Travaux pratiques - Algorithmes d'exploration et de mouvement

Algorithme *AlphaBeta*

Dans ce TP, nous étudions une implémentation possible de l'algorithme d'élagage `alpha_beta`.

1 Alpha-Beta

Nous avons vu dans l'ancien TP que l'algorithme `minimax` nous permettait de remonter avec certitude le meilleur choix. Cependant, si votre programme fonctionne correctement, le temps de calcul augmente rapidement (*exponentiellement*) quand on augmente la profondeur maximum d'exploration (ou la taille de la grille du Morpion). Or augmenter la profondeur d'exploration est le seul moyen de rendre votre programme (votre IA) meilleur. Existe-il une méthode qui nous permette d'optimiser l'algorithme initial (`minimax` ou `negamax`) ? Heureusement, l'algorithme `alpha_beta` permet d'optimiser la taille de l'arbre des possibilités à explorer. L'`alpha_beta` se base sur le principe suivant : si, sur un coup, l'adversaire a une possibilité de réponse qui rend ce coup mauvais, il est inutile d'examiner les autres possibilités de réponse à ce même coup, puisqu'il ne les choisira que si elles sont pires.

Algorithme 1: Algorithme *alpha-beta*.

```
procedure alpha_beta(n, α, β)
1  if ( isLeaf(n) ∨ depth = 0 ) then return f(n);
2  else
3      if ( isMinNode(n) ) then
4          β_n ← α;
5          foreach ( s_i ∈ children(n) ) do
6              α_i ← alpha_beta(s_i, β_n, β));
7              β_n ← max(α_i, β_n);
8              if ( β_n ≥ β ) then
9                  return β_n;
10         return β_n;
11     else                                        /* n is a max node */
12         α_n ← β;
13         foreach ( s_i ∈ children(n) ) do
14             β_i ← alpha_beta(s_i, α, α_n);
15             α_n ← min(β_i, α_n);
16             if ( α_n ≤ α ) then
17                 return α_n;
18         return α_n;
```

L'élagage `alpha_beta` (Algorithme 1) est une manière simple et efficace d'élaguer l'arbre de recherche dans le cas d'une recherche `minimax` ou `negamax`. Il permet en général de supprimer au moins la moitié de l'arbre de recherche. Le principe est d'ajouter deux paramètres à la fonction récursive `negamax`, appelés α et β, tels que :

α : représente le score minimum que le joueur maximisant (l'IA) est assuré d'obtenir ;

β : représente le score maximum que le joueur minimisant (l'adversaire du l'IA) peut obtenir.

À chaque fois qu'on explore le fils d'un nœud n, on met à jour β (si n est un nœud min) ou α (si n est un nœud max) et on arrête l'exploration si $\alpha > \beta_n$ (nœud min) ou $\beta_n < \alpha$ (nœud max). Autrement dit, à chaque fois qu'on essaye de chercher le minimum des fils d'un nœud (le père du nœud cherche donc un maximum) et que l'on trouve un minimum qui est inférieur à celui déjà trouvé par le père (α), on peut arrêter la recherche en élaguant le sous-arbre. On procède de la même manière pour les nœuds max.

Question. 1 Implémenter l'algorithme `alpha_beta` pour un morpion défini sur une grille de $m \times m$ cases (m est typiquement supérieur à 4 pour rendre le jeu un peu plus difficile). Comparez l'efficacité du votre `alpha_beta` par rapport à `minimax` et `negamax`.