INF4230 – Intelligence Artificielle

Algorithmes de recherche pour des jeux avec des adversaires (*adversarial search*)

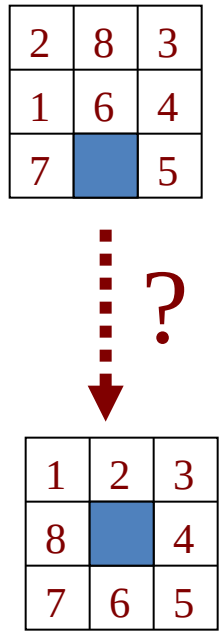
Hiver 2017

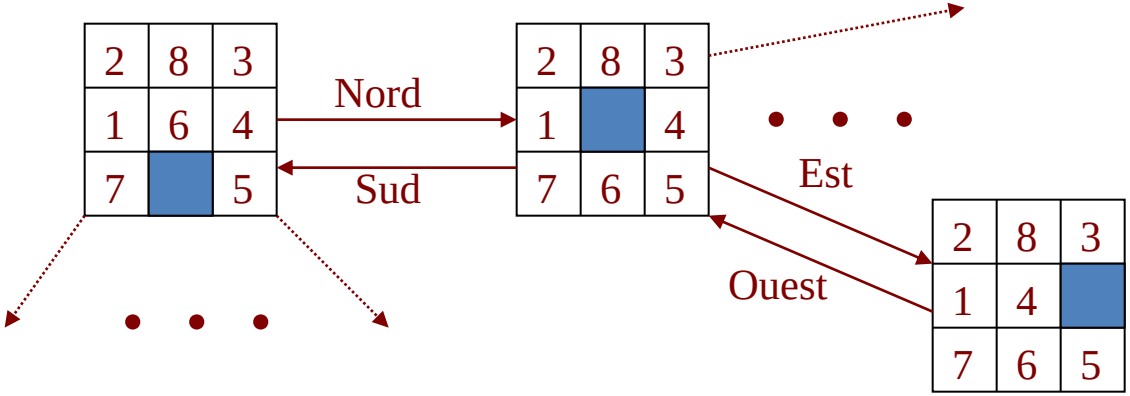
Objectifs

- Se familiariser avec les sujets suivants :
  - Jeux entre deux adversaires.
  - Algorithme minimax.
  - Élagage alpha-beta.
  - Décisions imparfaites en temps réelle.
  - Généralisation à 3+ joueurs.
  - Généralisation à l'incertitude.
- Se préparer au TP2
  - Implémentation d'un joueur artificiel pour un jeu.

Rappel sur Recherche et A*

- Notion d'espace d'états (configurations).
- État initial.
- Fonction de transition (successeurs).
- Fonction de but (configuration(s) finale(s)).





Rappel sur les environnements

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

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(s) sont les hypothèse(s) non satisfaite(s)?

R **:** Deux façon de voir les choses. (1) Dans les jeux, le monde n'est pas statique, mais **dynamique.** Le joueur adverse peut modifier l'environnement. (2) Plus spécifiquement, environnement **multi-agent** (l'autre joueur est agent non contrôlable et compétitif).

- Q : Comment peut-on résoudre ce problème ?

R : Supposer que l'adversaire joue de façon rationnelle…

Relation entre les joueurs

- Dans un jeu, des joueurs peuvent être :
  - Coopératifs.
    - Ils veulent atteindre le même but.
  - **En compétition direct (avec adversaires).**
    - 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, Connect5, etc.
    - C'est le type de jeux qui nous intéresse aujourd'hui.
  - Mixte.
    - Il y a tout un spectre entre les jeux purement coopératifs et les jeux avec adversaires. Exemple : jeux d'alliances.

Hypothèses

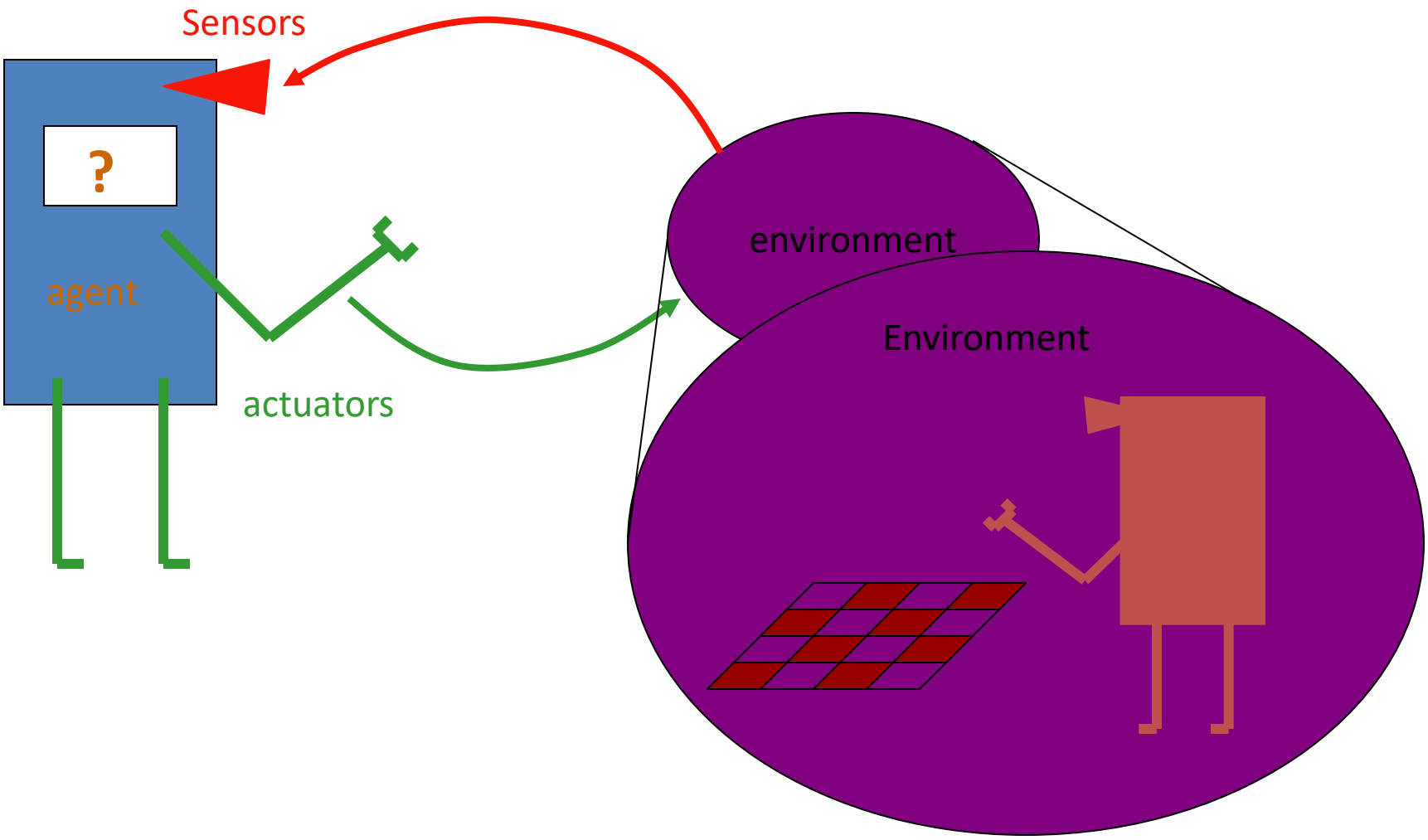
- Théorie des jeux (*Game Theory*) = sujet large.
- Pour l'instant, nous aborderons les :
  - Jeux à deux adversaires.
  - Jeux discrets à tour de rôle.
  - Jeux à somme nulle.
  - Jeux avec observation totale.
  - Jeux déterministes (sans hasard ou incertitude).
- Il sera possible de généraliser plus tard.

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
- Premiers algorithmes de jeu
  - algorithme pour jeu parfait, J. von Neumann (1944)
  - horizon fini, évaluation/approximation, K. Zuse (1945), C. Shannon (1950), A. Samuel (1952)
  - réduction de coût par élagage, McCarthy (1956)

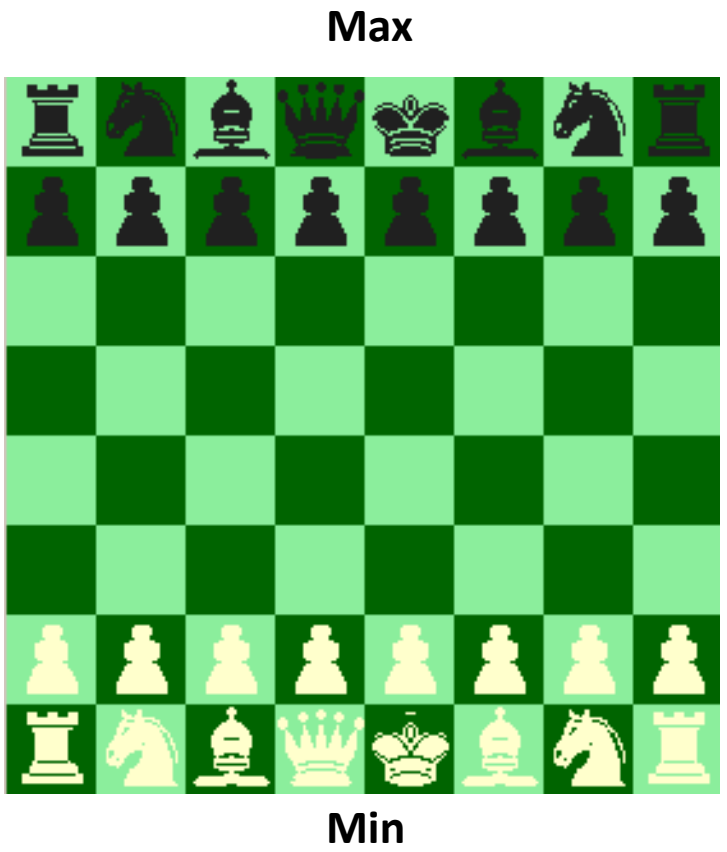
- Mais il y a d'importantes différences avec un problème standard de recherche

Agent Joueur



Jeux entre deux adversaires

- Noms des joueurs :
  Max *vs* **Min.**
  - **Max** est le premier* à jouer («notre joueur»).
  - **Min** est son adversaire.

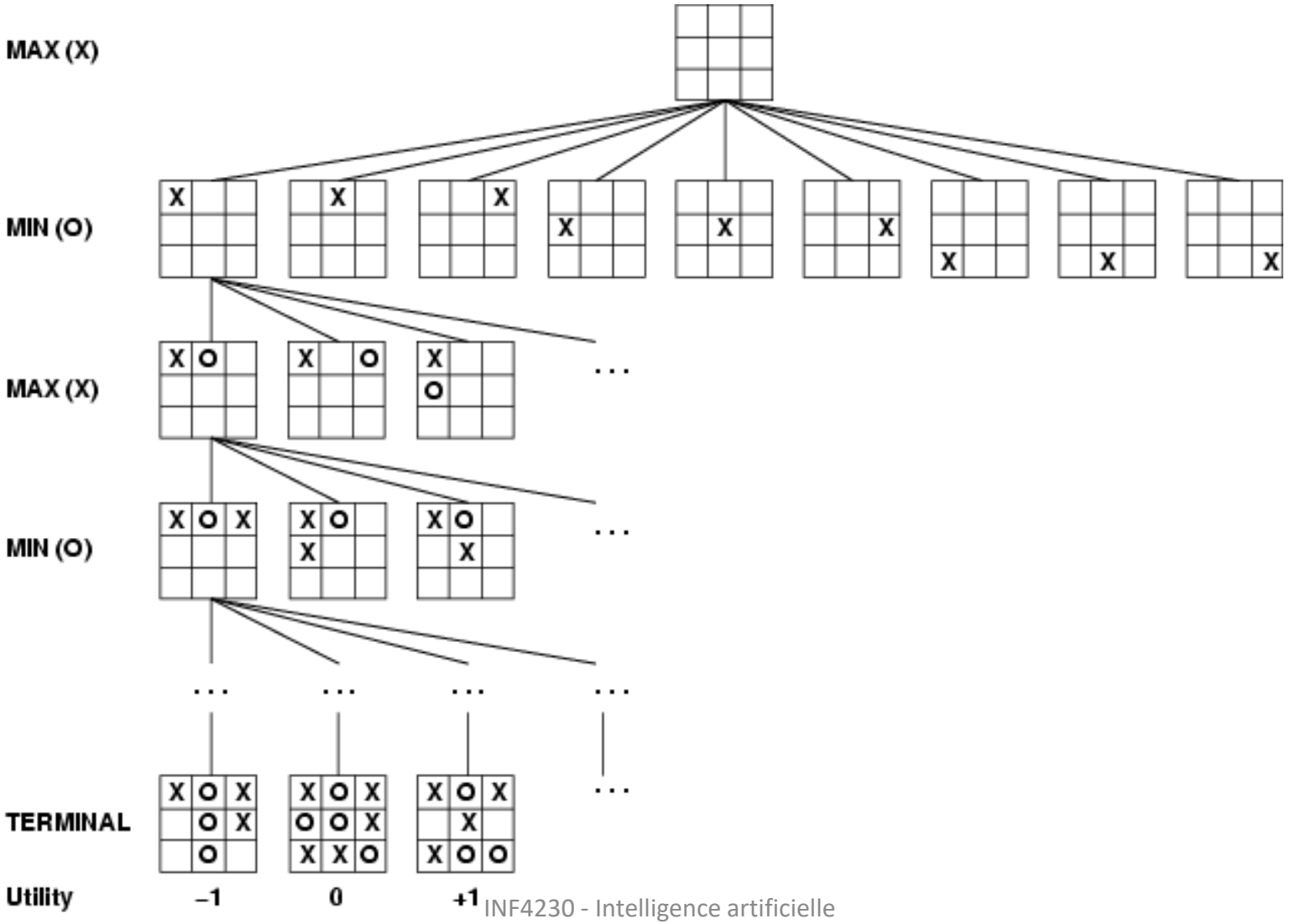


* *«Premier» au sens «prochain» dans la situation courante.*

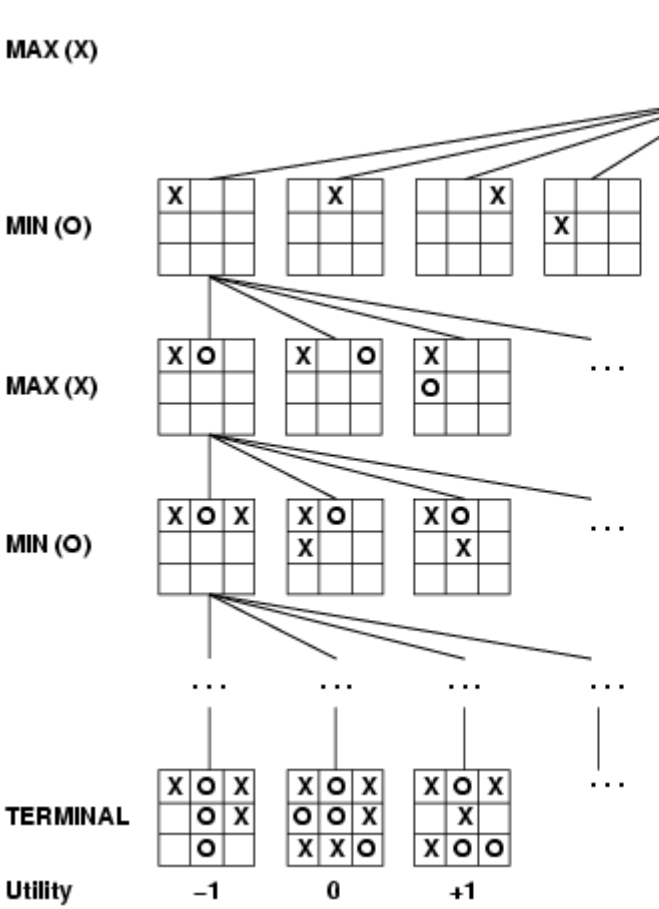
Arbre de recherche

- Problème de décision dans un jeu → problème de recherche dans un arbre :
  - Racine = Un état (configuration) initial.
  - Un ensemble d'opérateurs (coups légaux ou actions légales).
    - Ces opérateurs génèrent des transitions dans l'arbre.
  - Un test de terminaison.
    - Indique si le jeu est terminé.
  - Une fonction d'utilité pour les états finaux.

Arbre de recherche tic-tac-toe



Arbre de jeu (2-joueurs)



Max et Min sont les 2 joueurs. Un gain pour Max est une perte pour Min et vice-versa.
+1 pour un gain, -1 pour une perte, 0 pour un nul.

La somme des valeurs de la fonction d'évaluation pour les 2 joueurs est nulle à la fin du jeu.

Il faut disposer d'une fonction d'évaluation statique capable de mesurer la qualité d'une configuration par rapport à un joueur (gén. Max)
car il n'est pas possible de produire tout l'arbre de recherche jusqu'à la fin du jeu, càd au moment où la décision gain/perte/nul est claire.

Minimax

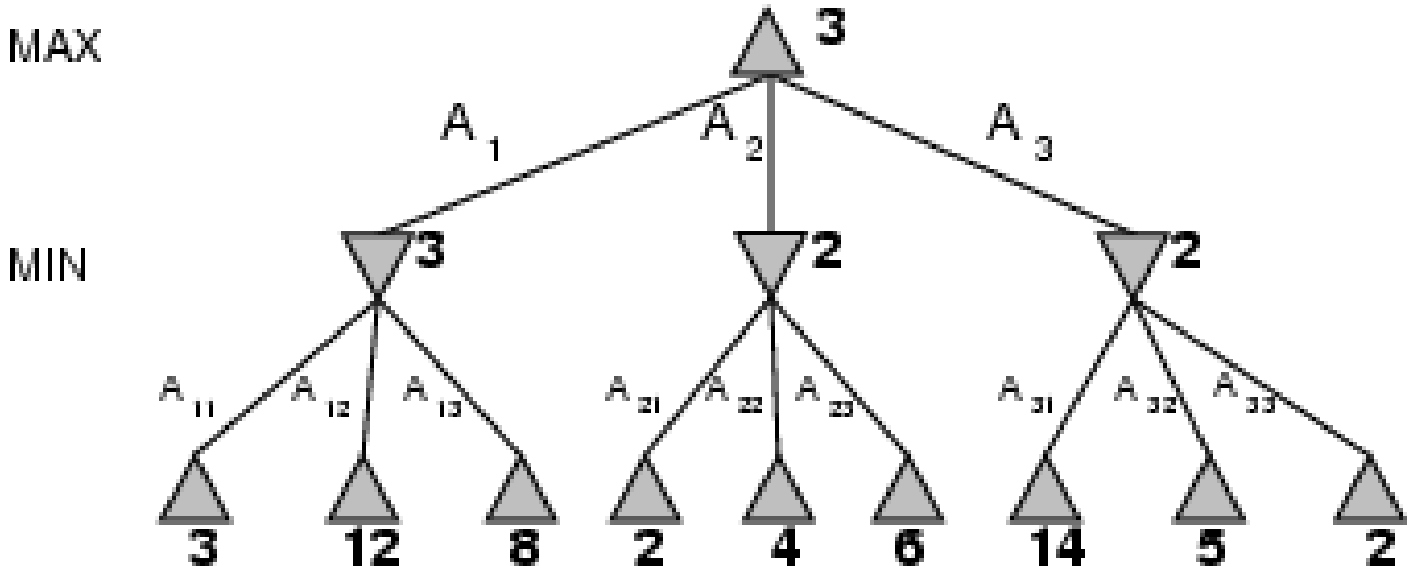
- **Objectif:** Trouver le coup parfait pour un jeu déterministe à information parfaite

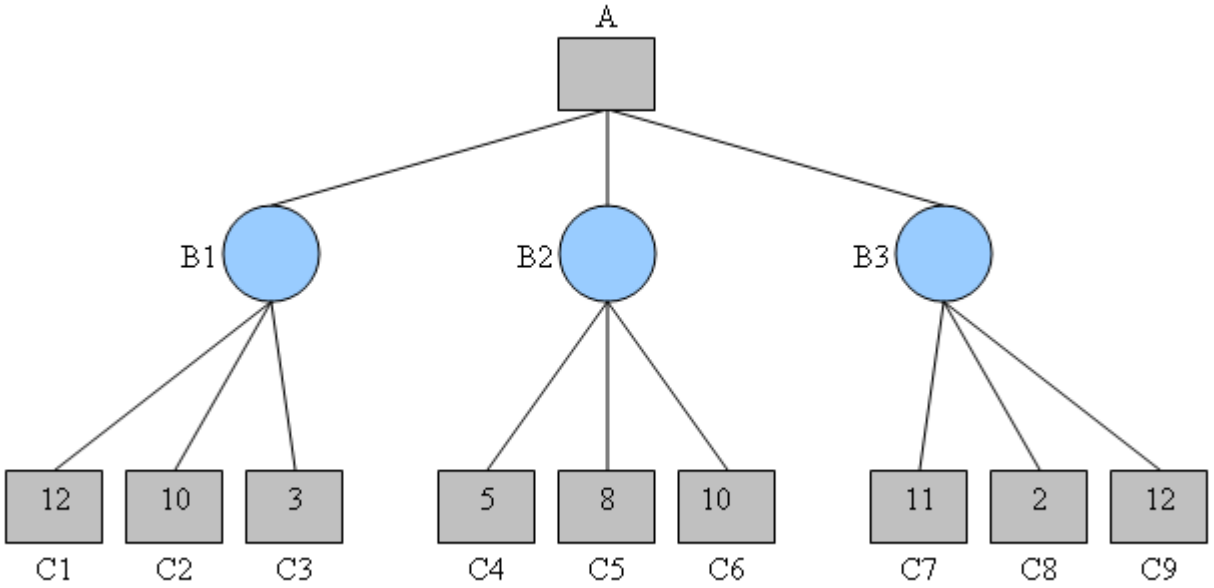
- **Principe:** maximiser la valeur d'utilité minimax pour Max avec l'hypothèse que Min joue parfaitement pour la minimiser; Ceci revient à choisir le coup qui mène vers l'état qui a la meilleure valeur possible contre le meilleur jeu de l'adversaire.
  1) étendre l'arbre de jeu
  2) calculer la valeur 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

- **Algorithme:** On visite l'arbre de jeu pour faire remonter à la racine une valeur (appelée « **valeur du jeu** ») qui est calculée récursivement de la façon suivante :
  - ***minimax(p) = f(p)*** *si p est une feuille de l'arbre où f est une fonction d'évaluation de la position du jeu*
  - ***minimax(p) = MAX(minimax(O1), …, minimax(On))*** *si p est un nœud Joueur avec O1…On comme fils*
  - ***minimax(p) = MIN(minimax(O1), …, minimax(On))*** *si p est un nœud Joueur avec O1…On comme fils*

Minimax

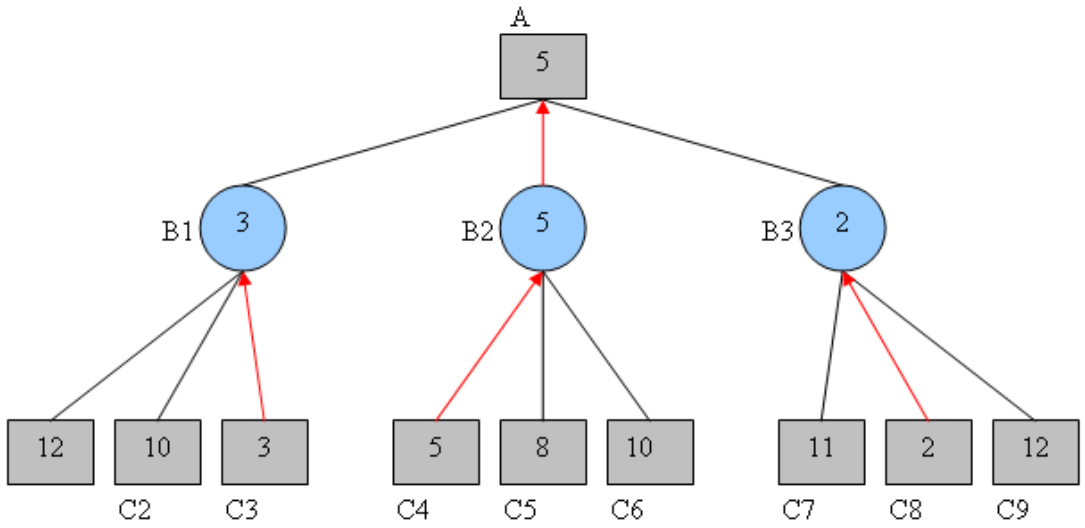
- Exemples

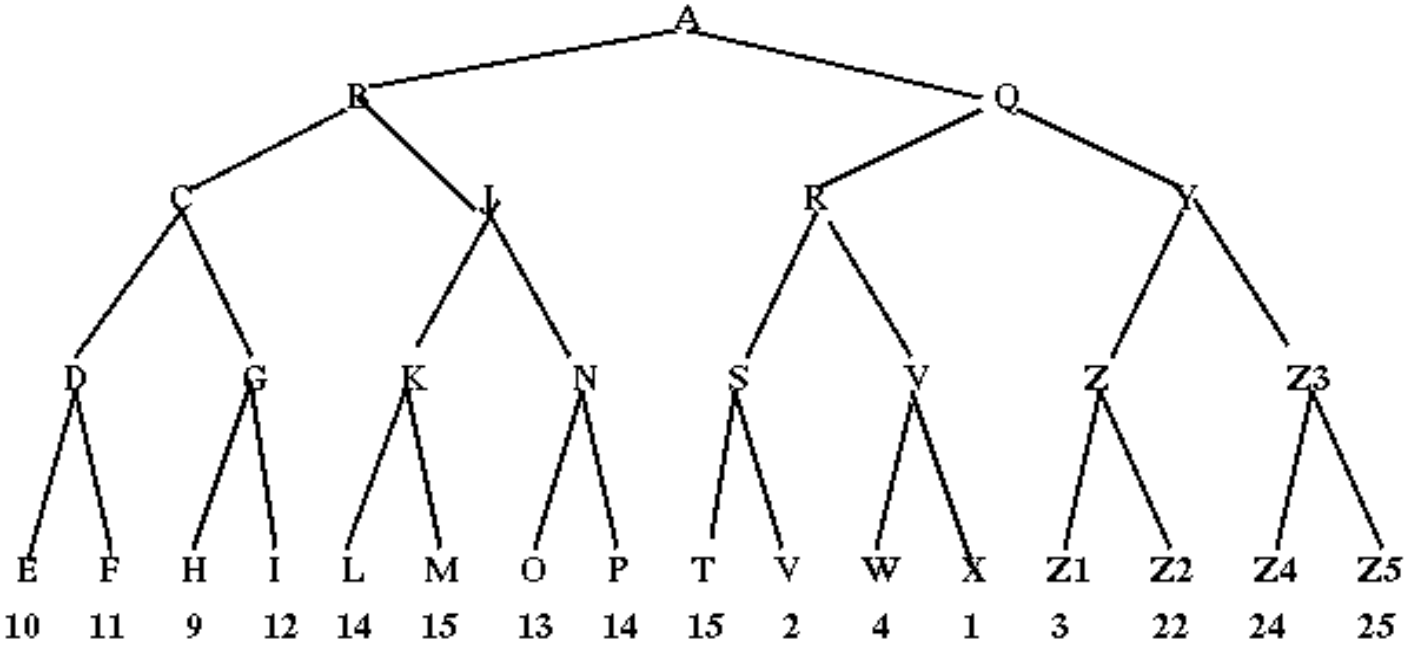




Minimax

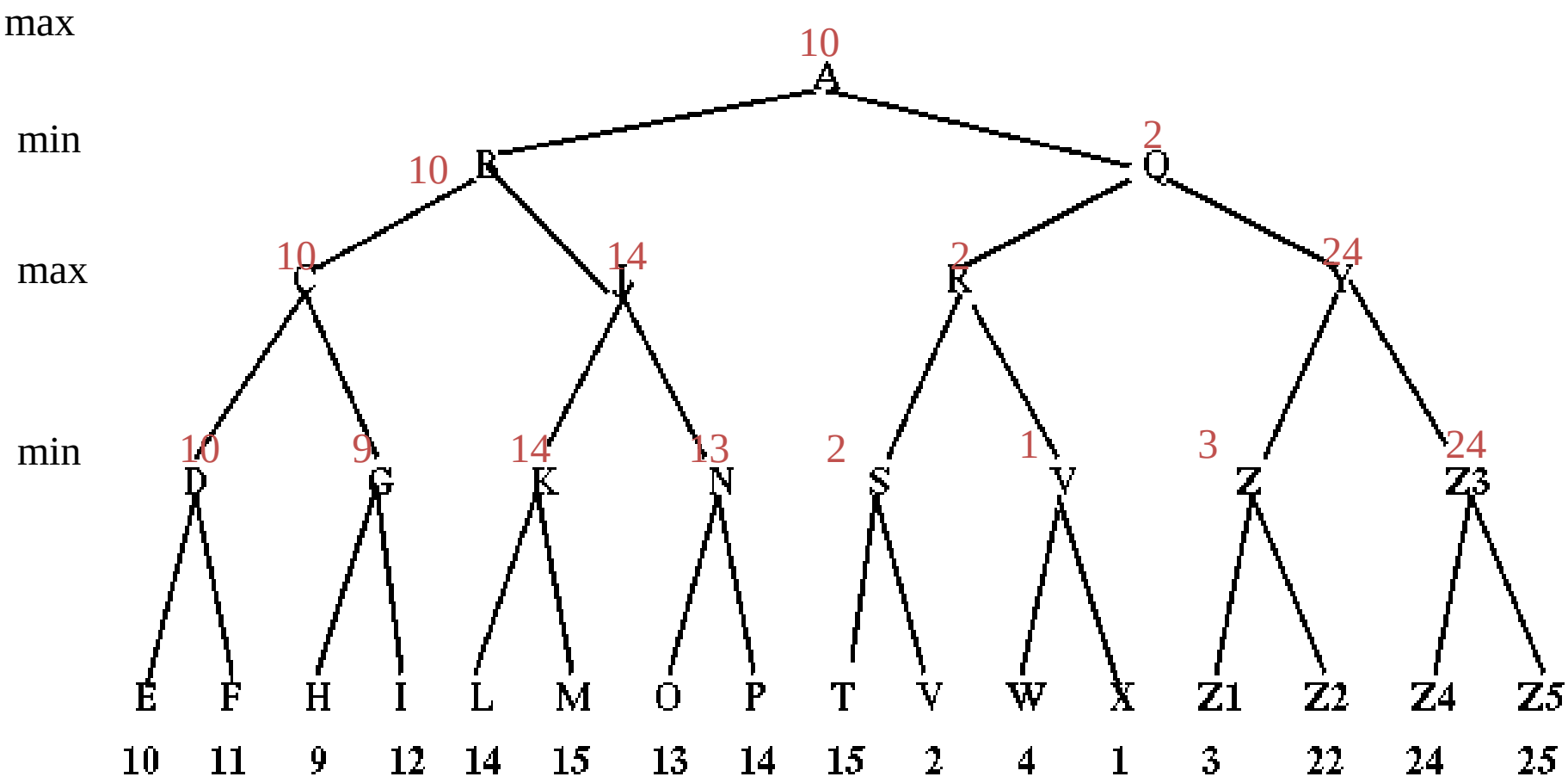
- Exemples





Minimax

- Exemples



Récapitulons: Minimax

Étant donné un arbre de jeu, la stratégie optimale peut être obtenue par le calcul comme suit d'une valeur minimax pour chaque noeud:

MINIMAX-VALUE(n)=

UTILITY(n)	*si n est un noeud feuille*
$max_{s \in successors(n)}$ *MINIMAX-VALUE(s)*	*si n est un noeud max*
$min_{s \in successors(n)}$ *MINIMAX-VALUE(s)*	*si n est un noeud min*

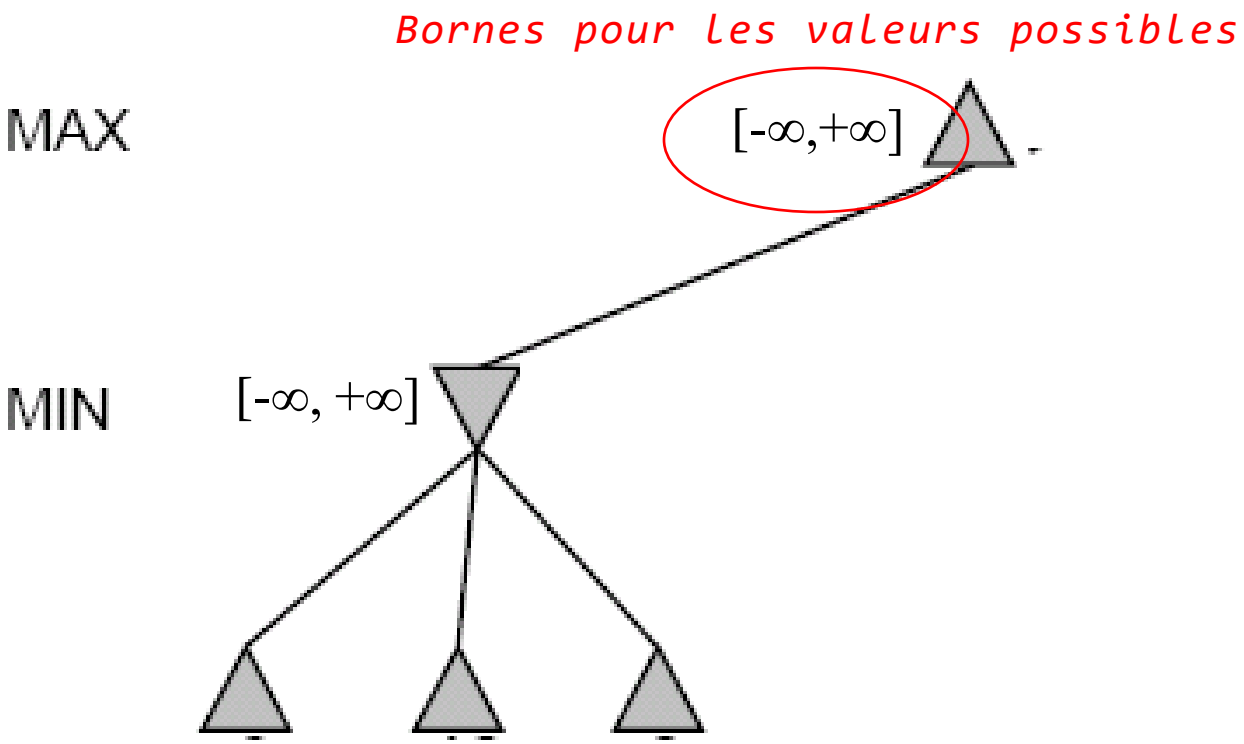
Propriétés de minimax

- Complet ?
  - **Oui** (si l'arbre est fini).
- Optimal ?
  - **Oui** (contre un adversaire qui joue de façon optimale).
- Complexité en temps ?
  - $O(b^m)$:
    - b: le nombre maximum coups (actions) légaux à chaque étape.
    - m: nombre maximum de coups dans un jeu (profondeur maximale de l'arbre).
- Complexité en espace mémoire ?
  - ***O(bm)***, vue qu'on fait une recherche en profondeur.

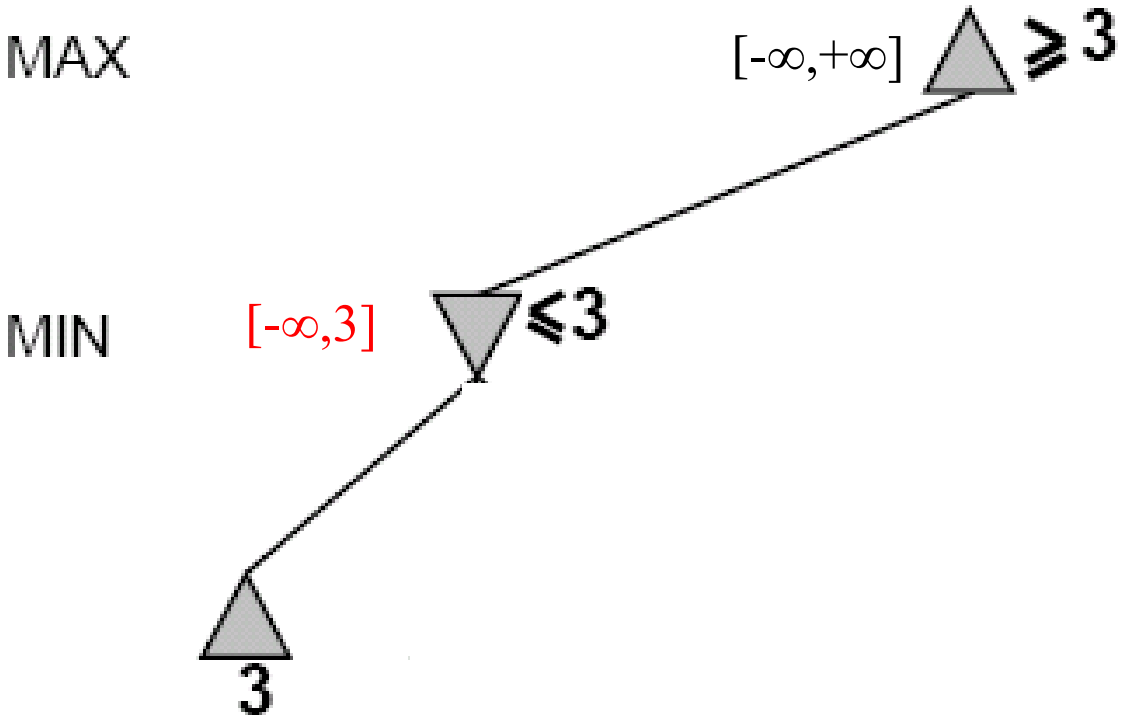
- Pour le jeu d'échecs : $b \approx 35$ et $m \approx 100$ pour une partie « raisonnable ».
  - Il n'est pas réaliste d'espérer trouver une solution exacte en un temps raisonnable. ➔ Problème non tractable (non-tractable problem).

MINIMAX AVEC ÉLAGAGE ALPHA-BETA

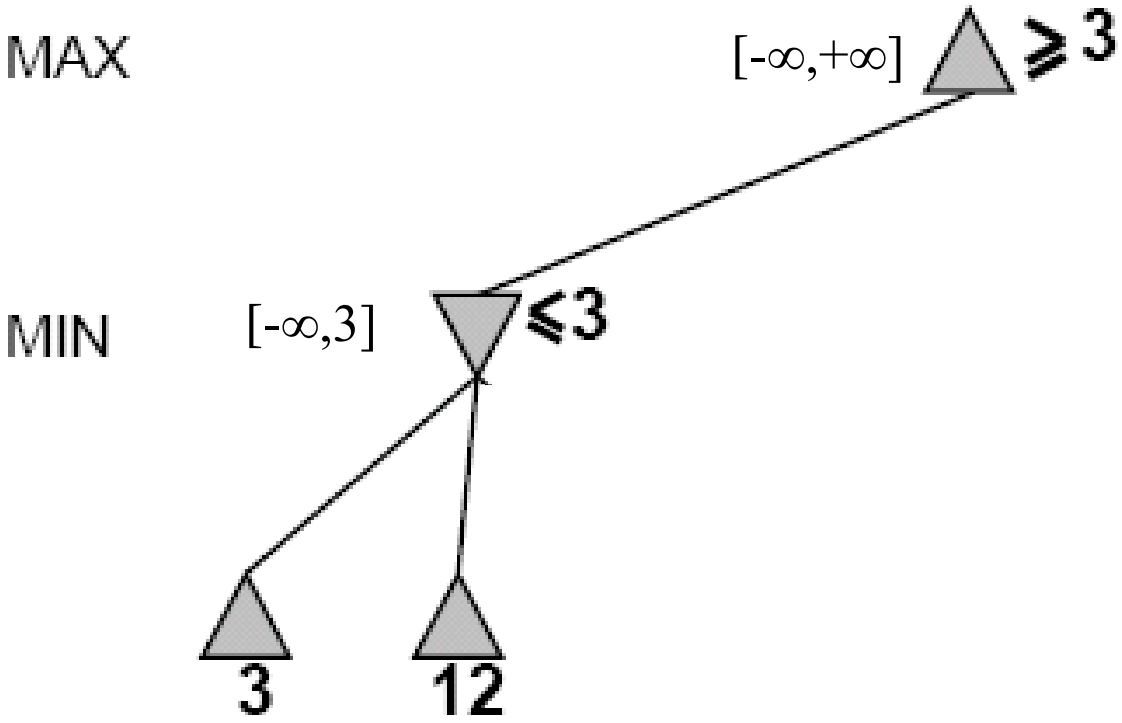
Élagage alpha-beta



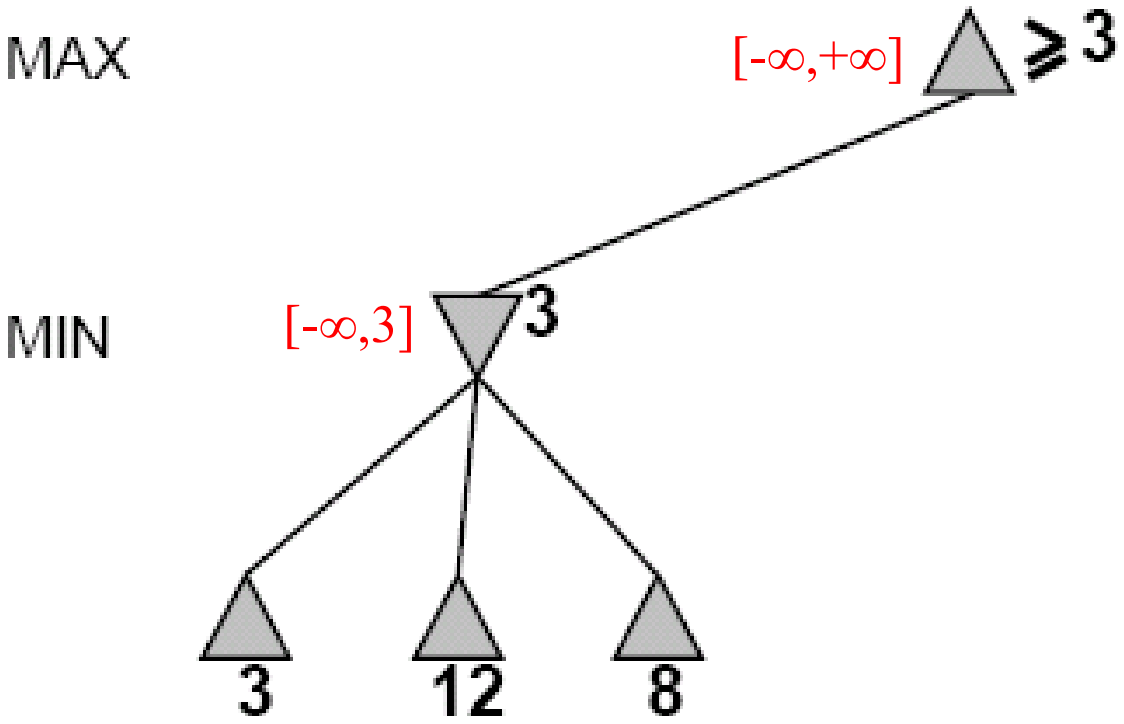
Élagage alpha-beta



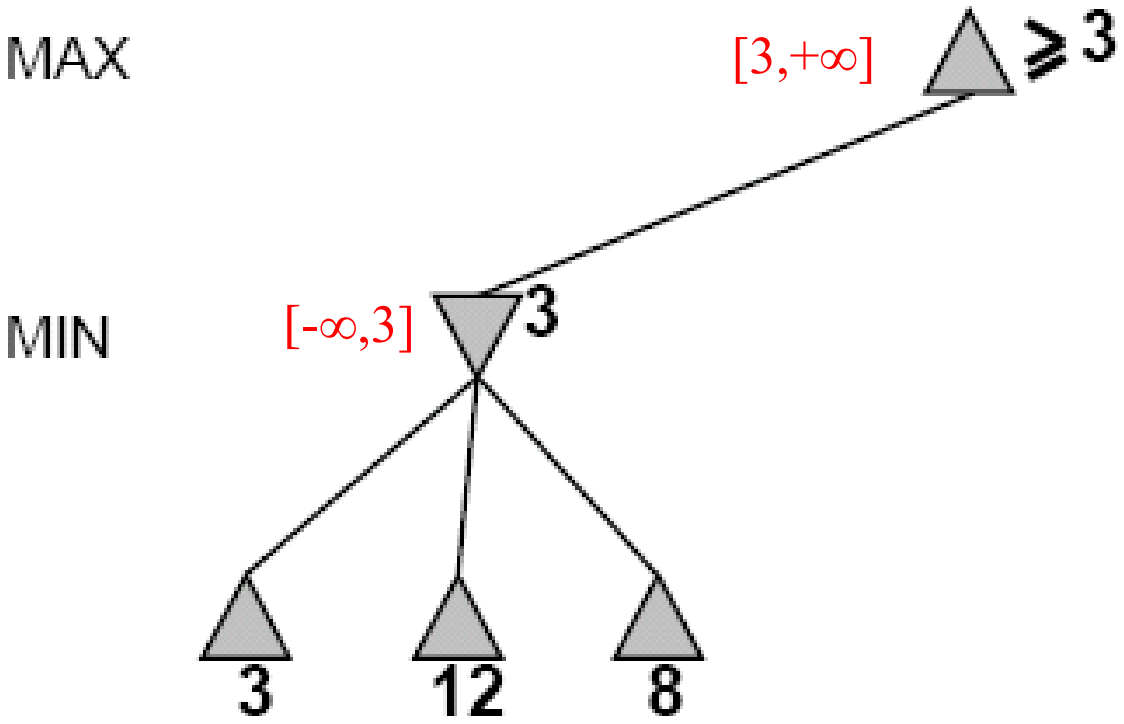
Élagage alpha-beta



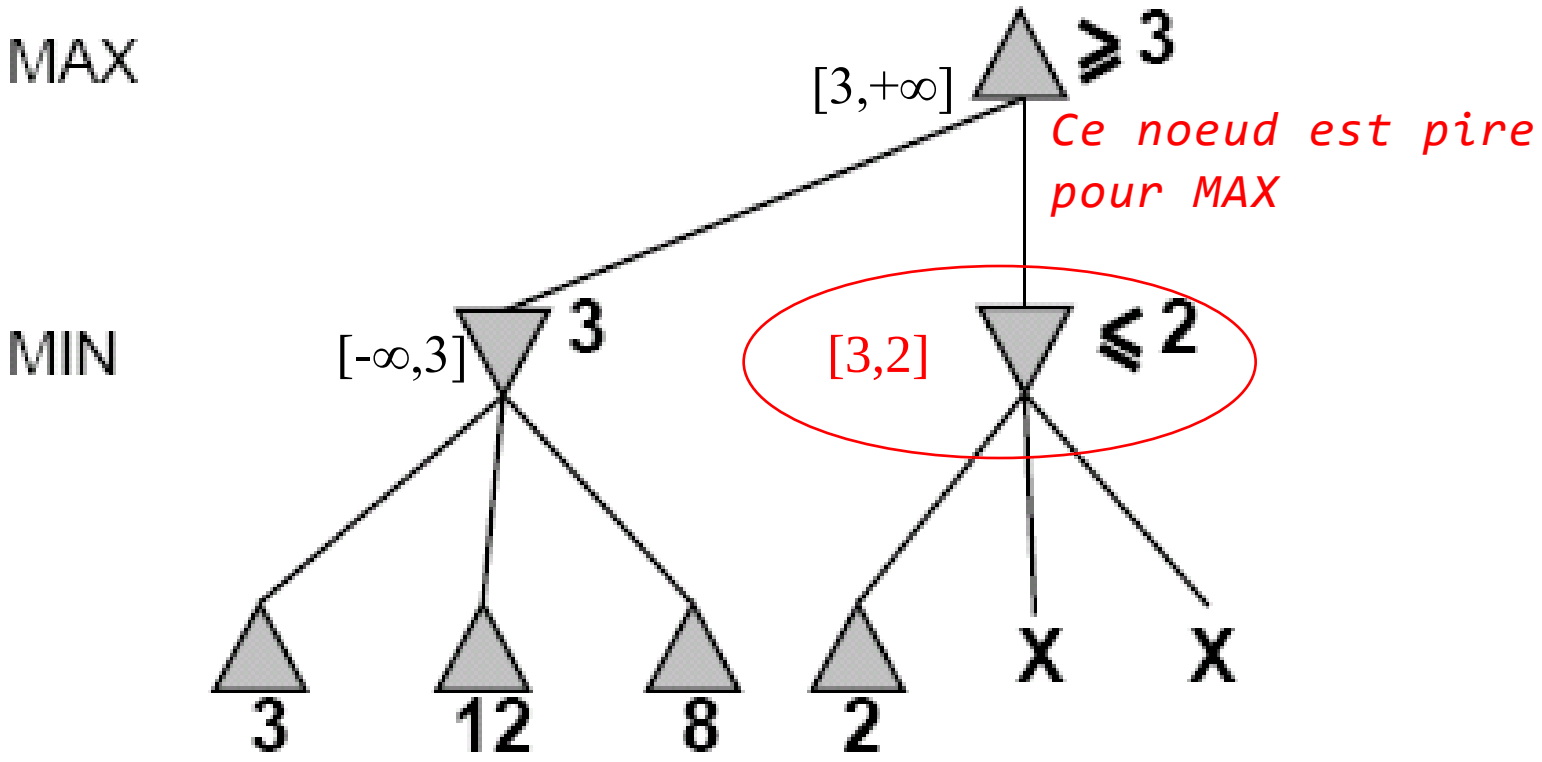
Élagage alpha-beta



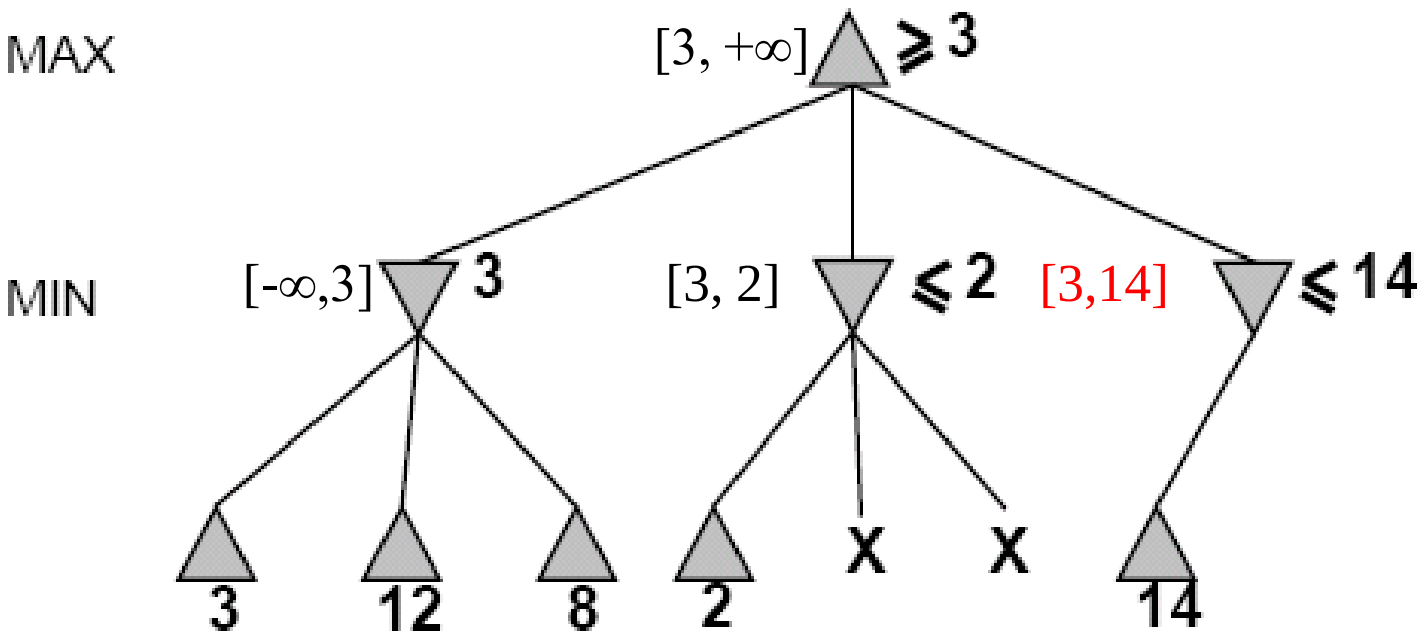
Élagage alpha-beta



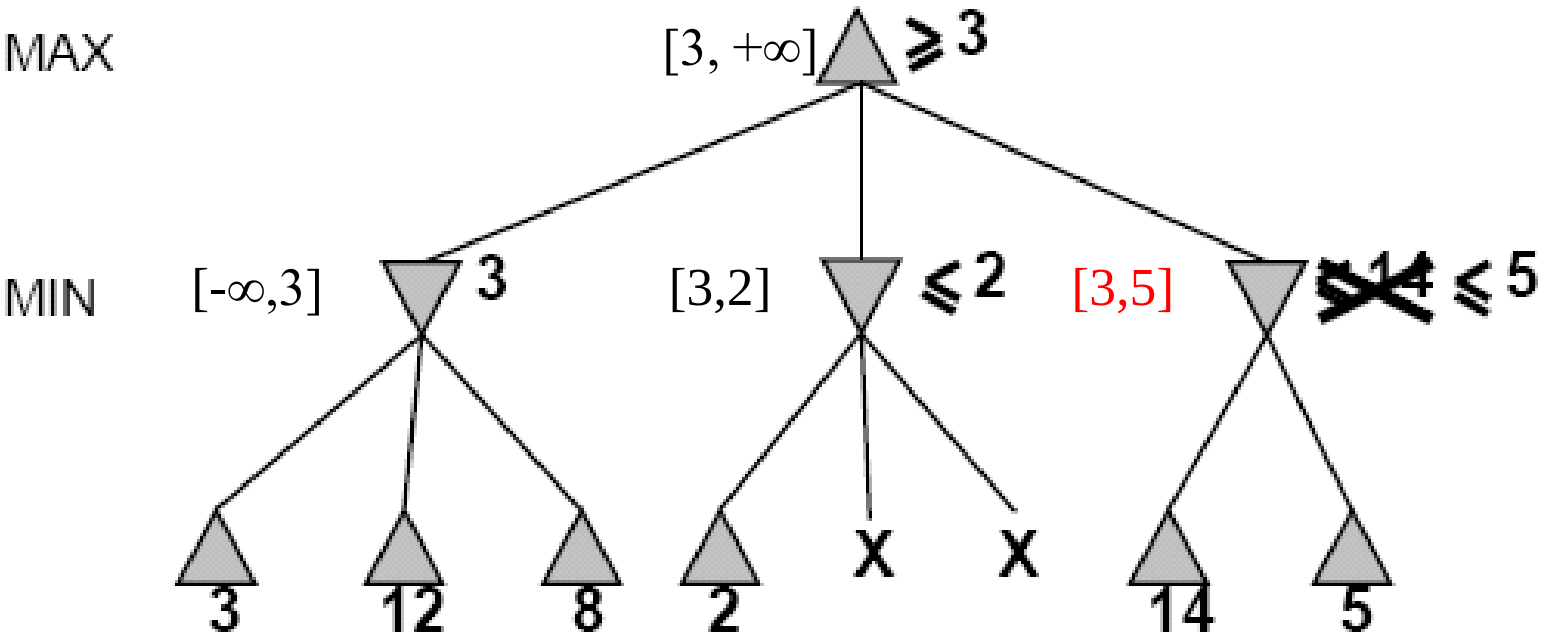
Élagage alpha-beta



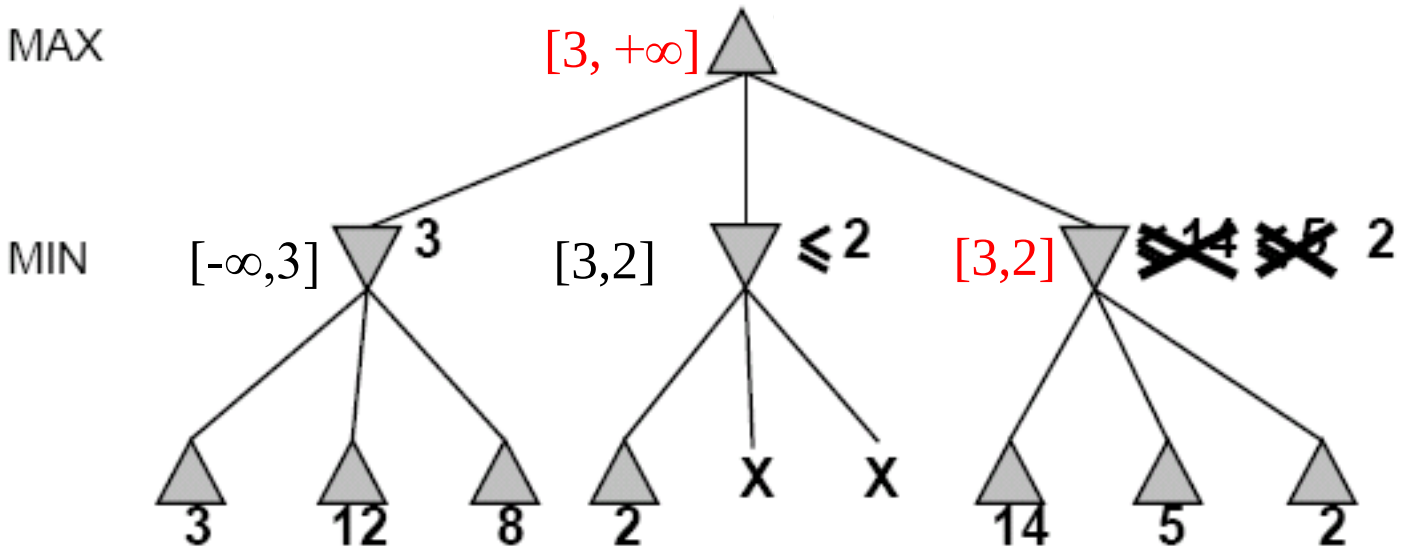
Élagage alpha-beta



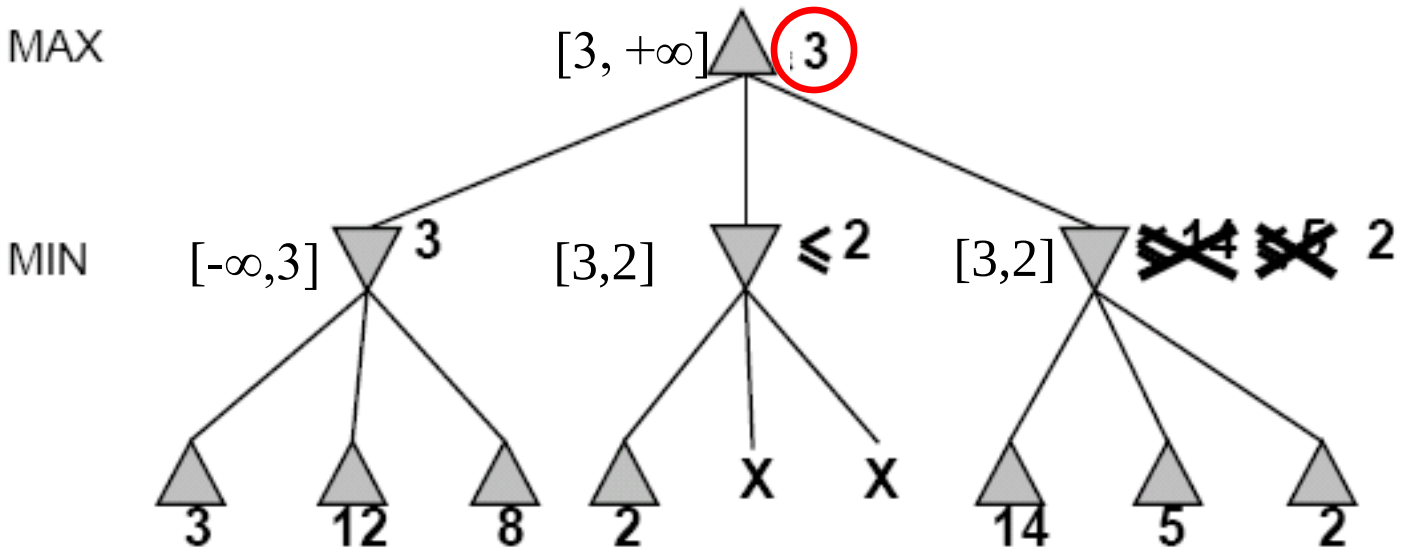
Élagage alpha-beta



Élagage alpha-beta

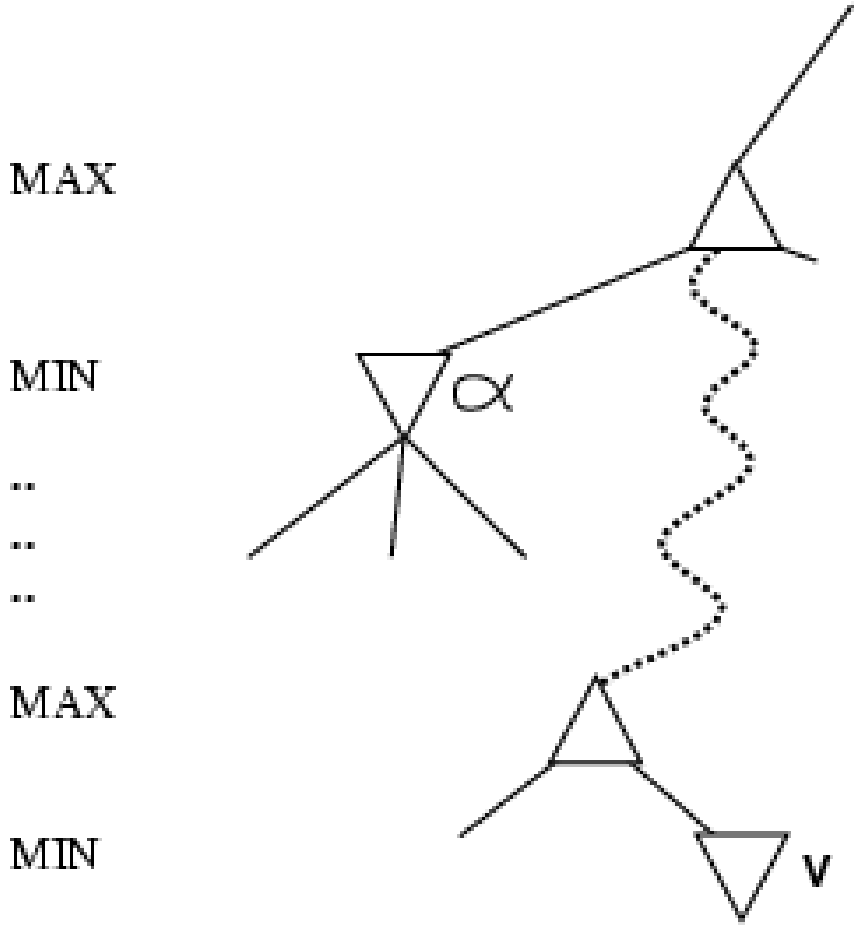


Élagage alpha-beta



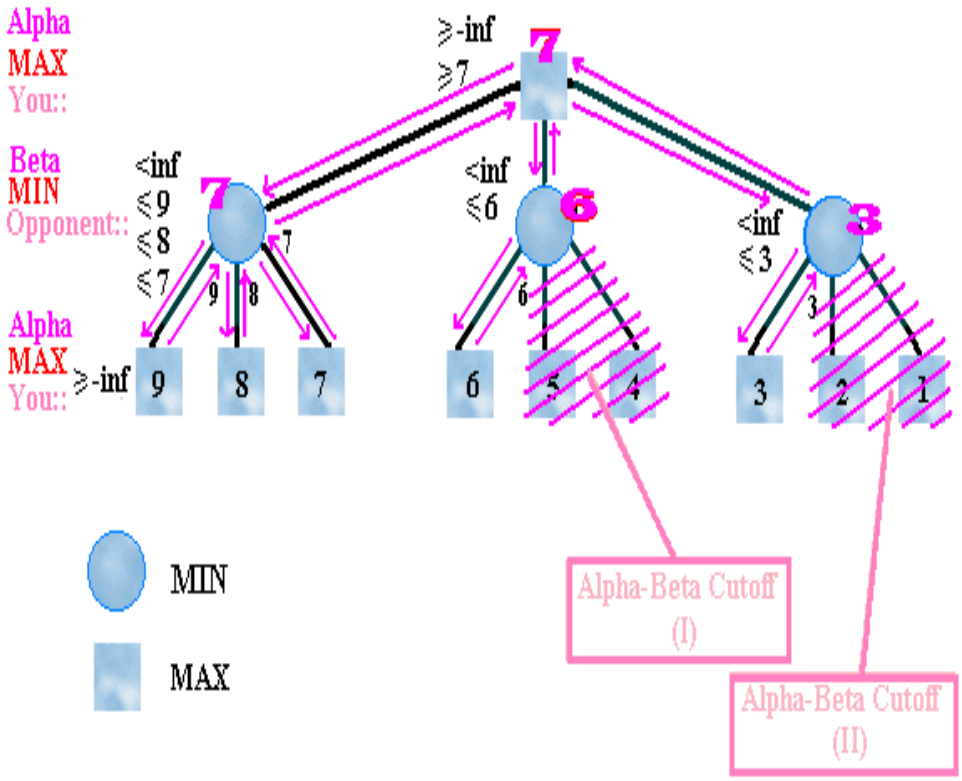
D'ou vient le nom alpha-beta ?

- L'algorithme alpha-beta tire son nom des paramètres α et β décrivant les bornes des valeurs d'utilité enregistrées durant le parcours.
  - α 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 α, *Max* n'ira jamais à *v: c*ouper la branche
  - β est défini de manière analogue pour *Min*.



Algorithme

```
fonction ALPHABETA(P, alpha, beta) /* alpha est toujours inférieur à beta */
  si P est une feuille alors
    retourner la valeur de P
  sinon
    si P est un nœud Min alors
      Val = infini
      pour tout fils Pi de P faire
        Val = Min(Val, ALPHABETA(Pi, alpha, beta))
        si alpha ≥ Val alors  /* coupure alpha */
          retourner Val
        beta = Min(beta, Val)
      finpour
    sinon
      Val = -infini
      pour tout fils Pi de P faire
        Val = Max(Val, ALPHABETA(Pi, alpha, beta))
        si Val ≥ beta alors /* coupure beta */
          retourner Val
        alpha = Max(alpha, Val)
      finpour
    finsi
  retourner Val
  finsi
fin
```



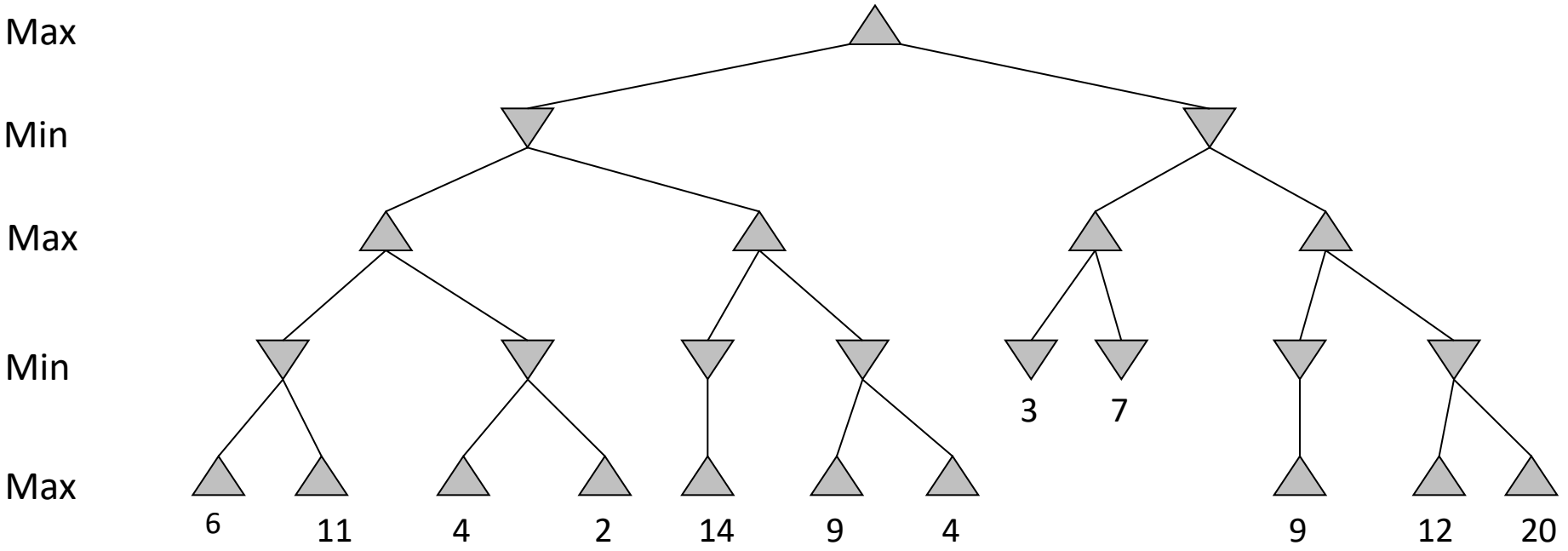
Algorithme avec élagage *alpha-beta*

```
function ALPHA-BETA-SEARCH(state) returns an action
    inputs: state, current state in game

    v ← MAX-VALUE(state, −∞, +∞)
    return the action in SUCCESSORS(state) with value v

---

function MAX-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state

    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s, α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v
```

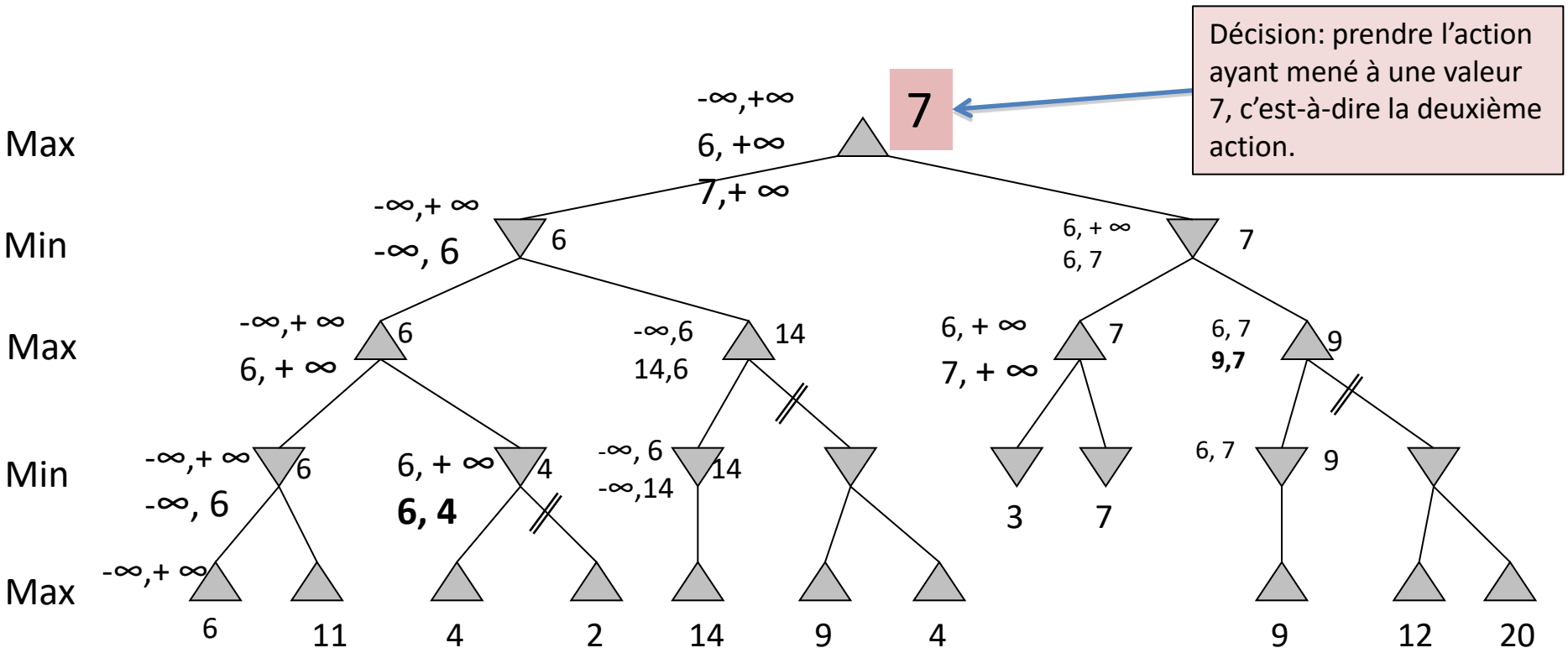
Algorithme avec élagage *alpha-beta (2)*

```
function MIN-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s, α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

Exercice



Exercice



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 l'algorithme *alpha-beta*

- **L'élagage n'affecte pas le résultat final de *minimax***.
- Dans le pire des cas, l'élagage alpha-beta (*alpha-beta prunning)* 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 (profondeur maximale de l'arbre).
- Un bon ordonnancement des actions à chaque nœud peut améliorer significativement l'efficacité.
- Dans le meilleur des cas (ordonnancement parfait), la complexité en temps est de $O(b^{m/2})$.
- Si le temps de réflexion est limité, la recherche peut être jusqu'à deux fois plus profonde comparé à *minimax*!
- Par exemple, si on peut explorer 104 nœuds par seconde et que l'on a 100 secondes
  - On peut regarder 106 nœuds par coup $\sim 35^{8/2}$
  - α-β peut facilement atteindre des profondeurs de l'ordre de 8 coups d'avance aux échecs
  - Ce qui est plutôt bon pour un programme d'´echecs

Version unifiée de l'algorithme minimax (*negamax*)

```
fonction SearchAlphaBeta(state, α, β)  /* α < β */
  if TerminalTest(state) then
     return Utility(state)
  else
     best ← -∞
     for (a,s) in Sucessors(state)
        v ← -SearchAlphaBeta(s,-β,-α)
        if v > best then
           best ← v
           if best > α then
              α ← best
              if α >= β then
                 return best
     return best
```

DÉCISIONS EN TEMPS RÉEL

Décisions imparfaites en temps réel

- En général, des décisions imparfaites doivent être prises en temps réel:
  - Supposons qu'on a 60 secs pour réagir et que l'algorithme explore 10^4 nœuds/sec
  - Cela donne 6×10^5 nœuds à explorer par coup.
- Approche standard
  - Couper la recherche
    - Exemple : limiter la profondeur de l'arbre.
    - Voir le livre pour d'autres idées.
  - Fonction d'évaluation
    - Estimer les configurations par rapport à leur «chance» (~~hasard~~) d'être gagnantes.

Exemple de fonction d'évaluation

- Pour le jeu d'échecs, une fonction d'évaluation typique est une somme pondérée des *features* estimant la qualité d'une configuration *s* :

$$\mathit{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

- Exemple:
  - $w_1 = 9$,
    $f_1(s)$ = (nb reines blanches) – (nb reines noires),
  - $w_2 = 5$,
    $f_2(s)$ = (nb tours blanches) – (nb tours noires),
  - etc

Exemple de fonction d'évaluation

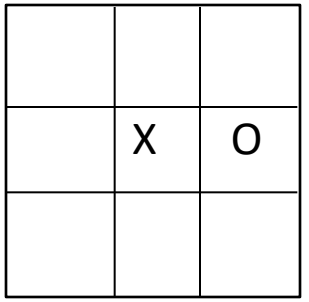
- Pour le *tic-tac-toe*, supposons que Max joue avec les X.

Eval(*s*) =

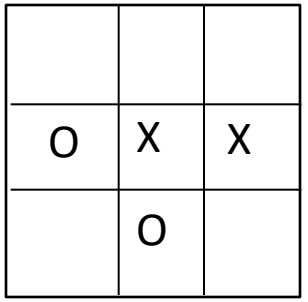
if s is win for Max, + ∞
if s is win for Min, - ∞

else

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



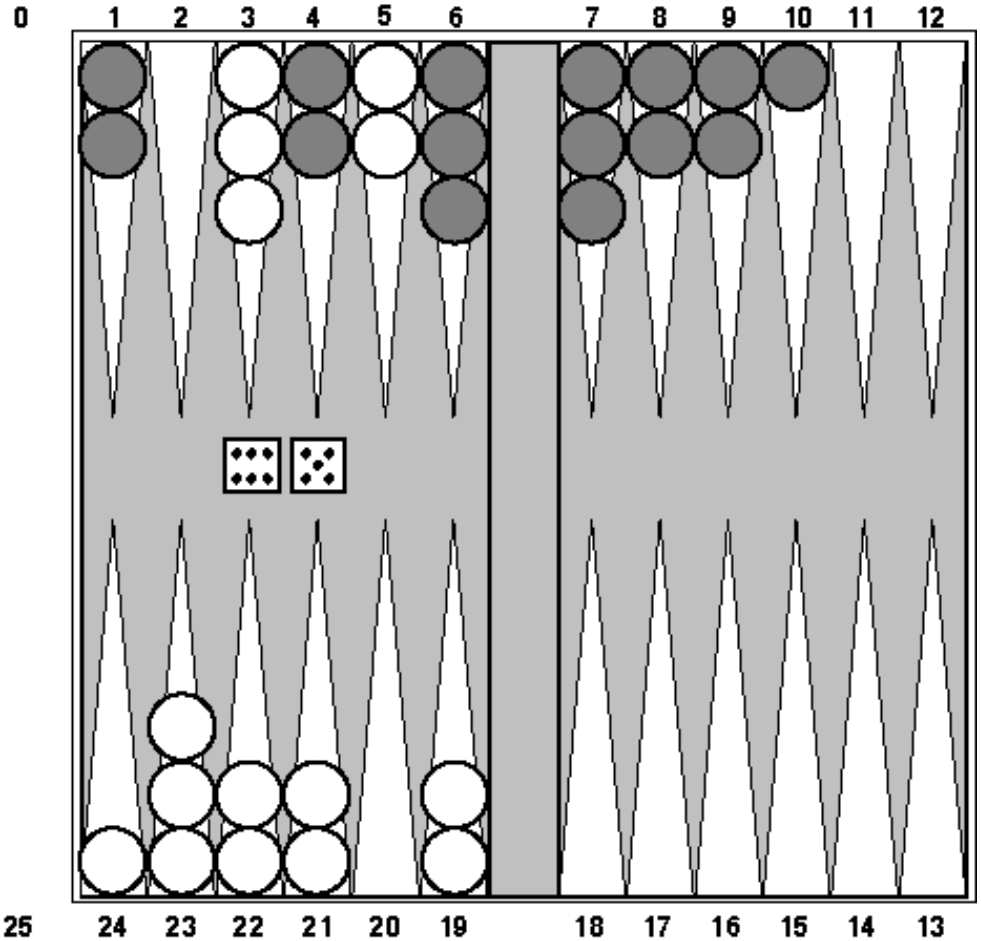
Eval(*s*) = 6 - 4 = 2



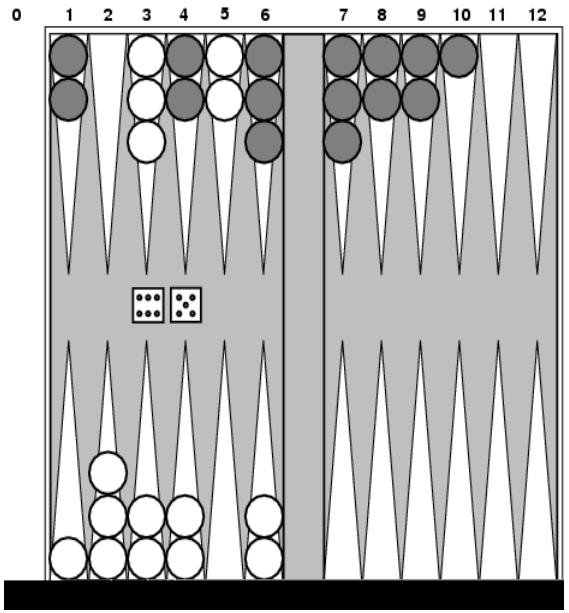
Eval(*s*) = 4 - 3 = 1

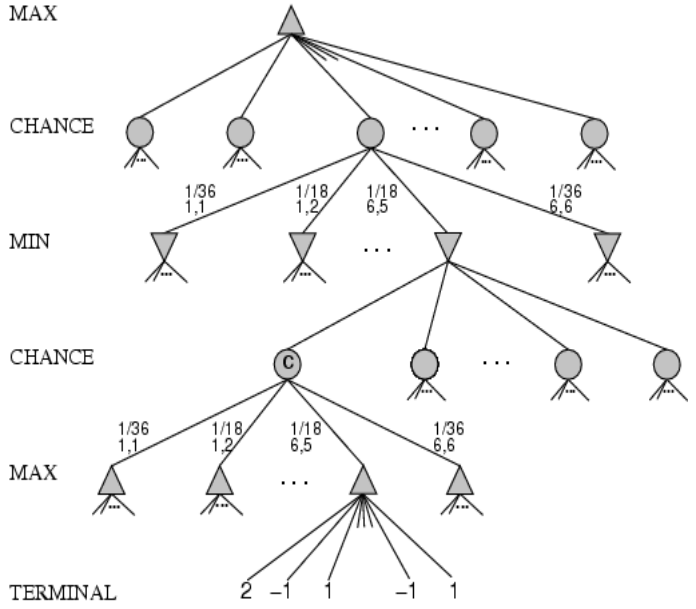
Généralisation à l'incertitude: jeux non-déterministes avec CHANCE

- Dans un jeu non-déterministe, la chance est introduite par un jeter de dés, une distribution de cartes, une pièce de monnaie . . . pour déterminer la prochaine action.
- Exemple: **backgammon**



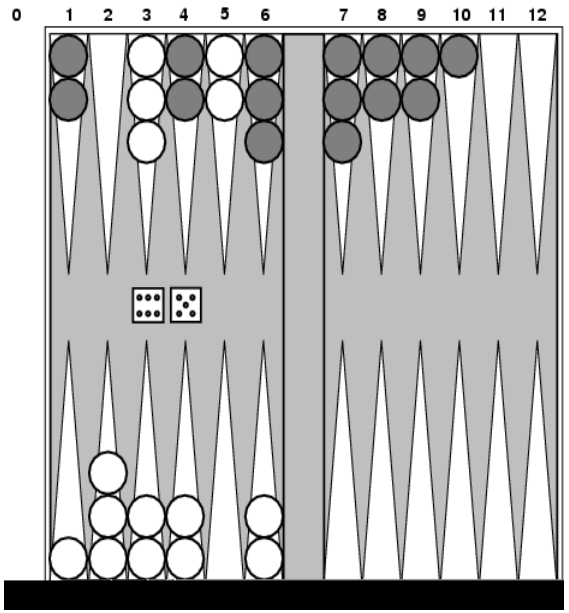
Généralisation à l'incertitude: jeux non-déterministes avec CHANCE

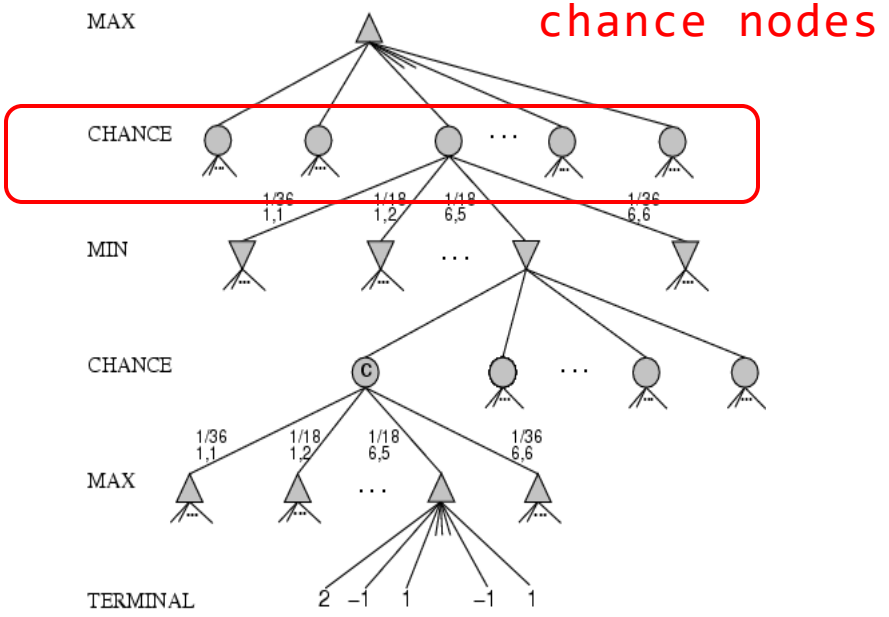




- Le Blanc a lancé (6,5)
- Coups possibles: (5-10,5-11), (5-11,19-24),(5-10,10-16) and (5-11,11-16)
- **Introduire des niveaux pour les noeuds CHANCE entre MAX et MIN**

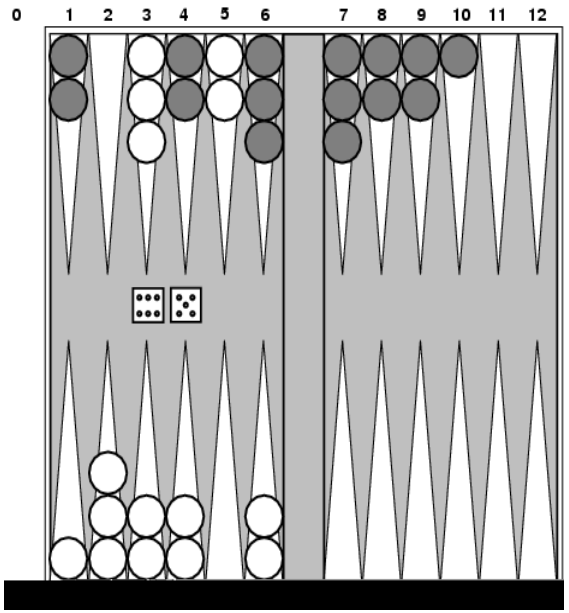
Généralisation à l'incertitude: jeux non-déterministes avec CHANCE

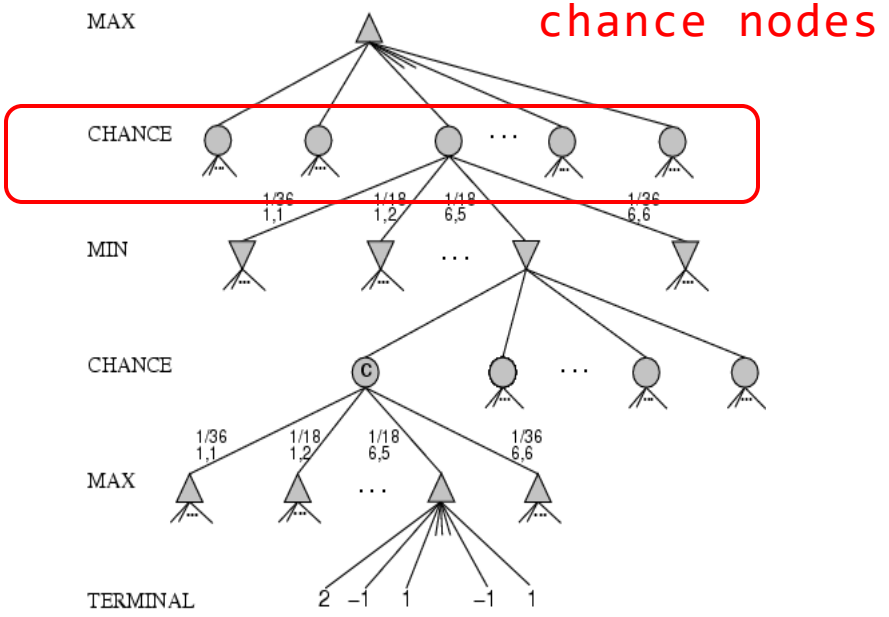




- Le Blanc a lancé (6,5)
- Coups possibles: (5-10,5-11), (5-11,19-24),(5-10,10-16) and (5-11,11-16)
- **Chance d'avoir [1,1], [6,6] …, tout autre chance …**

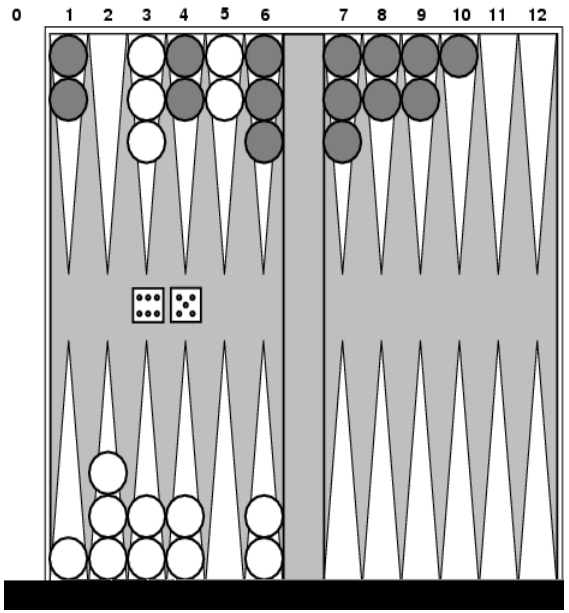
Généralisation à l'incertitude: jeux non-déterministes avec CHANCE

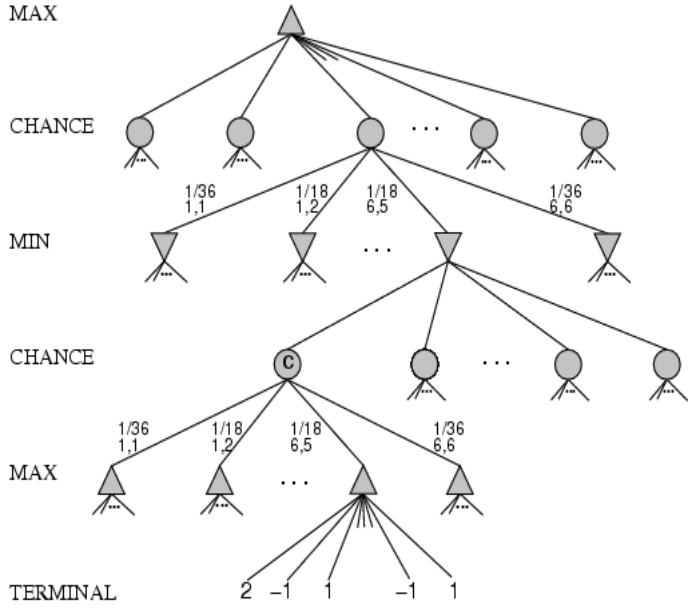




- Le Blanc a lancé (6,5)
- Coups possibles: (5-10,5-11), (5-11,19-24),(5-10,10-16) and (5-11,11-16)
- **Chance d'avoir [1,1], [6,6] 1/36, tout autre chance 1/18**

Généralisation à l'incertitude: jeux non-déterministes avec CHANCE





- Chance d'avoir [1,1], [6,6] 1/36, tout autre chance 1/18
- **On ne peut calculer une valeur minimax définitive, seulement une EXPECTÉE**

Généralisation à l'incertitude: jeux non-déterministes avec CHANCE

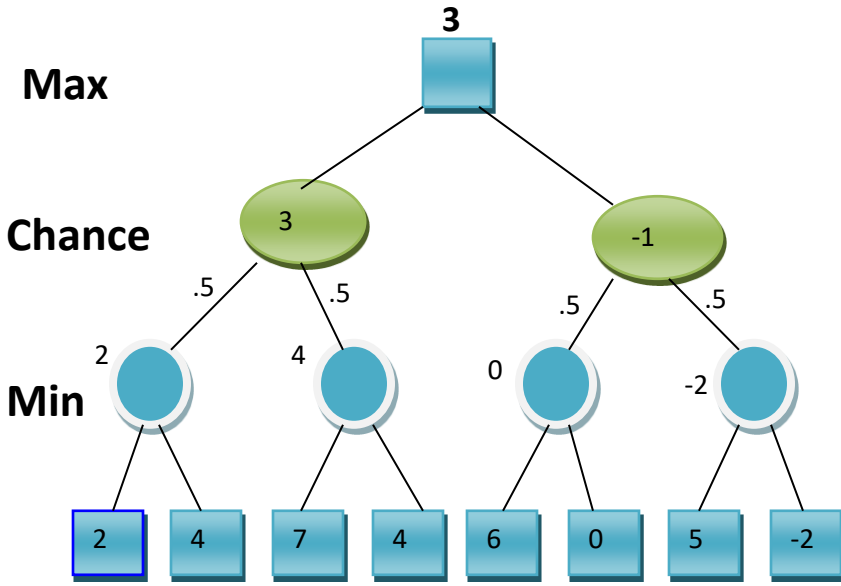
Algorithme de jeux non-déterministe:

Expected_minimax donne le coup parfait, comme MiniMax

- On introduit en plus des nœuds Min et Max, les nœuds Chance, et:

- ***if state is a terminal node then***
  return Utility(state)
  if state is a Max node then
  return the highest Expected_Minimax-Value of Successors(state)
  if state is a Min node then
  return the lowest Expected_Minimax-Value of Successors(state)
  if state is a Chance node then
  return $(\sum_{s \in successors(state)} P(s) *$ ***Expectiminimax-Value*** $(s))$

- Exemple simplifié avec le lancer d'une pièce de monnaie:



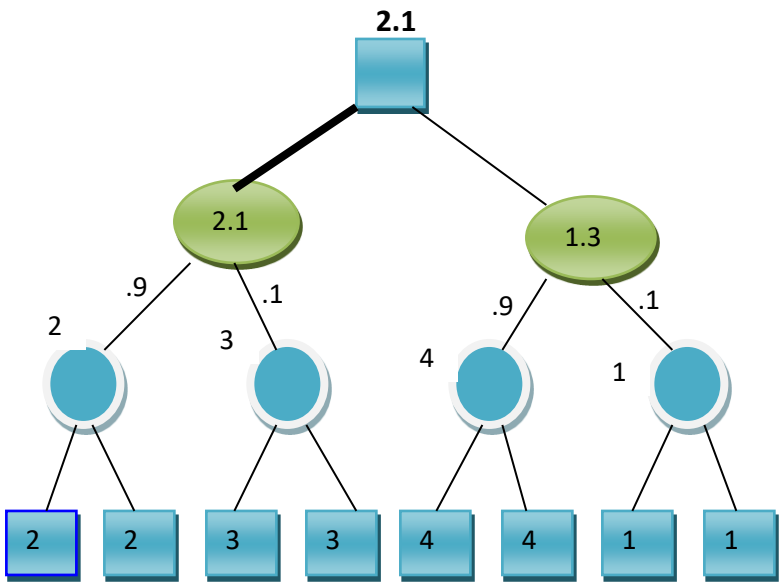
- Si la fonction d'évaluation (*Eval*) est bornée, α-β peut-être adaptée

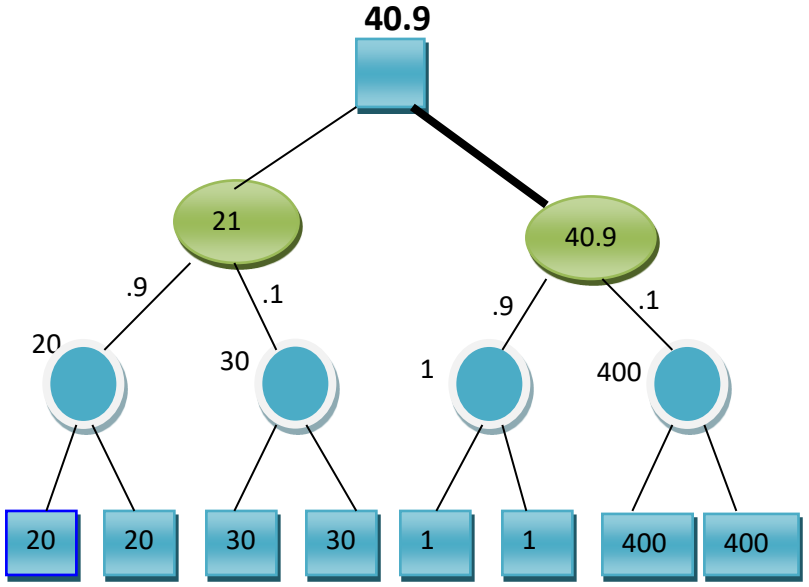
Généralisation à l'incertitude: jeux non-déterministes avec CHANCE

EXPECTED_MINIMAX-VALUE(n) =

UTILITY(n)	Si n est un nœud terminal
$\max_{s \in successors(n)}$ MINIMAX-VALUE(s)	Si n est un nœud Max
$\min_{s \in successors(n)}$ MINIMAX-VALUE(s)	Si n est un nœud Min
$\sum_{s \in successors(n)}$ P(s) * EXPECTED_MINIMAX(s)	Si n est un nœud chance

Ces équations définissent récursivement la valeur des nœuds jusqu'aux feuilles de l'arbre.





Généralisation à l'incertitude: jeux à information imparfaite

- **Exemple:** Jeux de cartes où les cartes de l'adversaire ne sont pas connues
- Il est possible de calculer une probabilité pour chaque distribution de cartes possible
- **Principe :** calculer la valeur Minimax de chaque action dans chaque distribution possible, puis choisir l'action qui la valeur espérée maximale parmi toutes les distributions
- **Cas spécial :** si une action est optimale pour toutes les distributions, c'est une action optimale.

Remarques:

- L'idée que la valeur d'une actions est la moyenne de ses valeurs possibles est **fausse**
- Avec une observation partielle, la valeur d'une action dépend de l'**état des connaissances** ou l'**état de croyance de l'agent**
- Il est possible de générer un arbre de recherche sur les états de connaissances qui mène à des **comportements rationnels** tels que :
  - Agir pour obtenir de l'information
  - Donner les informations que l'on a à son ou ses partenaire(s)
  - Agir au hasard pour minimiser les informations que l'on fournit à ses adversaires.

État de l'art: Quelques succès et défis

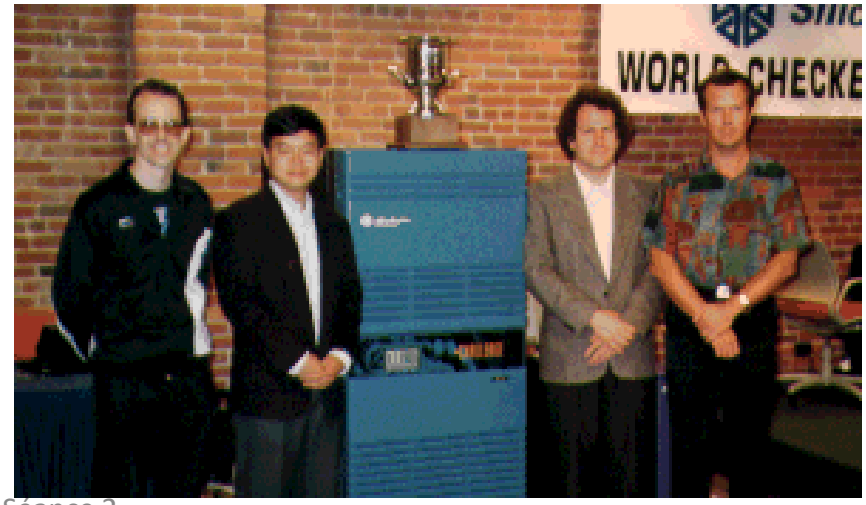
- **Jeu de dames** : En 1994, Chinook a mis fin aux 40 ans de règne du champion du monde Marion Tinsley. Chinook utilisait une base de données de coups parfaits pré-calculés pour toutes les configurations impliquant 8 pions ou moins : 444 milliards de configurations!
- **Jeu d'échecs**: En 1997, Deep Blue a battu le champion du monde Garry Kasparov dans un match de six parties. Deep Blue explorait 200 million de configurations par seconde.
- **Othello** : les champions humains refusent la compétition contre des ordinateurs, parce que ces derniers sont trop bons!
- **Go**: les champions humains refusaient la compétition contre des ordinateurs, parce que ces derniers sont trop mauvais! Dans le jeu GO, le facteur de branchement (*b)* dépasse 300! La plupart des programs utilisent des bases de règles empiriques pour calculer le prochain coup.
  - En janvier 2016,Google prétend détenir la solution …
  - Et en mars 2016, **AlphaGo**, l'IA développée par **Deep Mind,** l'a emporté 4 parties à 1 face au Sud-Coréen **Lee Sedol**, numéro 3 mondial. (VIDEO : https://youtu.be/mzpW10DPHeQ)
  - L'IA AlphaGo s'entraîne pour vaincre au jeu de stratégie en temps réel StarCraft II (Défis: environnement non entièrement observable; les joueurs ne voient pas ce que fait leur rival).

Jeu de dames: Tinsley vs. Chinook

Nom: Marion Tinsley
Profession: professeur de mathématiques
Hobby: le jeu de dames
Record: en plus de 42 ans n'a perdu que 3 (!) parties

Mr. Tinsley encaisse sa 4ème et 5ème défaite contre Chinook

Nom: Chinook
Record: premier programme informatique ayant gagné un championnat du monde face au champion en titre en 1994, invaincu depuis! Méthode: base de fins de parties de 8 pièces (ou moins), pour un total de 444 milliards de positions



Jeu d'échecs

Jeu qui a reçu la plus grande attention au début les progrès ont été très lents

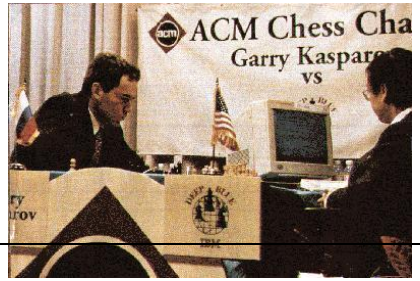
➢ 1970: 1er programme à gagner le championnat ACM d'échecs informatiques, utilisant une recherche α - β, des ouvertures classiques et des algorithmes de fins de parties infaillibles

➢1982: "Belle" est le premier ordinateur spécialisé dans le jeu d'échecs capable d'explorer quelques millions de combinaisons par coups

➢1985: "Hitech" se classe parmi les 800 meilleurs joueurs du monde, il est capable d'explorer plus de 10 millions de combinaisons par coups

➢1997: "*Deep Blue*" bat G. Kasparov. Il effectue une recherche sur 14 niveaux et explore plus de 1 milliard de combinaisons par coup à raison de plus de 200 millions de positions par seconde, utilise une fonction d'évaluation très sophistiquée et des méthodes non divulguées pour étendre certaines recherches à plus de 40 niveaux.

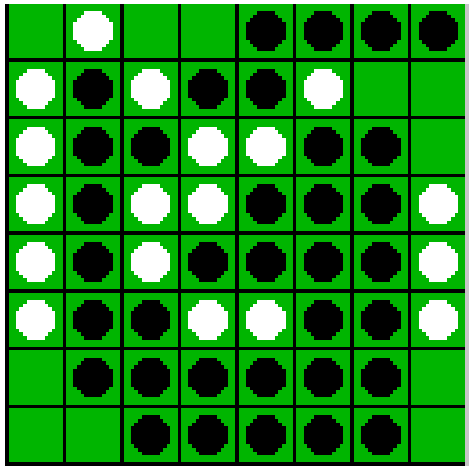
Kasparov vs. Deep Blue



Kasparov		Deep Blue
5'10"	Taille	6' 5"
176 lbs	Poids	2,400 lbs
34 ans	Age	4 ans
50 milliards	Ordinateur	512 processeurs
2 pos/sec	Vitesse	200,000,000 pos/sec
Extensive	Connaissance	Primitive
Electrique/chimique	Source d'énergie	Électrique
Démesuré	Ego	Aucun

1997: Deep Blue gagne par 3 victoires, 1 défaite, et 2 nuls

Othello: Murakami vs. Logistello



Takeshi Murakami
Champion du monde d'Othello

1997: Le logiciel Logistello défait Murakami
par 6 jeux à 0

Kasparov vs. Deep Blue Junior

Deep Junior

8 CPU, 8 GB RAM, Win 2000
2,000,000 pos/sec
Disponible à $100

2 Août 2003: Égalité (3-3)!

"Saying Deep Blue doesn't really think about chess is like saying an airplane doesn't really fly because it doesn't flap its wings"

Drew McDermott

Go: AlphaGo vs. Lee Sedol

Mars 2016: L'IA **AlphaGo** (de DeepMind) défait **Lee Sedol** (3e mondial) par 4 jeux à 1

AphaGo contre Ke Jie , numéro 1 mondial? A suivre…

StarCraft: AlphaGo vs. X ?

Défis:

- Environnement partiellement observable ➔ Exploration de l'environnement
- Gestion des ressources et de l'incertitude
-Développement de stratégies offensive et défensive sans voir ce que fait l'adversaire.

Défis

- Jeux stratégies en temps réel combinant éléments discrets et continue.
  - StarCraft
- Jeux avec lois de la physiques.
  - Angry Bird
  - Billard
- Méthodes probabilistes
  - *Random search trees,* Monte Carlo, Échantillonnage (sampling), etc.

Résumé

- Il est amusant de pratiquer l'IA avec les jeux.
- Les recherches sur les jeux révèlent des aspects fondamentaux intéressants applicables à d'autres domaines.
- La perfection est généralement inatteignable dans les jeux : il faut approximer.
- *Alpha-beta* a la même valeur pour la racine de l'arbre de jeu que *minimax*.
- Dans le pire des cas, *Alpha-beta* se comporte comme *minimax* (explore tous les nœuds).
- En général, *Alpha-beta* est beaucoup plus rapide que *minimax*.