EFREI 2010/2011
L3

Aide à la Décision

Jeux (à deux joueurs)

Intelligence Artificielle

Jeux (à deux joueurs)

Cours « Aide à la Décision »
EFREI – 2010/2011 – L3

Abstraction

- Espace d'états
  - Les situations de jeu
  - Un état initial :
    - Le début de partie
    - La position courante
  - Fonction « successeur »
    - Les règles du jeu
    - Connaissance parfaite (pas d'incertitude sur le résultat des actions)
  - Test solution
    - Fin de partie : gagné, perdu, nul
      - Fonction de score

 3

Mais, par rapport aux « recherches off-line dans un espace d'états »…

- Adversaire
  - Incertitude dans l'expansion d'un « arbre de recherche »
    - L'ordinateur ne contrôle pas tout
- Nécessité de faire face à l'imprévu
- Complexité : les solutions exhaustives sont vite trop complexes pour être envisagées
  - Ex. facteur de branchement pour un jeu d'échec de l'ordre de 35…
- Malgré l'incertitude, on cherche pas à pas à choisir le prochain « meilleur » coup à jouer

 4

Quoi qu'il en soit …

- Jeu de plateau
  - État initial
  - Transitions (règles de jeu, déplacement des pièces)
    - Donc une certaine « exploration » est possible
  - État gagnant / perdant / nul
    - Test solution
  - Possibilité de savoir si une situation est « meilleure » ou non qu'une autre
    - « évaluation » / « heuristique »
    - Mais dépend de nombreux paramètres, et notamment du jeu de l'adversaire !

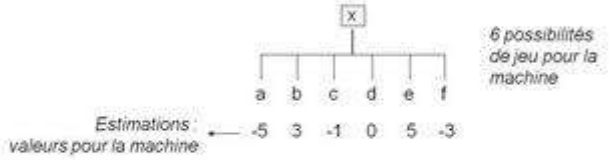
 5

- Adversaire imprévisible
  donc prendre en compte toutes les situations…
- Mais, par exemple aux échecs :
  - Facteur de branchement = 35
  - Nombre moyen de coups par joueur = 50
  - Nombre total d'états = 35^{100}
  - Nombre d'états dans l'espace d'analyse = 10^{120}
- Une exploration off-line est-elle envisageable ?

Composants de base

- Générateur de mouvements
  - La fonction « successeur »
- Le test de terminaison
  - Décision sur l'état courant
    - Gain, perte, nul, rien
- Fonction d'évaluation (gain / utilité)
  - Qualité d'un état donné indépendamment des coups passés ou futurs
  - Évaluation intrinsèque d'une situation de jeu
- Stratégie de contrôle
  - Choix entre différentes options de jeu
    - Stratégie de jeu la plus prometteuse dans l'objectif de vaincre

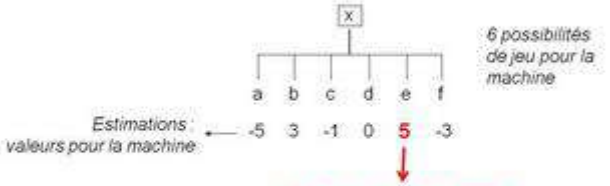
Approche intuitive : analyse à 1 niveau



→ *Quelle branche la machine doit-elle jouer ?*

☐ *position laissée à la machine qui doit jouer*

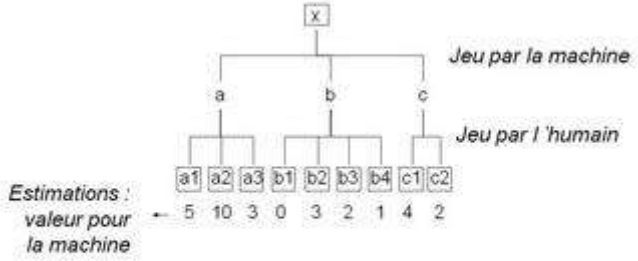
Approche intuitive : analyse à 1 niveau



Situation de jeu ayant la plus grande « valeur » pour la machine

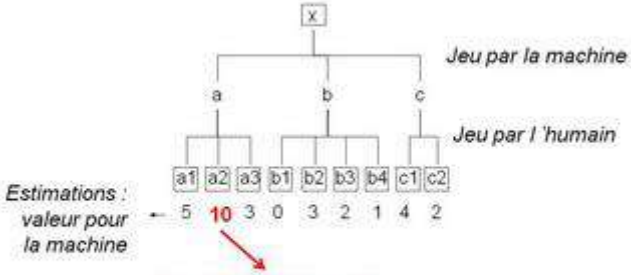
Donc jouer 'e' !

Analyse à 2 niveaux



→ *Quelle branche la machine doit-elle jouer ?*

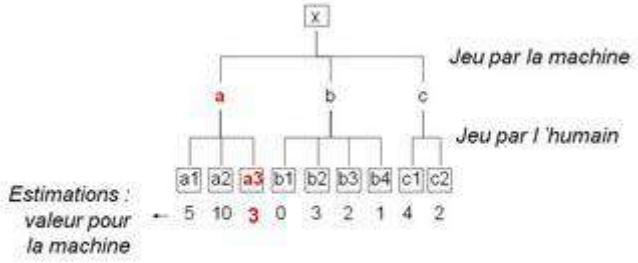
Analyse à 2 niveaux



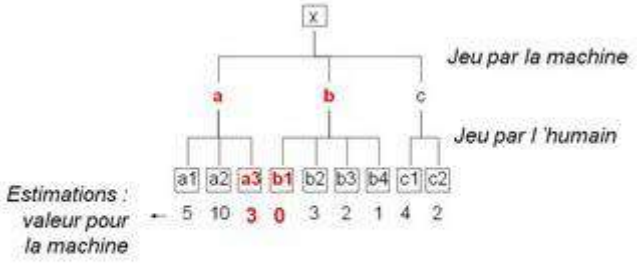
Situation de jeu ayant la plus grande « valeur » pour la machine

Mais…

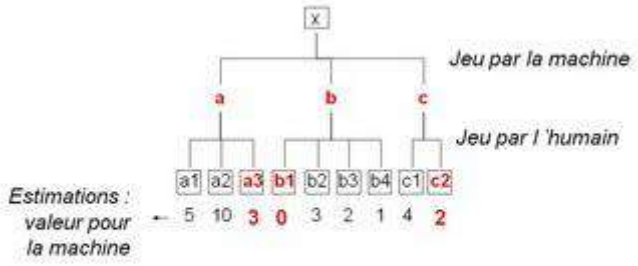
Analyse à 2 niveaux



Analyse à 2 niveaux



Analyse à 2 niveaux



Où est le « gain » de 10 ?!...

Fonction d'évaluation

- e(s) = +∞ si gain pour « Max »
  e(s) = - ∞ si gain pour « Min »
  e(s) = 0 si égalité

- Exemple tic-tac-toe :
  - e(s) =
    nombre de ligne / colonnes / diagonales 'ouvertes' pour « Max »
    -
    nombre de lignes / colonnes / diagonales 'ouvertes' pour « Min »

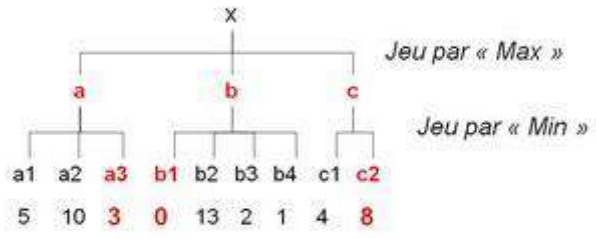
- Hypothèse :

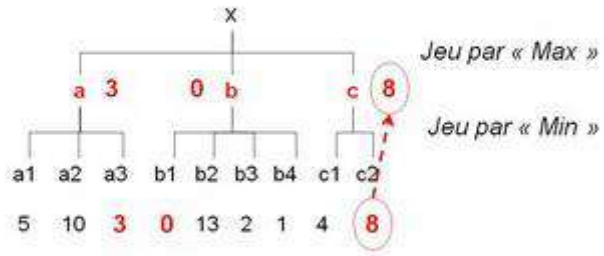
  Fonction d'évaluation de la quelité d'une sitution symétrique par rapport à 0

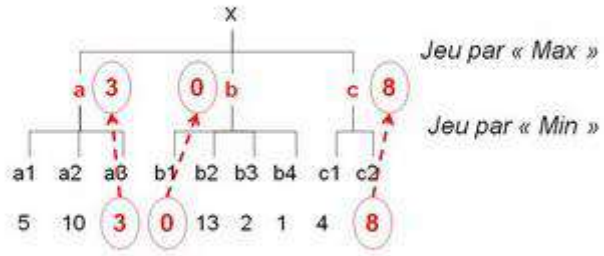
- « NegaMax »

MinMax

- Hypothèse pour « Max » :
  « Min » joue aussi bien que lui !
  - Et donc toute position que « Max » juge bonne pour lui sera jugée par « Min » comme « mauvaise »
    - Et vice-versa !... Mais ça je l'ai déjà dit.
- Principe de base :
  Supposons un arbre d'exploration fini,
  les valeurs sont « remontées » du bas (des feuilles) vers le haut (les successeurs directs de la position courante) pour permettre à « Max » de faire le meilleur choix







Généralisation de l'algorithme MinMax

- Nœuds « Max » et nœuds « Min »
- Remonter la valeur minimale des successeurs d'un nœud « Min »
  Remonter la valeur maximale des successeurs d'un nœud « Max »


```
function MinMax ( nœud )

Le nœud est à la profondeur maximale d'analyse
     ⇒ return  e ( nœud )

Le nœud est « Max »
     ⇒ return    MAX      [ MinMax(x) ]
                 x ∈ Γ(nœud)

Le nœud est « Min »
     ⇒ return    MIN      [ MinMax(x) ]
                 x ∈ Γ(nœud)
```


- Questions évidentes :
  - Qu'est-ce que la « profondeur » ?
  - Les branches d'exploration sont-elles toutes de même « profondeur » ?
    - Une branche peut amener plus rapidement qu'une autre à la fin de la partie…
    - Donc il faut pouvoir s'arrêter n'importe où ?...

```
Function MinMax ( nœud )
Returns a performance value

Partie terminée en 'nœud'
              ⇒ return  performance ( nœud )

¬ Partie terminée en 'nœud'
∧ 'nœud' est « Max »
              ⇒ return    MAX      [ MinMax(x) ]
                        x ∈ Γ(nœud)

¬ Partie terminée en 'nœud'
∧ 'nœud' est « Min »
              ⇒ return    MIN      [ MinMax(x) ]
                        x ∈ Γ(nœud)
```


- Complétude
  - Oui si l'arbre de jeu est fini
- Optimalité
  - Oui si l'adversaire est aussi optimal
- Complexité en temps
  - $\mathcal{O}(b^m)$

aux échecs :
b = facteur de branchement → 35
m = horizon de la recherche → 100

- Complexité en espace
  - $\mathcal{O}(b.m)$

Réaliste ?

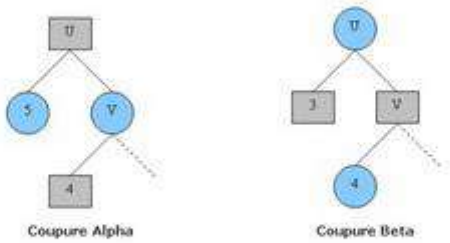
- Solution 1 : profondeur limitée
- Solution 2 : élagage alpha-béta

MinMax en profondeur limitée

- Explorer l'arbre sur une profondeur limitée
- Calculer les valeurs des feuilles
- Les remonter selon l'alternance Min/Max

- Nécessite une bonne fonction d 'évaluation
  - D'autant plus qu'on peut être encore loin de la fin de la partie…
  - Ex. pondération de la valeur des pièces
    - Mais cela peut être beaucoup plus compliqué
      - Par exemple valeurs et pondérations différentes selon le moment de la partie…

MinMax et élagage α - ß



- Nœud « Max » : α-valeur
  - Valeur de son « meilleur » successeur trouvé jusqu'à présent
  - init = - ∞
- Nœud « Min » : ß-valeur
  - Valeur de son « plus mauvais » successeur trouvé jusqu'à présent
  - init = + ∞

```
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
           pour tout fils Pi de P faire
               Val = ALPHABETA(Pi, alpha, beta)
               beta = Min(beta, Val)
               si alpha >= beta alors  /* coupure alpha */
                   retourner beta
           finpour
           retourner beta
       sinon
           pour tout fils Pi de P faire
               Val = ALPHABETA(Pi, alpha, beta)
               alpha = Max(alpha, Val)
               si alpha >= beta alors /* coupure beta */
                   retourner alpha
           finpour
           retourner alpha
       finsi
   finsi
fin
```

Vu sur wikipedia...


```
fonction ALPHABETA(P, A, B) /* A < B */
   si P est une feuille alors
       retourner la valeur de P
   sinon
       Meilleur = -INFINI
       pour tout fils Pi de P faire
           Val = -ALPHABETA(Pi,-B,-A)
           si Val > Meilleur alors
               Meilleur = Valeur
               si Meilleur > A alors
                   A = Meilleur
                   si A > B alors
                       retourner Meilleur
       finpour
       retourner Meilleur
```

Vu sur wikipedia...

- L'élagage n'affecte pas le résultat final

...ouff !!!

- L'efficacité de l'élagage est fonctiond e l'ordre d'apparition / prise en compte des positions « successeurs »
- Au pire, on fait du MinMax « normal »