IFT 615 – Intelligence artificielle

Recherche pour jeux à deux adversaires

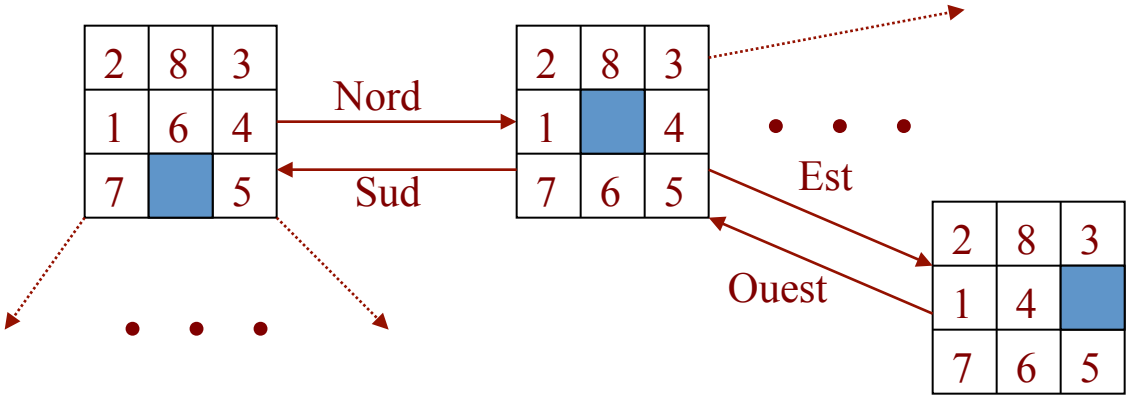
Hugo Larochelle
Département d'informatique
Université de Sherbrooke
http://www.dmi.usherb.ca/~larocheh/cours/ift615.html

Objectifs

- Se familiariser avec les sujets suivants:
  - Jeux entre deux adversaires
  - Algorithme minimax
  - Élagage alpha-beta
  - Décisions imparfaites en temps réelle
  - Généralisation à des jeux avec incertitude

Rappel sur A*

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



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?
- R: C'est le sujet d'aujourd'hui !

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 (ex.: alliances)

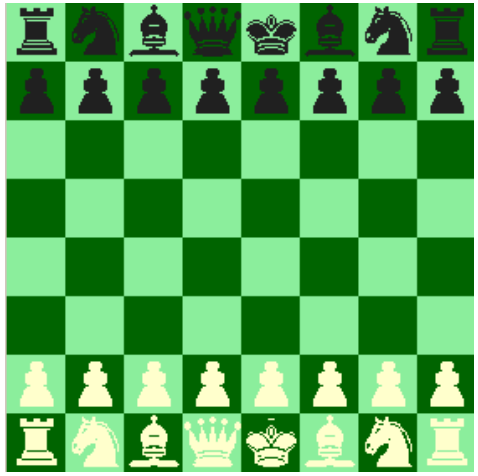
Hypothèses

- Pour l'instant, nous aborderons les:
  - Jeux à deux adversaires
  - Jeux 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 entre deux adversaires

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

Min

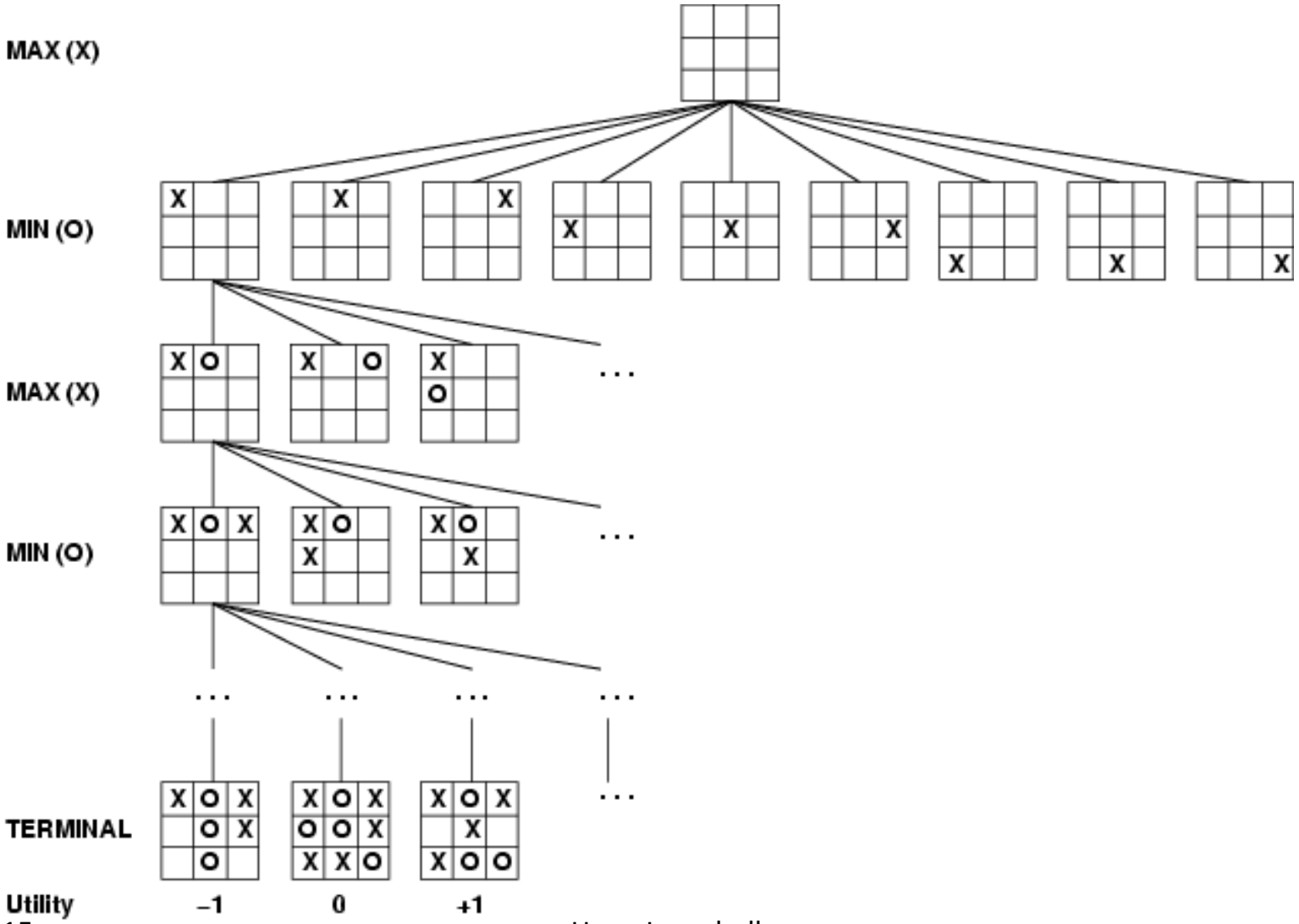


Max

Arbre de recherche

- Un problème de jeu peut être vu comme un problème de recherche dans un arbre:
  - Un **état (configuration) initial**
  - Un ensemble d'opérateurs:
    - » retournant l'**ensemble des actions possibles** (légales)
    - » retournant l'**état après avoir exécuté une action donnée**
  - Un **test de terminaison**
    - » Indique si le jeu est terminé
  - Une **fonction d'utilité** pour les états finaux

Arbre de recherche tic-tac-toe



Algorithme minimax

- Idée: à chaque tour, choisir l'action qui correspond à la plus grande valeur minimax

EXPECTED-MINIMAX-VALUE(n) =

UTILITY(n)	Si n est un nœud terminal
$\max_{s \in successors(n)}$ EXPECTED-MINIMAX-VALUE(s)	Si n est un nœud Max
$\min_{s \in successors(n)}$ EXPECTED-MINIMAX-VALUE(s)	Si n est un nœud Min

- Ces équations donnent la programmation récursive des valeurs pour tous les noeuds dans l'arbre de recherche

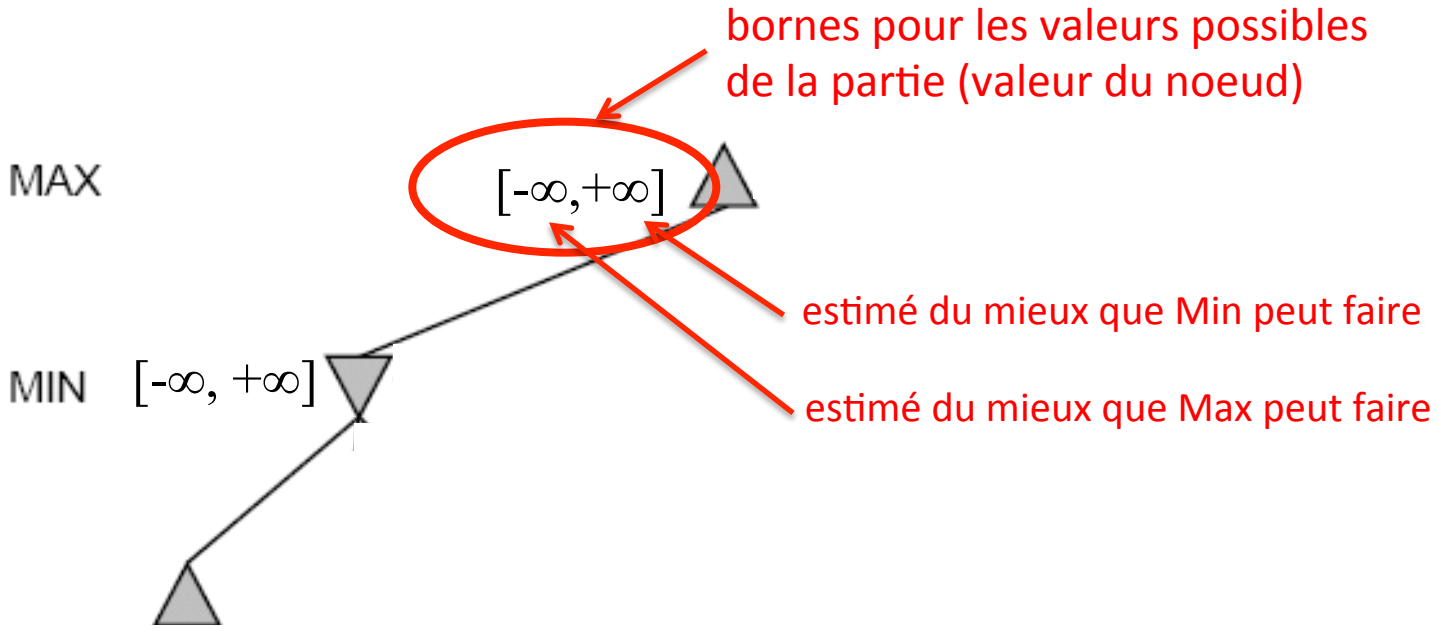
Algorithme minimax

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

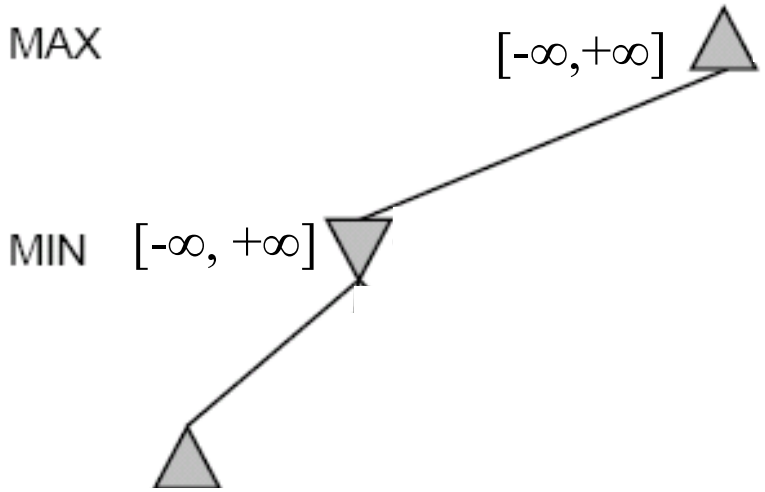
Propriétés de minimax

- 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 de 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), puisque **recherche en profondeur**
- Pour le jeu d'échec: b ≈ 35 et m ≈100 pour une partie « raisonnable »
  - il n'est pas réaliste d'espérer trouver une solution exacte en un temps raisonnable

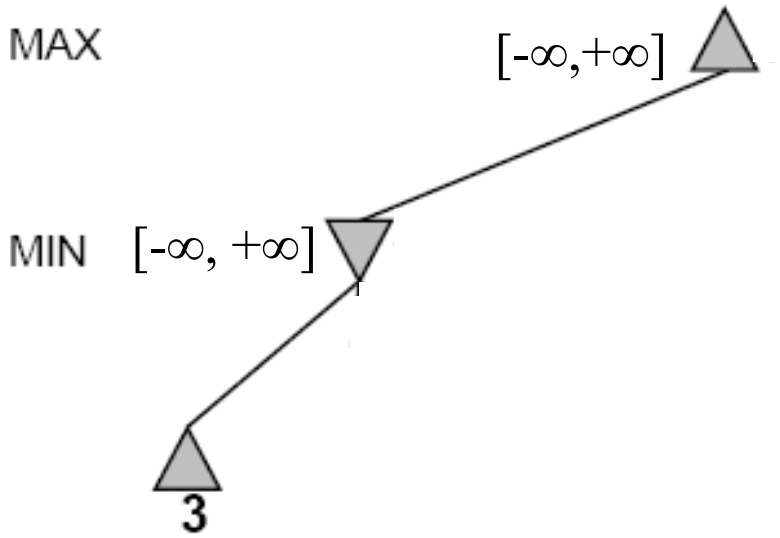
Alpha-beta pruning



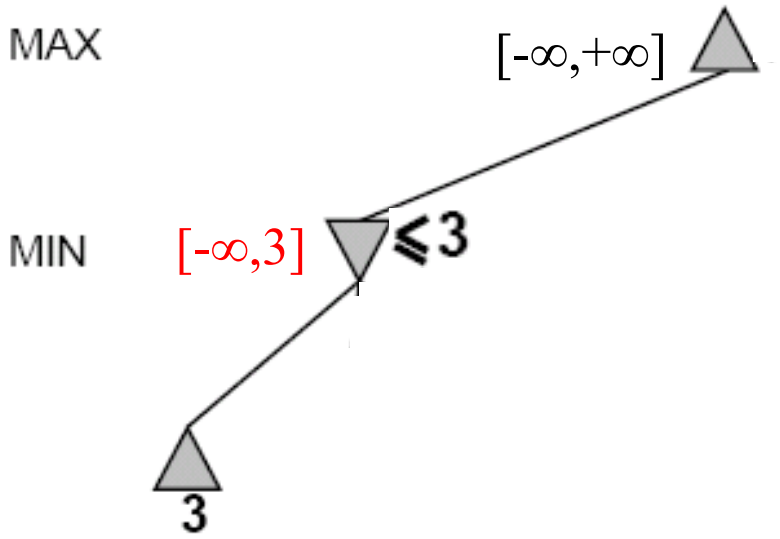
Alpha-beta pruning



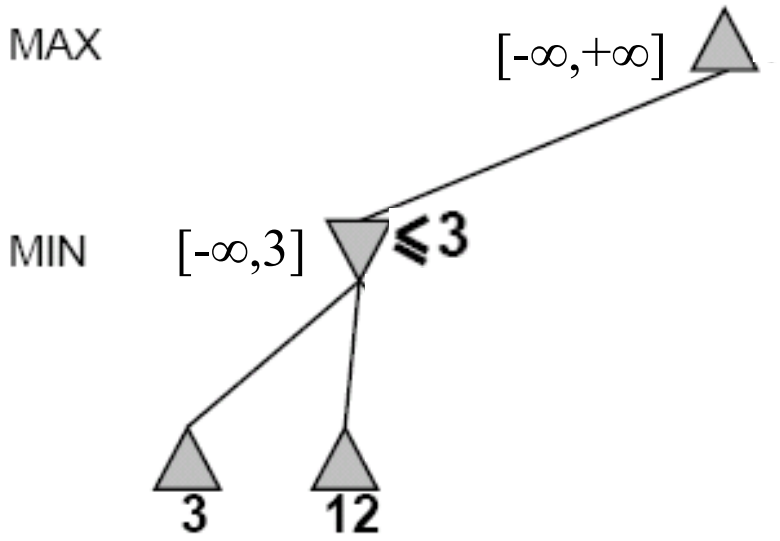
Alpha-beta pruning



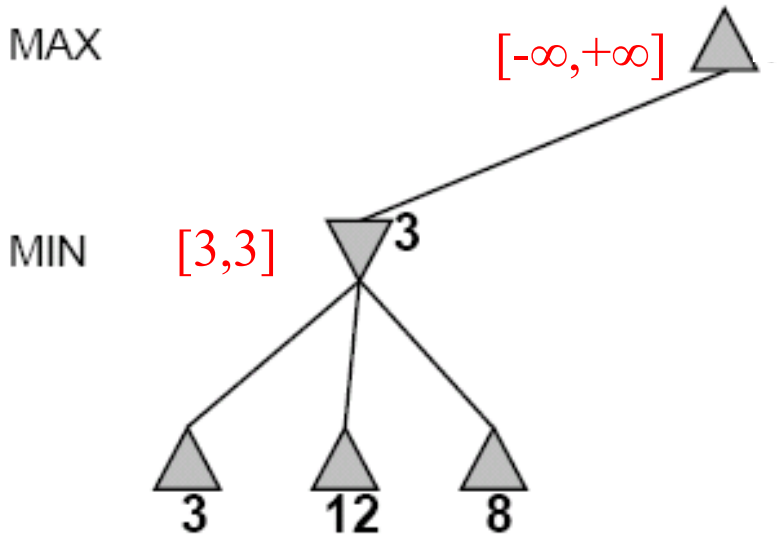
Alpha-beta pruning



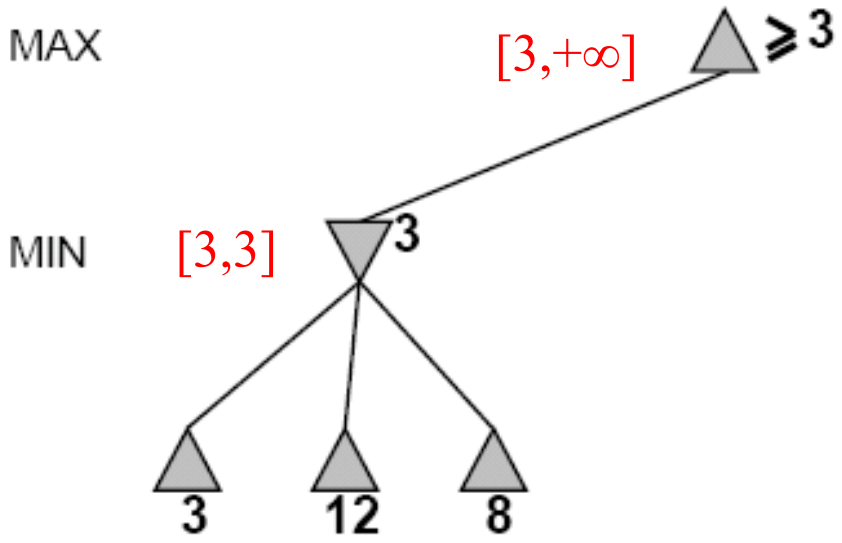
Alpha-beta pruning



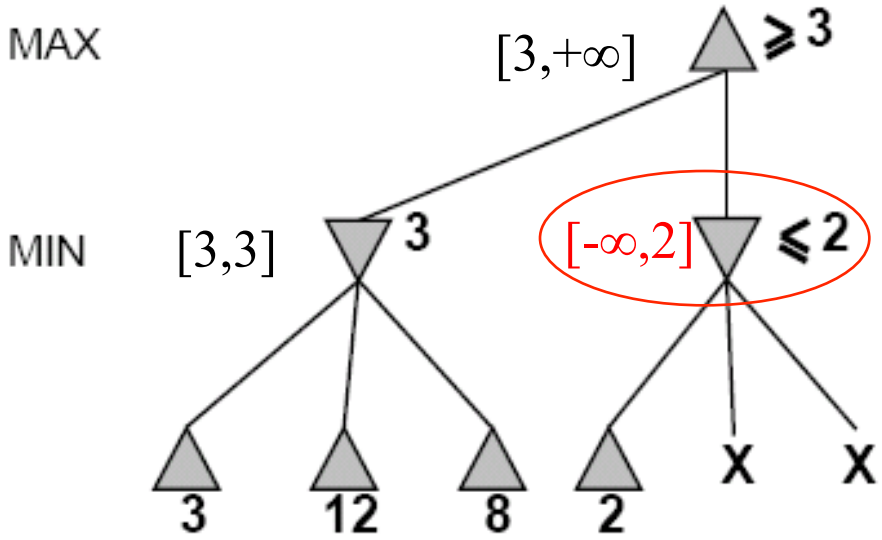
Alpha-beta pruning



Alpha-beta pruning

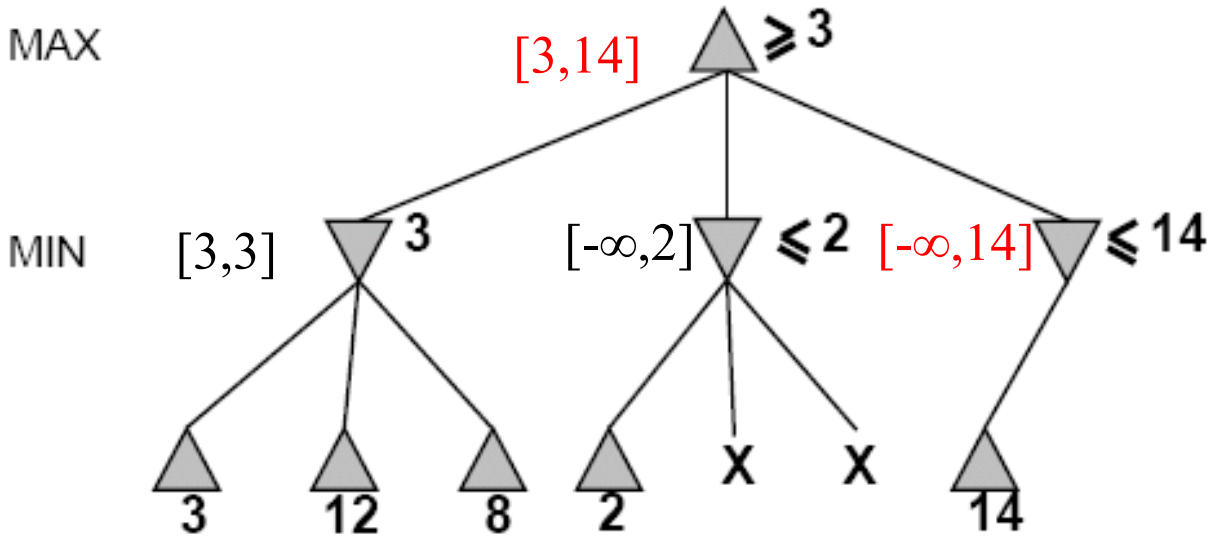


Alpha-beta pruning

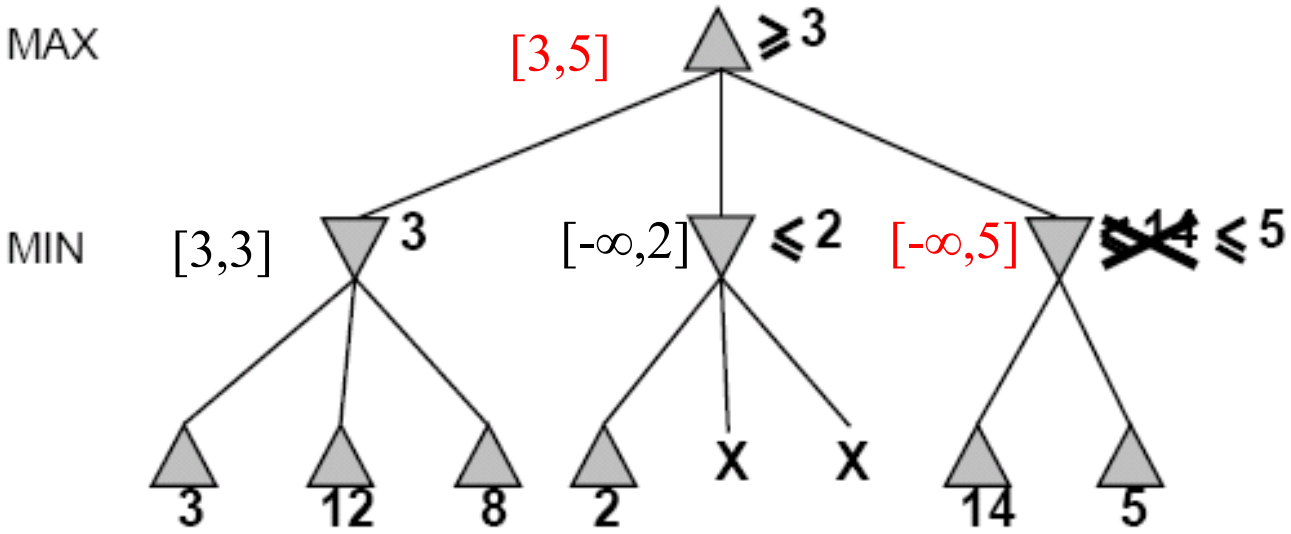


Ce noeud est pire pour Max

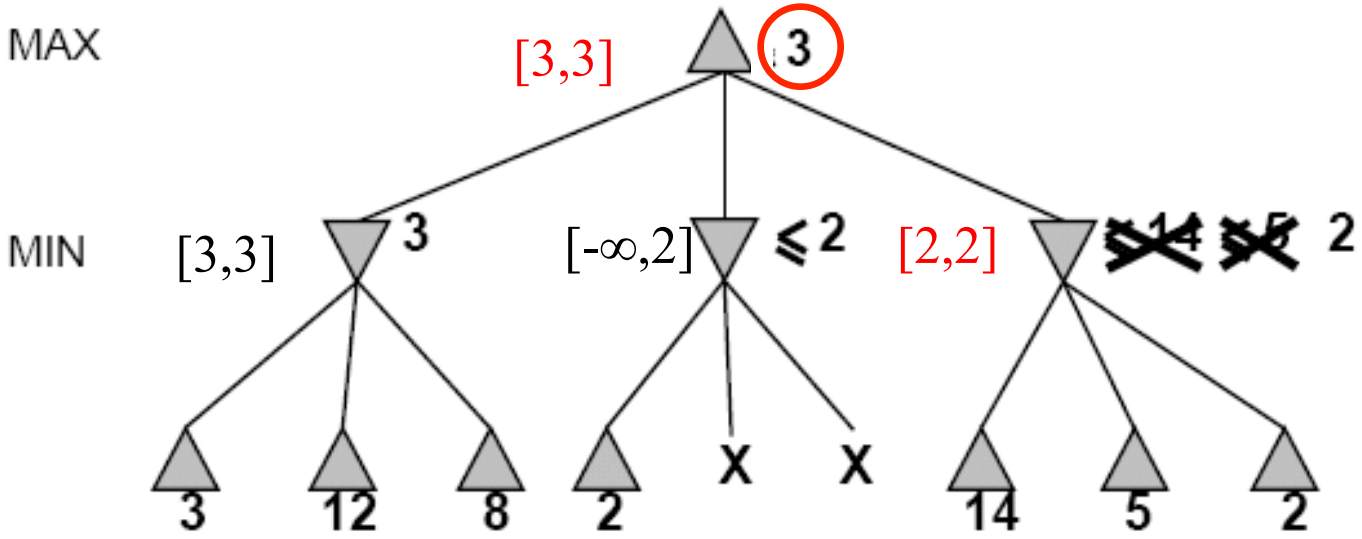
Alpha-beta pruning



Alpha-beta pruning

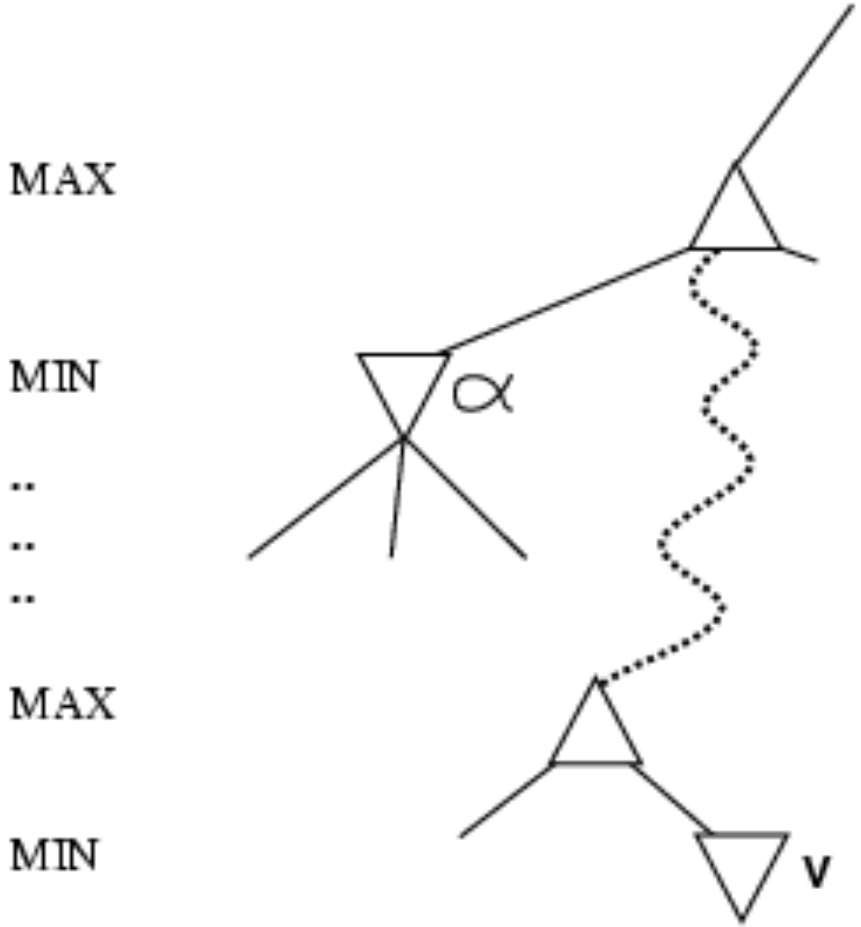


Alpha-beta pruning



D'ou vient le nom alpha-beta?

- L'algorithme alpha-beta tire son nom des paramètres suivants, décrivant les bornes des valeurs d'utilité enregistrée durant le parcourt
  - α est la valeur du meilleur choix pour Max (c.-à-d., plus grande valeur) trouvé jusqu'ici:
    - » si le nœud v a une valeur pire que α, Max n'ira jamais à v: couper la branche
  - β est défini de manière analogue pour Min



Algorithme alpha-beta pruning

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a), α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

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 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 améliore 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!
  - Dans le cas moyen d'un ordonnancement aléatoire: $O(b^{3m/4})$

Version unifiée d'alpha-beta pruning

```
1.  fonction SearchAlphaBeta(state, α, β)  /* α < β */
2.    if TerminalTest(state) then
3.          return Utility(state)
4.    else
5.       best ← -∞
6.       for (a,s) in Sucessors(state)
7.          v ← -SearchAlphaBeta(s,-β,-α)
8.          if v > best then
9.             best ← v
10.            if best > α then
11.               α ← best
12.               if α >= β then
13.                  return best
14.    return best
```

Décisions 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:
    - » par exemple, limiter la profondeur de l'arbre
    - » voir le livre pour d'autres idées
  - Fonction d'évaluation
    - » estimation du alpha ou beta qui aurait été obtenu en faisait une recherche complète
    - » on peut voir ça comme une estimation de la « chance » qu'une configuration mènera à une victoire

Exemple de fonction d'évaluation

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

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

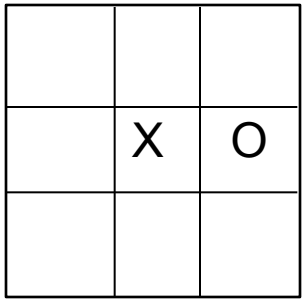
- Par exemple:
  - $w_1 = 9$, $f_1(s) =$ (*number of white queens*) – (*number of black queens*)
  - etc.

Exemple de fonction d'évaluation

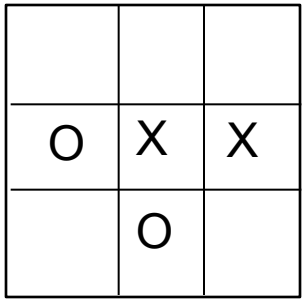
- Pour le tic-tac-toe, supposons que Max joue avec les X

Eval(*s*) =

if s is win for Max, + ∞
elif s is win for Min, - ∞
else
(nombre de ligne, colonnes et diagonales disponibles pour Max) - (nombre de ligne, colonnes et diagonales disponibles pour Min)

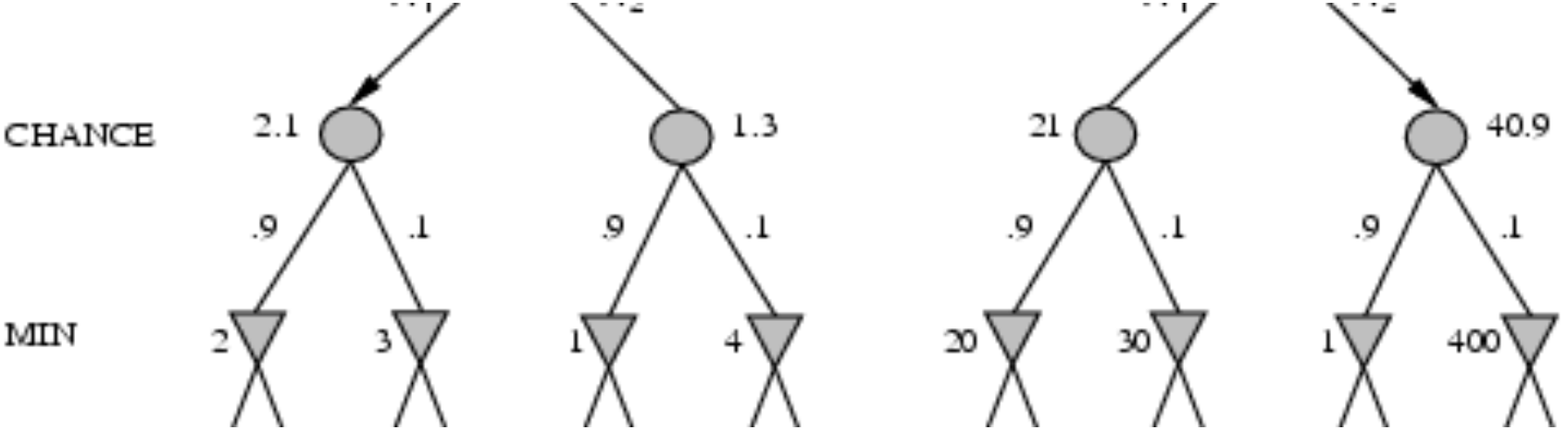


Eval(*s*) = 6 - 4 = 2



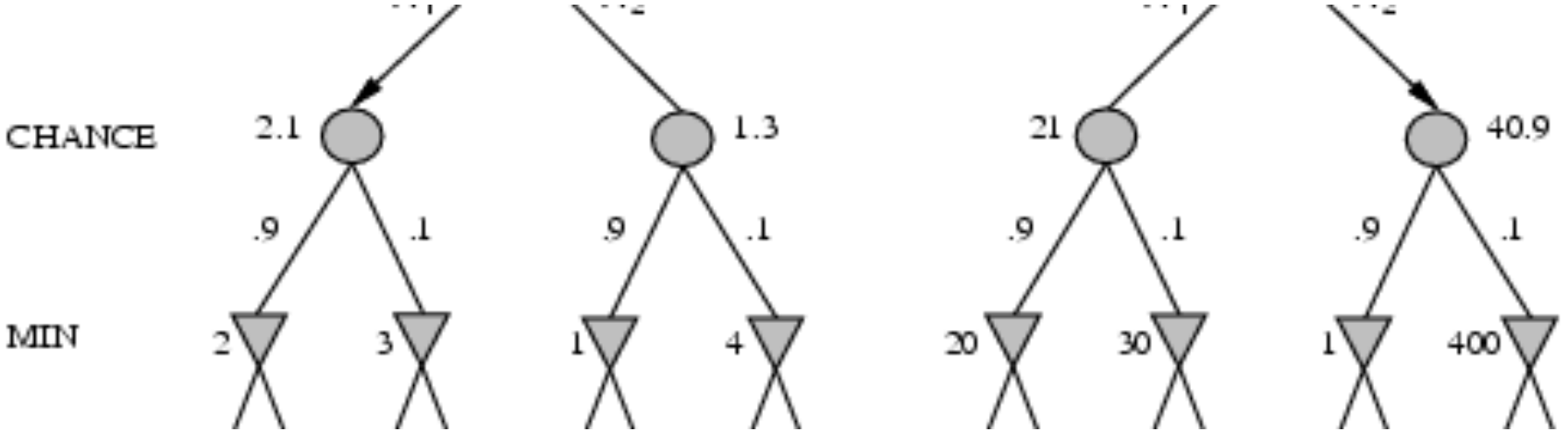
Eval(*s*) = 4 - 3 = 1

Généralisation aux actions aléatoires



- Par exemple, des jeux où on lance un dé pour déterminer la prochaine action

Généralisation aux actions aléatoires



EXPECTED-MINIMAX-VALUE(n) =

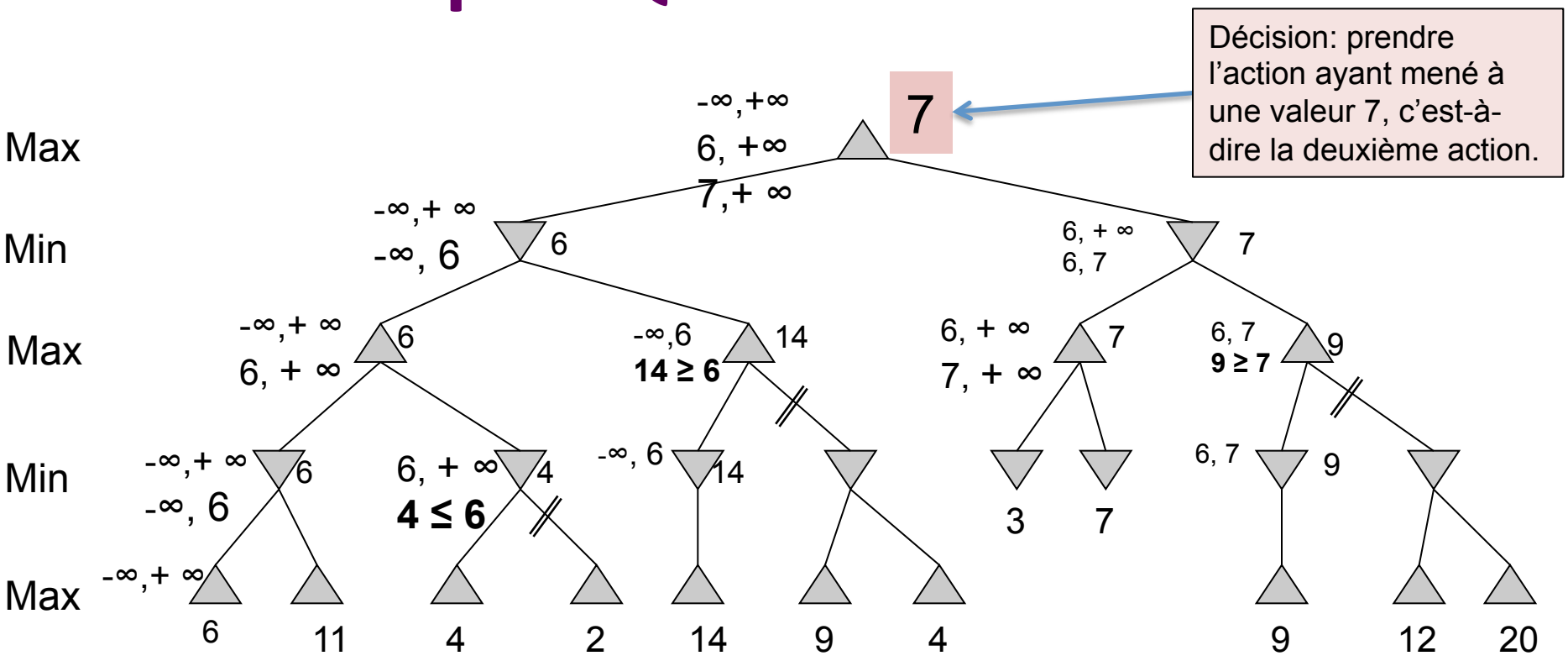
UTILITY(n)	Si n est un nœud terminal
$\max_{s \in successors(n)}$ MINIMAX-VALUE(s)	Si n est un nœud Max
$\min_{s \in successors(n)}$ MINIMAX-VALUE(s)	Si n est un nœud Min
$\sum_{s \in successors(n)} P(s)$ * EXPECTEDMINIMAX(s)	Si n est un nœud chance

Ces équations donne la programmation récursive des valeurs jusqu'à la racine de l'arbre

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 millions 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 refusent 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 programmes utilisent des bases de règles empiriques pour calculer le prochain coup.

Autre exemple: Question #2 – 2009H



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

Résumé

- La recherche sur les jeux révèlent des aspects fondamentaux applicables à d'autres domaines
- La perfection est 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, 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