AgroParisTech UC 2A « La place des bandits en IA »

TP « MinMax, Alpha-Beta et MCTS »

https://antoinecornuejols.github.io/teaching/AGRO/UC-2A-Bandits/UC-2A_IA_Bandits.html

---

1. Introduction

A - Dans ce TP, en utilisant le jeu de Puissance4 (*"Connect4" en anglais*), nous allons d'abord **explorer des variations sur l'algorithme MinMax** :

1 L'effet de l'élagage par Alpha-Beta
2 Une étude « d'ablation » sur une fonction d'évaluation fournie
3 L'effet de modification de paramètres dans cette fonction d'évaluation
4 Les idées que vous pourriez avoir pour d'autres indicateurs d'évaluation que ceux codés dans la fonction d'évaluation fournie

B - Dans un deuxième temps, nous étudierons l'effet du nombre de simulations ("*roll outs*") dans la performance de **l'algorithme MCTS**.

Pour cela, il vous faut télécharger les codes correspondant à :

— `main.py`
— `bot.py`
— `common.py`
— `connect4game.py`
— `minimax.py`
— `minimax_2.py`
— `mcts.py`

2. L'algorithme MinMax avec ou sans élagage par Alpha-Beta

2.1 L'effet de la profondeur d'exploration sur la performance de MinMax

En premier lieu, nous allons lancer un petit tournoi entre l'algorithme MinMax (avec élagage Alpha-Beta) contre un joueur aléatoire.

La commande est : `python main.py -p1 minimax -p2 random`

Ici, l'algorithme minimax est initialisé avec une profondeur d'exploration $d = 5$ (voir la ligne 67 dans le code `main.py`) et le nombre de parties a été fixé à 10 (voir la variable `nb_Games` à la ligne 40).

Vous devriez voir que l'algorithme minimax l'emporte largement.

Maintenant, nous allons faire jouer deux algorithmes MinMax l'un contre l'autre en fixant la profondeur d'exploration d'un des deux algorithmes, disons 'player1', à $d = 5$ et en faisant varier la profondeur d'exploration de 'player2' de 1 à 5.

Fixez 'nb_Games' à 20 (ou à une valeur plus importante si vos ressources calculatoires vous le permettent). *Quelles performances observez-vous dans ces tournois* ?

2.2 Effet de l'élagage Alpha-Beta

Nous avons vu en cours le principe de l'élagage par Alpha-Beta. Il s'agit d'un élagage qui ne modifie pas les valeurs retournées à la racine de jeu, et donc ne modifie pas les coups choisis (à un tirage aléatoire près si plusieurs branches ont la même valeur à la racine).

Dans ces expériences, nous allons chronométrer les temps de jeux quand l'un des deux joueurs n'utilise pas l'élagage par Alpha-Beta. Pour cela, il faut aller en ligne 43 de `connect4game.py`, et remplacer la valeur 'True' par 'False'. Ici, c'est le joueur 1 qui n'utilisera pas Alpha-Beta.

Faites varier la profondeur de jeu des deux joueurs de $d = 1$ à $d = 5$ (voir la ligne 67 dans le code `main.py`).

Avec $nb\_Games = 20$, chronométrez le temps pour que les 20 parties soient jouées pour chacune de ces profondeurs.

1. Est-ce que la performance des joueurs (en termes de parties gagnées) varies sensiblement lorsque d varie ?
2. Pourquoi ?
3. Quel est le nombre de feuilles explorées en moyenne par MinMax sans élagage pour chacune des profondeurs $d = 1$ $d = 5$?
4. Quelles durées avez-vous mesuré pour chacune des profondeurs ?
5. Quelles durées observez-vous quand les deux joueurs utilisent l'élagage Alpha-Beta ?
6. Pour quelle profondeur MinMax sans élagage utilise à peu près le même temps que MinMax avec élagage Alpha-Beta, lui jouant avec une profondeur $d = 5$?
7. Et dans ce cas, quelle est la proportion de parties gagnées par MinMax avec élagage ?

3. Le rôle de la fonction d'évaluation dans l'algorithme MinMax

Il y a deux fonctions d'évaluation distinctes dans l'algorithme MinMax :

1. La première correspond aux gains obtenus en fin de partie
2. La seconde à la fonction qui évalue la « promesse » d'une position du point de vue du joueur qui doit jouer.

3.1 La fonction retournant le gain en fin de partie

La fonction 'is_terminal_node' (ligne 97 du module `minimax.py`) retourne 'True' si la partie est terminée, soit que l'un des deux joueurs ait gagné (la fonction 'winning_move' en ligne 46 a retourné 'True') soit qu'il n'y ait plus de case où jouer.

Le gain de la partie pour le joueur qui a le trait vaut 'math.inf', et la perte vaut '- math.inf', tandis qu'une partie nulle vaut 0. (Voir les lignes 192 à 198 de `minimax.py`).

3.2 La fonction d'évaluation d'une position

C'est la fonction 'score_position' commençant en ligne 133 de `minimax.py` qui évalue une position (correspondant à la variable 'board').

1. Quels sont les critères que cette fonction utilise ?

Nous allons maintenant procéder à ce que l'on appelle une étude d'*ablation*. Pour cela, vous allez modifier le module `minimax2.py`. Ce module est une copie de `minimax.py` permettant de faire jouer un algorithme MinMax avec une autre fonction d'évaluation que celle codée dans `minimax.py`.

2. Fixez à $d = 3$ la profondeur d'exploration pour les deux joueurs (voir la ligne 67 dans le code `main.py`).
   Puis mettez en commentaire l'un ou l'autre des critères d'évaluation de la fonction 'evaluate_window' (ligne 109) (par exemple les lignes 123 et 124).
   Lancez alors la commande `python main.py -p1 minimax -p2 minimax2` pour 20 parties par exemple ('nb_Games = 20 en ligne 40 de `main.py`). Observez-vous une différence de performance entre les deux algorithmes ?
3. Mettez en commentaires d'autres critères et mesurez l'effet sur les performances.

Nous allons maintenant revenir à la version initiale de `minimax2.py`, et modifier les paramètres de la fonction 'evaluate_window' (par exemple en remplaçant 'score += 5' en ligne 124 par 'score += 50', ou bien en modifiant le multiplicateur dans 'score += center_count * 3' en ligne 146).

4. Qu'observez-vous comme impact sur les performances ?
5. Finalement, pouvez-vous coder d'autres critères à prendre en compte dans la fonction d'évaluation ?

4. L'algorithme MCTS

Le module `monte_carlo.py` est appelé si l'un des joueurs utilise l'algorithme Monte Carlo Tree Search. Le nombre d'itérations (d'appels à *roll out*) pour chaque évaluation d'une position est fixé par défaut à la ligne 67 de `main.py`.

Mais il est possible de régler ce nombre pour chaque joueur dans le module `connect4game.py` aux lignes 68 et 76.

1. Modifiez le nombre d'itérations du joueur 1 (ligne 68) en le mettant à 10. Qu'observez vous en terme de gain de partie pour un nombre de parties de 20 ? ('nb_Games' en ligne 40 de `main.py`)
2. À partir de quelle valeur de ce nombre d'itérations pour joueur 1, le nombre de gains de partie est à peu près équilibré entre les deux joueurs (le second utilisant le nombre d'itérations par défaut de 1000) ?

Nous allons maintenant comparer l'algorithme MCTS avec l'algorithme MinMax en essayant d'équilibrer la profondeur d'exploration de ce dernier et le nombre d'itérations de MCTS.

3. En ayant fixé la profondeur d'exploration de 'player2' (minimax) à 3, combien faut-il environ d'itérations pour MCTS pour que le nombre de gains de parties soit à peu près équilibré entre les deux joueurs ?
4. Même chose si la profondeur de 'player2' (minimax) est fixée à 4.
5. Même chose si la profondeur de 'player2' (minimax) est fixée à 5.