Cours 5: Algorithme MinMax[1]

1. Introduction

La famille des algorithmes MinMax permet la sélection d'un coup dans les jeux dits « à deux joueurs à somme nulle et information complète », famille dans laquelle se trouve la plupart des jeux de réflexion (Othello, échecs, morpion, puissance 4, Go, etc) :

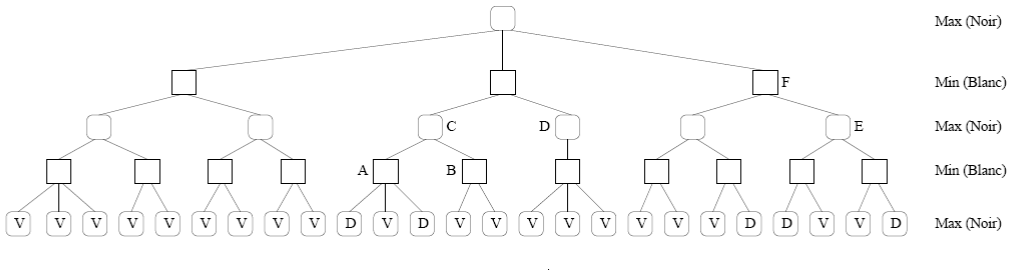
Somme nulle : les gains d'un joueur sont exactement l'opposé des gains de l'autre joueur (ces gains peuvent donc être négatifs).

Information complète : lors de sa prise de décision (i.e. du choix d'un coup à jouer dans le cas qui nous intéresse), chaque joueur connaît précisément:

- ses possibilités d'action,
- les possibilités d'action de son adversaire
- les gains résultants de ces actions,
- les motivations de son opposant.

2. MinMax

L'algorithme MinMax est associé à un arbre de jeu.



Cet arbre est composé:

- d'un nœud racine représentant la configuration de jeu à analyser,
- de nœuds fils qui représentent chacun une configuration du jeu atteignable depuis le nœud racine,
- de branches qui symbolisent les coups possibles dans une configuration données,
- de feuilles qui représentent chacune une configuration finale du jeu.

A chaque feuille (configuration finale) est associée une valeur:

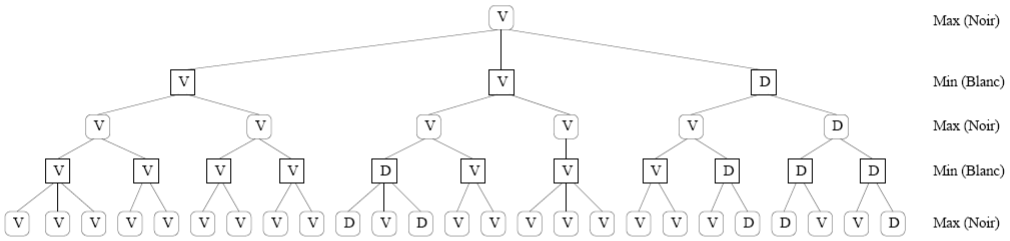
- positive en cas de victoire,
- 0 en cas de match nul,
- négative en cas de défaite.

Le principe de l'algorithme dissocie les nœuds où l'algorithme pourra jouer (nœuds MinMax) des nœuds où l'adversaire pourra jouer (nœuds adverses):

- nœuds MinMax: on associe à ces nœuds la plus grande valeur associée aux nœuds suivants,
- nœuds adverse: on associe à ces nœuds la plus petite valeur associée aux nœuds suivants.

En effectuant cette analyse jusqu'aux feuilles, on peut déterminer quel est le meilleur coup à jouer.

1 Certain passages de ce document sont issus du cours de Liva Ralaivola [http://www.lif.univ-mrs.fr/~liva/],



Comme il n'est pas toujours possible d'explorer l'arbre jusqu'aux feuilles, il est fréquent d'avoir recourt à des fonctions d'évaluations qui retournent une estimation d'une partie inachevée.

Algorithme MinMax:

```
int MinMax(int Profondeur)
{
    if (Game->GameOver() or Profondeur=0)
        return Game->Evaluation() ;

    int Meilleur_Score;
    int Meilleur_Coup;

    if (Noeud==Joueur) {                          // Max node
    {
        Meilleur_Score = -INFINI;
        for (chaque coup c possible) {
             jouer le coup c ;
             int Score=MinMax (Profondeur – 1);
             annuler le coup c ;
             if (Score > Meilleur_Score) {
                Meilleur_Score=Score;
                Meilleur_Coup=c;
            }
        }
    }
    else  {                                       // Min node
        Meilleur_Score = INFINI;
        for (chaque coup c possible) {
             jouer le coup c ;
             int Score=MinMax (Profondeur – 1);
             annuler le coup c ;
             if (Score < Meilleur_Score) {
                Meilleur_Score=Score;
                Meilleur_Coup=c;

            }
        }
    }
    return Meilleur_Score [ & Meilleur_Coup ];
}
```

3. NegaMax

Minimiser une valeur revient à maximiser l'opposée de cette valeur. En s'appuyant sur cette constatation, il est possible d'écrire une version plus compacte de l'algorithme:

Algorithme NegaMax:

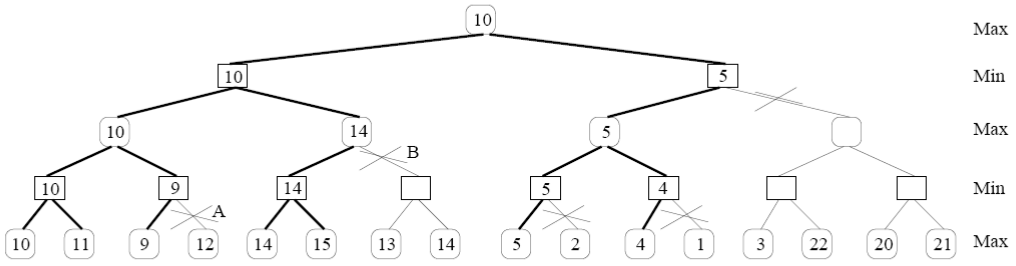
```
int NegaMax(int Profondeur)
{
    if (Game->GameOver() or Profondeur=0) {
        if (Noeud==Joueur)
            return  Game->Evaluation();
        else
            return -Game->Evaluation();

    int Meilleur_Score;
    int Meilleur_Coup;

    Meilleur_Score = -INFINI;
    for (chaque coup c possible) {
        jouer le coup c;
        int Score=-NegaMax (Profondeur – 1);
        annuler le coup c;
        if (Score > Meilleur_Score) {
           Meilleur_Score=Score;
           Meilleur_Coup=c;
        }
    }
    return Meilleur_Score  [ & Meilleur_Coup ];
}
```

4. Elagage Alpha-Bêta

Imaginons que la première branche explorée conduise systématiquement à une victoire, l'algorithme MinMax explorera inutilement les autres branches. Pour diminuer le temps d'analyse, il est possible de supprimer les branches inutiles.



Le principe est de tenir à jour deux variables α et β qui contiennent respectivement la valeur minimale que le joueur peut espérer obtenir pour le coup à jouer étant donné la position où il se trouve et la valeur maximale. Certains développements de l'arbre sont arrêtés car ils indiquent que l'adversaire a l'opportunité de faire des coups qui violent le fait que α est obligatoirement la note la plus basse que le joueur sait pouvoir obtenir ou que β est la valeur maximale que l'adversaire autorisera le joueur à obtenir.

Algorithme NegaMax avec élagage Alpha-Bêta:

```
int AlphaBeta(int Profondeur, int Alpha, int Beta)
{
    if (Game->GameOver() or Profondeur=0) {
        if (Noeud==Joueur)
            return Game->Evaluation();
        else
            return -Game->Evaluation();

    int Meilleur_Score;
    int Meilleur_Coup;

    Meilleur_Score = -INFINI;
    for (chaque coup c possible) {
        jouer le coup c;
        int Score=-AlphaBeta (Profondeur – 1, Alpha, Beta);
        annuler le coup c;
        if (Score > Meilleur_Score) {
           Meilleur_Score=Score;
           Meilleur_Coup=c;
           if (Score>=Alpha) {
                Alpha=Score;
                if (Alpha >= Beta)
                    return Meilleur_Score [ & Meilleur_Coup ];
           }
        }
    }
    return Meilleur_Score  [ & Meilleur_Coup ];
}
```