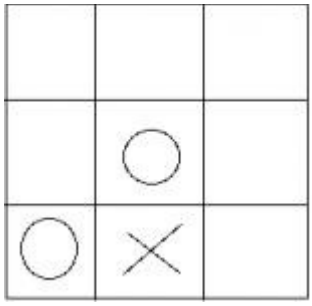
Algorithme MIN MAX

Principe

L'algorithme min-max de Von Neumann est particulièrement adapté aux jeux se jouant à tour de rôle, comme par exemple le morpion ou puissance 4, parmi les plus simples ou les échecs ou le shogui, pour les plus complexes etc....

Le but de min-max est de trouver le meilleur coup à jouer à partir d'un état donné du jeu. Exemple avec un jeu de morpion :

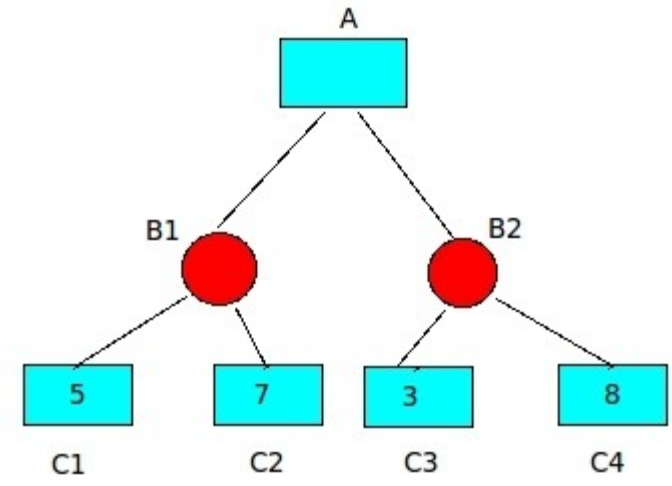


Ici, le meilleur coup à jouer pour le joueur des croix est la case en haut à droite, comment notre machine va pouvoir arriver à ce résultat ?.

On va noter par MAX l'agent qu'on cherche à faire gagner et son adversaire par MIN. Les deux joueurs désirent gagner le jeu. On suppose que le joueur MIN joue logiquement et qu'il ne va jamais rater une occasion de gagner. Si pour gagner le joueur MAX essaie de maximiser son score, le joueur MIN désire aussi maximiser son propre score (ou de minimiser le score du joueur MAX).
L'algorithme MINIMAX, à comme but l'élaboration d'une stratégie optimale pour le joueur MAX.

À chaque tour le joueur MAX va choisir le coup qui va maximiser son score, tout en minimisant les bénéfices de l'adversaire. Ces bénéfices sont évalués en termes de la *fonction d'évaluation statique* utilisée pour apprécier les positions pendant le jeu.

Le principe consiste à construire un arbre de jeu : c'est un arbre qui représente toutes les évolutions possibles du jeu à partir de la configuration actuelle.



La racine (le nœud A) représente l'état actuel du jeu, et la question est ici de savoir si le meilleur coup consiste en B1 ou B2.

On va donc se demander ce que va jouer l'adversaire MI si l'on joue B1, puis B2, et choisir le coup qui nous avantagera le plus, c'est-à-dire donner un "poids" à B1 et B2.

En effet, si on joue B1, l'adversaire MIN va choisir le coup qui nous avantage le moins entre C1 et C2, c'est-à-dire C1, qui à le poids le plus faible.

Donc si on joue B1, on arrivera à la configuration C1, dont le poids est 5.

De même, si on joue B2, l'adversaire nous mènera en C3, dont le poids est 3.

On peut donc dire que le poids de B1 est 5, et celui de B2, 3.
On choisira donc B1, et on recommence à partir de C1.

Nb : On remarque que l'algorithme suppose que l'adversaire jouera toujours le meilleur coup possible.

Pour résumer

On va choisir le maximum des nœuds fils(B) de A, sachant que chaque nœud fils(B) de A est lui-même le minimum de ses nœuds fils(C), et que chaque nœud fils(C) des nœuds fils(B) de A est lui-même le maximum de ses propres nœuds fils, etc... Jusqu'à arriver aux feuilles qui symbolisent la fin de la partie.

Pour un jeu de morpion 3x3, parcourir tout l'arbre à chaque fois n'est pas gênant car il y à n-1! Solutions ou n=9 cases donc 8! => 40320 états, mais par contre, pour des jeux plus complexes tels que les échecs, on peut atteindre les 35^{100} états, ce qui représente beaucoup d'attente avant chaque coup.

On restreint donc le parcours de l'arbre à une certaine profondeur n.
L'algorithme min-max est donc caractérisé par sa profondeur.

On applique donc un processus qu'on répétera jusqu'à atteindre les feuilles terminale ou la profondeur voulue.

Comment donner un poids aux feuilles ?

Effectivement, quand on arrive aux feuilles, on n'a plus de nœuds fils dont tirer un poids.
Il faut donc arriver à évaluer une configuration du jeu, à lui donner un poids.

C'est justement ce qui différencie deux algorithmes min-max : la fonction d'évaluation.

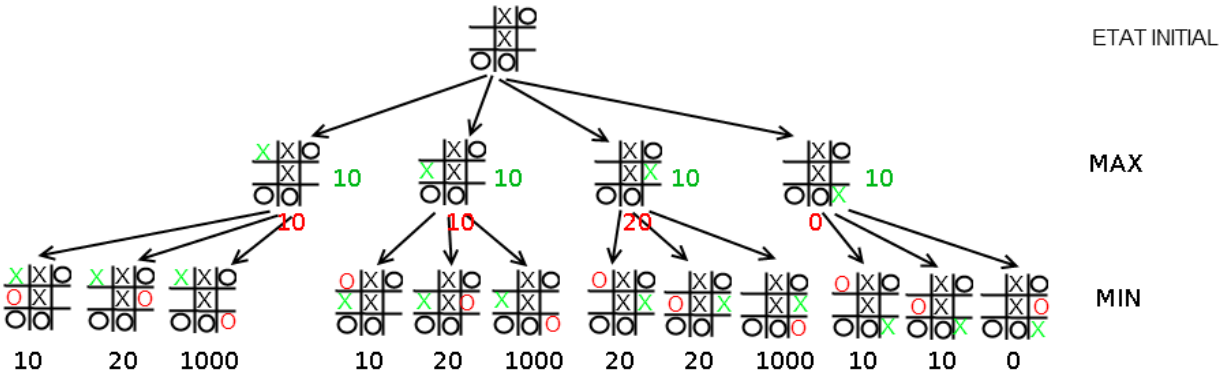
Il est possible d'isoler deux cas :

- Le jeu est fini : si on a gagné, on associe un poids élevé, et si c'est l'adversaire qui à gagné, on associe un poids très négatif, et 0 si personne ne gagne

- Le jeu n'est pas fini : à ce moment-là, il faut trouver un moyen de quantifier l'avantage que nous procure la configuration du jeu.

On a vu que l'évaluation statique des positions terminales de jeu est simple et précise : il faut apprécier selon les règles du jeu s'il s'agit d'une victoire, d'une défaite ou d'une égalité. Pour les positions intermédiaires cette fonction est imprécise, à cause des critères plus au moins subjectifs qui ont été utilisé pour l'évaluation. Le manque d'exactitude de la fonction d'évaluation statique est compensée par l'analyse rigoureuse des conséquences de chaque coup des deux joueurs.

Exemple simple et basique sur une situation au morpion



Dans cet exemple on recherche le min du joueur adverse et on recherche le max de notre coups, le min de l'adversaire à pour valeur 0 et le max pour nous à pour valeur 10, on choisira donc de jouer le coups en bas à droite.

L'algorithme

Une des idées fondamentales de min-max est de simuler les coups, pour analyser leur impact sur le déroulement de la partie. On aboutit donc aisément à l'algorithme suivant, dont nous allons détailler ensemble les fonctions utilisées :

Fonction jouer

```
fonction jouer : void

     max_val <- -infini

     Pour tous les coups possibles
          simuler(coup_actuel)
          val <- Min(etat_du_jeu, profondeur)

          si val > max_val alors
               max_val <- val
               meilleur_coup <- coup_actuel
          fin si

          annuler_coup(coup_actuel)
     fin pour

     jouer(meilleur_coup)
fin fonction
```

Fonction Min

Le but de la fonction Min est de trouver le minimum des nœuds-fils du nœud envoyé en paramètre. Or chaque nœud-fils est le maximum de ses propres nœuds-fils

```
fonction Min : entier

     si profondeur = 0 OU fin du jeu alors
          renvoyer eval(etat_du_jeu)

     min_val <- infini

     Pour tous les coups possibles
          simuler(coup_actuel)
          val <- Max(etat_du_jeu, profondeur-1)

          si val < min_val alors
               min_val <- val
          fin si

          annuler_coup(coup_actuel)
     fin pour

     renvoyer min_val
fin fonction
```

Fonction Max

Elle est en tout point semblable à la fonction Min, sauf qu'au lieu de chercher le minimum de

ses nœuds-fils, elle en cherche le maximum

```
fonction Max : entier

     si profondeur = 0 OU fin du jeu alors
          renvoyer eval(etat_du_jeu)

     max_val <- -infini

     Pour tous les coups possibles
          simuler(coup_actuel)
          val <- Min(etat_du_jeu, profondeur-1)

          si val > max_val alors
               max_val <- val
          fin si

          annuler_coup(coup_actuel)
     fin pour

     renvoyer max_val
fin fonction
```

Fonction eval

Contrairement aux autres fonctions, la fonction d'évaluation dépend presque entièrement du jeu. Une bonne fonction d'évaluation peut faire une grande différence entre deux algorithmes min-max

Une fonction d'évaluation pour le morpion possible est :

- Si la partie est finie :

Si on a gagné, on renvoie un grand nombre (1000 par exemple) - le nombre de coups (on va donc essayer de gagner le plus vite possible).
Si on a perdu, on renvoie un nombre très négatif (-1000 par exemple) + le nombre de coups (pour essayer de survivre le plus longtemps possible).

Et si personne n'a gagné, on renvoie 0.

- Si la partie n'est pas finie :

  On renvoie le nombre de séries de deux pions alignés du joueur, moins celui de l'adversaire

Complexité de l'algorithme

Au échecs, pour une exploration de l'arbre de jeu en profondeur d'abord $\boldsymbol{O}(b^P)$

b=35, p=100 en moyenne soit 35^{100} états.

Au Morpion, complexité - $O((c-1)!/((c-1)-p)!)$

c=9, p<=(c-1)

pour c=3x3=9 et p=8 on 8!/1!=8!=40320 états.

Lien utile : un simulateur de morpion sous forme d'une applet java, qui démontre le fonctionnement de l'algorithme Min Max (http://lwh.free.fr/pages/algo/minmax/minmax.htm)