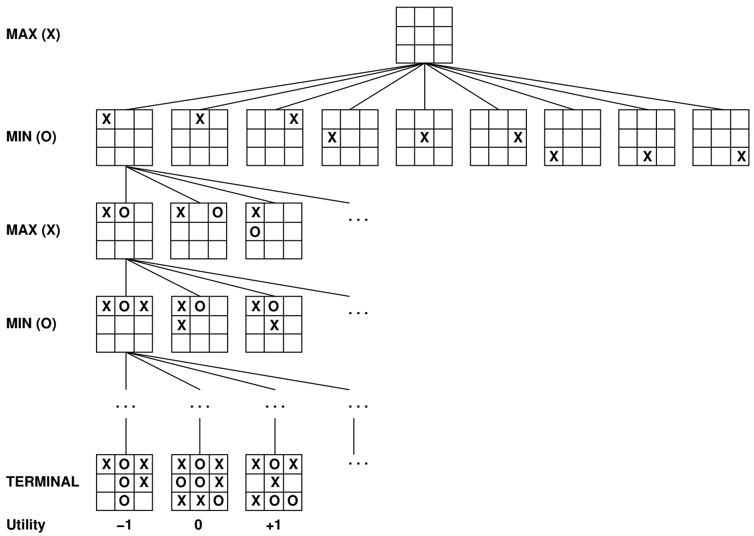
Les jeux de stratégie

Les types de jeu:

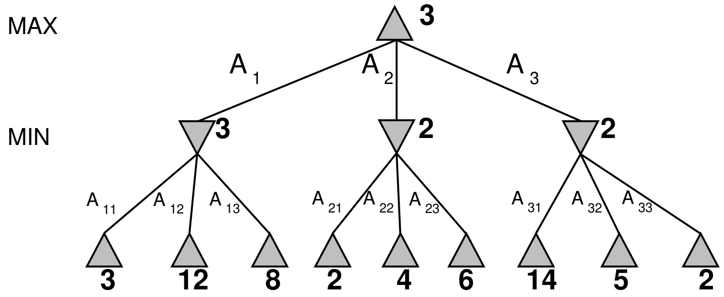
	déterministe	**hasard**
information parfaite	échecs, reversi go, morpion	backgammon monopoly
information imparfaite		bridge, poker scrabble

Arbre de jeux morpion



L'algorithme Minimax

- Donne le coup parfait pour un jeu déterministe à information parfaite
- Idée: choisir le meilleur coup vers la position avec la meilleure valeur **minimax** = meilleur valeur possible contre le meilleur jeu de l'adversaire
- Exemple d'un jeu à deux coups:



L'algorithme minimax

function MINIMAX-DECISION(*game*) **returns** *an operator*

  for each *op* **in** OPERATORS[*game*] **do**
    VALUE[*op*] ← MINIMAX-VALUE(APPLY(*op*, *game*), *game*)
  end
  return the *op* with the highest VALUE[*op*]

---

function MINIMAX-VALUE(*state, game*) **returns** *a utility value*

  if TERMINAL-TEST[*game*](*state*) **then**
    return UTILITY[*game*](*state*)
  else if MAX is to move in *state* **then**
    return the highest MINIMAX-VALUE of SUCCESSORS(*state*)
  else
    return the lowest MINIMAX-VALUE of SUCCESSORS(*state*)

Propriétés de minimax

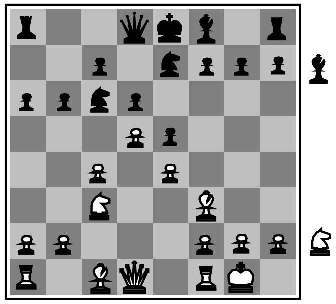
- s'arrête toujours si l'arbre est fini
- Optimal, si l'adversaire est optimale.
- Si b est le nombre de coups possibles par situation et m la profondeur maximale de l'arbre, minimax a une complexité en temps $O(b^m)$ et en espace $O(b * m)$.
- Pour les échecs par exemple: $b \approx 35$, $m \approx 100 \Rightarrow$ solution exacte impossible.

Les ressources sont limitées

Par exemple, on a 100 secondes et on peut explorer 10^4 noeuds par secondes. Donc, on peut regarder 10^6 noeuds par coup. Approche standard:

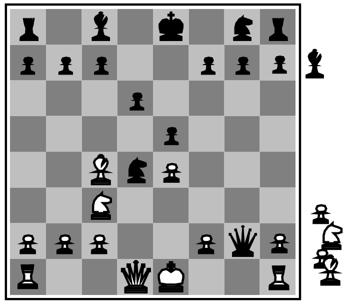
- Test d'arrêt: Par exemple limiter la profondeur (peut-être ajouter recherche "silencieuse", p.e. aux échecs on ne s'arrête pas si la reine vient d'être prise)
- Fonction d'évaluation = valeur estimé d'une position

Fonction d'évaluation



Black to move

White slightly better



White to move

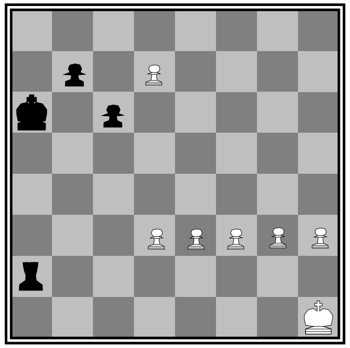
Black winning

Aux échecs on choisit par exemple une somme **linéaire** pondéré de caractéristiques.

$eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$

avec p.e. $w_1 = 9$ et $f_1 =$ le nombre de reines blanches $-$ le nombre de reines noires, etc.

L'effet horizon



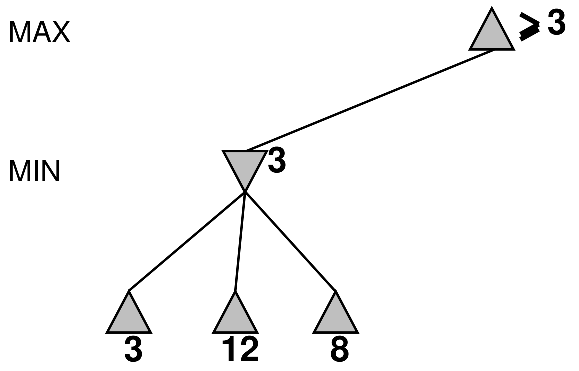
Black to move

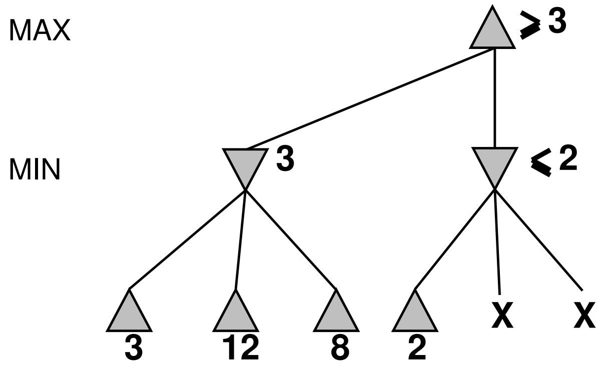
Dans cette position Noir peut mettre le roi blanc en échec un certain nombre de fois mais le pion blanc va se transformer inévitablement en reine. On en s'aperçoit très tard.

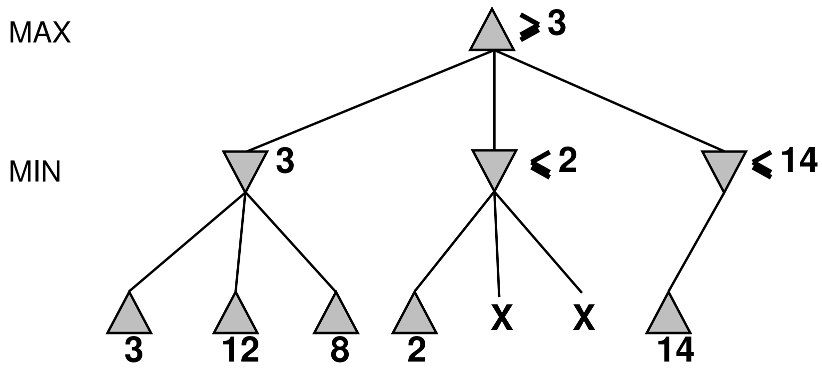
Arrêter la recherche

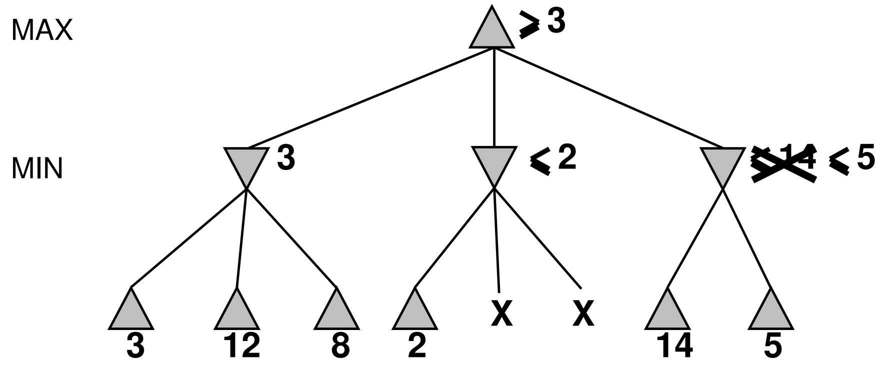
- On peut facilement modifier l'algorithme minimax en ajoutant un test d'arrêt (remplacer TERMINAL-TEST par CUTOFF-TEST) et en remplaçant UTILITY par EVAL.
- En pratique: problèmes de performances
  p.e. $b^m = 10^6$ et $b = 35$. Donc $m = 4$.
  Aux échecs on ne pourrait que regarder 4 demi-coups en avance.
- Idée: Élaguer des branches inutiles.

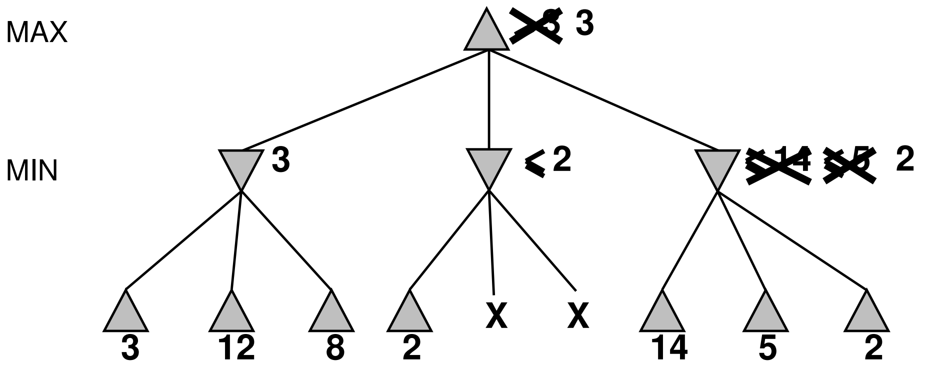
Exemple d'élagage α-β



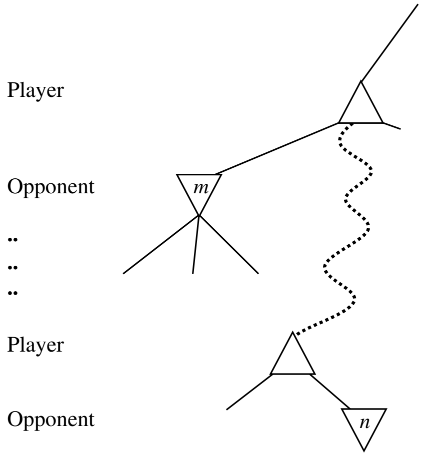








En général



- α est la meilleure valeur (la plus grande) pour MAX trouvée jusqu'à présent en dehors du chemin actuel
- Si V est pire que α MAX va l'éviter $\Rightarrow$ élaguer la branche
- Définir β d'une manière similaire pour MIN: β est la meilleur valeur (la plus petite) pour MIN jusqu'à présent.

L'algorithme α-β

function MAX-VALUE(*state, game*, α, β) **returns** the minimax value of *state*
  inputs: *state*, current state in game
    game, game description
    α, the best score for MAX along the path to *state*
    β, the best score for MIN along the path to *state*

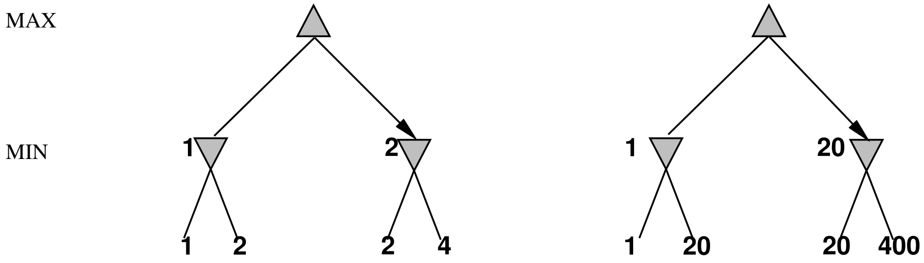
  if CUTOFF-TEST(*state*) **then return** EVAL(*state*)
  for each *s* **in** SUCCESSORS(*state*) **do**
    $\alpha \leftarrow$ MAX(α, MIN-VALUE(*s, game*, α, β))
    if $\alpha \geq \beta$ **then return** β
  end
  return α

---

function MIN-VALUE(*state, game*, α, β) **returns** the minimax value of *state*

  if CUTOFF-TEST(*state*) **then return** EVAL(*state*)
  for each *s* **in** SUCCESSORS(*state*) **do**
    $\beta \leftarrow$ MIN(β, MAX-VALUE(*s, game*, α, β))
    if $\beta \leq \alpha$ **then return** α
  end
  return β

α-β: Valeur exacte des noeuds n'est pas importante



- Seulement l'ordre est important
- Le comportement est préservé pour chaque transformation monotone de EVAL

Remarques algorithme α-β

- L'ordre dans lequel on visite les fils est important
- Si on trouve rapidement une bonne valeur, on élague plus de noeuds
- On peut trier les fils par leur utilité
- Il y a d'autres améliorations
- Au mieux, on visite $\sqrt{n}$ noeuds au lieu de $n = b^m$ pour minimax
- Aux échecs on utilise au début du jeu des bases de données d'ouverture, au milieu α-β et à la fin des algorithmes spéciaux

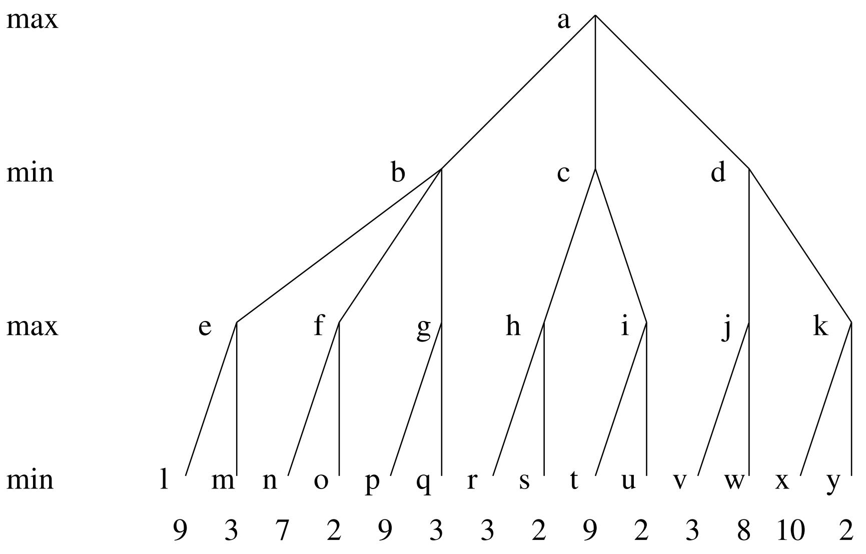
L'algorithme SSS*

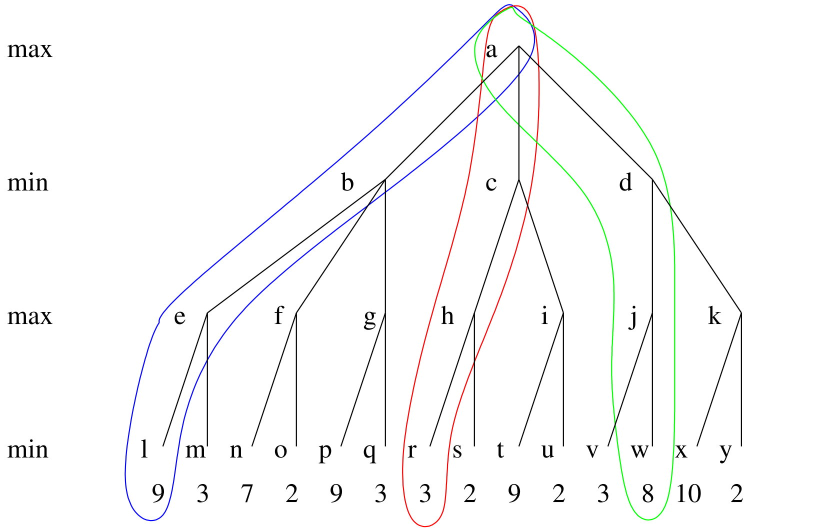
- Définition: Une stratégie (complète) pour le joueur max, étant donné un arbre de jeux A, est un sous-arbre, qui
  - contient la racine de A
  - dont chaque noeud Max a exactement un fils
  - dont chaque noeud Min a tous ses fils
- Une stratégie partielle pour le joueur max, étant donné un arbre de jeux A, est un sous-arbre de A, qui
  - contient sa racine
  - dont chaque noeud Max a au plus un fils

L'algorithme SSS*

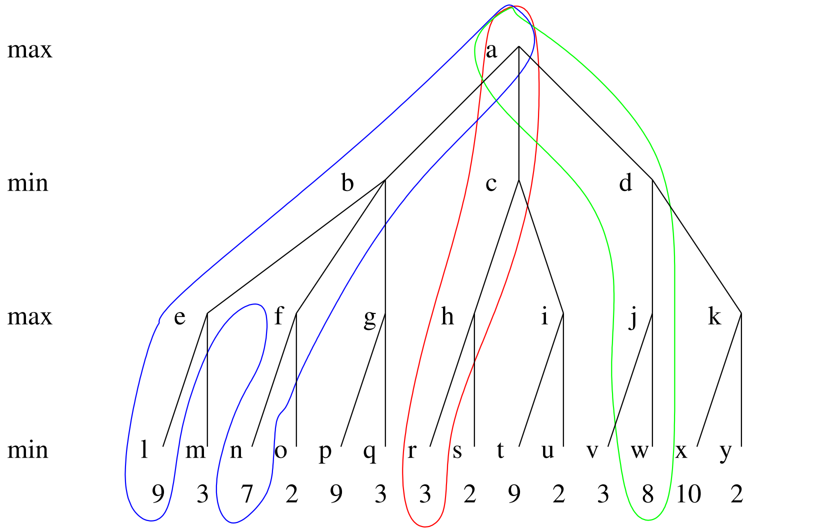
- Une stratégie partielle S représente implicitement toutes les stratégies complètes C_S auxquelles on aboutie en développant S
- La valeur d'une stratégie est le minimum des valeurs des feuilles
- La valeur d'une stratégie partielle S donne une borne supérieur pour toutes ses stratégies complètes C_S.
- Idée: On recherche a compléter les stratégies partielles suivant leur valeur jusqu'à présent.
- Une fois quand a trouvé la stratégie optimale complète à partir d'un noeud x on ne considère plus les autres de x.

Exemple

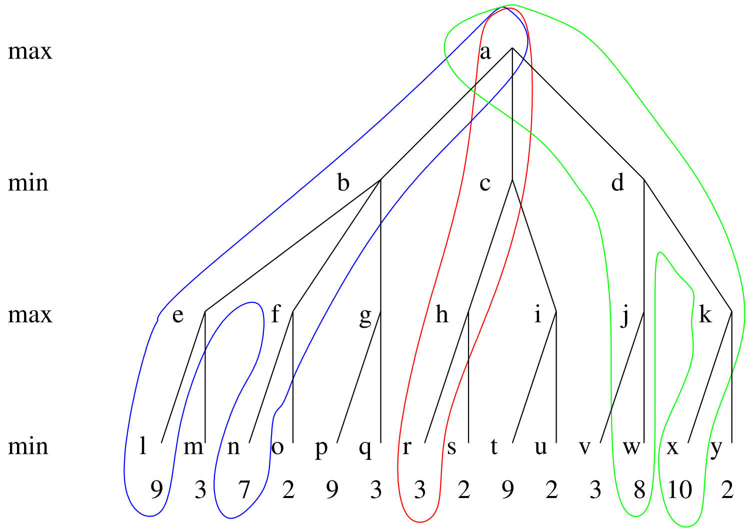




Trois stratégies partielles



On développe la meilleure stratégie, puis on voit que la troisième pourrait être meilleure



On a trouvé une stratégie complète (par construction la meilleure).

L'algorithme SSS* en détail

- S(x) donne pour tout noeud x la liste des successeurs
- e(x) donne l'évaluation d'un noeud x
- Max(x) est vrai si et seulement si x est un noeud max
- On utilise une pile P pour mettre les noeuds encore à traités
- $ranger(P, \langle x, f, h \rangle)$ met $\langle x, f, h \rangle$ dans la pile, ordonnée en ordre descendant par les valeurs h. Si dans P il y a déjà un élément avec la même valeur h on range $\langle x, f, h \rangle$ avant.
- $ représente la valeur infini.
- Il y a deux types de noeuds:
  - f: fermé (la stratégie optimale pour lui est connu)
  - v: vivant

Le programme

```
fonction valeur(x0 : noeud) : réél
début
   P := (<x0,v,$>)
répéter
  <x,s,h> := premier(P)
  P := reste(P)
  x1,x2,...,xm := S(x)
  si s = v alors
    si m = 0 alors
      ranger(P,<x,f,min(h,e(x))>)
    sinon
      si Max(x) alors
        empiler(P,<x1,v,h,>,...,<xm,v,h>)
      sinon empiler(P,<x1,v,h>)
  sinon
    si Max(x) alors
      si x a frère cadet y alors
        empiler(P,<y,v,h>)
      sinon
        z := père(x)
        empiler(P,<z,f,h>)
    sinon
      z := père(x)
      empiler(P,<z,f,h>)
      supprimer de P tous
           les descendants de z
jusqu'à P = (<x0,f,h>)
return h
```


Exemple

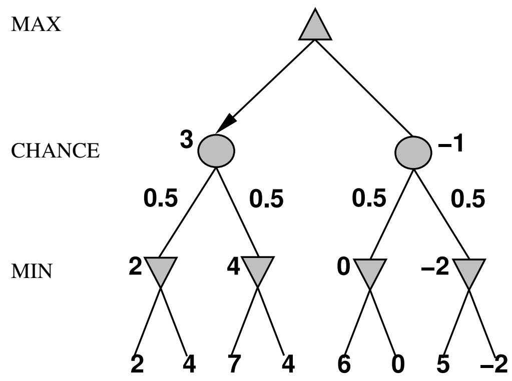
```
<a,v,$>
<b,v,$> <c,v,$> <d,v,$>
<e,v,$> <c,v,$> <d,v,$>
<l,v,$> <m,v,$> <c,v,$> <d,v,$>
<m,v,$> <c,v,$> <d,v,$> <l,f,9>
<c,v,$> <d,v,$> <l,f,9> <m,f,3>
<h,v,$> <d,v,$> <l,f,9> <m,f,3>
<r,v,$> <s,v,$> <d,v,$> <l,f,9> <m,f,3>
<s,v,$> <d,v,$> <l,f,9> <r,f,3> <m,f,3>
<d,v,$> <l,f,9> <r,f,3> <m,f,3> <s,f,2>
<j,v,$> <l,f,9> <r,f,3> <m,f,3> <s,f,2>
<v,v,$> <w,v,$> <l,f,9> <r,f,3> <m,f,3> <s,f,2>
<w,v,$> <l,f,9> <v,f,3> <r,f,3> <m,f,3> <s,f,2>
<l,f,9> <w,f,8> <v,f,3> <r,f,3> <m,f,3> <s,f,2>
<e,f,9> <w,f,8> <v,f,3> <r,f,3> <s,f,2>
<f,v,9> <w,f,8> <v,f,3> <r,f,3> <s,f,2>
<n,v,9> <o,v,9> <w,f,8> <v,f,3> <r,f,3> <s,f,2>
<o,v,9> <w,f,8> <n,f,7> <v,f,3> <r,f,3> <s,f,2>
<w,f,8> <n,f,7> <v,f,3> <r,f,3> <o,f,2> <s,f,2>
<j,f,8> <n,f,7> <r,f,3> <o,f,2> <s,f,2>
```



```
<k,v,8> <n,f,7> <r,f,3> <o,f,2> <s,f,2>
<x,v,8> <y,v,8> <n,f,7> <r,f,3> <o,f,2> <s,f,2>
<x,f,8> <y,v,8> <n,f,7> <r,f,3> <o,f,2> <s,f,2>
<k,f,8> <n,f,7> <r,f,3> <o,f,2> <s,f,2>
<d,f,8> <n,f,7> <r,f,3> <o,f,2> <s,f,2>
<a,f,8>
```


Jeux non-déterministes

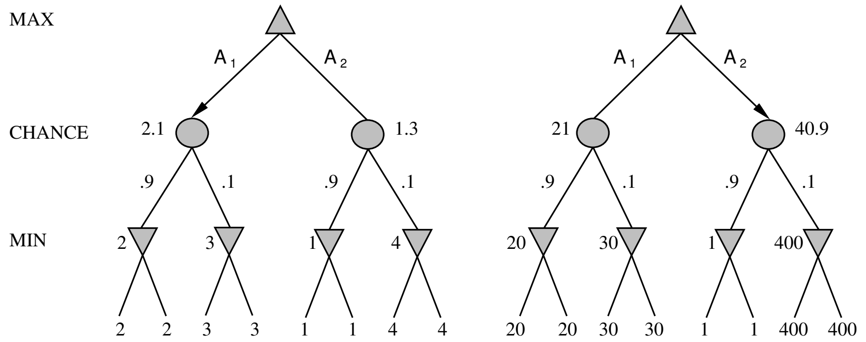
Dans Backgammon par exemple, le lancer des dés détermine les coups corrects. Exemple simplifié avec lancer d'une pièce de monnaie:



Algorithmes pour jeux non-déterministes

- EXPECTIMAX donne coup parfait comme MINIMAX
- On doit considérer les noeuds CHANCE
- On doit ajouter dans l'algorithme: **if** state is a chance node **then return** average of EXPECTIMINIMAX-Value of SUCCESSORS(state)
- L'algorithme α-β peut être adapté, si EVAL est borné.

Valeur exacte des noeuds est importante



- Le comportement est préservé seulement pour chaque transformation positive et linéaire de EVAL.
- EVAL doit être proportionnelle au gain attendu.