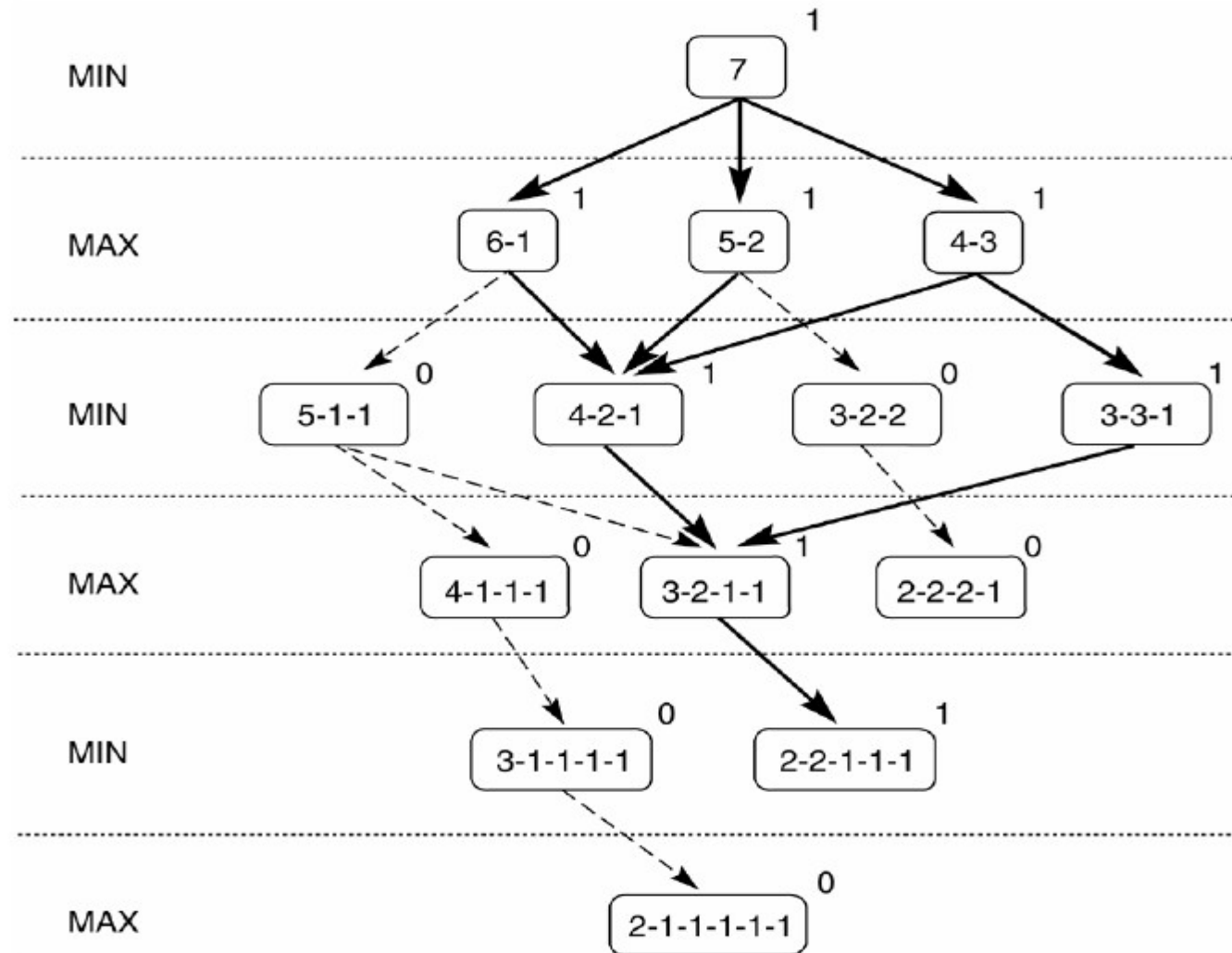
Application : variante du jeu de nim (jeu de Grundy)

- ❑ Des jetons sont places sur une table entre deux adversaires.
- ❑ Un déplacement consiste à diviser une pile de jetons en 2 piles non vides de tailles différentes.
- ❑ Par exemple, 6 jetons peuvent être divisés en de piles de 5 jetons et l'autre de 1 jetons, ou de 4 et 2, mais pas de 3 et 3.
- ❑ Le premier joueur qui ne peut plus diviser la pile a perdu.

L'espace d'état de la variante du jeu de Grundy



Minimax pour jeu de Grundy



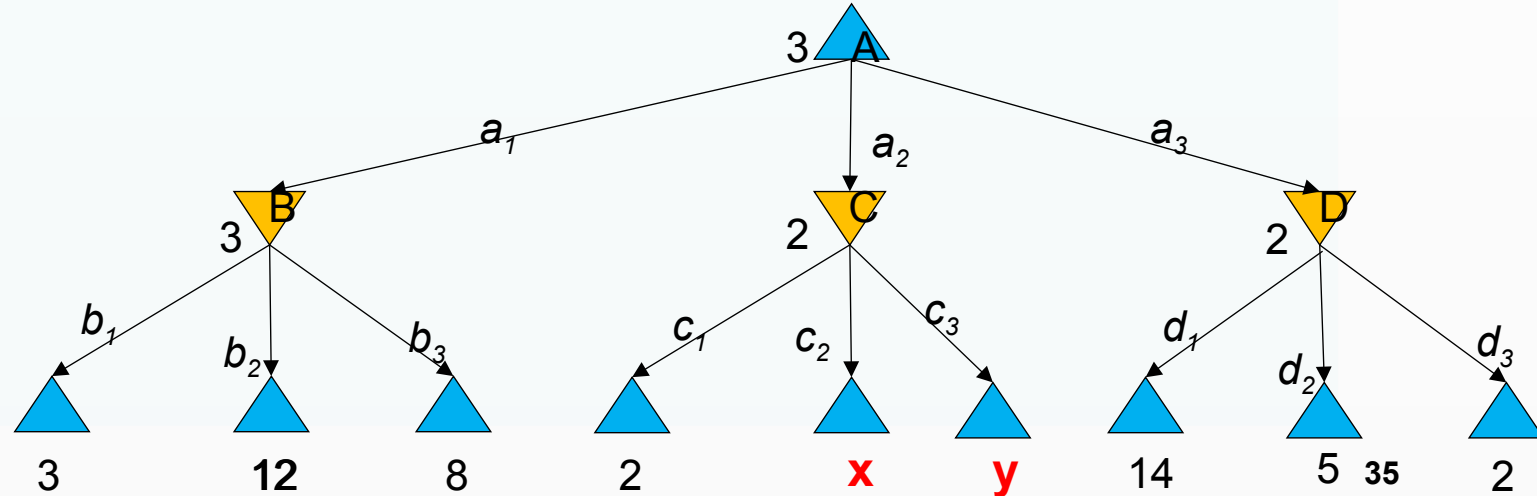
Elagage alpha-bêta (pruning)

Elagage alpha-bêta : principe

- ❑ L'élagage permet d'éviter l'exploration systématique de tout l'arbre de jeu.
- ❑ Appliqué à un arbre minimax standard, il retourne le même coup renvoyé par minimax, mais en élaguant des branches qui n'influencent pas la décision finale.

Elagage alpha-bêta : principe

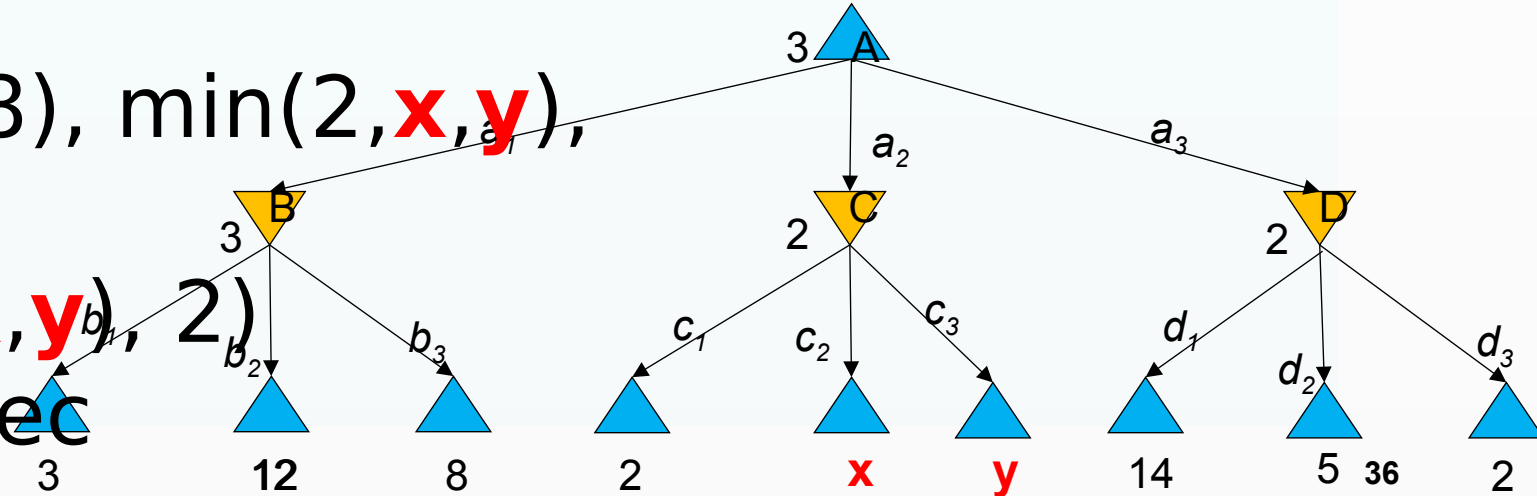
- Supposons que deux successeurs du nœud C ne seront pas évalués, on leur attribue les valeurs x et y .



Elagage alpha-bêta : principe

- Supposons que deux successeurs du nœud C ne seront pas évalués, on leur attribut les valeurs x et y.

$$\begin{aligned}
 \text{MINIMAX}(\text{racine}) &= \max(\min(3, 12, 8), \min(2, \mathbf{x}, \mathbf{y}), \min(14, 5, 2)) \\
 &= \max(3, \min(2, \mathbf{x}, \mathbf{y}), 2) \\
 &= \max(3, \mathbf{z}, 2) \text{ avec } \mathbf{z} = \min(2, \mathbf{x}, \mathbf{y})
 \end{aligned}$$



Elagage alpha-bêta : principe

- Supposons que deux successeurs du nœud C ne seront pas évalués, on leur attribut les valeurs x et y .
- Les valeurs de la racine et donc de minimax sont indépendantes des valeurs des feuilles x et y , et donc ces dernières peuvent être élaguées.

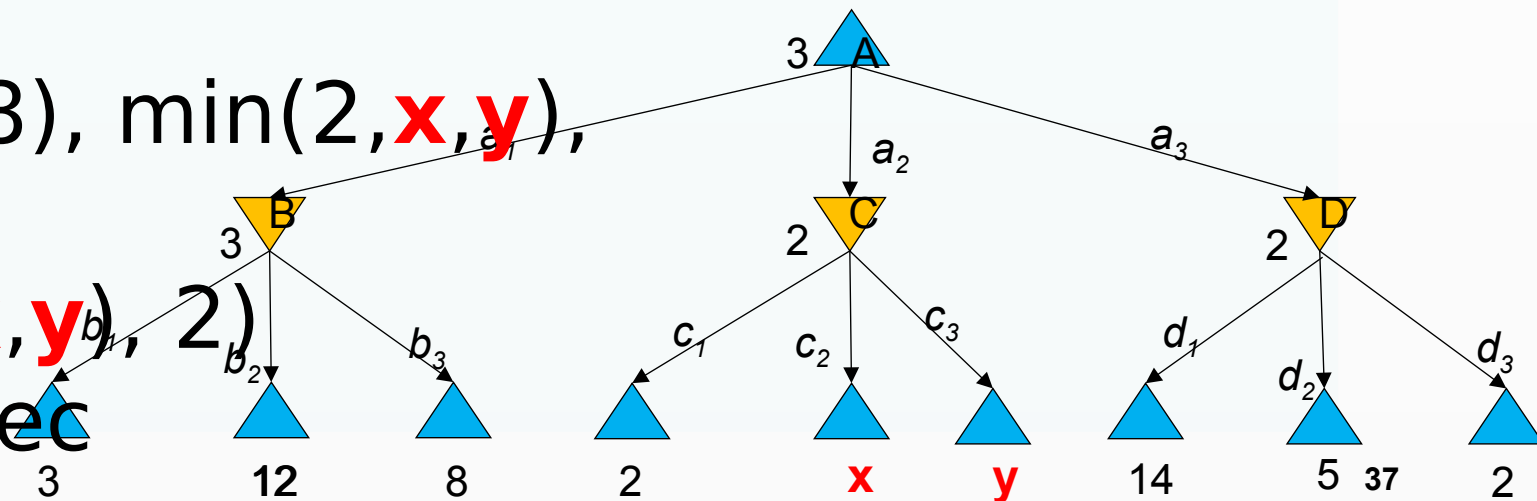
MINIMAX(*racine*)

$$= \max(\min(3, 12, 8), \min(2, \mathbf{x}, \mathbf{y}), \min(14, 5, 2))$$

$$= \max(3, \min(2, \mathbf{x}, \mathbf{y}), 2)$$

$$= \max(3, \mathbf{z}, 2) \text{ avec}$$

$$\mathbf{z} = \min(2, \mathbf{x}, \mathbf{y}) \neq 2$$



Elagage alpha-bêta : principe

- Supposons que deux successeurs du nœud C ne seront pas évalués, on leur attribue les valeurs x et y .
- Les valeurs de la racine et donc de minimax sont indépendantes des valeurs des feuilles x et y , et donc ces dernières peuvent être élaguées.

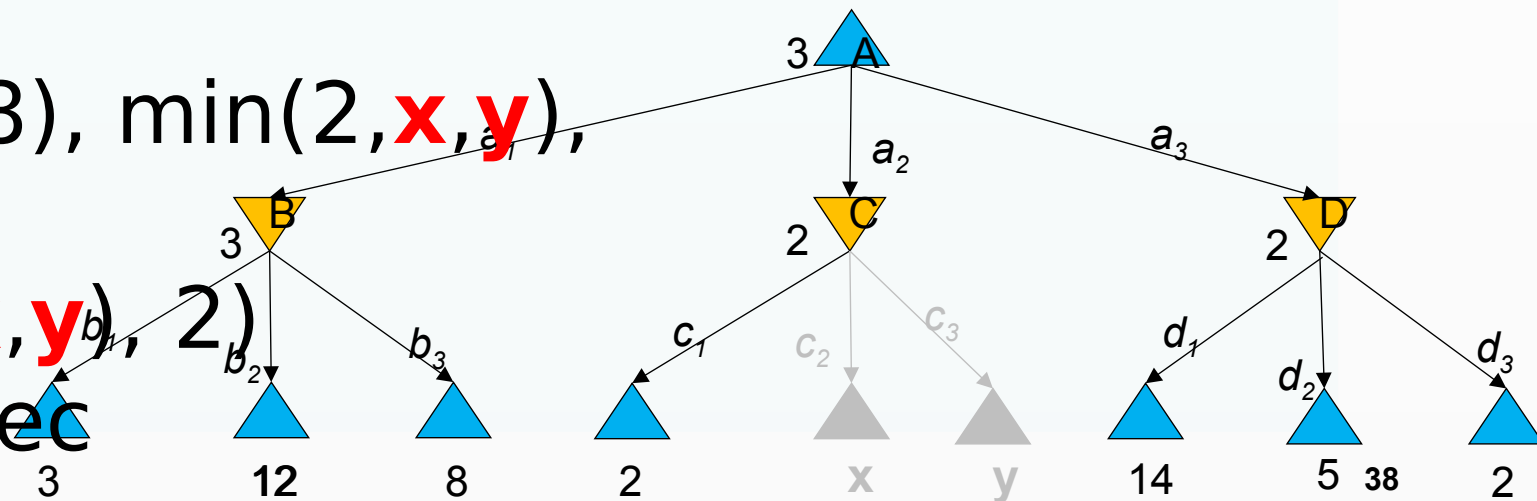
MINIMAX(*racine*)

$$= \max(\min(3, 12, 8), \min(2, \mathbf{x}, \mathbf{y}), \min(14, 5, 2))$$

$$= \max(3, \min(2, \mathbf{x}, \mathbf{y}), 2)$$

$$= \max(3, \mathbf{z}, 2) \text{ avec}$$

$$z = \min(2, x, y) \neq 2$$



Elagage alpha-bêta : quand couper ?

- L'élagage alpha-bêta tire son nom de deux bornes qui sont remontées tout au long du chemin :
- γ_x : valeur du meilleur choix (valeur la plus élevée) trouvée jusqu'ici pour les choix effectués le long du chemin par MAX.
- β_x : valeur du meilleur choix (valeur la plus basse) trouvée jusqu'ici pour les choix effectués le long du chemin par MIN.

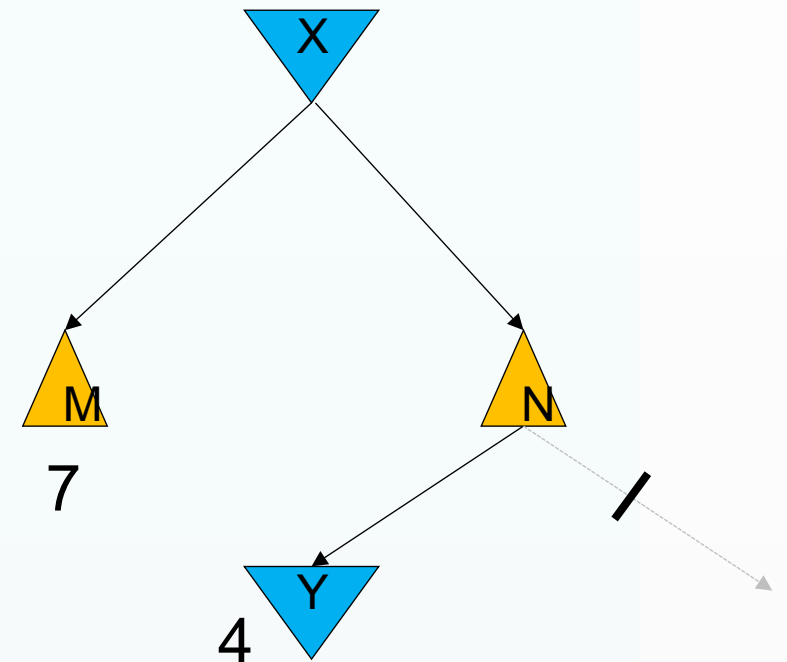
Elagage alpha-bêta : couper dans un nœud MIN

- γ_X est la valeur du meilleur choix pour Max (c.-à-d., plus grande valeur) trouvé jusqu'ici:
- Si on est **dans un nœud Min** et que sa valeur **v devient inférieure α** (donc « pire que γ_X » du point de vue de Max), il faut arrêter la recherche (couper la branche).
- La valeur v du nœud $N = 4$ et $\gamma_X = 7$ (meilleur nœud fils de X), donc $v < \gamma_X$: coupure.

MAX

MIN

MAX



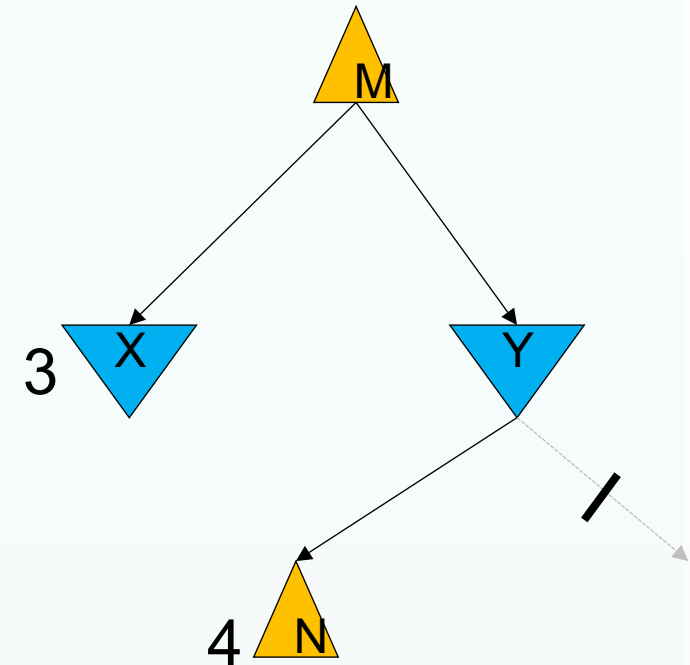
Elagage alpha-bêta : couper dans un nœud MAX

- β est la valeur du meilleur choix pour Min (c.-à-d., plus petite valeur) trouvé jusqu'ici:
- Si on est **dans un nœud Max** et que sa valeur **devient supérieur à β** (donc « pire que β » du point de vue de Max), il faut arrêter la recherche (couper la branche).
- La valeur v du nœud $Y = 4$ et $\beta = 3$ (meilleur nœud fils de M), donc $v \geq \beta$: coupure.

MIN

MAX

MIN



Algorithme alpha-bêta

fonction EXPLORATION-ALPHA-BÊTA(état) **retourne** une action

VALEUR-MAX(état,)

retourner l'action dans ACTIONS(état) avec la valeur

fonction VALEUR-MAX(état, α , β) **retourne** une valeur d'utilité

si TEST-TERMINAL(état) **alors retourner** UTILITÉ(état)

pour chaque a **dans** ACTIONS(état) **faire**
MAX(, VALEUR-MIN(RÉSULTAT(s, a), α , β))

si alors retourner

MAX(,)

retourner

fonction VALEUR-MIN(état, α , β) **retourne** une valeur d'utilité

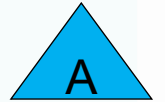
si TEST-TERMINAL (état) **alors retourner** UTILITÉ(état)

pour chaque a **dans** ACTIONS(état) **faire**
MIN(, VALEUR-MAX(RÉSULTAT(s, a), α , β))

Elagage alpha-bêta : application de l'algorithme

- Appel de la fonction `EXPLORATION-ALPHA-BETA(A)`.
- Elle retourne la meilleure transition pour `A` afin de maximiser son utilité (v).
- L'exploration minimax est de type DFS (profondeur d'abord).

EXPLORATION-ALPHA-BETA(A)



Elagage alpha-bêta : application de l'algorithme

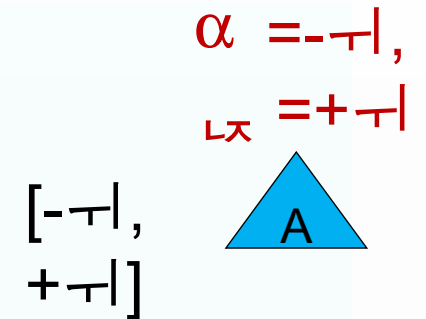
- Pour calculer cette utilité v , la fonction EXPLORATION-ALPHA-BETA(A) appelle la fonction VALEUR-MAX avec les valeurs de α , β et A .

- Initialement :

- $\alpha = -\infty$

- $\beta = +\infty$

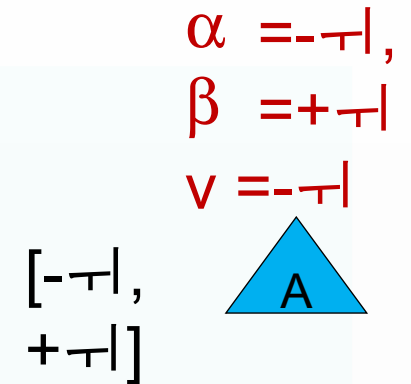
- Entre $[\alpha, \beta]$: l'intervalle des valeurs possibles des nœuds visités. C'est une information indicative et ne fait pas partie de l'algorithme lui-même.



Appel de la fonction VALEUR-MAX avec les valeurs de α , β et A : $VALEUR-MAX(-\infty, +\infty, A)$. Elle doit retourner la meilleure valeur d'utilité v pour A .

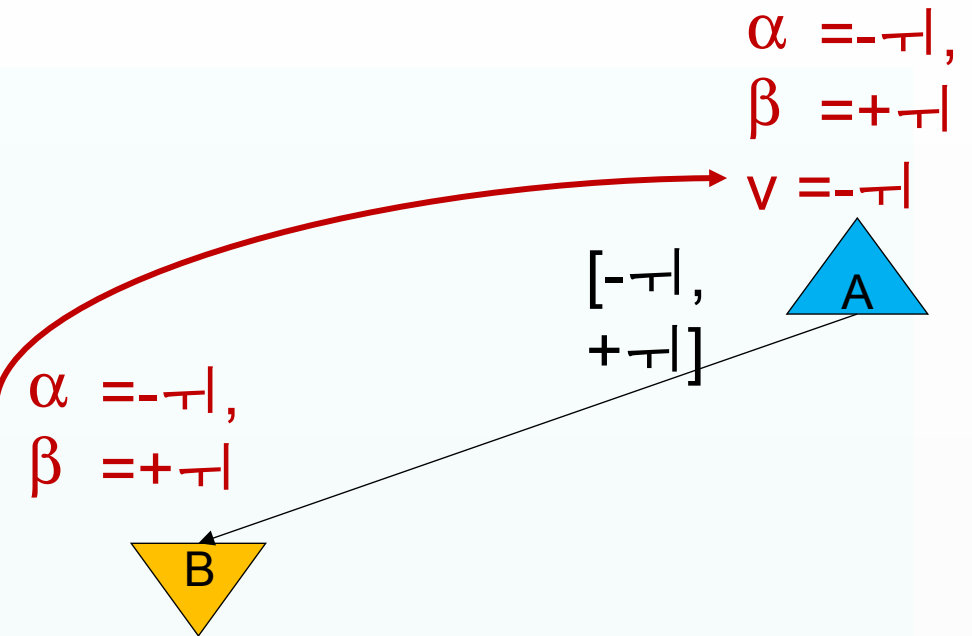
Elagage alpha-bêta : application de l'algorithme

- ❑ Comme A n'est pas un nœud terminal, il ne retourne pas la valeur d'utilité v de l'état et propage l'appel récursif pour ses successeurs.
- ❑ Initialisation (dans la fonction VALEUR-MAX appelée pour A) de v à $-\tau$.
- ❑ C'est cette valeur de v qui sera remontée récursivement comme minimax du nœud racine A



Elagage alpha-bêta : application de l'algorithme

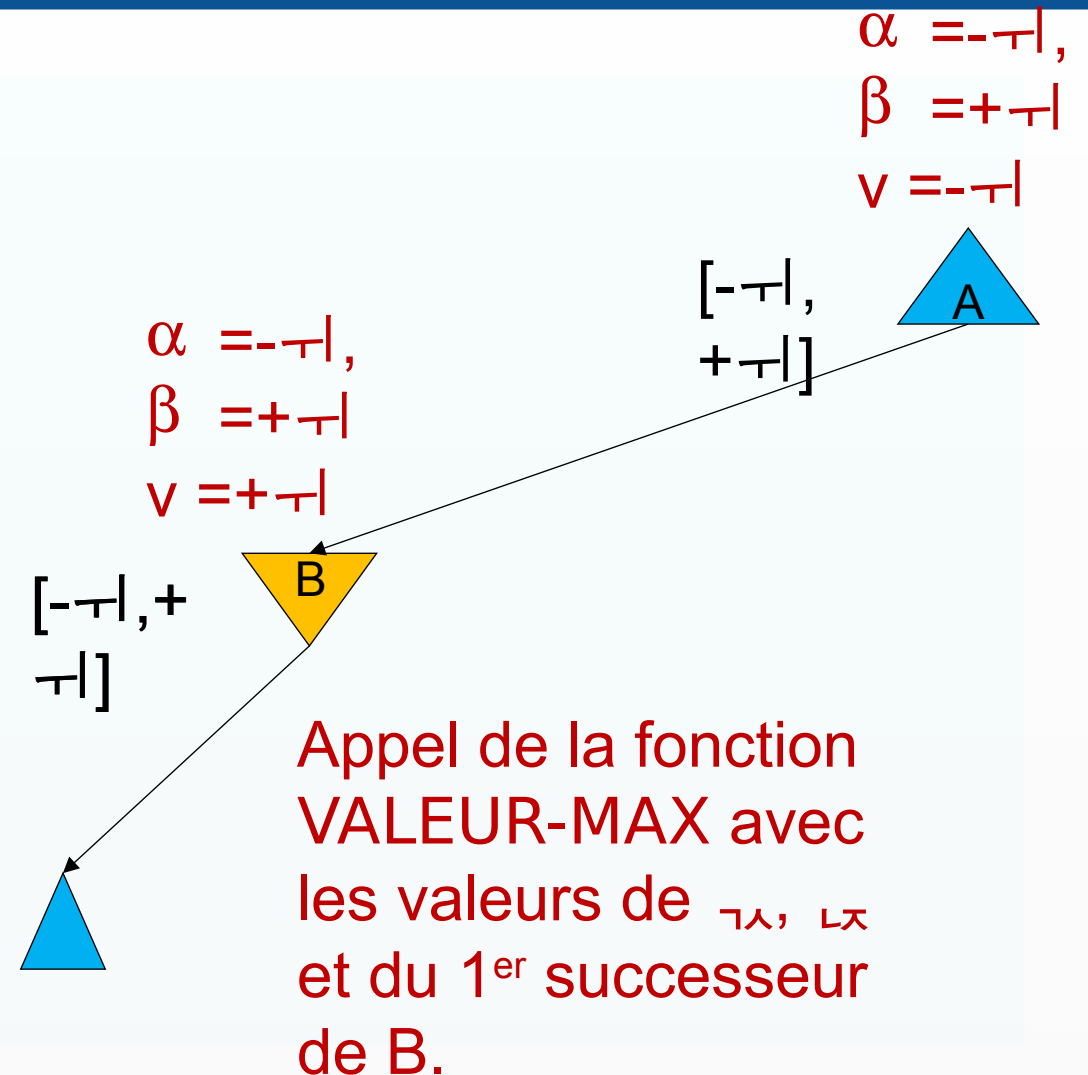
- La fonction VLAEUR-MAX appelée avec l'état A appelle donc la fonction VALEUR-MIN pour tous les fils de ce nœud, pour garder à la fin la meilleure valeur minimax trouvée dans les successeurs.
- C'est l'appel $v = \text{MAX}(v, \text{VALEUR-MIN}(\text{RESULTAT}(B, a), \neg\lambda, \neg\lambda)) = \text{MAX}(-\tau, \text{VALEUR-MIN}(\text{RESULTAT}(B, a), \neg\lambda, \neg\lambda))$ qui permet de garder en tout temps la meilleure valeur minimax pour A parmi ses fils.



Appel de la fonction VALEUR-MIN avec les valeurs de $\neg\lambda$, $\neg\lambda$ et du 1^{er} successeur B de A : $\text{VALEUR-MIN}(B, -\tau, +\tau)$. Elle doit retourner la meilleure valeur d'utilité v pour B.

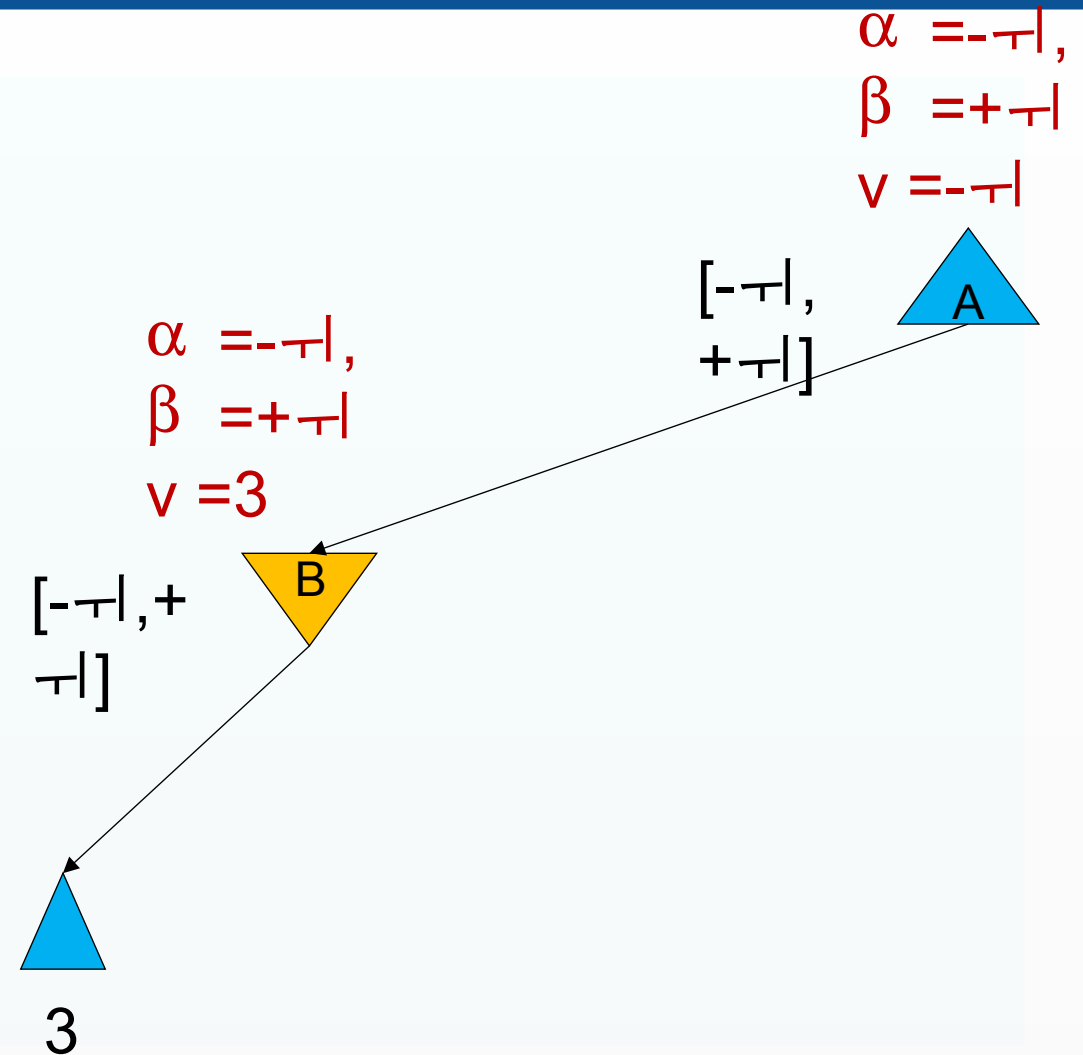
Elagage alpha-bêta : application de l'algorithme

- ❑ La fonction VLAEUR-MIN appelée avec l'état B ne retourne pas sa fonction d'utilité puisque B n'est pas un état terminal.
- ❑ Initialisation (dans le fonction VALEUR-MIN appelée pour B) de v à $+\infty$.
- ❑ Appelle donc la fonction VALEUR-MAX pour tous les fils de ce nœud B, pour garder à la fin la meilleure valeur minimax trouvée dans les successeurs.
- ❑ C'est l'appel $v = \text{MIN}(v, \text{VALEUR-MAX}(\text{RESULTAT}(s,a), \gamma, \lambda))$ qui permet de garder en tout temps la meilleure valeur minimax pour B parmi ses fils.



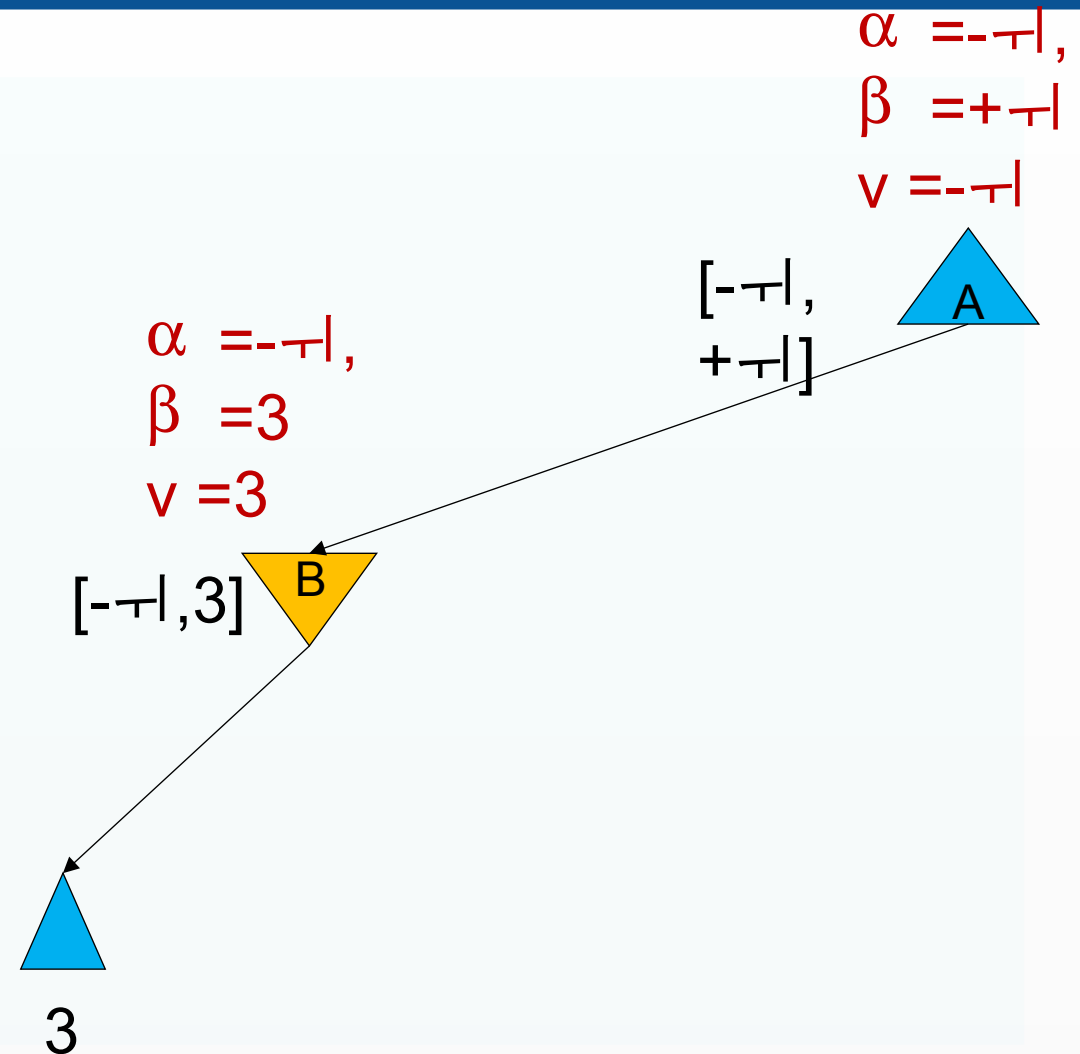
Elagage alpha-bêta : application de l'algorithme

- ❑ La première feuille sous B est un nœud terminal donc l'appel VALEUR-MAX(fils de B, $-\tau|, +\tau|$) pour le 1^{er} fils de B retourne la valeur d'utilité de l'état soit 3 et l'appel récursif ne se poursuit pas pour ce nœud (terminal).
- ❑ Cette première feuille sous B a la valeur d'utilité 3, l'appel $\text{MIN}(v, \text{VALEUR-MAX}(\text{RESULTAT}(s, a), \tau\lambda, \tau\lambda))$ qui n'est autre que $\text{MIN}(+\tau|, 3)=3$ et donc v aussi.
- ❑ Comme $v > \tau\lambda$, v ne sera pas retournée comme valeur d'utilité de B.



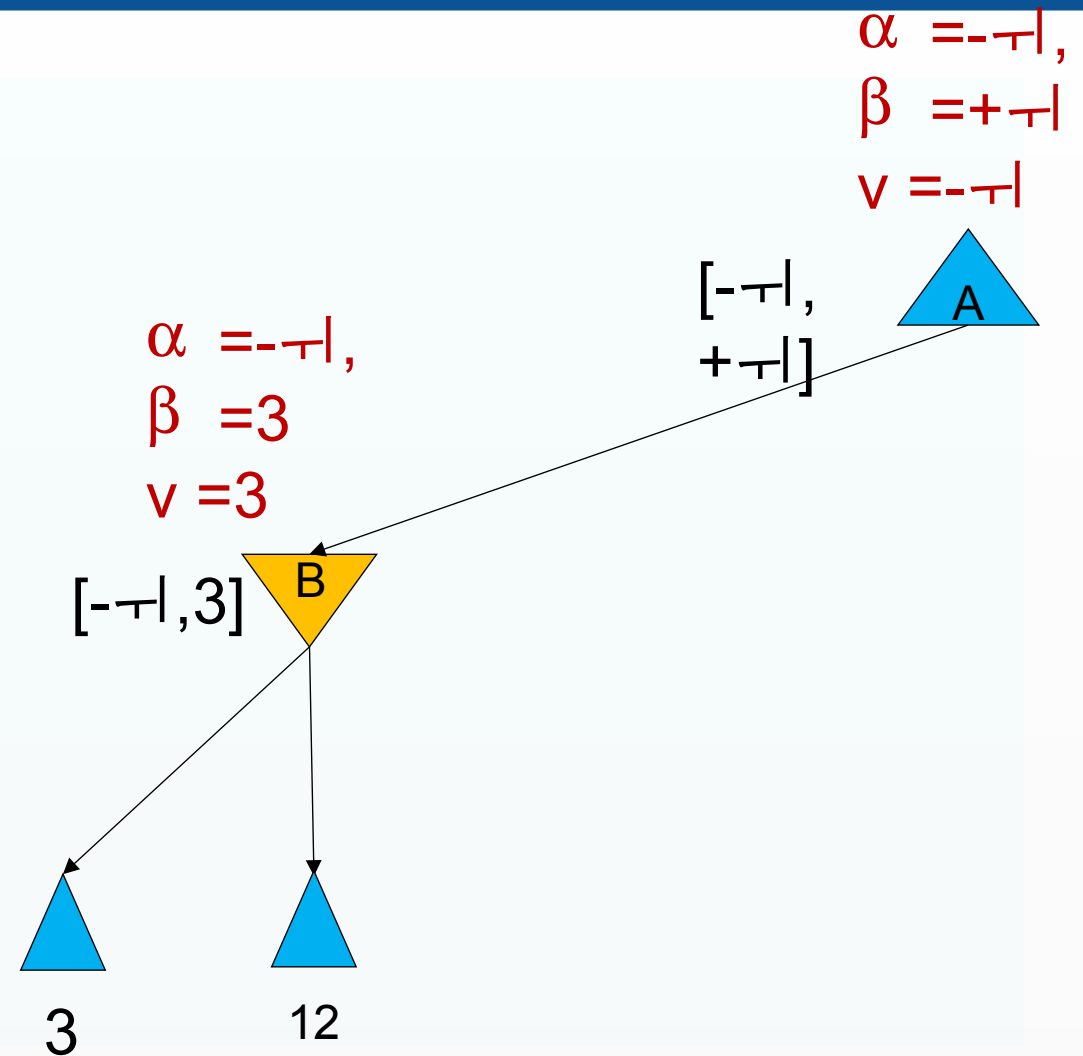
Elagage alpha-bêta : application de l'algorithme

- α doit recevoir maintenant $\text{MIN}(\alpha, v) = \text{MIN}(+\infty, 3) = 3$.
- C'est cette actualisation de la valeur de α et le test $v > \beta$ qui font la différence avec l'algorithme minimax et permettent de savoir en tout temps si une branche sera élaguée.



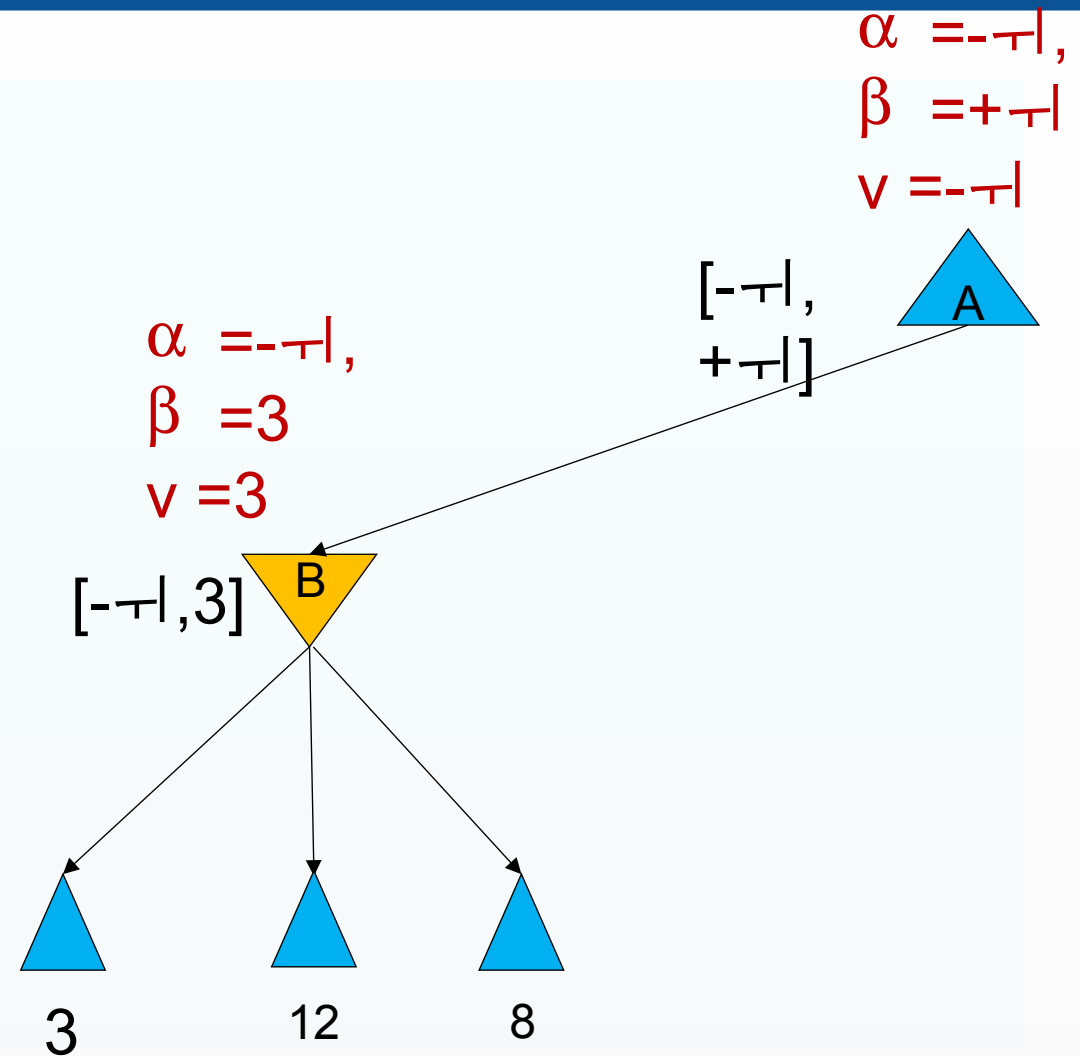
Elagage alpha-bêta : application de l'algorithme

- La 2^{ème} feuille sous B est un nœud terminal donc l'appel VALEUR-MAX(fils de B, $-\tau$, 3) pour le 2^{ème} fils de B retourne la valeur d'utilité de l'état soit 12 et l'appel récursif ne se poursuit pas pour ce nœud (terminal).
- L'appel $\text{MIN}(v, \text{VALEUR-MAX}(\text{RESULTAT}(s, a), \tau, \lambda))$ qui n'est autre que $\text{MIN}(3, 12) = 3$ et donc v aussi.
- MIN évite donc ce coup et la valeur de B maximale demeure 3.
- Comme $v > \tau$, v ne sera pas retournée comme valeur d'utilité de B.
- λ doit recevoir maintenant $\text{MIN}(\lambda, v) = \text{MIN}(3, 3) = 3$.



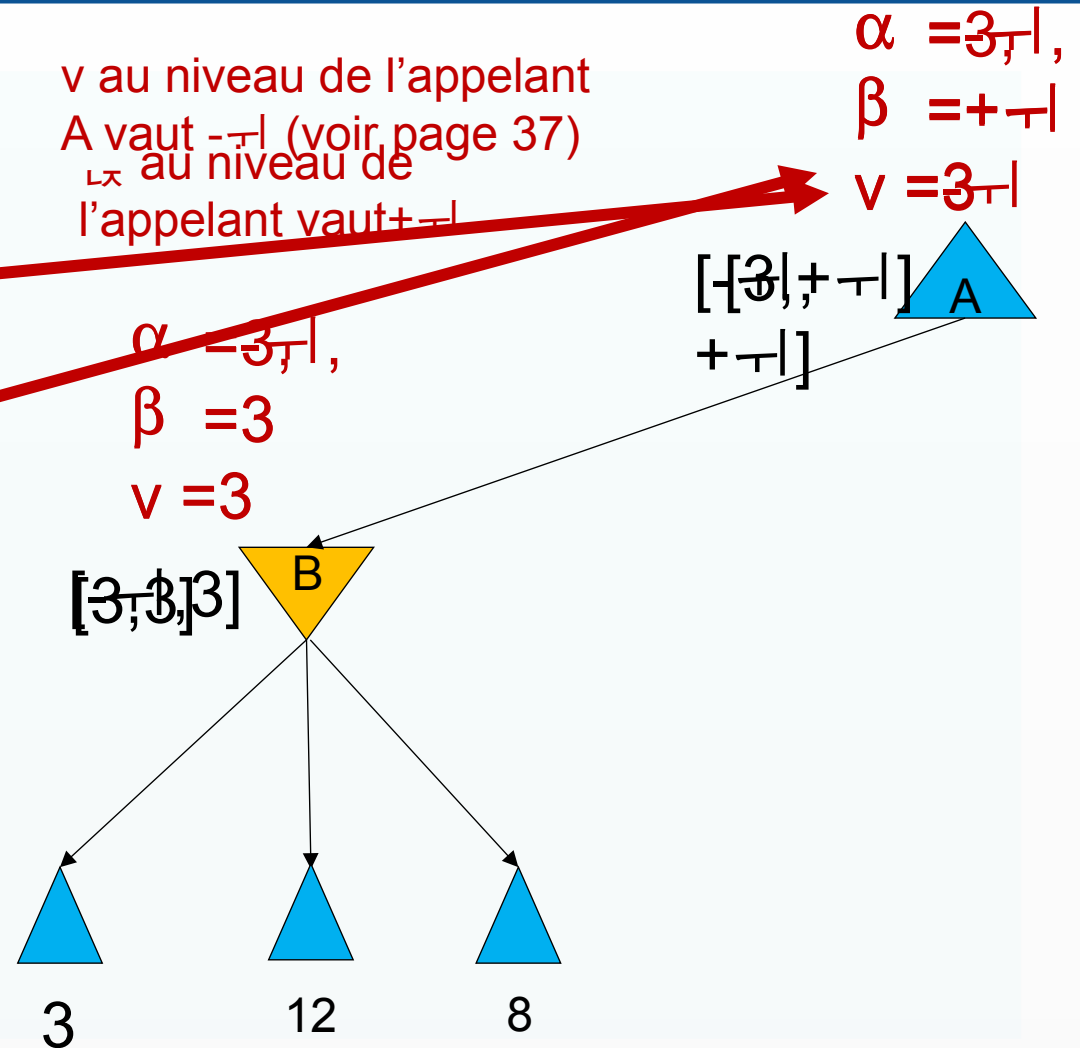
Elagage alpha-bêta : application de l'algorithme

- La 3^{ème} feuille sous B est un nœud terminal donc l'appel VALEUR-MAX(fils de B, $-\tau$, 3) pour le 3^{ème} fils de B retourne la valeur d'utilité de l'état soit 8 et l'appel récursif ne se poursuit pas pour ce nœud (terminal).
- L'appel $\text{MIN}(v, \text{VALEUR-MAX}(\text{RESULTAT}(s,a), \tau, \lambda))$ qui n'est autre que $\text{MIN}(3, 8)=3$ et donc v aussi.
- MIN évite donc ce coup et la valeur de B maximale demeure 3.
- Comme $v > \tau$, v ne sera pas retournée comme valeur d'utilité de B.
- λ doit recevoir maintenant $\text{MIN}(\lambda, v) = \text{MIN}(3, 3) = 3$.



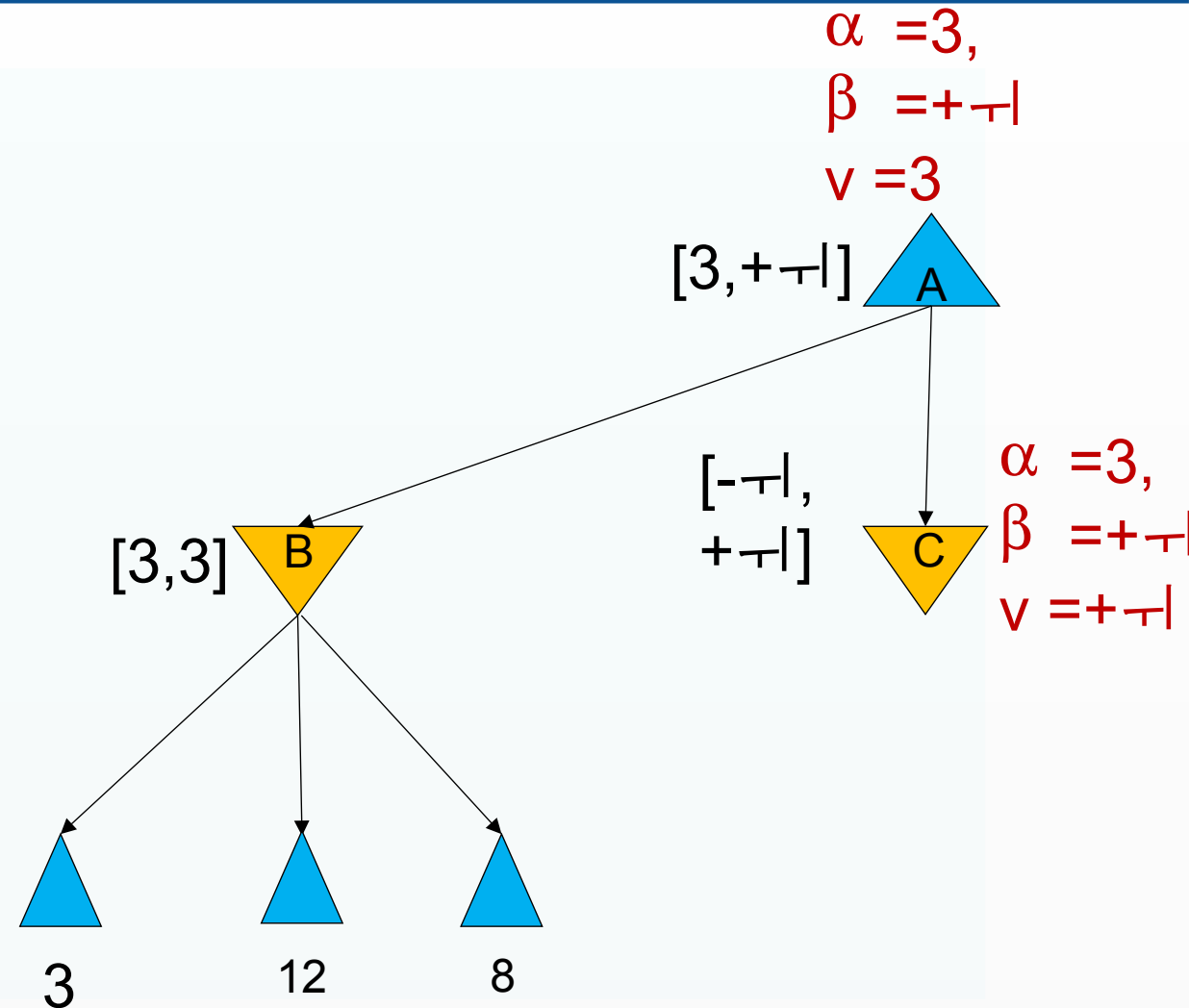
Elagage alpha-bêta : application de l'algorithme

- Tous les successeurs de B ont été visités.
- La valeur d'utilité $v=3$ sera donc retournée à l'appelant : $\text{VALEUR-MIN}(\text{RESULTAT}(B,a), \alpha, \beta)$.
- La fonction $\text{MAX}(v, \text{VALEUR-MIN}(\text{RESULTAT}(B,a), \alpha, \beta)) = \text{MAX}(-\infty, 3) = 3$ est donc appelée à fin de calculer la meilleure valeur d'utilité pour le 1^{er} coup de A : B.
- Affection $v = \text{MAX}(-\infty, 3) = 3$.
- $v = 3 < \beta = +\infty$, pas d'élagage (v ne sera pas retournée comme valeur d'utilité).
- $\alpha = \text{MAX}(\alpha, v) = \text{MAX}(-\infty, 3) = 3$.
- On peut même inférer que la valeur de la racine est au minimum 3.



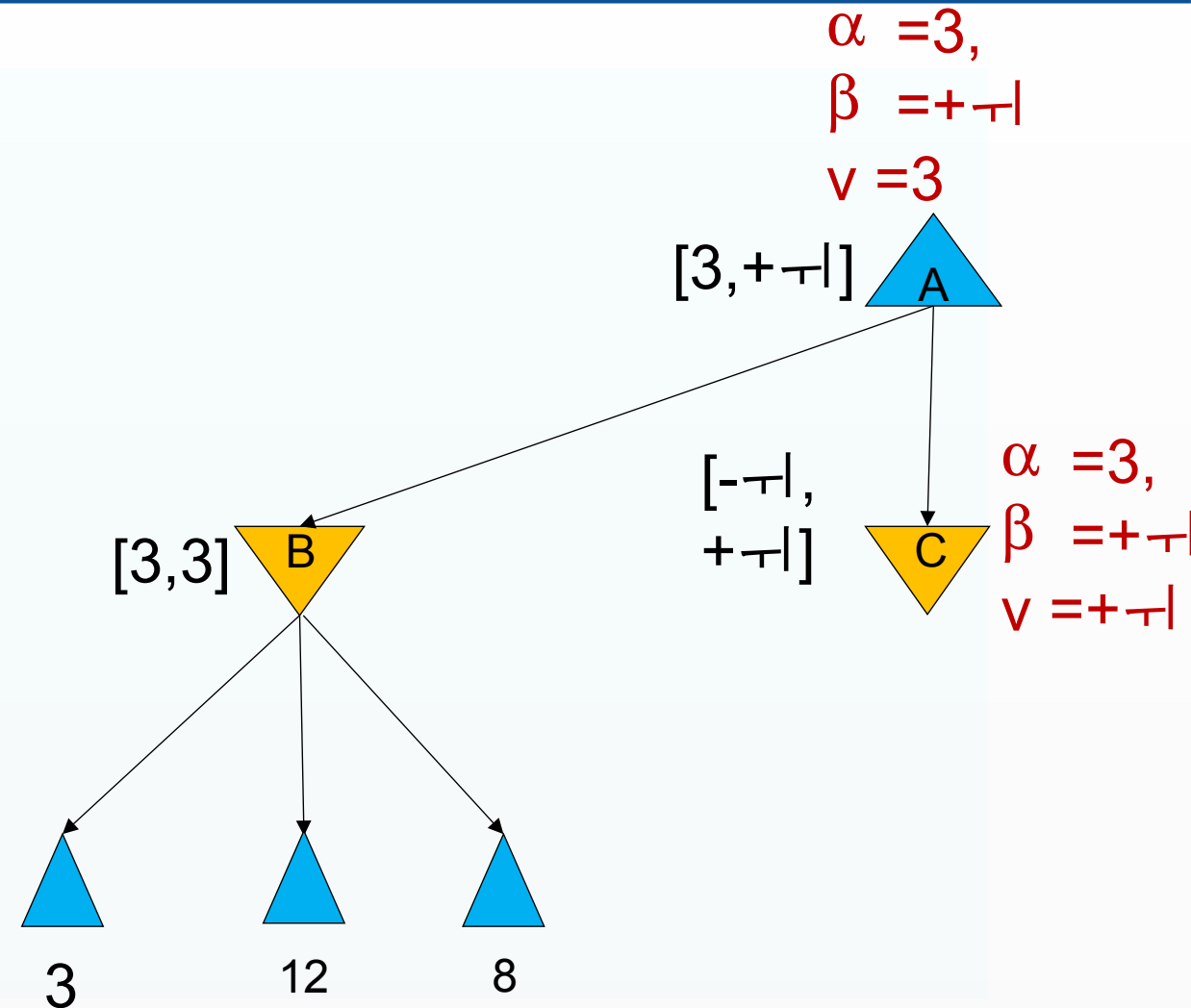
Elagage alpha-bêta : application de l'algorithme

- ❑ Maintenant c'est au tour du 2^{ème} fils de A, C, qui sera traité :
 $VLAEUR-MIN(C, \tau, \tau) = VLAEUR-MIN(C, 3, +\tau)$.
- ❑ $VLAEUR-MIN(C, 3, +\tau)$ ne retourne pas sa fonction d'utilité puisque C n'est pas un état terminal.
- ❑ Initialisation (dans la fonction VALEUR-MIN appelée pour C) de v à $+\tau$.



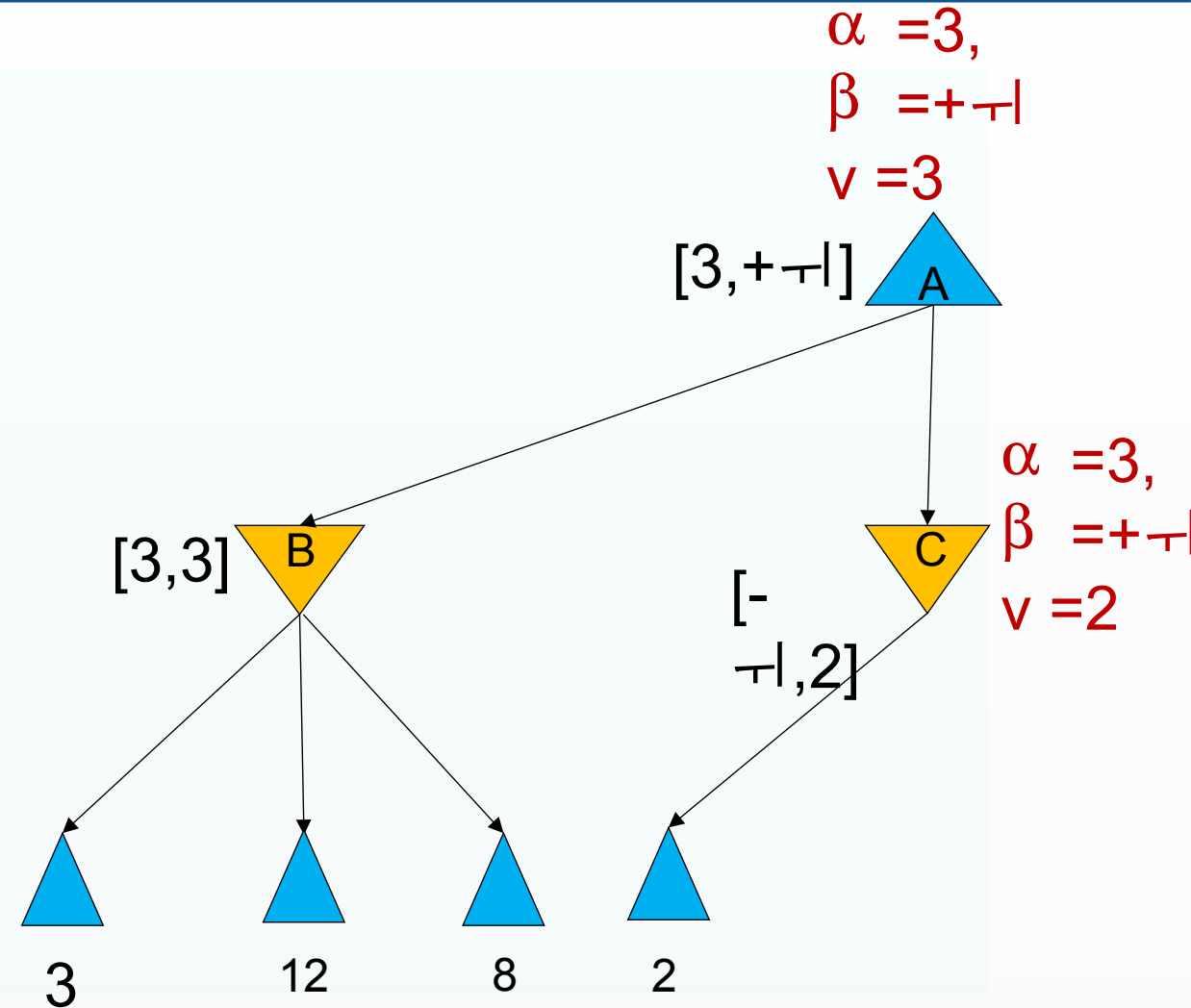
Elagage alpha-bêta : application de l'algorithme

- Appelle la fonction VALEUR-MAX pour tous les fils de ce nœud C, pour garder à la fin la meilleure valeur minimax trouvée dans les successeurs.
- C'est l'appel $v = \text{MIN}(v, \text{VALEUR-MAX}(\text{RESULTAT}(s, a), \tau, \perp))$ qui permet de garder en tout temps la meilleure valeur minimax pour C parmi ses fils.



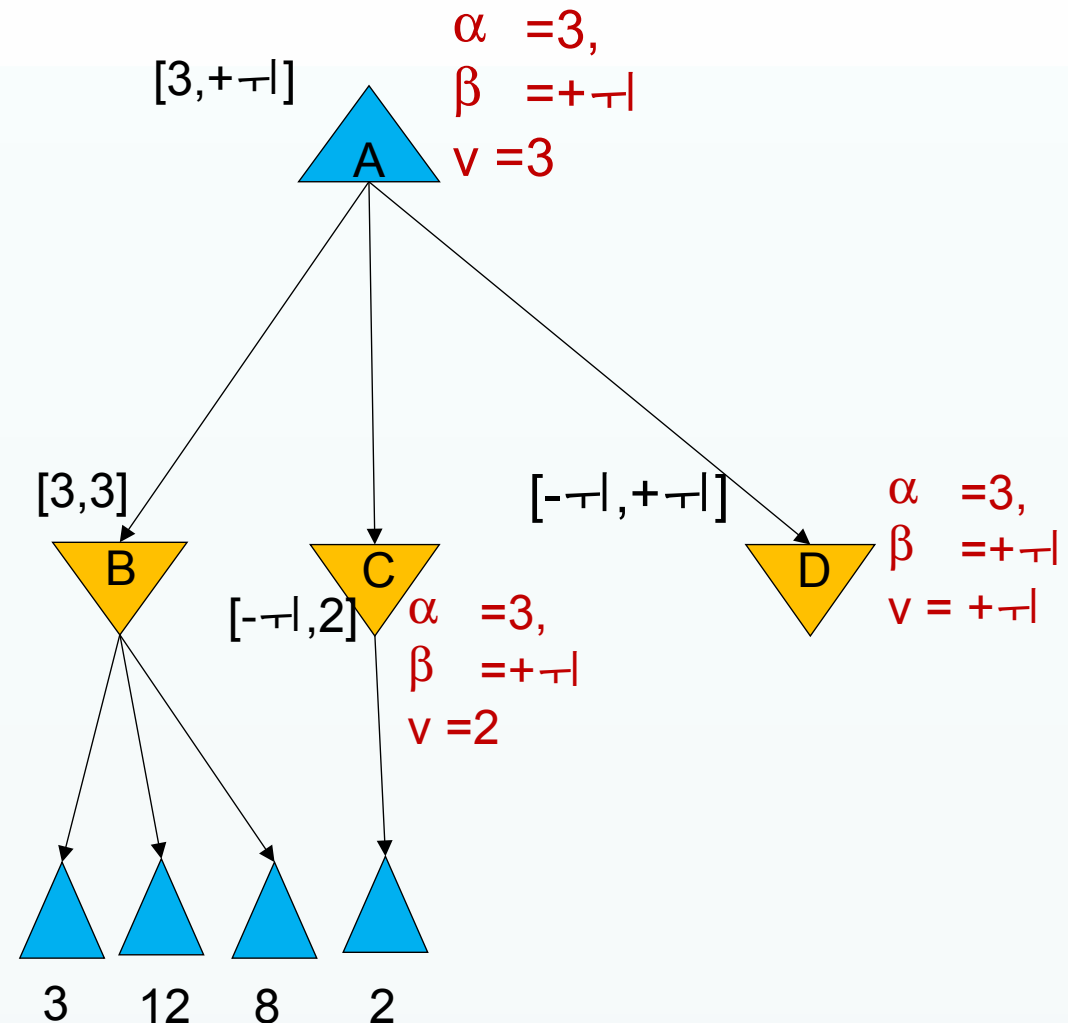
Elagage alpha-bêta : application de l'algorithme

- La première feuille sous C est un nœud terminal donc l'appel VALEUR-MAX(fils de C, 3, $+\infty$) pour le 1^{er} fils de B retourne la valeur d'utilité de l'état soit 2 et l'appel récursif ne se poursuit pas pour ce nœud (terminal).
- Cette première feuille sous C a la valeur d'utilité 2, l'appel $\text{MIN}(v, \text{VALEUR-MAX}(\text{RESULTAT}(s,a), \alpha, \beta))$ qui n'est autre que $\text{MIN}(+\infty, 2)=2$ et donc v aussi.
- Comme $v \geq \alpha : 2 \geq 3$, la valeur v sera retournée comme valeur d'utilité de C et les autres nœuds fils de C ne seront pas explorés : c'est l'élagage.
- En effet, Comme B vaut 3, MAX préférera ce nœud là au nœud C : C ne sera jamais choisi : $v \geq \alpha$.



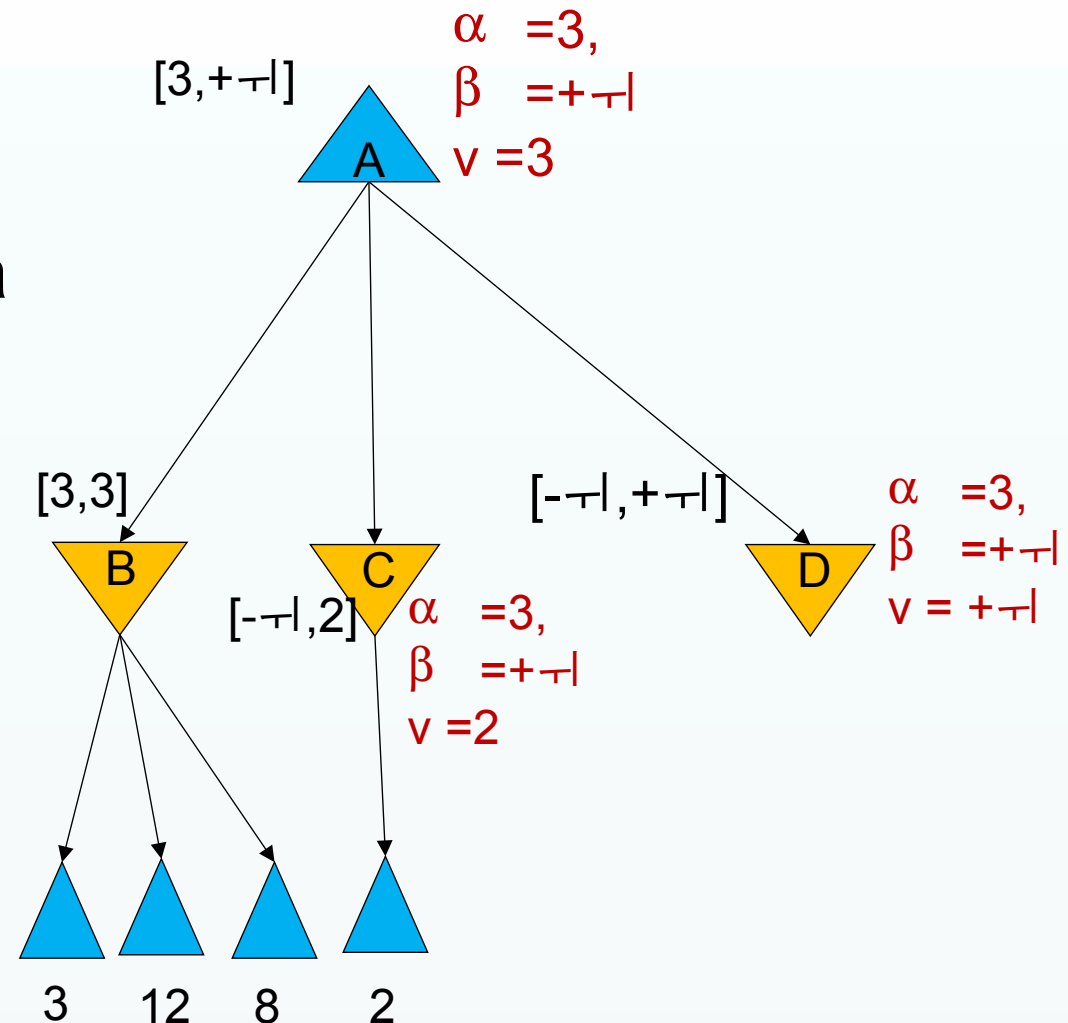
Elagage alpha-bêta : application de l'algorithme

- ❑ Maintenant c'est au tour du 3^{ème} fils de A, D, qui sera traité :
 $VLAEUR-MIN(D, \tau, \tau) = VLAEUR-MIN(C, 3, +\tau)$.
- ❑ $VLAEUR-MIN(C, 3, +\tau)$ ne retourne pas sa fonction d'utilité puisque C n'est pas un état terminal.
- ❑ Initialisation (dans la fonction VALEUR-MIN appelée pour D) de v à $+\tau$.



Elagage alpha-bêta : application de l'algorithme

- Appelle la fonction VALEUR-MAX pour tous les fils de ce nœud D, pour garder à la fin la meilleure valeur minimax trouvée dans les successeurs.
- C'est l'appel $v = \text{MIN}(v, \text{VALEUR-MAX}(\text{RESULTAT}(s, a), \tau, \perp))$ qui permet de garder en tout temps la meilleure valeur minimax pour D parmi ses fils.

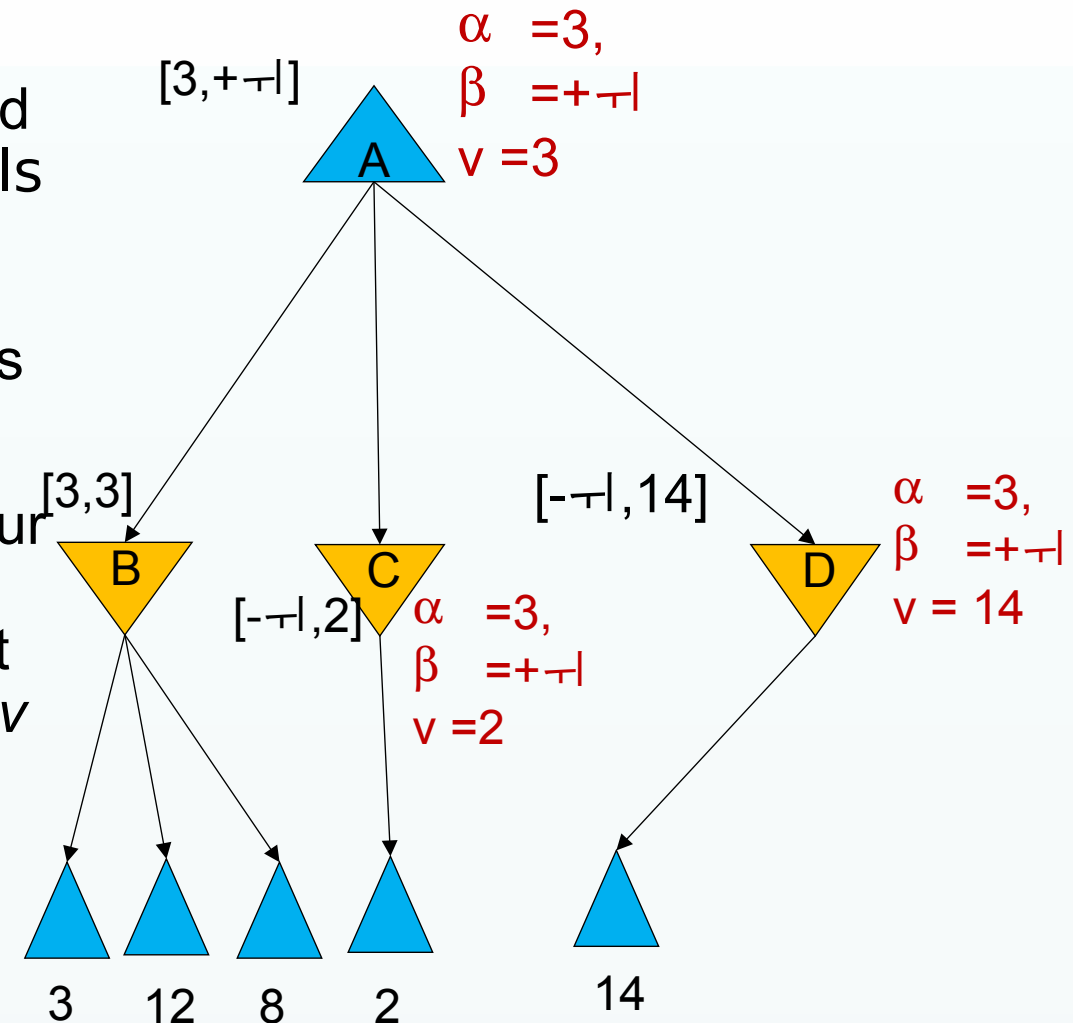


Elagage alpha-bêta : application de l'algorithme

□ La première feuille sous D est un nœud terminal donc l'appel VALEUR-MAX(fils de D, $3, +\infty$) pour le 1^{er} fils de D retourne la valeur d'utilité de l'état soit 14 et l'appel récursif ne se poursuit pas pour ce nœud (terminal).

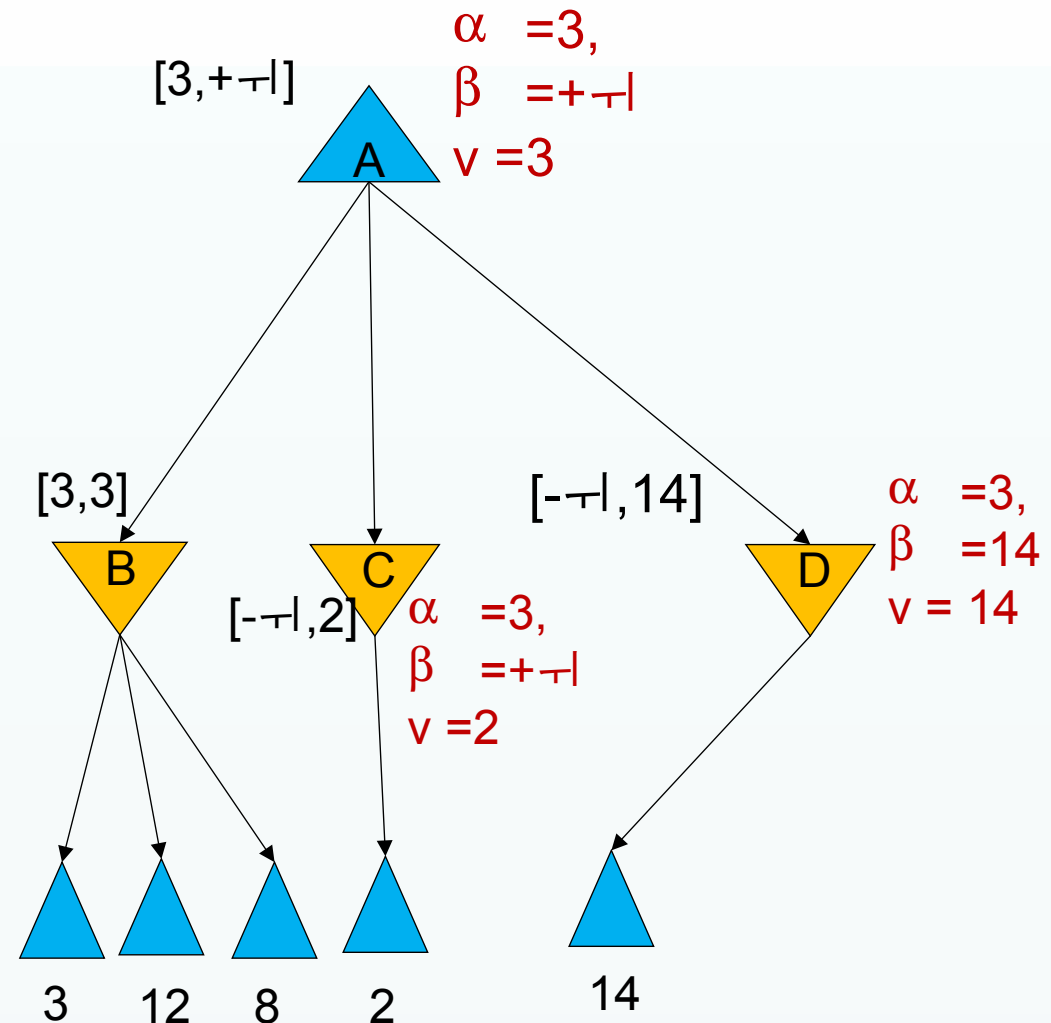
□ Cette première feuille sous D a la valeur d'utilité 14, l'appel $\text{MIN}(v, \text{VALEUR-MAX}(\text{RESULTAT}(s, a), \gamma, \infty))$ qui n'est autre que $\text{MIN}(+\infty, 14) = 14$ et donc v aussi.

□ Comme $v > \gamma$, v ne sera pas retournée comme valeur d'utilité de D.



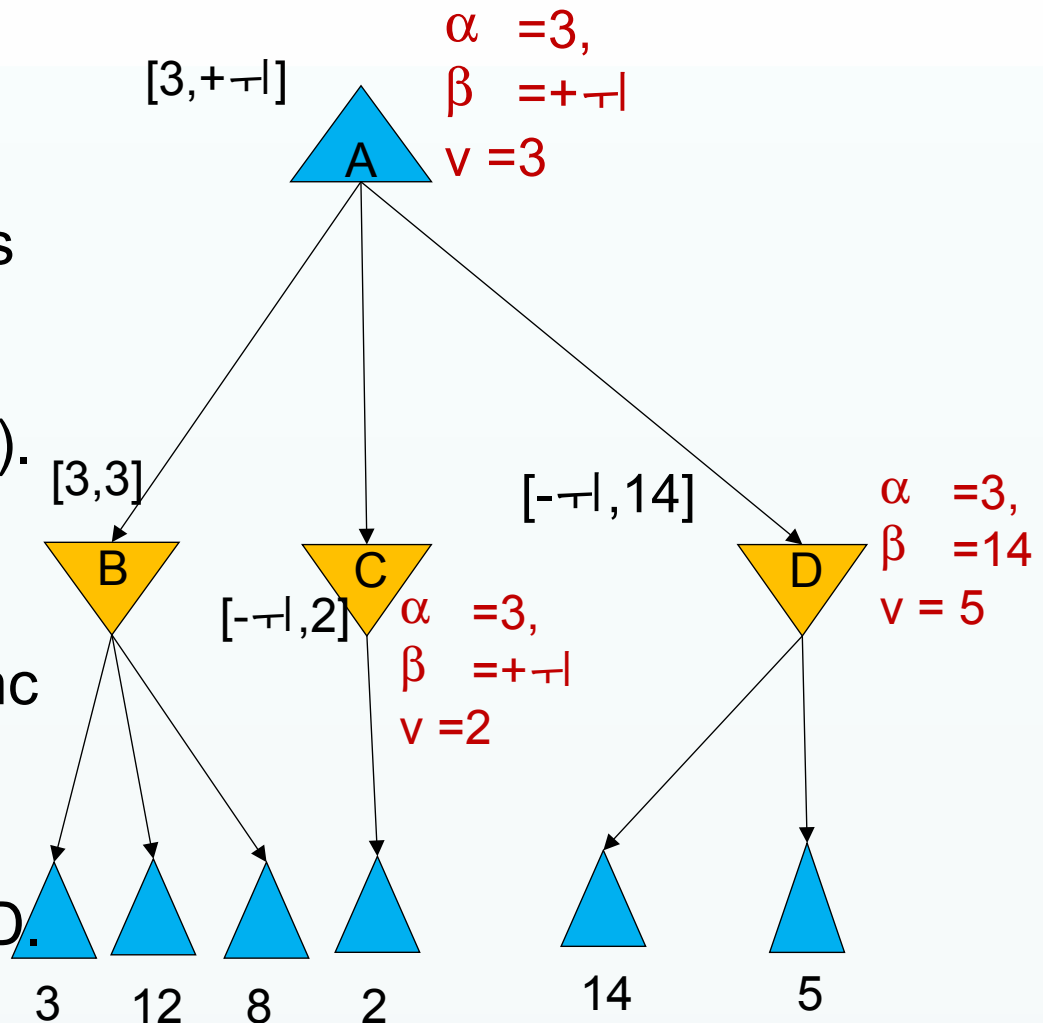
Elagage alpha-bêta : application de l'algorithme

□ L_{π} doit recevoir
maintenant $\text{MIN}(L_{\pi}, v) =$
 $\text{MIN}(+\infty, 14) = 14$.



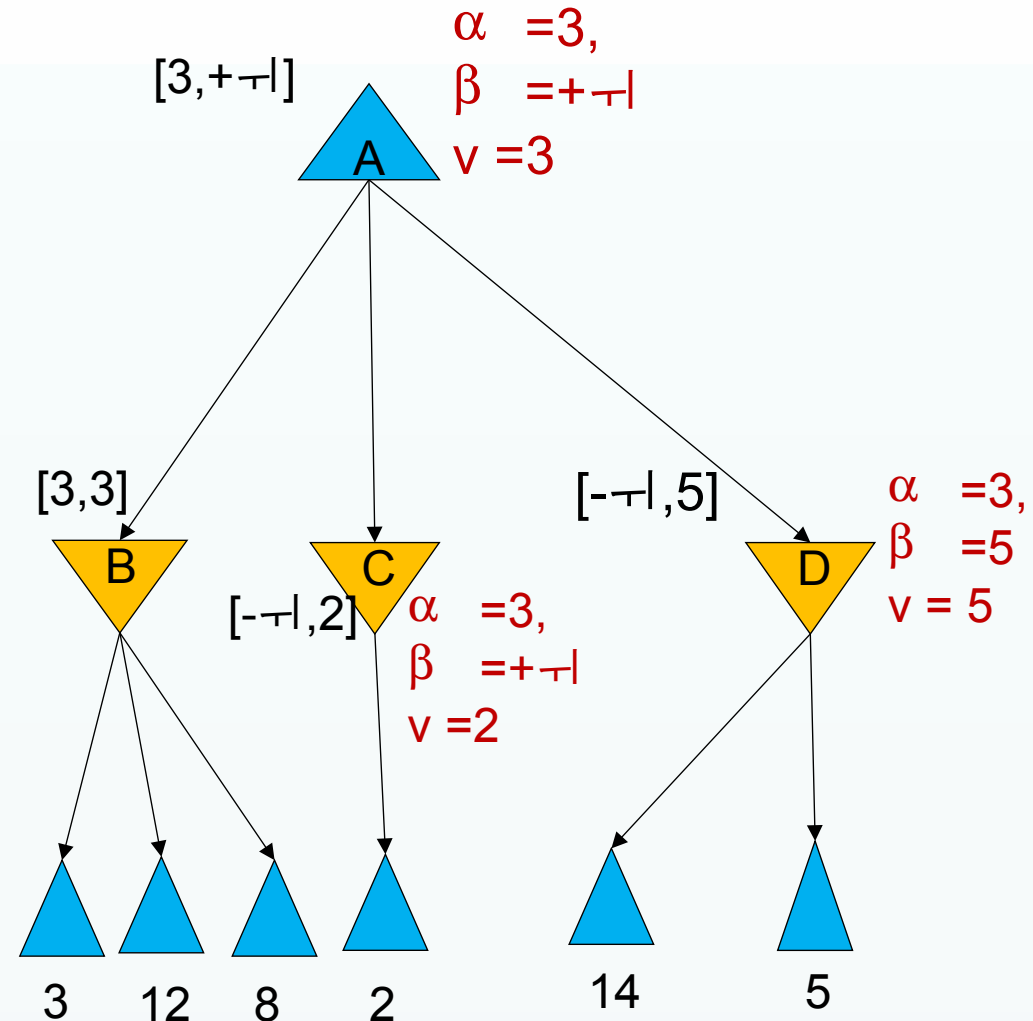
Elagage alpha-bêta : application de l'algorithme

- La 2^{ème} feuille sous D est un nœud terminal donc l'appel VALEUR-MAX(fils de D, 3, 14) pour le 2^{ème} fils de B retourne la valeur d'utilité de l'état soit 5 et l'appel récursif ne se poursuit pas pour ce nœud (terminal).
- L'appel MIN(v , VALEUR-MAX(RESULTAT(s, a), α , β))) qui n'est autre que MIN(14, 5)=5 et donc v aussi.
- Comme $v > \alpha$, v ne sera pas retournée comme valeur d'utilité de D.



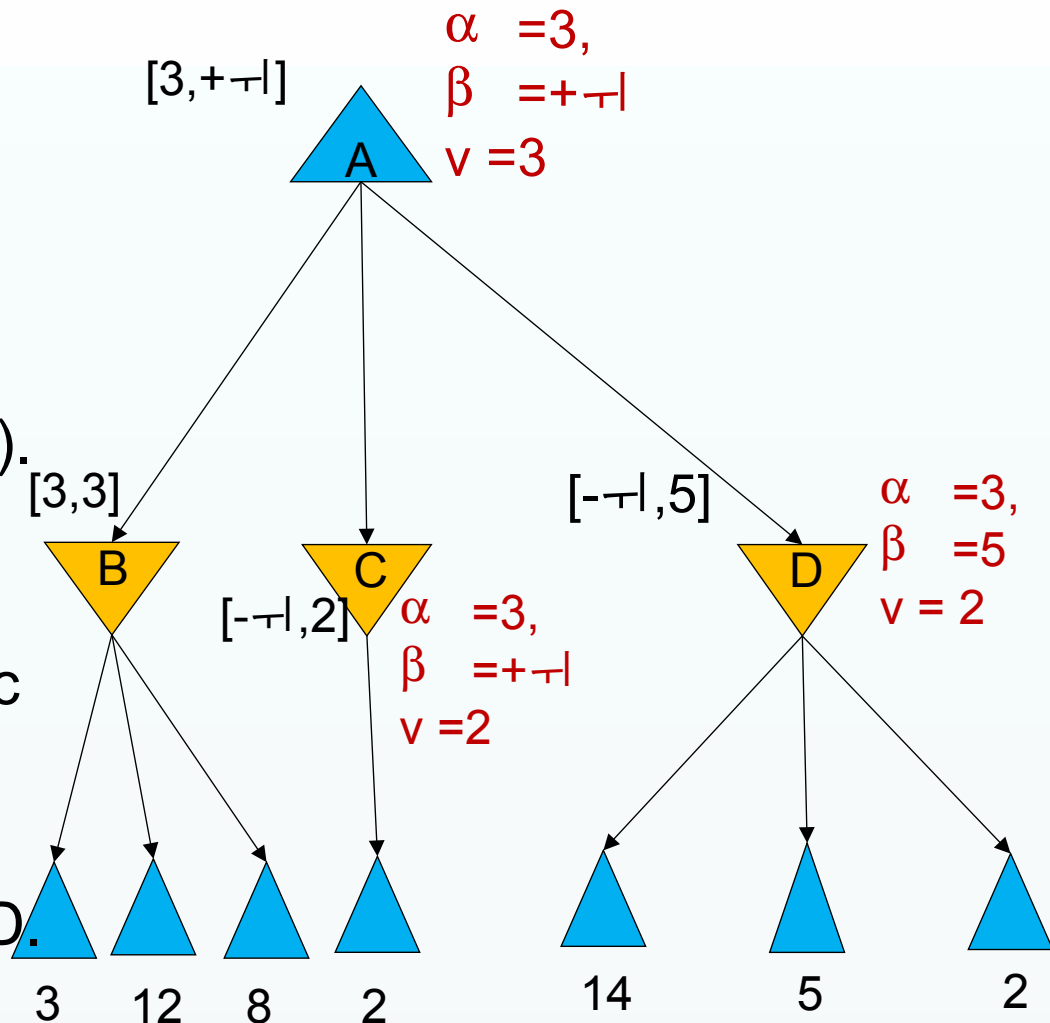
Elagage alpha-bêta : application de l'algorithme

□ L_{π} doit recevoir maintenant $\text{MIN}(L_{\pi}, v) = \text{MIN}(14, 5) = 5$.



Elagage alpha-bêta : application de l'algorithme

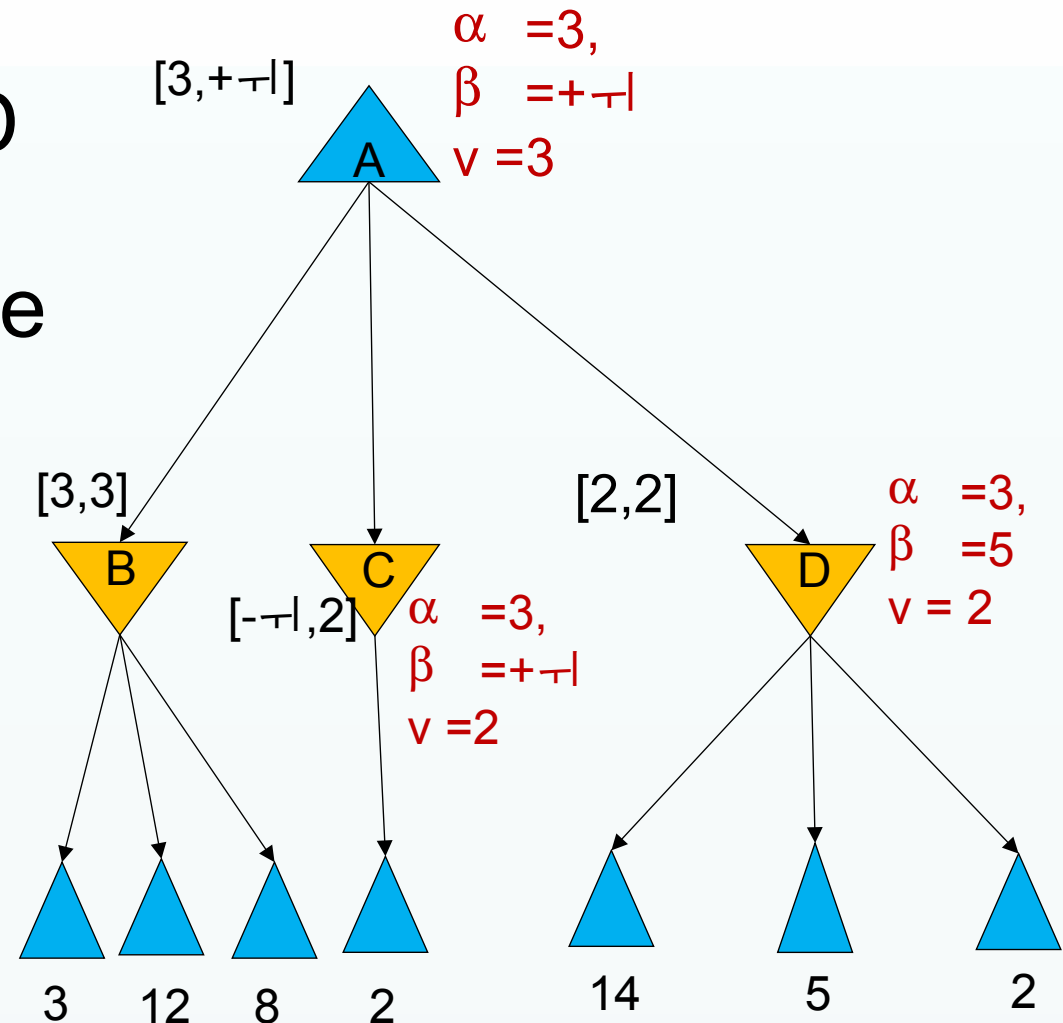
- La 3^{ème} feuille sous D est un nœud terminal donc l'appel VALEUR-MAX(fils de D, 3, 5) pour le 3^{ème} fils de B retourne la valeur d'utilité de l'état soit 2 et l'appel récursif ne se poursuit pas pour ce nœud (terminal).
- L'appel MIN(v , VALEUR-MAX(RESULTAT(s, a), α , β)) qui n'est autre que MIN(5, 2)=2 et donc v aussi.
- Comme $v \leq \alpha$, la valeur de v est retournée comme valeur d'utilité de D.



Elagage alpha-bêta : application de l'algorithme

□ L'exploration des fils de D est terminée, donc D vaut 2 : retournée par l'appel de VLAEUR-MIN sur D.

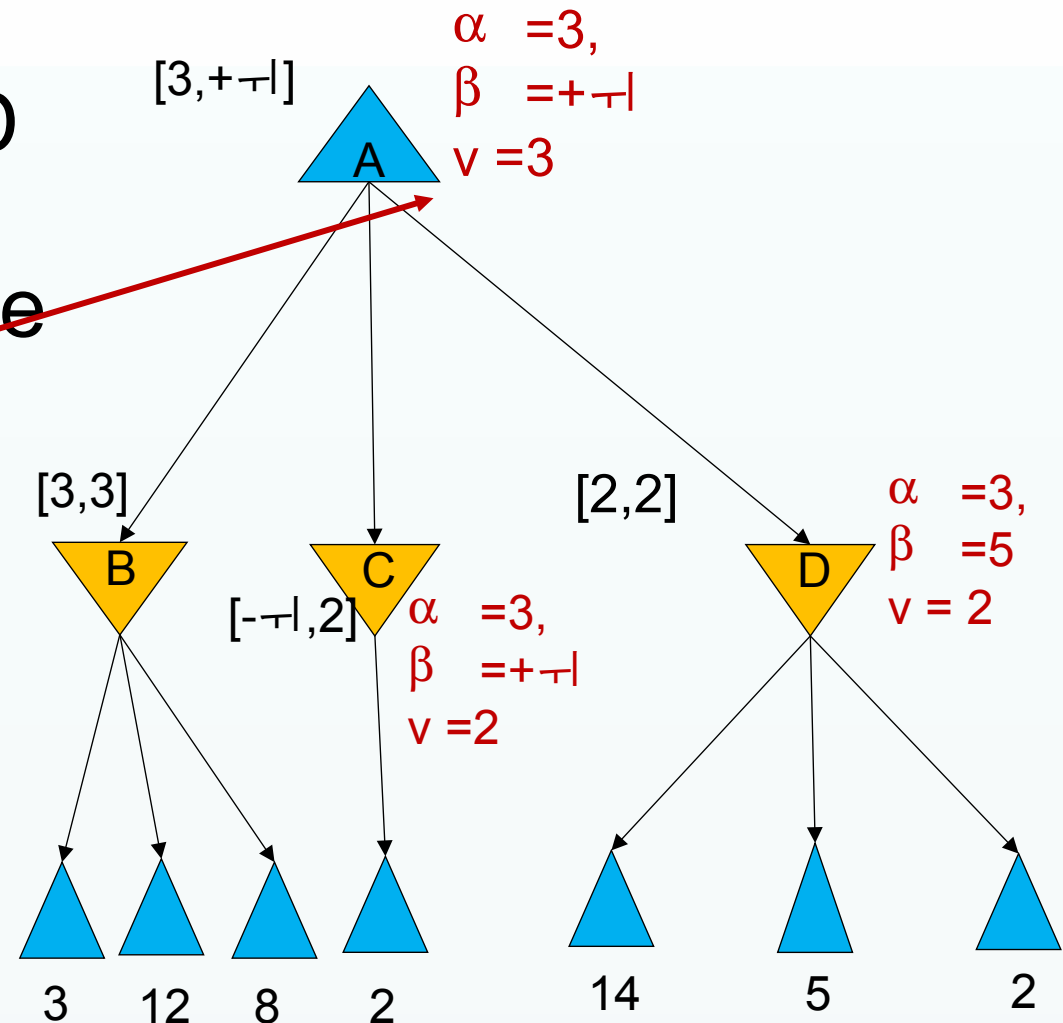
□ $v = \text{MAX}(v, 2) = \text{MAX}(3, 2)$



Elagage alpha-bêta : application de l'algorithme

□ L'exploration des fils de D est terminée, donc D vaut 2 : retournée par l'appel de VLAEUR-MIN sur D.

□ $v = \text{MAX}(v, 2) = \text{MAX}(3, 2)$



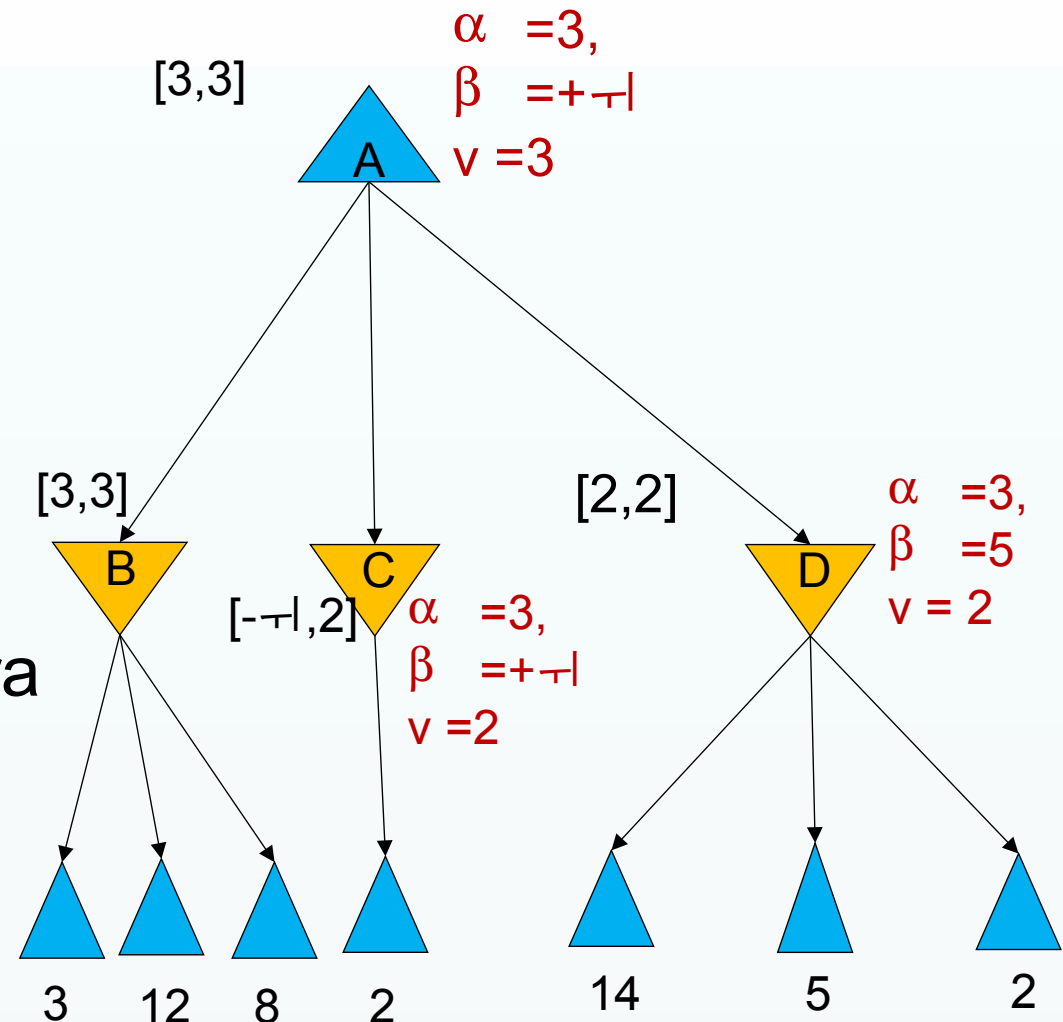
Elagage alpha-bêta : application de l'algorithme

□ L'exploration des fils de D est terminée, donc D vaut 2 : retournée par l'appel de VLAEUR-MIN sur D.

□ $v = \text{MAX}(v, 2) = \text{MAX}(3, 2) = 3$.

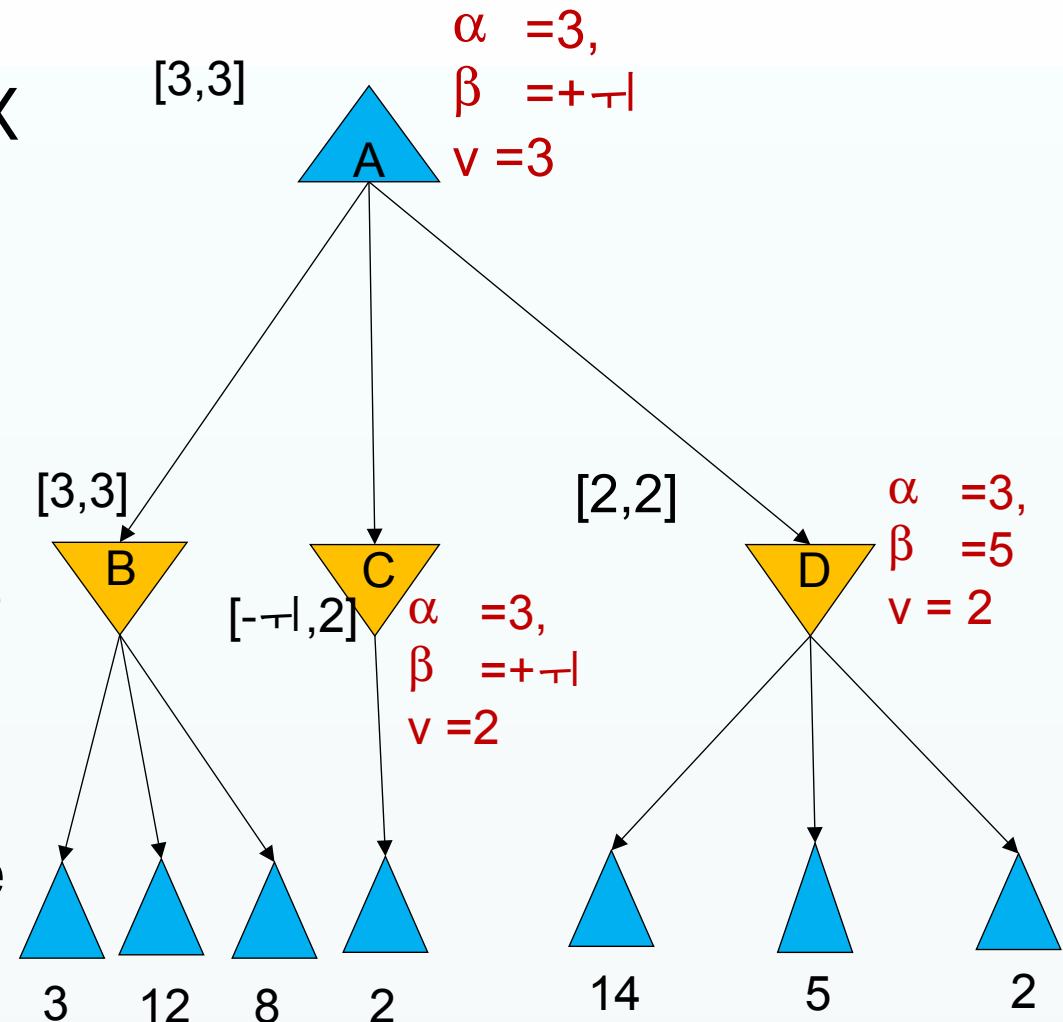
□ $v < \alpha$ donc la valeur de v ne sera pas retournée.

□ $\alpha = \text{MAX}(\alpha, v) = \text{MAX}(3, 3) = 3$



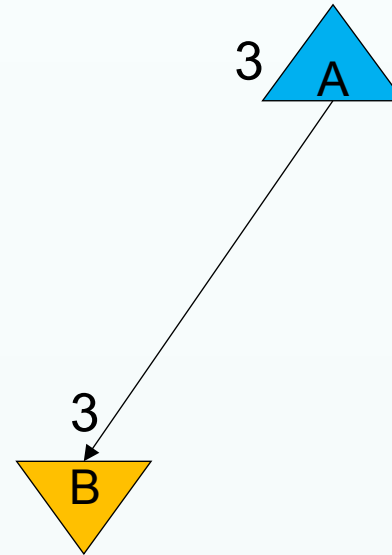
Elagage alpha-bêta : application de l'algorithme

- Fin de l'appel de VALEUR-MAX par le retour de $v = 3$.
- Cette valeur sera récupéré par la fonction EXPLORATION-ALPHA-BETA sur l'état A.
- Elle sera assignée à la variable v .
- EXPLORATION-ALPHA-BETA retourne alors l'action qui mène l'état ayant cette valeur, soit B.

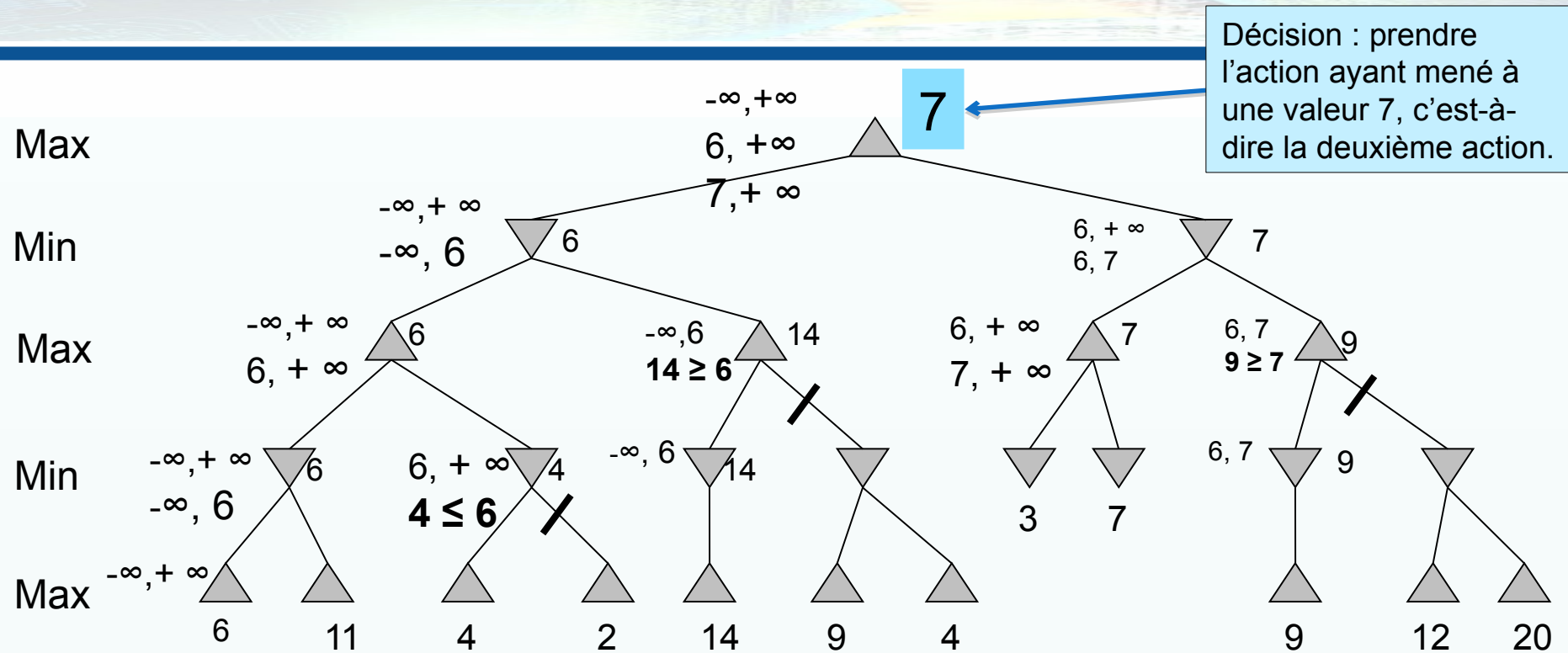


Elagage alpha-bêta : application de l'algorithme


- ❑ Fin de l'appel de VALEUR-MAX par le retour de $v = 3$.
- ❑ Cette valeur sera récupéré par la fonction EXPLORATION-ALPHA-BETA sur l'état A.
- ❑ Elle sera assignée à la variable v .
- ❑ EXPLORATION-ALPHA-BETA retourne alors l'action qui mène l'état ayant cette valeur, soit B.



Application




Légende de l'animation

 Nœud de l'arbre pas encore visité

 Nœud en cours de visite (sur pile de récursivité)

 Nœud visité

 Arc élagué (pruning)

α, β  Valeur retournée

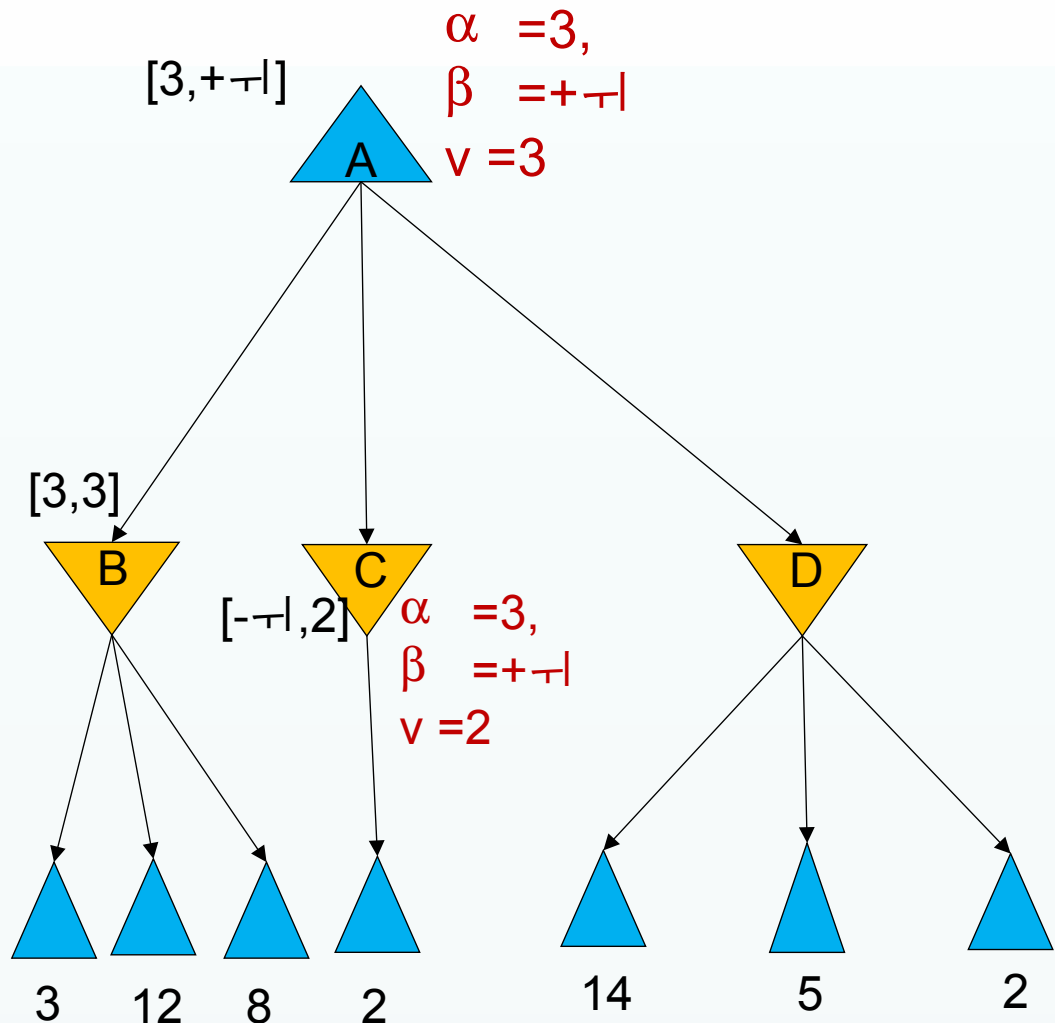
Valeur si
feuille

Propriétés de $\min\max$

- ❑ L'élagage n'affecte pas le résultat final de *minimax*.
- ❑ Dans le pire des cas, alpha-beta ne fait aucun élagage ; il examine b^m nœuds terminaux comme l'algorithme *minimax* :
 - b : le nombre maximum d'actions/coups légales à chaque étape.
 - m : nombre maximum de coup dans un jeu.
- ❑ Un bon choix de **l'ordre** de l'examen des actions améliore l'efficacité de l'élagage :

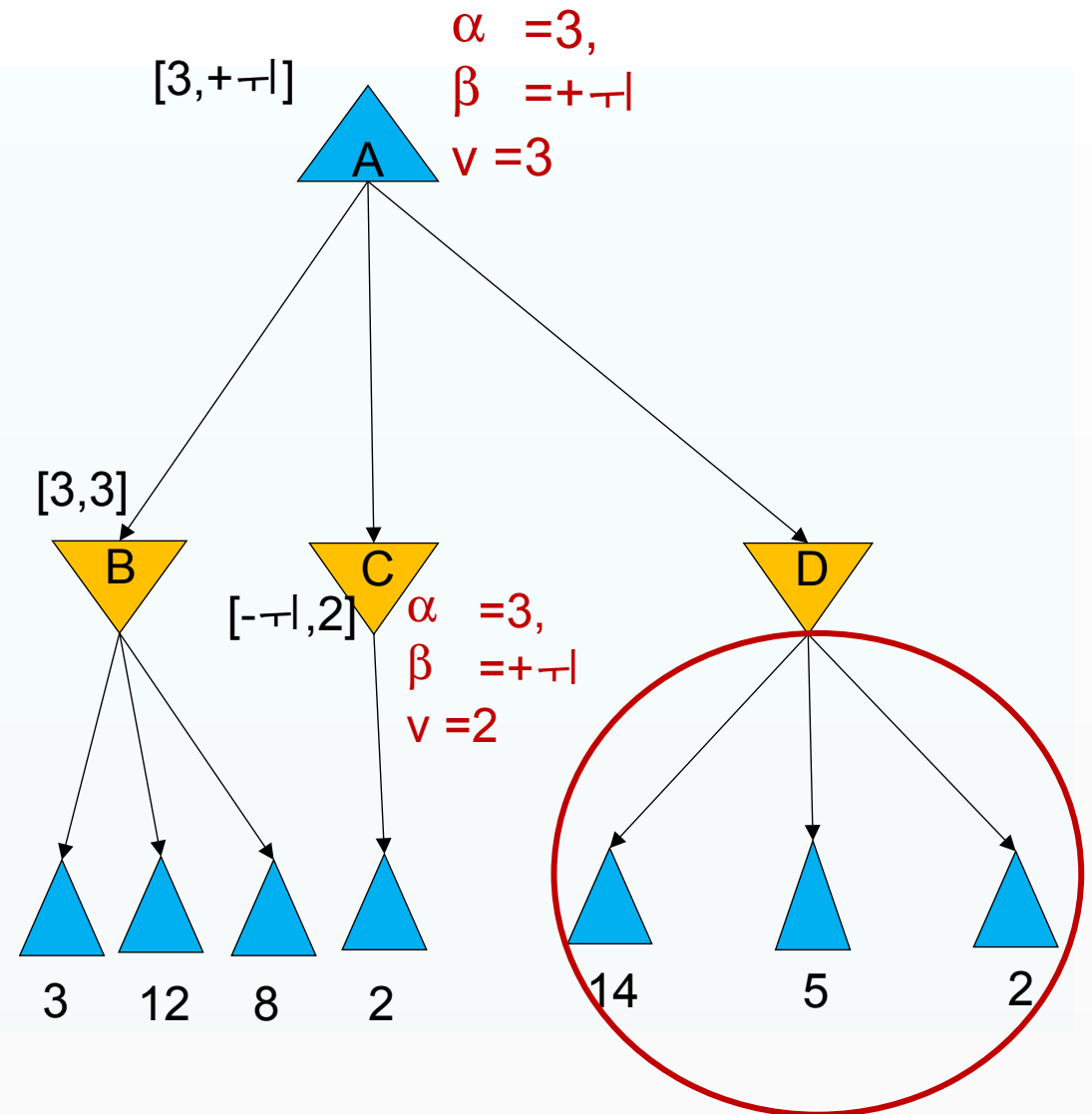
Attention à l'ordre des coups

□ Revenons à l'application de l'algorithme (page 63).



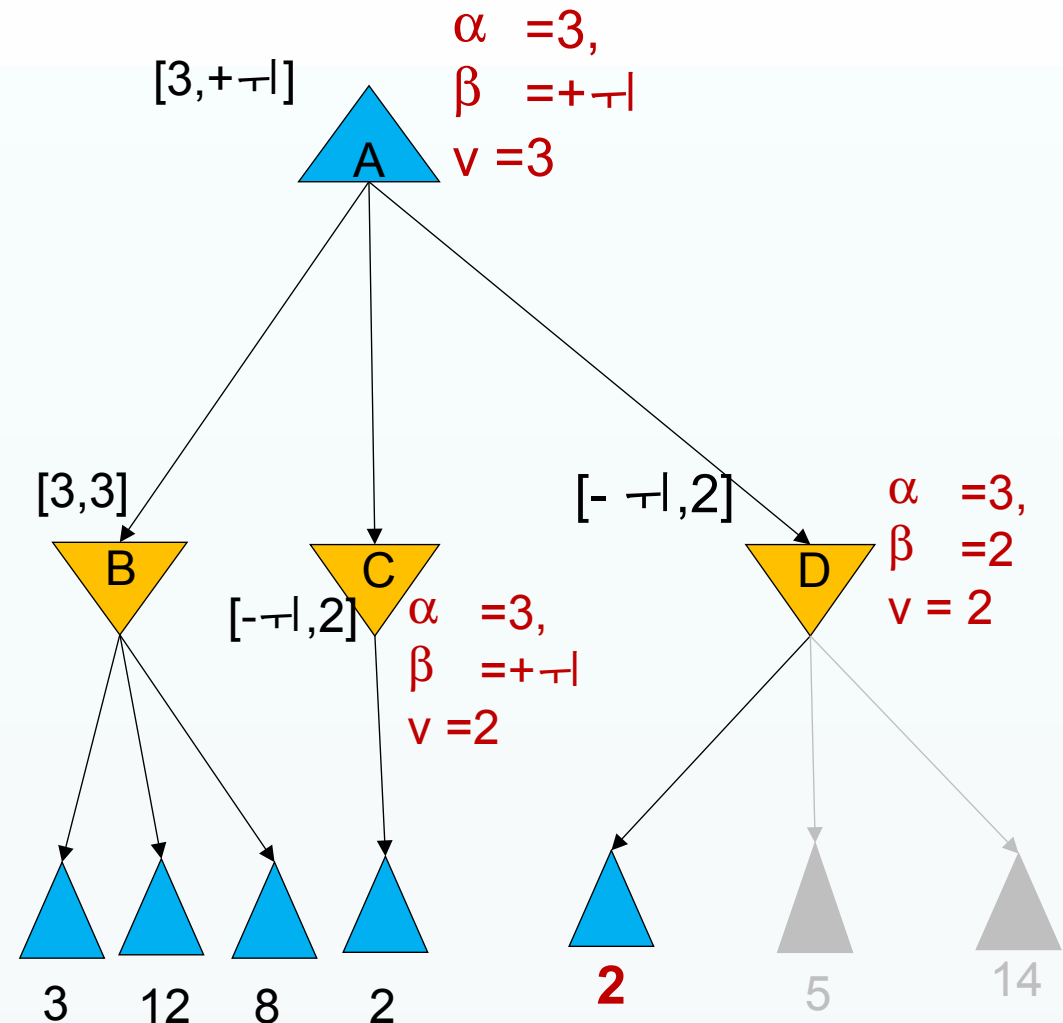
Attention à l'ordre des coups

- Revenons à l'application de l'algorithme (page 63).
- Aucun successeur de D ne peut être élagué, puisque les plus mauvais successeurs (pour MIN) ont été générés en premier.



Attention à l'ordre des coups

- Si on change l'ordre des feuilles : le dernier devient premier.
- $v_{\neg\Gamma} \neg\Delta$ donc v sera retourné et les autres feuilles seront élaguées.
- Reste à trouver la fonction d'ordonnement pour jouer à la perfection !



Propriétés de $\neg \wedge \neg \vee$

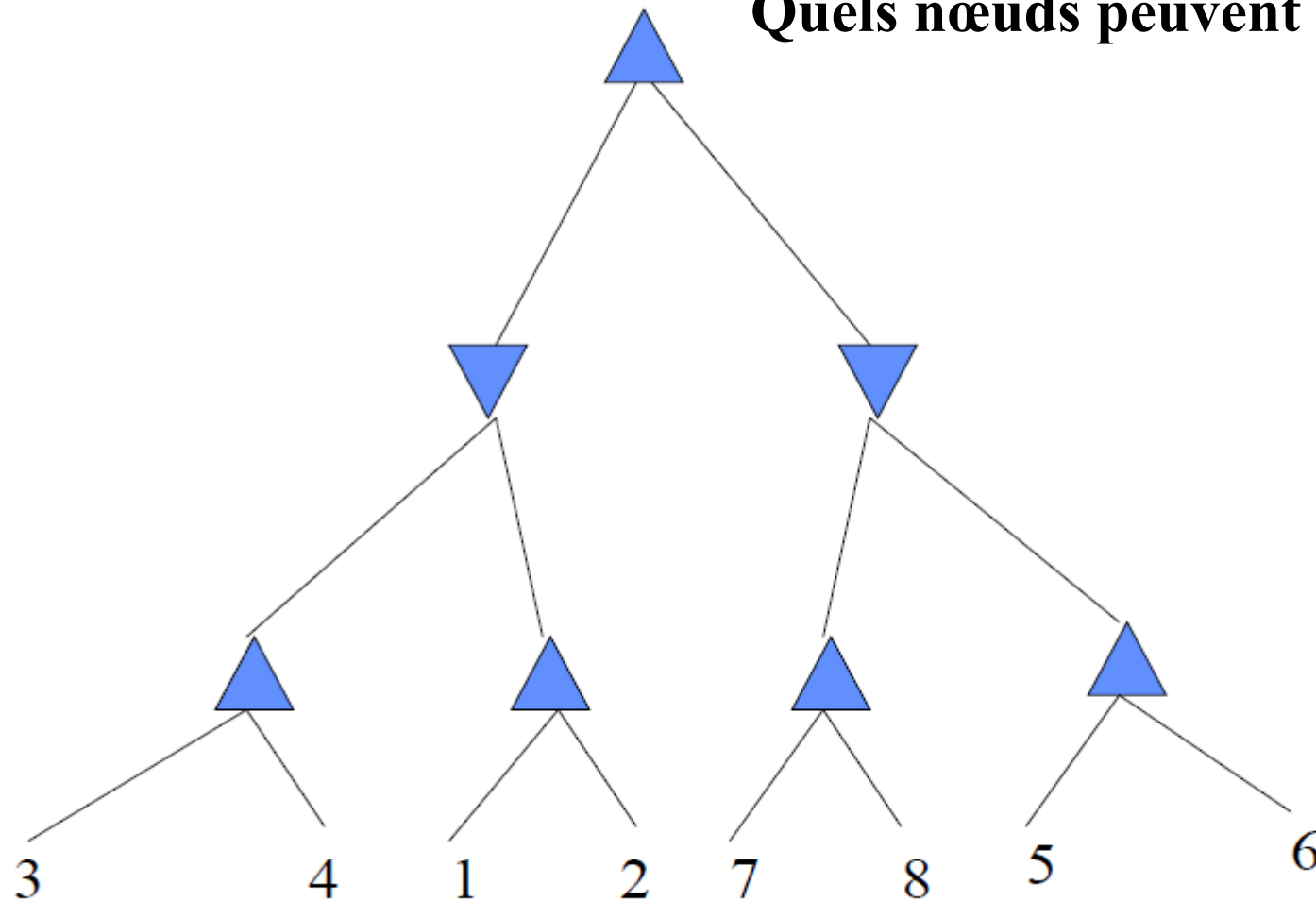
- Un bon choix de l'ordre de l'examen des actions améliore l'efficacité de l'élagage :
 - Dans le meilleur des cas (ordonnancement parfait), la complexité en temps est de $O(b^{m/2})$ au lieu de $O(b^m)$: On peut faire une recherche deux fois plus profondément comparé à *minimax*!
 - C'est la même chose que d'avoir un facteur de branchement de : $(b)^{m/2} = b^{m/2}$.
 - Pour les échecs, on passe de $b=35$ à $b \approx 6$, mais la complexité reste quand même assez élevée : 35^{50} .
- Si les nœuds sont examinés aléatoirement, le nombre de nœuds visités serait $\approx O(b^{3m/4})$

Se rapprocher de la limite théorique

- ❑ Par exemple, pour calculer dynamiquement l'ordonnancement des coups :
 - ❑ Choisir en premier lieu les meilleurs mouvements trouvés ultérieurement.
 - ❑ Utiliser une exploration itérative en profondeur pour tirer les nœuds selon les valeurs trouvées dans les itérations précédentes : ajoute une fraction constante du temps total d'exploration, mais on y gagne en ordonnancement (favoriser des **coups de maître**).
- ❑ Éviter les états répétés engendrés par des transpositions : par une table de hachage (table des transpositions), équivalente à la liste explorée du chapitre 2. Mais attention au nombre d'états (imaginons si on évaluait 1 million de nœuds/s !) : choisir ceux à garder.

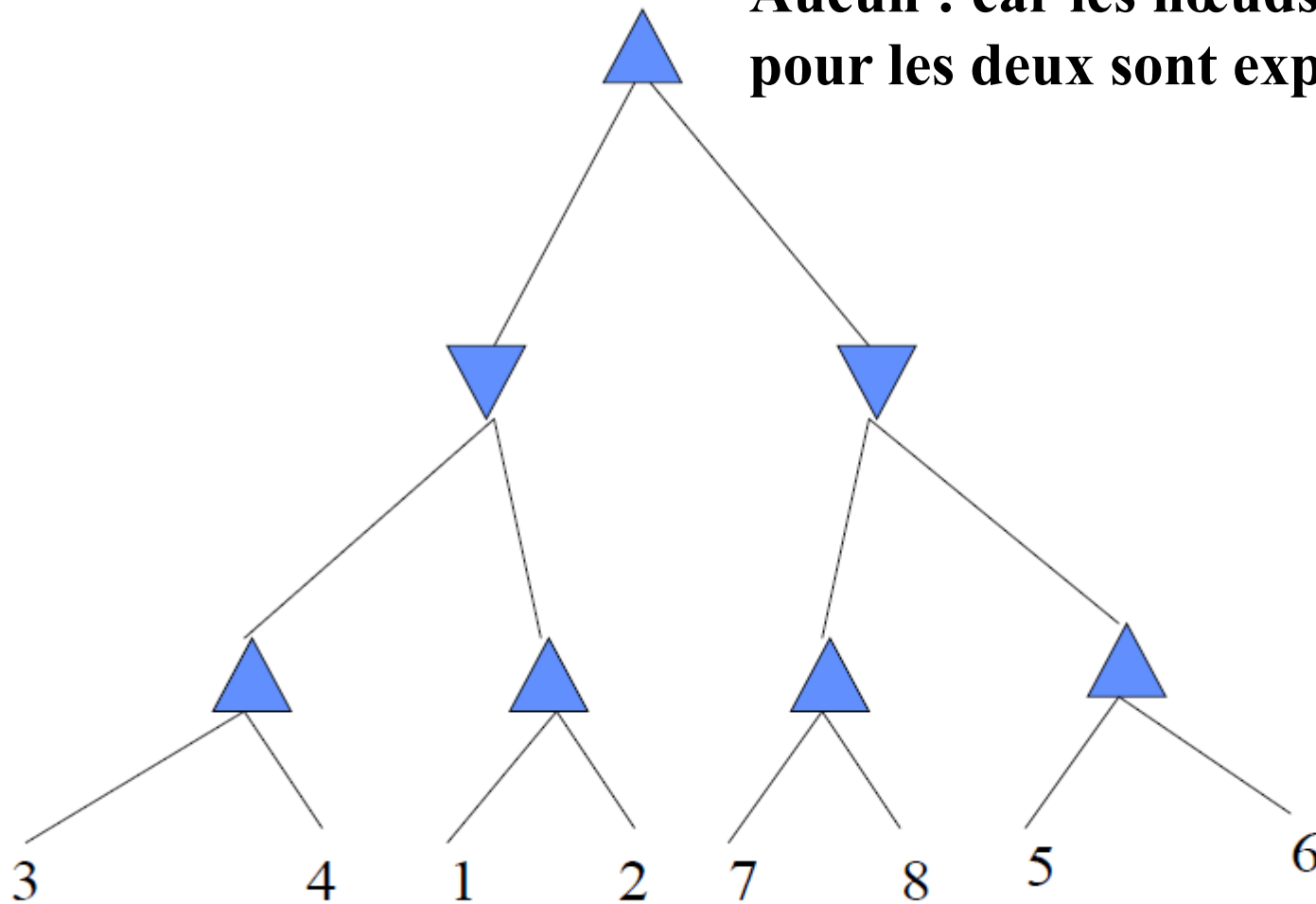
Application 1

Quels nœuds peuvent être élagués ?

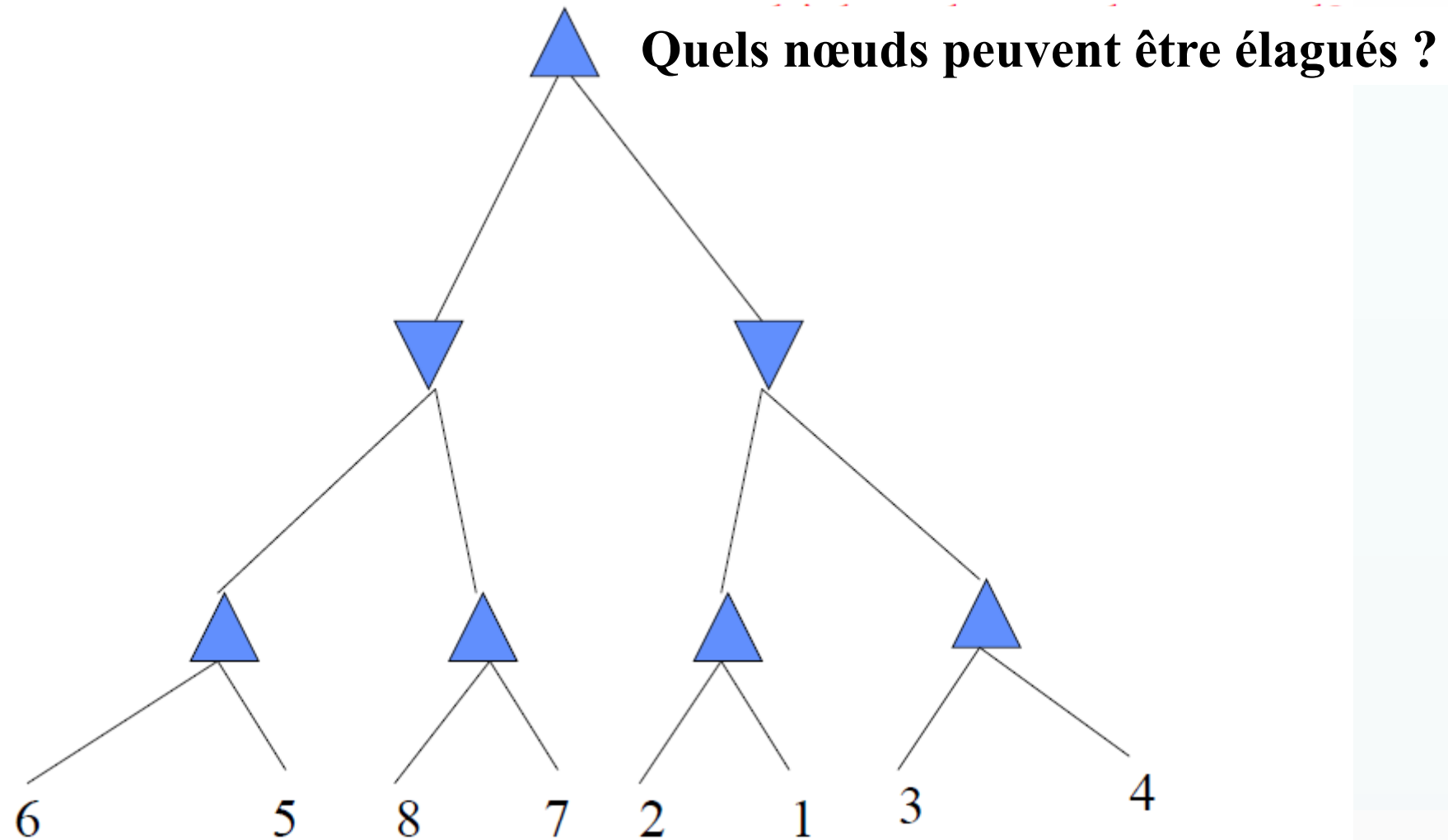


Application 1

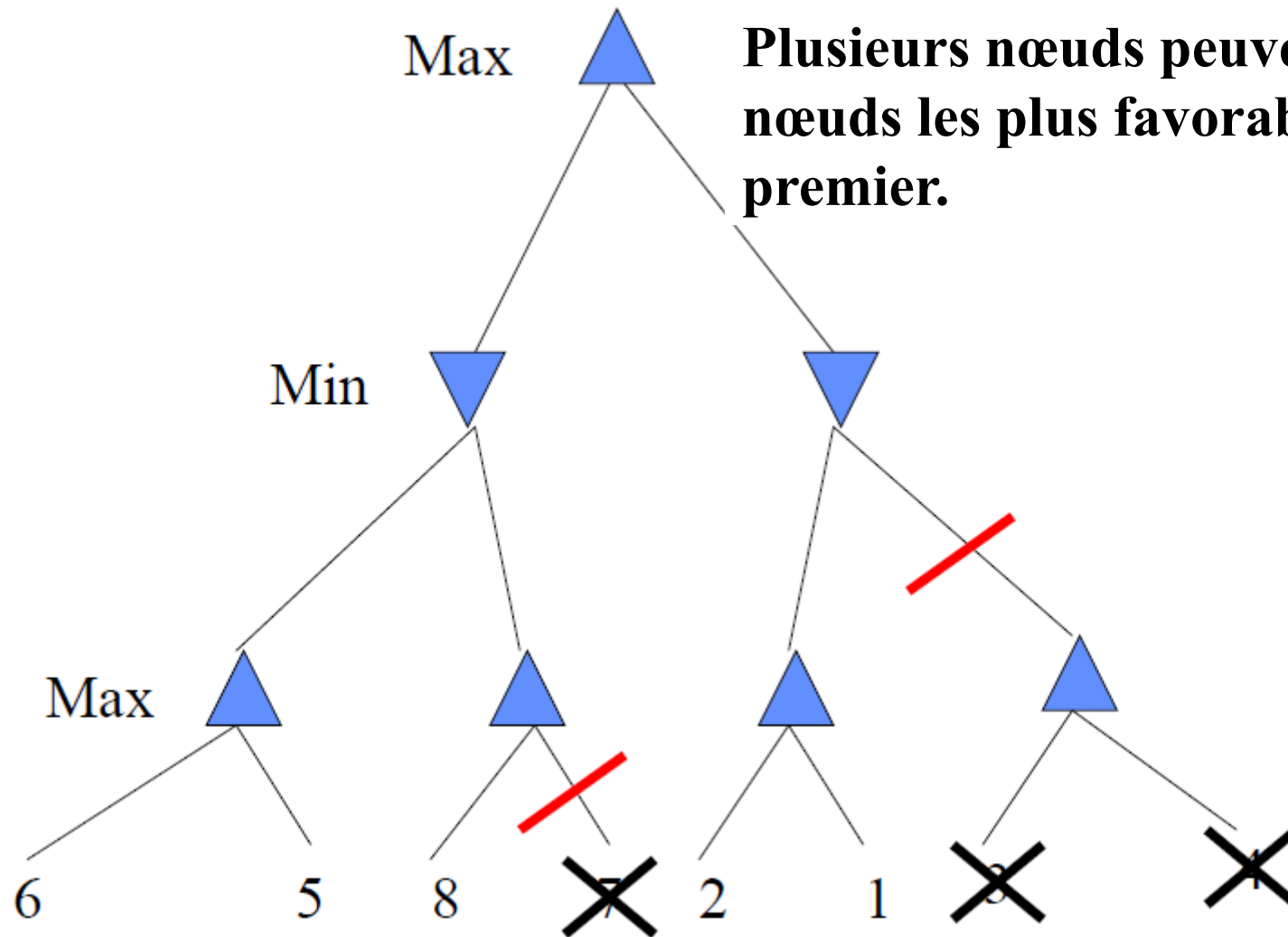
Aucun : car les nœuds les plus favorables pour les deux sont explorés en dernier



Application 2



Application 2



Plusieurs nœuds peuvent être élagués car les nœuds les plus favorables sont explorés en premier.

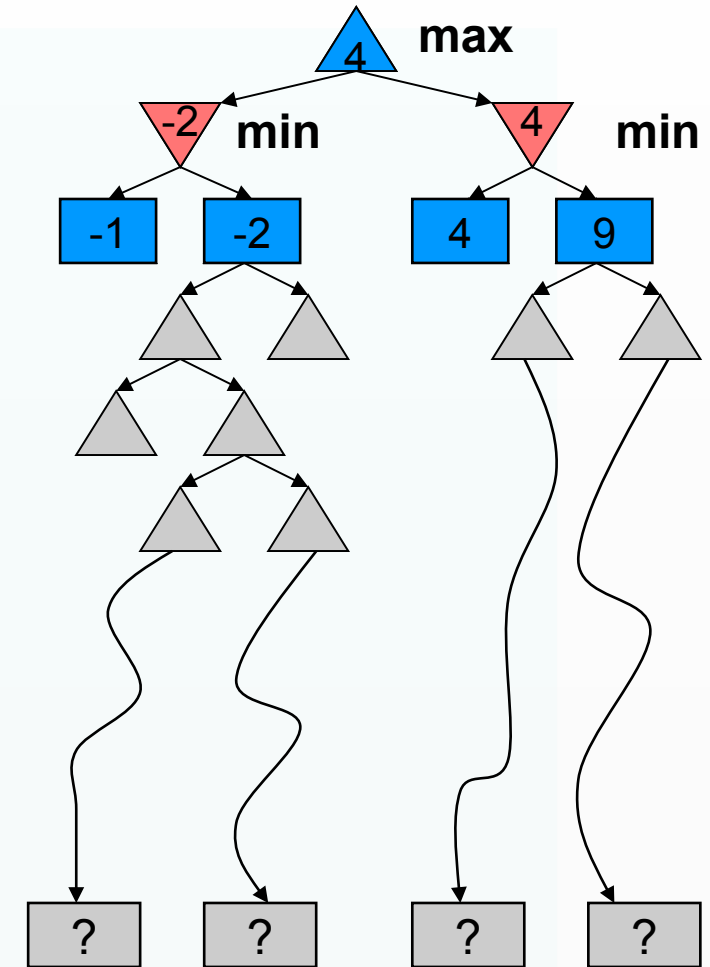
Décisions en temps réel (ressources limitées)

Décisions imparfaites en temps réel

- ❑ L'algorithme alpha-bêta permet d'élaguer une grande partie de l'arbre de jeu, mais il doit explorer au moins une partie de l'espace jusqu'aux états terminaux.
- ❑ Cette profondeur est atteinte difficilement en un temps raisonnable (contrainte du temps réel).

Approche standard : selon Claude Shannon

- ❑ Interrompre l'exploration plus tôt et remplacer la fonction d'utilité par une **fonction d'évaluation** heuristique EVAL (qui va considérer des états non terminaux comme tels) :
 - ❑ donne une estimation de l'utilité d'une position qui aurait été obtenue en faisant une recherche complète (comme l'heuristique h estimant la distance au but)
 - ❑ on peut voir ça comme une estimation de la « chance » qu'une configuration mènera à une victoire
- ❑ Remplacer le test de terminaison TESTTERMINAL par un test d'arrêt (**cutoff test**), qui va décider quand appliquer EVAL (exemple limiter la profondeur).



Fonction d'évaluation

- ❑ Elle doit ordonner les états terminaux de la même manière que la fonction d'utilité.
- ❑ Les calculs doivent se faire rapidement (temps réel).
- ❑ Pour les états non terminaux, elle doit être corrélée avec les « **chances de victoire** ».

Concevoir une bonne fonction d'évaluation

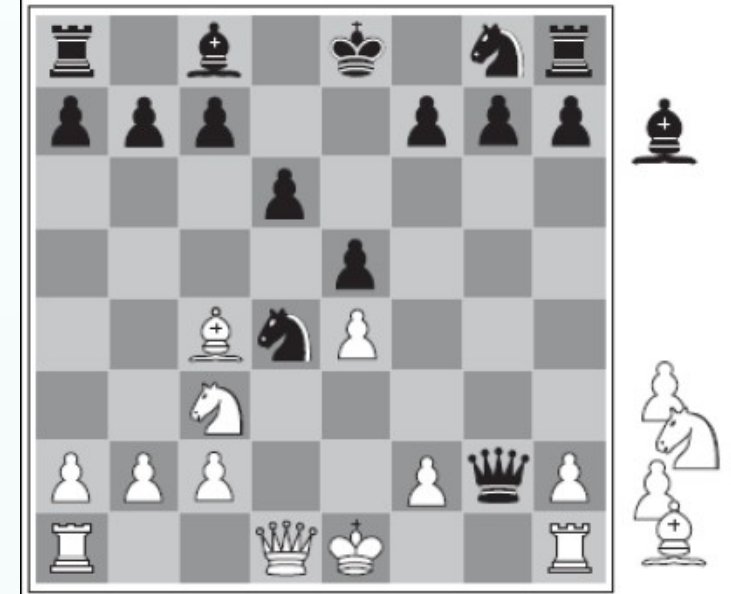
- ❑ Repose sur le calcul des différents attributs d'un état (ex. aux échecs : nombre de pions blancs, de pions noirs, de reines blanches, de reines noires).
- ❑ Pris ensemble, ces attributs définissent différentes catégories.
- ❑ Les états de chaque catégorie ont les mêmes valeurs pour tous les attributs (ex. une catégorie contient toutes les fin de partie à 2 pions contre un pion).
- ❑ Chaque catégorie contient des états conduisant à une victoire, une perte ou à un match nul.
- ❑ Si par exemple nous pensons que 72% des états d'une catégorie conduisent à une victoire (+1), que 20% à une perte (0) et 8% à une partie nulle ($\frac{1}{2}$), la valeur pourrait être : $(0,72 \times +1) + (0,20 \times 0) + (0,08 \times \frac{1}{2}) = 0,76$.
- ❑ Mais c'est très difficile de la calculer ! A la place on calcule les **contributions numériques distinctes** de chaque attribut et on les combine pour déterminer la valeur totale.

Fonction linéaire pondérée pour le jeu d'échec

- ❑ Chaque pièce a une valeur matérielle approximative : un pion vaut 1, un cavalier ou un fou vaut 3, une reine vaut 9, etc.
- ❑ Si un joueur d'échecs a un avantage d'un pion a une bonne probabilité de victoire et si cet avantage équivaut à 3 pions, la victoire est presque certaine.
- ❑ La fonction d'évaluation typique est une somme (linéaire) pondérée de *métriques* estimant la qualité de la configuration:

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

- ❑ w_i peut être = poids du pion,
- ❑ $f_i(s)$ peut être = (nombre d'occurrence d'un type de pion d'un joueur) – (nombre d'occurrence du même type de pion de l'opposant).



C'est aux blancs de jouer
Les noirs ont de fortes chances de gagner

Exploration avec arrêt

- ❑ Modifier EXPLORATION-ALPHA-BÊTA pour qu'il appelle la fonction heuristique EVAL lorsqu'il faut arrêter l'exploration (**cut off**).
- ❑ Il faut modifier la fonction TEST-ARRÊT qui doit retourner vrai si la limite est atteinte ou si l'état rencontré est terminal.
- ❑ Pour état s et profondeur maximale d :

$$\text{H-MINIMAX}(s, d) = \text{EVAL}(s)$$

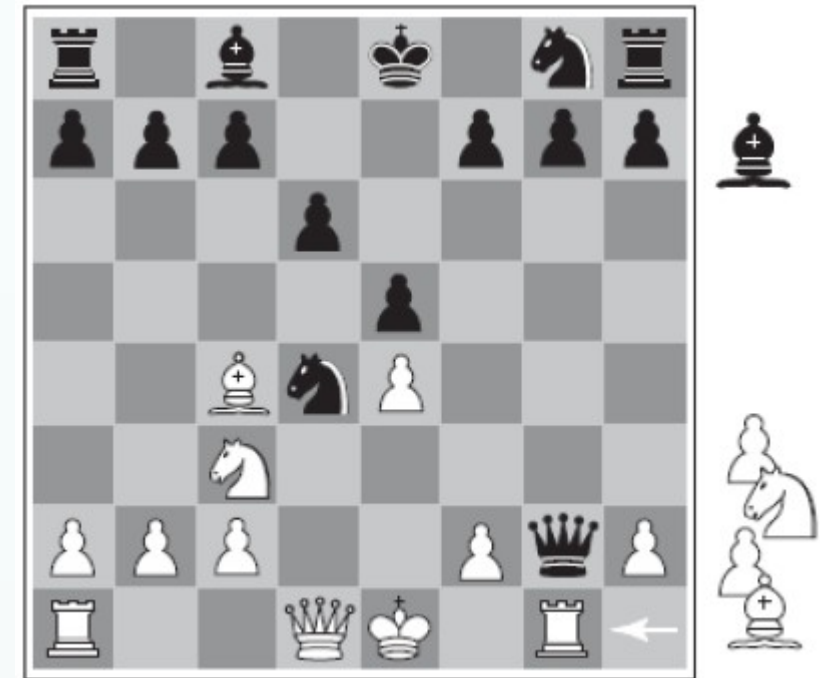
si TEST-
ARRÊT(s, d)

$$\text{H-MINIMAX}(s, d) = \max_{a \in \text{Actions}(s)} \text{H-MINIMAX}(\text{RÉSULTAT}(s, a), d+1)$$

si
JOUEUR(s)=M

Choix de la profondeur d

- ❑ La limite d peut être fixée à l'avance de telle manière qu'un coup puisse être choisi dans le temps imparti.
- ❑ Il est possible d'appliquer l'exploration itérative en profondeur d'abord (IDS) : lorsque le temps s'écoule, le programme retourne le coup renvoyé par l'exploration la plus profonde achevée : c'est intéressant surtout que l'IDS définit également l'ordre des coups.
- ❑ Mais il faut faire attention aux erreurs dues à la nature approximative de la fonction d'évaluation.
- ❑ Un programme a un **horizon limité**.
- ❑ Solution : appliquer la fonction qu'à des états **stables**.



C'est aux blancs de jouer
Les blancs ont de fortes chances de gagner

Fonction d'évaluation pour le tic-tac-toe

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

$Eval(s) = (\text{nombre de lignes, colonnes et diagonales disponibles pour Max}) - (\text{nombre de lignes, colonnes et diagonales disponibles pour Min})$



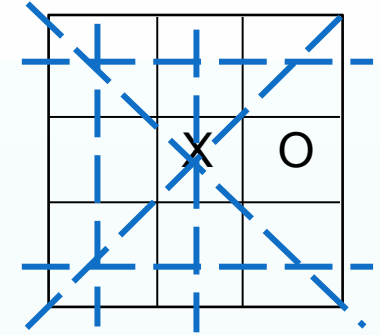
Fonction d'évaluation pour le tic-tac-toe

	X	O

$$Eval(s) = 6 - 4 = 2$$

X a 6 Chemins possibles pour gagner

O a 5 Chemins possibles pour gagner



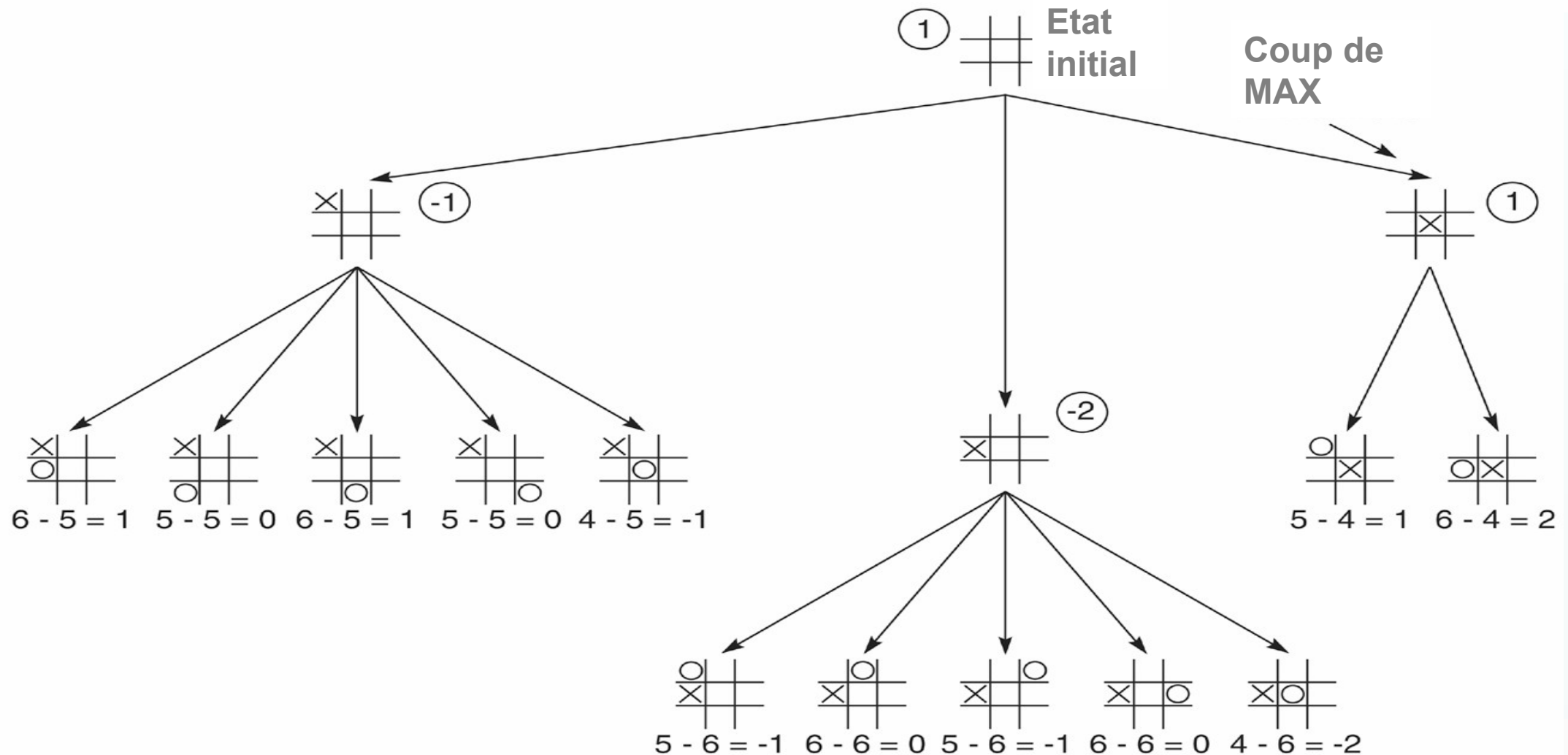
O	X	X
	O	

$$Eval(s) = 4 - 3 = 1$$

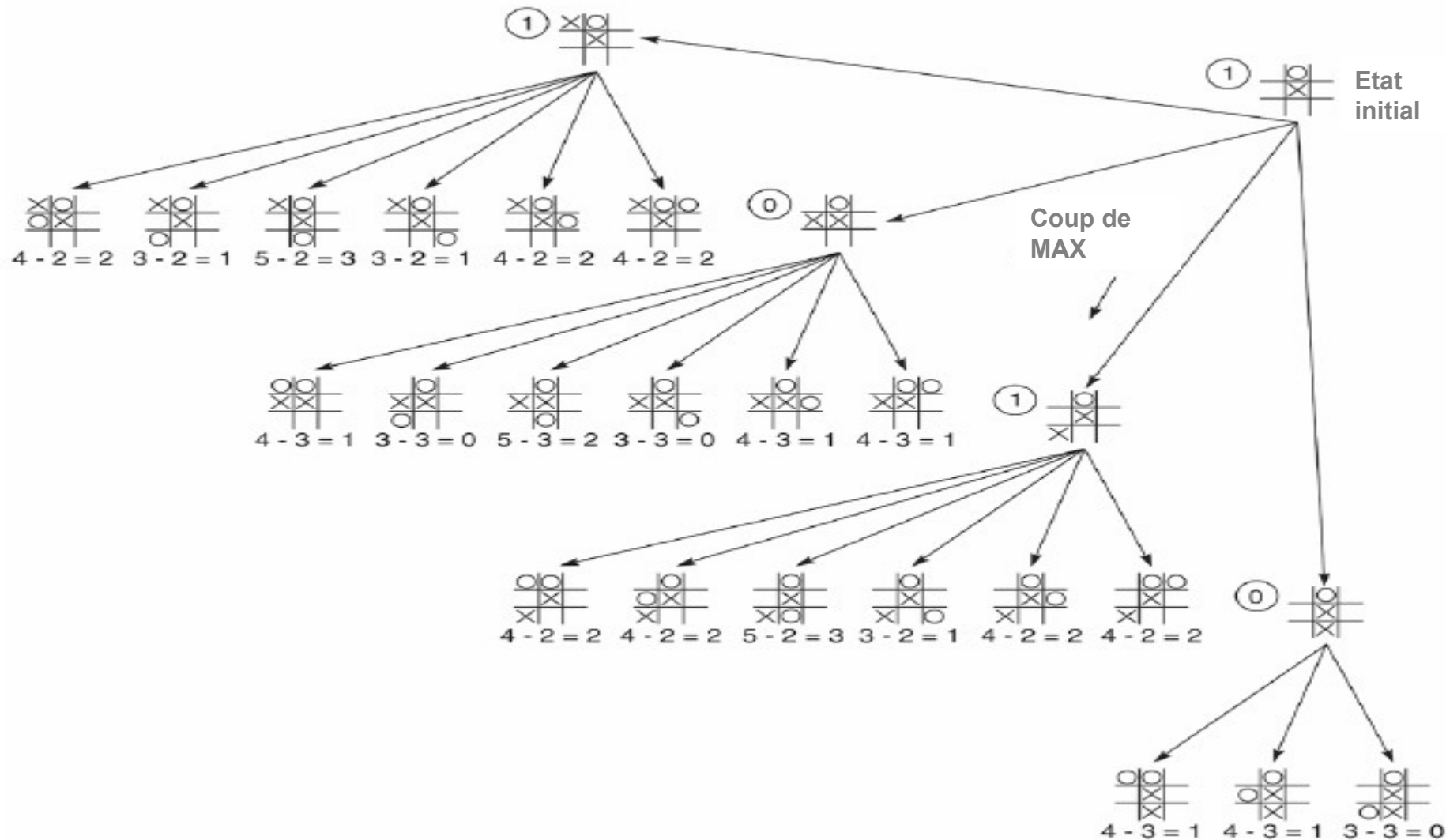
X a 4 Chemins possibles pour gagner

O a 3 Chemins possibles pour gagner

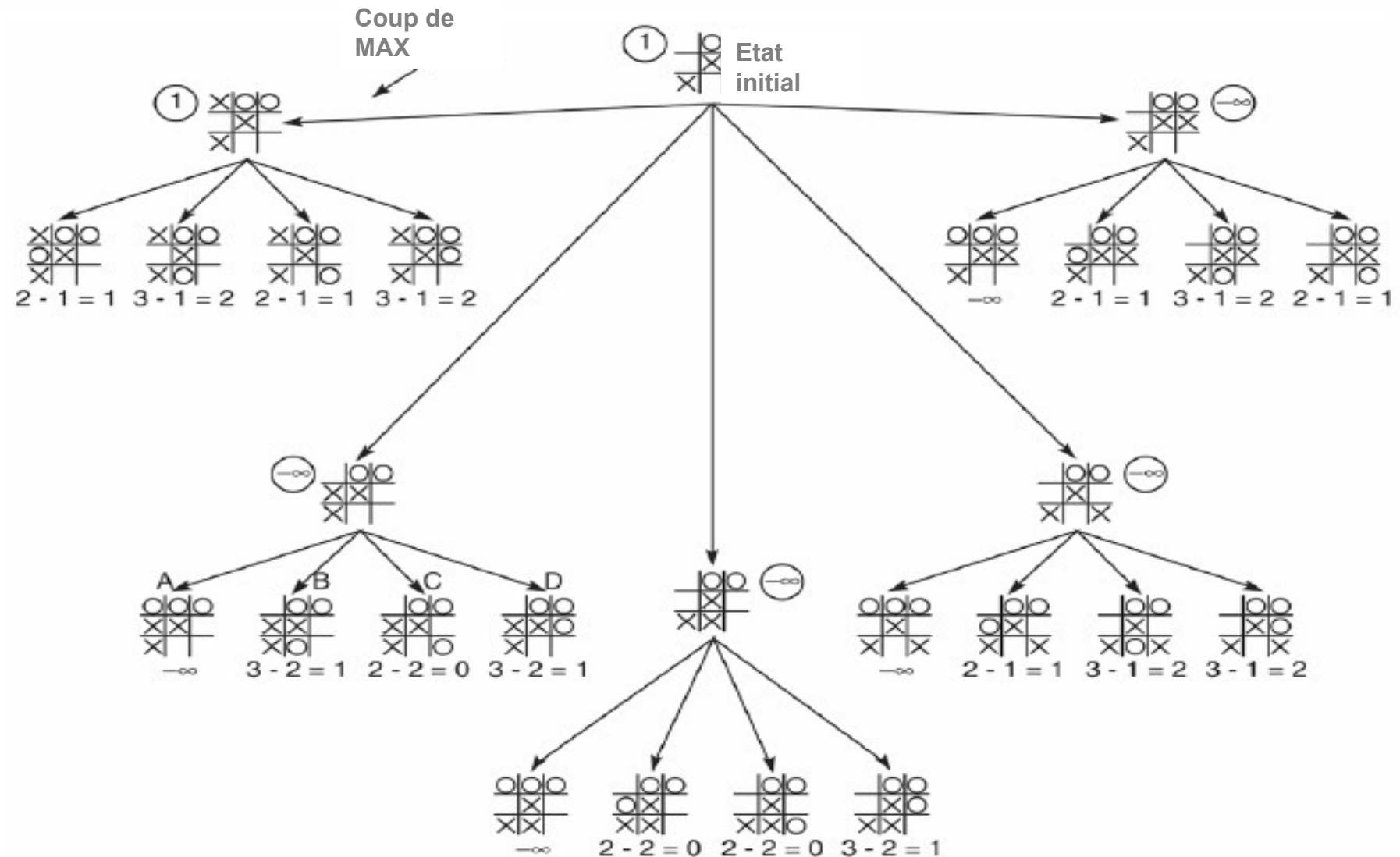
Minimax appliqué au premier tour du tic-tac-toe (Nilsson, 1971)



Minimax appliqué au deuxième tour et un des 2 coups possibles de Max (Nilsson, 1971)



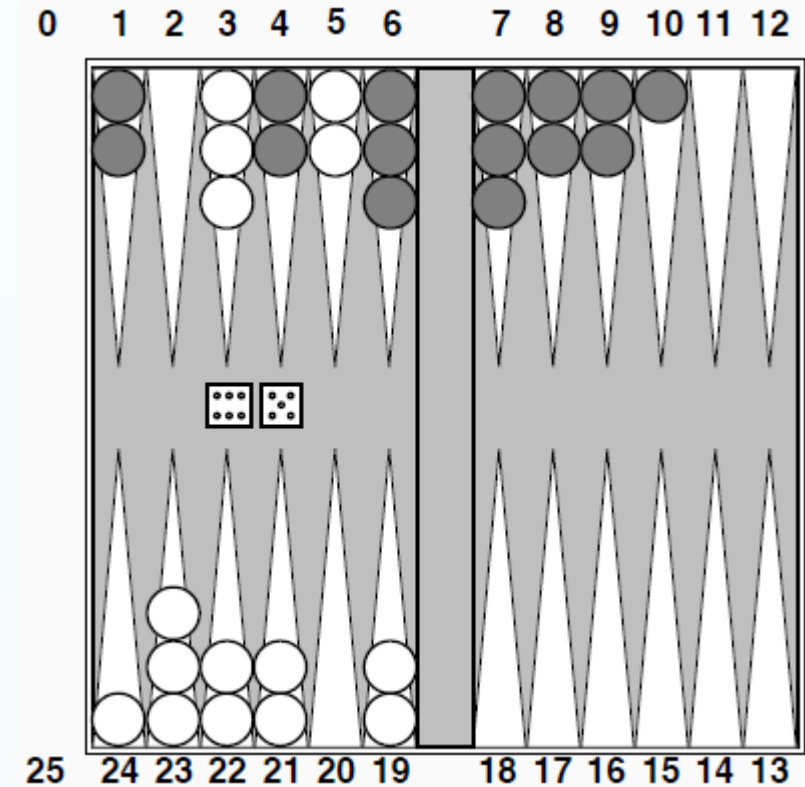
Minimax appliqué à un coup de MAX presque à la fin de la partie Nilsson, 1971)



Jeux stochastiques (non déterministes)

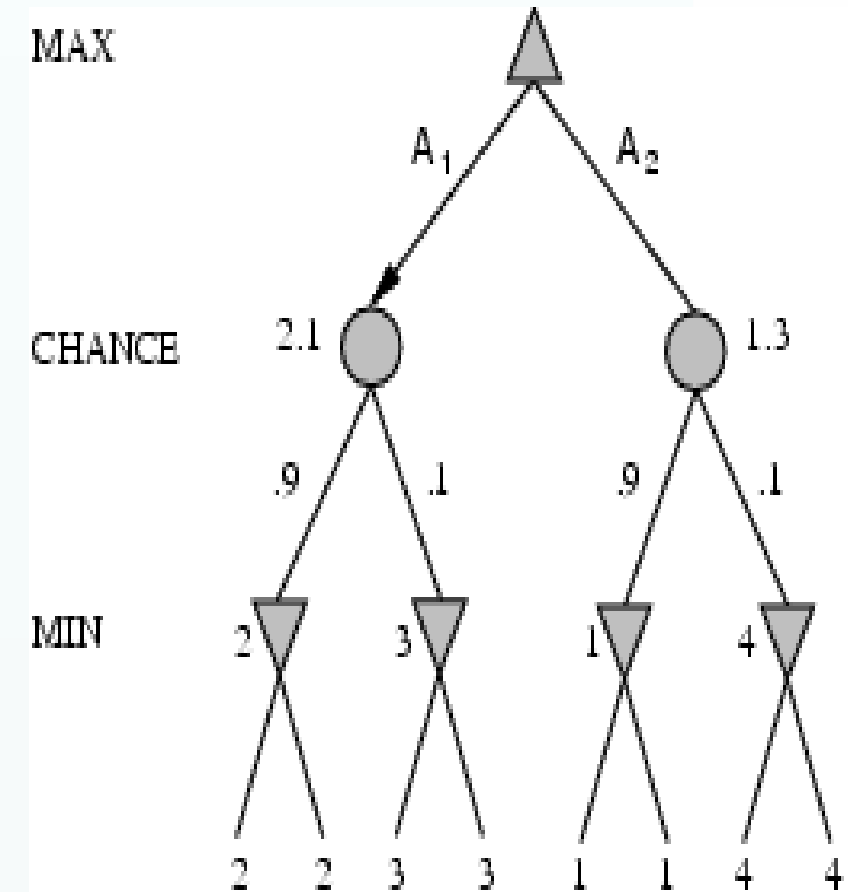
Jeux stochastiques

- ❑ Les jeux stochastiques sont imprévisibles qui font intervenir un élément aléatoire comme un lancer de dés ou les actions des fantômes dans pacman.
- ❑ Le backgammon est un jeu typique combinant hasard et savoir faire.

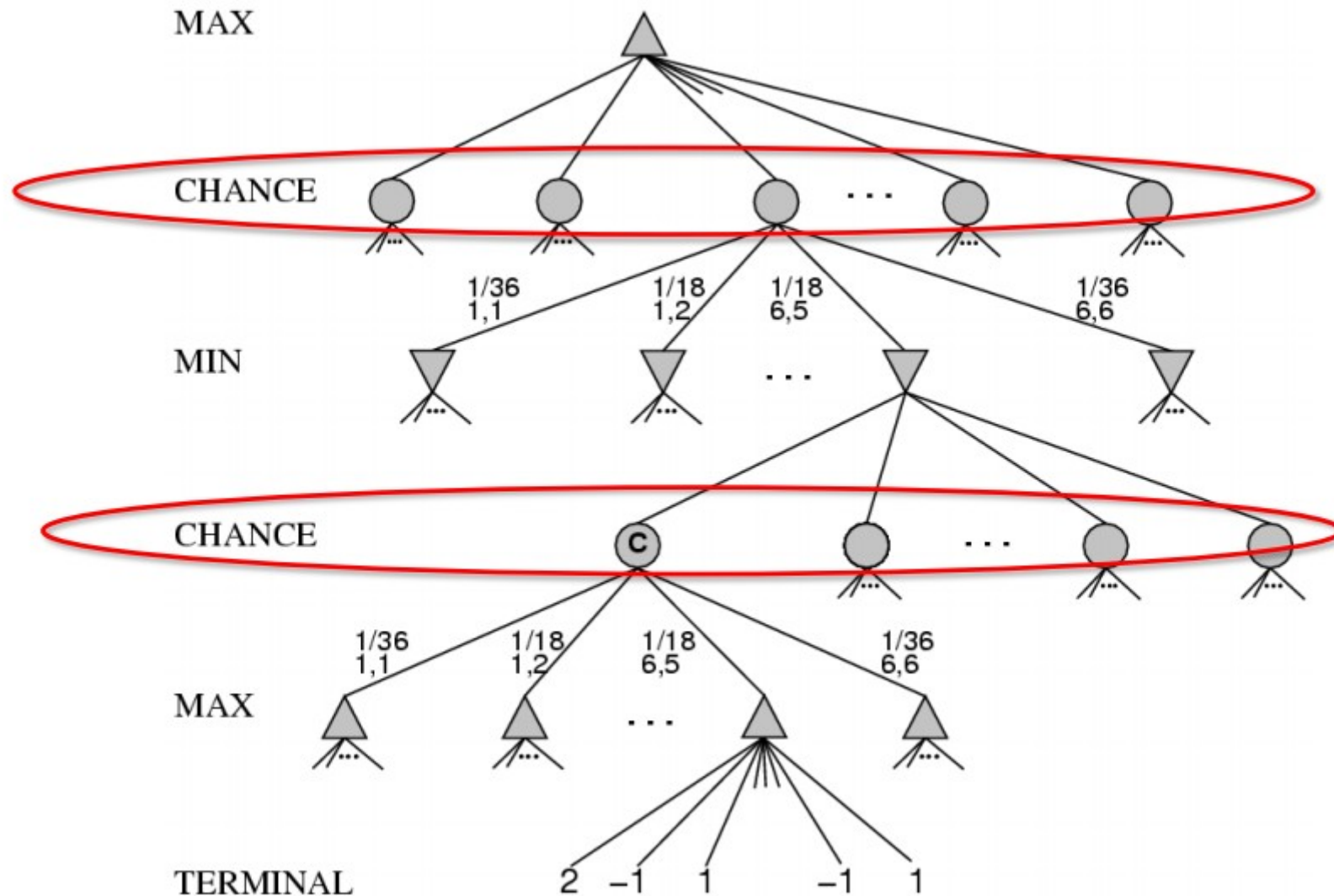


Arbre de jeu pour les jeux non déterministes

- Un arbre de jeu pour les jeux stochastiques doit inclure des nœuds de hasard en plus des nœuds MAX et MIN.
- Ces nœuds de hasards indiquent l'élément aléatoire comme un lancer de dés.
- L'utilité d'un nœud de chance est **l'utilité espérée**, c.-à-d., la moyenne pondérée des utilités de ses enfants (les valeurs sont pondérées par la probabilité de chaque tirage).
- Exemple : le nœud $2,1 = 0,9*2+0,1*3$.



Arbre de jeu d'une position du backgammon



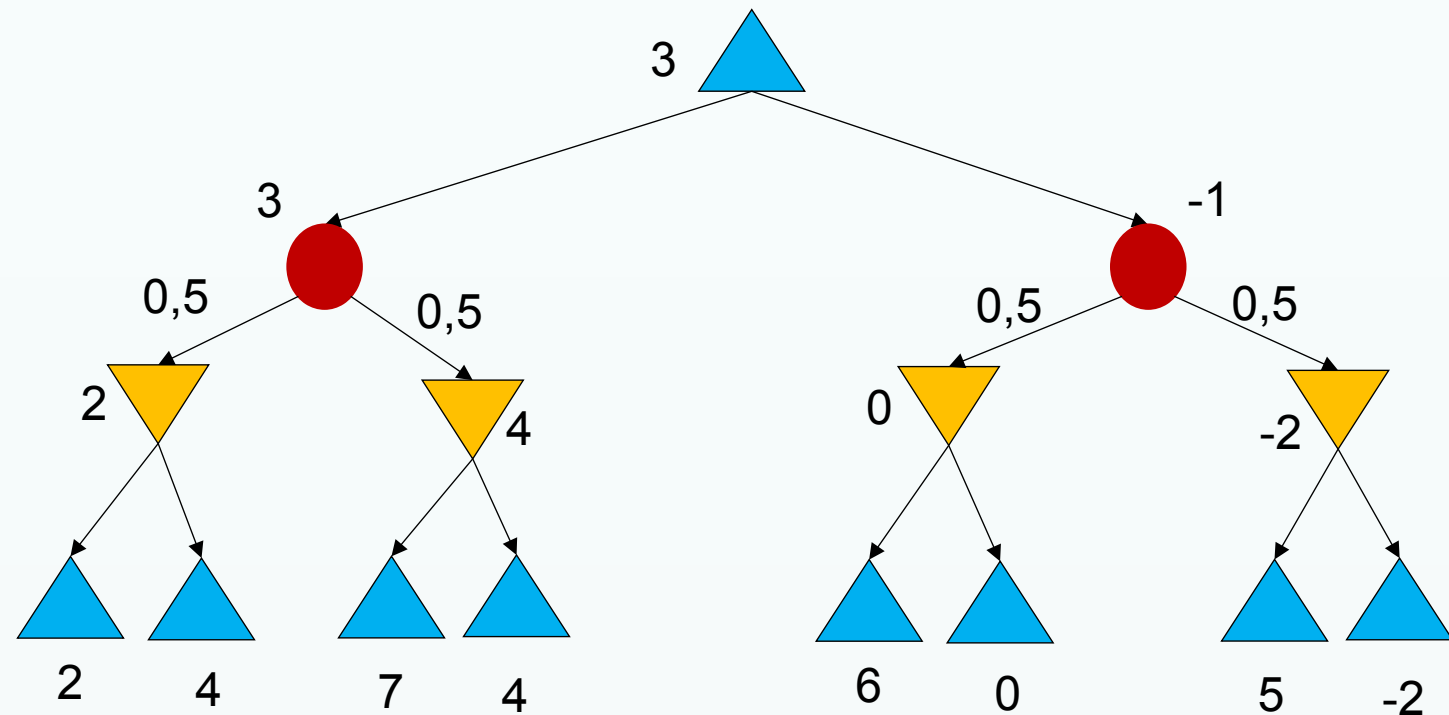
Arbre de jeu pour un lancer d'une pièce de monnaie

MAX

CHANCE

MIN

ETATS TERMINAUX



Algorithme EXPECTIMINMAX

- Un model probabiliste des comportements des opposants :
 - 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 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.

Algorithme EXPECTIMINMAX

EXPECTIMINMAX(s) =

UTILITÉ(s) Si n est un nœud terminal

$\max_a \text{EXPECTIMINMAX}(\text{RÉSULTAT}(s, a))$

Si n est un nœud Max

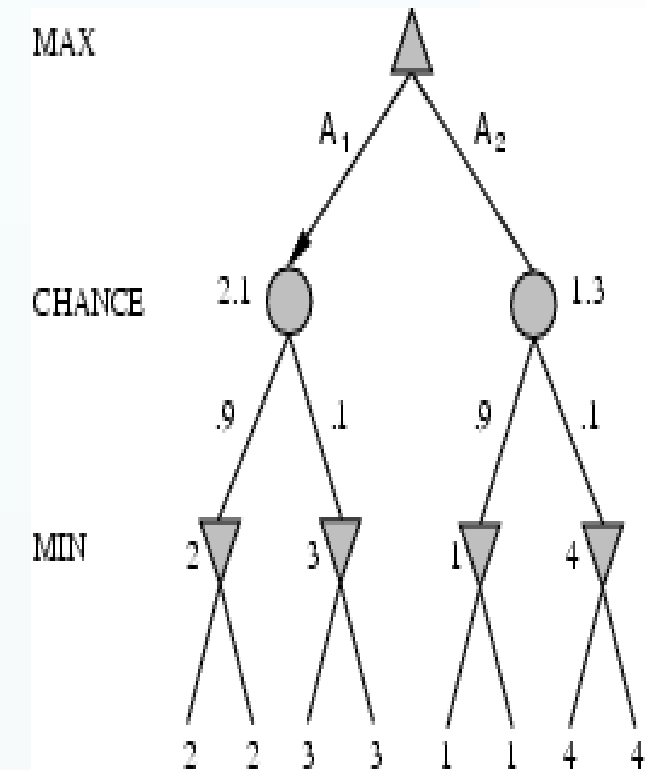
□ $\min_g \text{EXPECTIMINMAX}(\text{RÉSULTAT}(s, a))$
Si n est un nœud Chance

□ $\text{RÉSULTAT}(s, r)$ est l'état avec le fait que le résultat du lancer est r .

□ Ces équations donnent la programmation récursive des valeurs jusqu'à la racine de l'arbre.

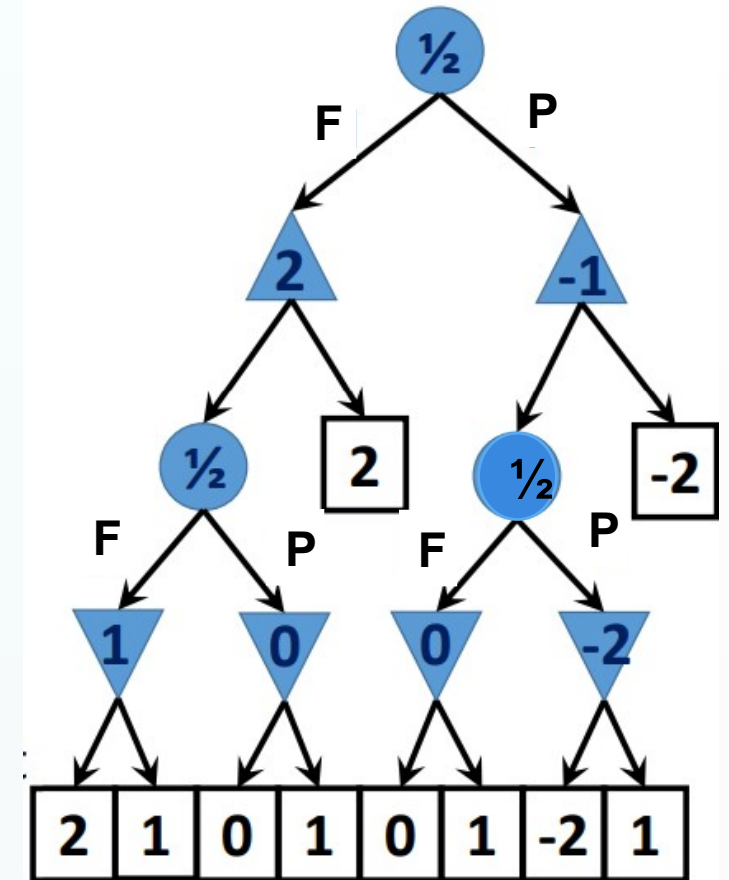
Si n est un nœud chance

□ Attention quand au choix de la fonction d'utilité !



Exemple Expectiminimax

- ❑ CHANCE : Max tire à pile ou face
- ❑ Max : Max arrête de jouer ou décide de continuer
 - ❑ S'il arrête à face : le jeu se termine, Max gagne (valeur = 2D).
 - ❑ S'il arrête à pile : le jeu se termine, Max perd (valeur = -2D)
- ❑ CHANCE : Min tire à pile ou face
 - ❑ FF : valeur = 2D
 - ❑ PP : valeur = -2D
 - ❑ FP ou PF : valeur = 0
- ❑ MIN : Min décide ou bien de garder la valeur présentée ci-haut ou de payer une pénalité (valeur = 1D)



Quelques exploits

- ❑ 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!

Conclusion

- ❑ La recherche sur les jeux révèle des aspects fondamentaux applicables à d'autres domaines
- ❑ La perfection est inatteignable dans les jeux : il faut approximer
- ❑ Alpha-bêta a la même valeur pour la racine de l'arbre de jeu que minimax
- ❑ Dans le pire des cas, il se comporte comme minimax (explore tous les nœuds)
- ❑ Dans le meilleur cas, il peut résoudre un problème de profondeur 2 fois plus grande dans le même temps que minimax