L'algorithme Minimax : intuitions

- Jeux à deux joueurs ayant des objectifs antagonistes
- Information totale
- On modélise le problème à l'aide d'un arbre.
- Comment construire l'arbre?
  - À partir d'un état on génère l'ensemble des positions que le joueur 1 peut atteindre en jouant un coup (niveau 1).
  - À partir de chacune de ces positions on génère l'ensemble des positions que le joueur 2 peut atteindre en jouant un coup (niveau 2).
  - On s'arrête à une *profondeur maximale* qui modélise la fin du jeu (*arbre de jeu*) ou qui est fixée à l'avance (*arbre de recherche*).

- Comment utiliser l'arbre?
  - On donne une valeur à chaque feuille de l'arbre par une *estimation heuristique*. C'est une valeur *statique*
  - En partant de la racine de l'arbre, il faut joueur (aller vers le niveau 1) de telle sorte que le gain soit *maximal* contre toute défense de l'adversaire.
  - On suppose que l'adversaire joue aussi de manière *optimale* de telle sorte que le gain du joueur 1 soit *minimal*.
  - Ainsi, chaque état de l'arbre est *étiqueté* avec une information `MAX` ou `MIN`.
  - Aussi, chaque état de l'arbre est distingué par une valeur *dynamique* qui est propagée en faisant remonter les valeur statiques au long de l'arbre.

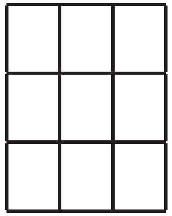
Exemple

L'algorithme Minimax

```
Minimax (s:arbre de recherche): reel

Selon le cas de s:

1) feuille(s):
         renvoyer valeur(s)

2) non_feuille(s) et etiquette(s)=MAX:
         soient (f1,...fn) = succ(s) dans
         Calculer E= {Minimax(fi) | 1 <= i <= n}
         renvoyer Max(E)
3) non_feuille(s) et etiquette(s)=MIN:
         soient (f1,...fn) = succ(s) dans
         Calculer E= {Minimax(fi) | 1 <= i <= n}
         renvoyer Min(E)
```

Arbre de jeux Morpion

État initial:



Fonction d'estimation :

- Perdu : -1
- Nul : 0
- Gagné : 1

Les fonctions d'estimation

Aux échecs :

```
valeur(a) = w1*f1(a) + ...... + wn*fn(a)

wi= un entier
fi= nb de pieces de type i blanches -
    nb de pieces de type i noires
```

Propriétés de Minimax

- S'arrête toujouts si l'arbre est fini
- Optimal si l'adversaire est optimal.
- Si b est le nb de successeurs possibles d'un état 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 : $b \sim 35$, $m \sim 100$, donc solution exacte impossible.

Efficacité de Minimax

- Les ressources sont limitées : si on a 100 secondes pour jouer et on peut explorer uniquement 10^4 noeuds par seconde, alors on peut explorer uniquement 10^6 noeuds par coup.

- Aux échecs : si $b^m = 10^6$ et $b = 35$, alors $m = 4$.

- Idée : élaguer des branches inutiles $\Rightarrow$ Algorithme α-β.

Exemple d'élagage

À partir de l'arbre : on obtient l'élagage :

```
            4               MAX                 4
          /   \                               /   \
         /     \                             /     \
        /       \                           /       \
       /         \                         /         \
      /           \                       /           \
     4             1        MIN          4            <=2
    / \           / \                   / \           / \
   /   \         /   \                 /   \         /   \
  4     6       2     1     MAX       4    >=5      2    XX
 /\     /\     /\     /\             /\     /\     /\
1  4   5  6   2  1   1  1           1  4   5 XX   2  1
```

Propriétés de l'élagage

- L'information de certains noeuds devient *approximative*.
- Ces approximations suffisent à déterminer la valeur *exacte* de la racine.
- Est-ce qu'on *réduit* le nombre de noeuds visités (le nombre d'évaluations statiques)
  - Plus on descend dans l'arbre, plus on réduit le nombre de noeuds visités.
  - Si l'ordre dans lequel les positions sont parcourues est défavorable, alors on visite *toutes les positions*.
  - Sinon, dans le meilleur des cas on obtient $\sqrt{n}$, où n est le nb de positions visitées par Minimax.

L'algorithme α-β : Intuitions

- α est la plus grande valeur que MAX est déjà assuré d'obtenir.
- β est la plus petite valeur que MIN est déjà assuré d'obtenir.

L'algorithme α-β

```
Alfa-Beta(s:arbre de recherche,a,b:reels):reel
Selon le cas de a:

1) feuille(s):
          renvoyer valeur(s)
2) non_feuille(s) et etiquette(s)=MAX:
          soient (f1,.....,fn) = succ(s) dans:
            pour j de 1 a n, tant que a<b
                 a <- max(a,Alfa-Beta(fj,a,b))
          renvoyer a
3)non_feuille(s) et etiquette(s)=MIN:
          soient (f1,.....,fn) = succ(s) dans:
            pour j de 1 a n, tant que a<b
                 b <- min(b,Alfa-Beta(fj,a,b))
          renvoyer b
```

Le premier appel est de la forme : `Alfa-Beta(arbre, -inf, + inf)`

Les notations utilisées dans l'algorithme SSS^*

- On utilise une pile `P` pour les noeuds à traiter.
- Il y a deux types de noeuds : `v` (vivant) et `r` (résolu)
- Chaque noeud est représenté par un triplet `<x,e,h>`, où `x` est le nom du noeud, `e` dit si le noeud est vivant ou résolut et `h` est la valeur du noeud.
- `empiler(P,<x,e,h>)` empile l'élément `<x,e,h>` dans la pile `P` par *ordre décroissant* selon `h`. S'il y a déjà un élément `<x',e',h>` on range `<x,e,h>` `avant` `<x',e',h>`.
- `depiler(P)` retourne le premier élément de la pile `P` et enlève cet élément de `P`.

l'algorithme SSS^*

```
SSS(s:noeud):reel
P:= empiler(pile_vide,<s,v,inf>)
repeter
    <x,e,h> := depiler(P)
    soient (x1,.....,xm) = succ(x)
    selon le cas de e:
    e=v: si m=0  alors empiler(P,<x,r,min(h,valeur(x))>)
                 sinon selon le cas de etiquette(x)
                       MAX: empiler(P,<xm,v,h>,...,<x1,v,h>)
                       MIN: empiler(P,<x1,v,h>)
    e=r: selon le cas de etiquette(x)
         MAX: Si x a frere droit y
                 alors empiler(P,<y,v,h>)
                 sinon empiler(P,<pere(x),r,h>)
         MIN: empiler(P,<pere(x),r,h>)
              supprimer tous les fils de pere(x)
jusqu'a P = <s,r,_>|RESTE
retourner valeur(s)
```

Un exemple

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

```
<n,v,9><o,v,9><w,r,8><v,r,3><r,r,3><s,r,2>
<o,v,9><w,r,8><n,r,7><v,r,3><r,r,3><s,r,2>
<w,r,8><n,r,7><v,r,3><r,r,3><o,r,2><s,r,2>
<j,r,8><n,r,7><r,r,3><o,r,2><s,r,2>
<k,v,8><n,r,7><r,r,3><o,r,2><s,r,2>
<x,v,8><y,v,8><n,r,7><r,r,3><o,r,2><s,r,2>
<x,r,8><y,v,8><n,r,7><r,r,3><o,r,2><s,r,2>
<k,r,8><n,r,7><r,r,3><o,r,2><s,r,2>
<d,r,8><n,r,7><r,r,3><o,r,2><s,r,2>
<a,r,8><n,r,7><r,r,3><o,r,2><s,r,2>
```