Stratégie Minimax

Alexandre CHOURA

Juliette LEROUX

Anaëlle MORTAGNE-CODERCH

Noé WESCHLER

---

1. Théorie

L'algorithme de recherche Minimax représente le pilier de méthodes de recherche plus complexes comme l'élagage Alpha-Beta. Cette recherche récursive suppose que l'adversaire jouera de façon optimale, c'est-à-dire, qu'il cherchera à effectuer le meilleur coup possible dans toutes les situations. Cette hypothèse que nous faisons permet ainsi de décider le coup à effectuer, amenant à minimiser la perte de la pire éventualité qui pourrait survenir à la suite d'un *nœud* donné dans un arbre de jeu. Cette recherche peut-être mieux illustrée avec un exemple comme illustré à la figure 1.

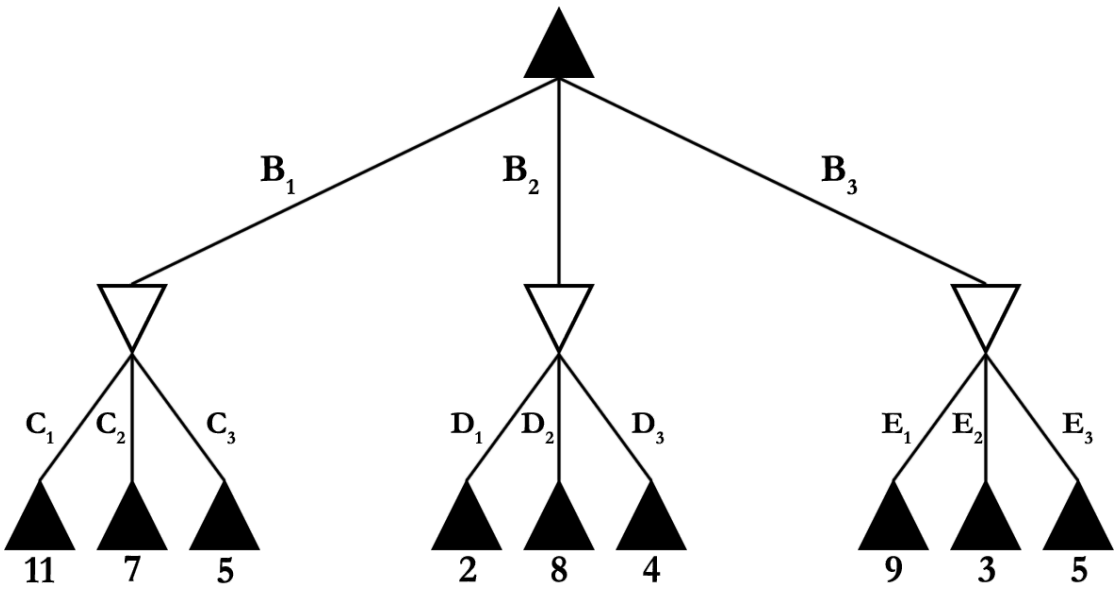

Figure 1: Exemple de Minimax

Pour Othello et sans perte de généralité, nous assignerons le joueur noir au joueur **MAX**, et le joueur blanc au joueur **MIN**. Les 9 *feuilles* de l'exemple d'arborescence de jeu illustré à la Figure 1 montrent l'*utilité* que le joueur max recevra pour atteindre chaque nœud respectif. Le joueur max veut atteindre l'utilité maximale possible et doit choisir la branche qui permettra d'atteindre cette utilité. D'autre part, le joueur min va essayer de minimiser l'utilité de max. Il s'agit d'un point d'information crucial qui permettra de déterminer quelle branche le joueur max prendra. L'utilité des nœuds non-feuilles peut-être calculée de manière récursive à l'aide de l'équation 1.

$$ValeurMinimax(noeud) = \begin{cases} Utilité(noeud) & si\ noeud\ terminal \\ \max\limits_{v \in successeurs(noeud)}(ValeurMinimax(v)) & si\ joueur\ max \\ \min\limits_{v \in successeurs(noeud)}(ValeurMinimax(v)) & si\ joueur\ min \end{cases}$$

Equation 1

Supposons que le joueur noir prenne la branche B_1. Dans ce scénario, le joueur min a le choix entre 3 options: C_1, C_2 et C_3. Puisque le but du joueur min est de minimiser l'utilité de max, et que nous avons supposé que le joueur min joue de façon optimale, alors le joueur min choisira la branche C_3. Ainsi, nous savons que choisir la branche B_1 en tant que joueur max nous mènera à une utilité de 5.

Dans le cas où le joueur max choisit la branche B_2, le joueur min choisira la branche D_1, puisque c'est la branche avec l'utilité la plus basse. Il en résultera par conséquent une utilité de 2 pour le joueur max.

Enfin, si le joueur max suit la branche B_3, le joueur min sélectionnera alors la branche E_2 puisqu'elle mène au nœud avec la plus basse utilité. Cela signifie que la branche B_3 propose une utilité de 3 pour le joueur max. L'ensemble de ces résultats est résumé dans le tableau ci-dessous:

Coup de MAX	Coup de MIN	Utilité finale
B_1	C_3	5
B_2	D_1	2
B_3	E_2	3

Tableau 1: Résultats correspondant à la Figure 1

Comme le montre le Tableau 1, l'utilité la plus élevée que le joueur max peut recevoir est 5, donc ce dernier choisira la branche B_1 et le joueur min la branche C_3. Cette arborescence est mise en valeur en rouge sur la Figure 2.

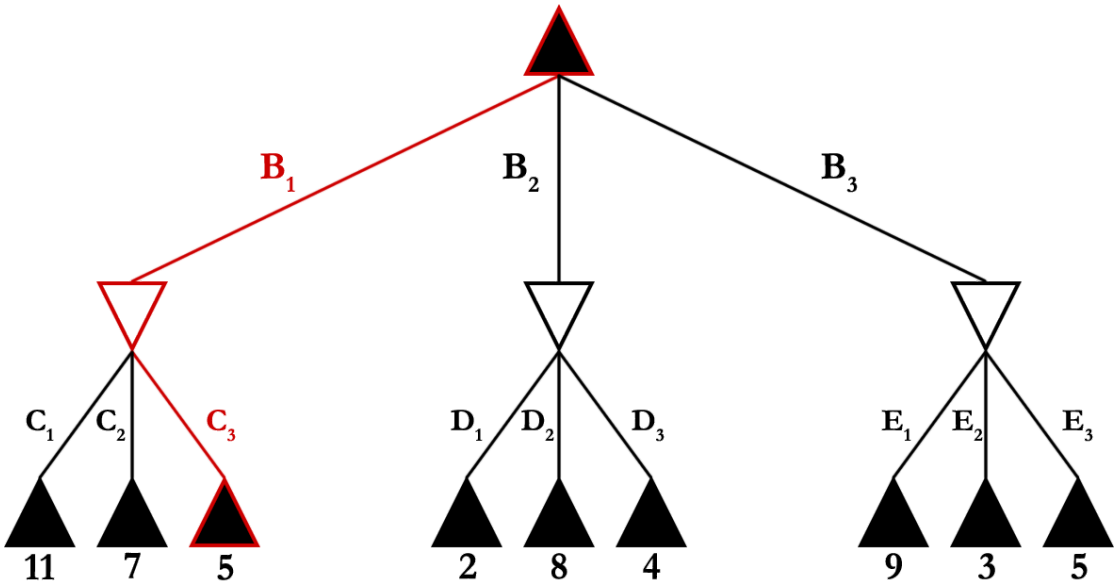

Figure 2: Chemin optimal

2. Implémentation

Une fois la théorie de la recherche Minimax comprise, le prochain obstacle est de pouvoir la mettre en œuvre. Ce n'est pas une tâche triviale, et il est recommandé de se référer à un pseudocode qui est facilement disponible en ligne, représenté ci-dessous.

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth − 1, FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1, TRUE))
        return value
```

Pseudocode de Wikipédia

Vous en trouverez une implémentation en Python dans le fichier nommé MinimaxPlayer.py basée notamment sur l'équation 1.

On notera la nature récursive de la fonction. Nous cherchons récursivement à travers une profondeur *depth* de l'arbre de jeu. Il est aussi important de remarquer la symétrie des situations *if/else* car les stratégies pour le joueur max (noir) et le joueur min (blanc) sont identiques, sauf que l'un essaie de maximiser les utilités, tandis que l'autre les minimisent.

Ainsi, pour chaque nœud (*node*), nous effectuons une recherche récursive dans l'arbre de jeu à travers ses différents successeurs (*child*) en alternant entre le joueur max et min. Dès lors que l'algorithme est arrivé à la profondeur de recherche désirée ou que l'arborescence ne peut pas continuer à partir de ce nœud (devenant donc une feuille), l'utilité du plateau de jeu est déterminée à l'aide d'une fonction heuristique rendant compte de l'état actuel du plateau pour un joueur donné. Ainsi, chaque utilité "remonte", permettant de déterminer le coup à effectuer, comme illustré sur la Figure 2.

3. L'élagage Alpha-Beta

Le projet propose aussi l'implémentation de l'algorithme d'élagage Alpha-Bêta, disponible dans le fichier AlphaBetaPlayer.py.

L'élagage Alpha-Bêta est une amélioration par rapport à la recherche Minimax. Il repose sur le fait qu'il est possible de calculer la décision Minimax exacte sans avoir à élargir la portée de tous les nœuds de l'arborescence du jeu, car on peut souvent savoir à l'avance que certaines branches ne peuvent pas mener à un nœud avec une utilité optimale. Revenons à la Figure 1 pour voir comment tailler notre arbre.

Si le joueur max prend la branche B_1, alors les branches C_1, C_2 et C_3 doivent encore être explorées. Il en résulte une utilité de 5 et il n'y a pas d'optimisations à effectuer (par l'élagage)

Après avoir pris la branche B_1, considérons la branche B_2. Si la branche B_2 est prise, la branche D_1 sera ensuite explorée. La branche D_1 nous dit que le joueur min (blanc) peut sélectionner la branche D_1 pour donner au joueur max (noir) une utilité de 2. Dans ce contexte, la branche B_2 donnera une utilité inférieure ou égale à 2. Ainsi, peu importe l'utilité des branches D_2 et D_3, le joueur max ne pourra jamais avoir une utilité plus grande que 3 à partir de la branche B_2 puisque le joueur min sélectionnera la plus basse utilité des branches D_1, D_2 et D_3. Or, puisque je joueur max à déjà une utilité de 5 après avoir exploré la branche B_1, le joueur max peut ignorer la branche B_2, et ainsi ne pas perdre de temps à explorer les branches D_1, D_2 et D_3. On dit que les branches D_2 et D_3 ont été taillés.

On peut effectuer un raisonnement similaire sur la branche B_3. Après avoir exploré les branches B_1 et B_2, nous allons explorer la branche B_3, nous amenant tout d'abord à la branche E_1. Cette branche fournit une utilité de 9. Puisque $9 > 5$, où 5 était la meilleure utilité trouvé jusqu'à présent (à travers la branche B_1), nous devons explorer la branche E_2. Cette branche amène à une utilité de 3. Nous pouvons maintenant réalisé que l'utilité provenant de l'exploration de la branche B_3 est inférieure ou égale à 3. Or, puisque nous avons déjà une utilité garantie de 5 à la branche B_1, nous pouvons économiser du temps de calcul en élaguant la branche E_3, puisque le joueur max ne prendra pas le chemin B_3. L'arbre de jeu taillé, ainsi que le chemin optimal, sont illustrés par la Figure 3.

Il est important de noter que nous arrivons exactement au même résultat que lorsque nous avons utilisé la recherche Minimax. Ce n'est pas une coïncidence, car l'élagage Alpha-Bêta offre toujours une solution optimale, puisque l'élagage de l'arbre de jeu n'a pas d'effet sur le résultat final.

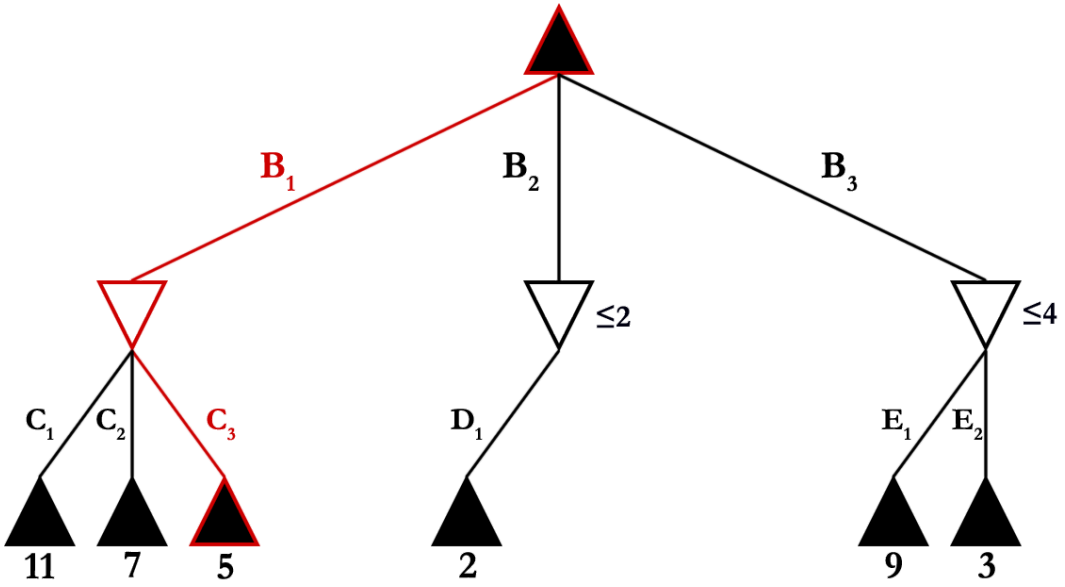

Figure 3: Exemple d'élagage Alpha-Beta

L'implémentation de cet algorithme est extrêmement similaire à celle de Minimax. Il suffit d'ajouter une petite modification au code de l'algorithme Minimax, suivant l'évolution de α et β (tel que défini dans l'équation 2), permettant de sauvegarder les valeurs extrêmes d'utilités trouvé.

$$\begin{cases} \alpha = la\ plus\ grande\ valeur\ d'utilité\ trouvé\ jusqu'ici \\ \beta = la\ plus\ petite\ valeur\ d'utilité\ trouvé\ jusqu'ici \end{cases}$$

Equation 2

Pour chaque nœud exploré lors de la recherche à travers l'arbre de jeu, on compare son utilité à α et β pour savoir si l'on peut élaguer des branches. Nous mettons aussi à joue les valeurs de α et β avec les meilleures valeurs trouvées jusqu'alors durant la recherche dans l'arborescence de jeu.