TP3 IA – Projet Minimax, Alpha-beta, MCTS

LE TRUNG Ethan – SEREK Matthieu

Dans le cadre du Projet Minimax, Alpha-beta, MCTS, nous nous étions tout d'abord orienté vers le jeu de Hex. Nous avons par la suite décidé de réaliser ce projet sur le jeu du Puissance 4 en Java.

Execution du jeu :

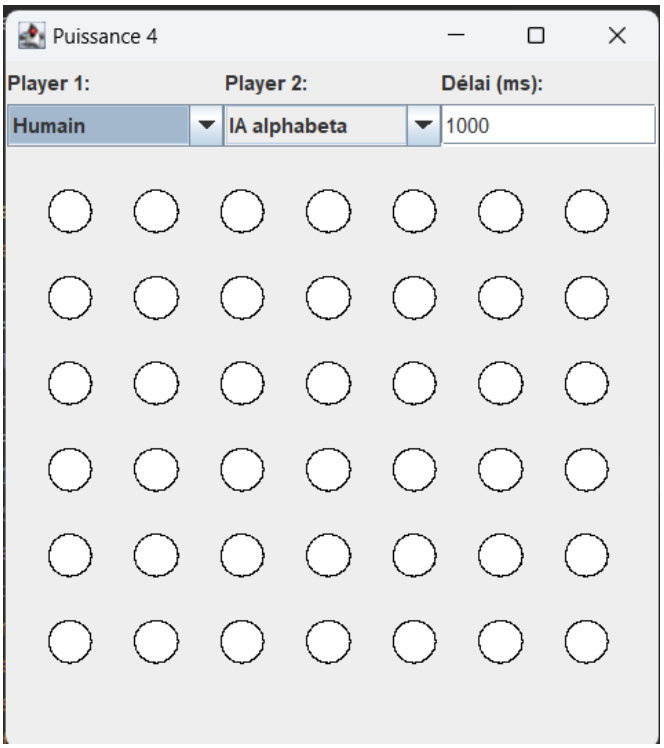
Il vous suffit d'ouvrir le projet « ConnectFourAI » dans votre IDE Java et d'éxécuter la classe Puissance4.java. Assurez-vous de mettre un JDK relativement récent (de notre côté le projet a été codé en OpenJDK 19 et 22).

En outre, il est également possible de saisir le délai souhaité en milli-secondes entre chaque coup de l'IA pour voir le Puissance 4 au tour par tour. Attention, il faut faire « Entrer » après avoir saisi la valeur pour que celle-ci change.

Finalement, pour modifier la profondeur des algorithmes Minimax ou Alpha-beta ou le nombre d'itérations de MCTS, il faut se rendre dans les classes correspondantes (MinimaxIAPlayer, AlphabetaIAPlayer, MonteCarloPlayer) dans la fonction appelée « do_turn », et modifier la valeur de la profondeur ou du nombre d'itérations dans les appels des algorithmes.

Interface Graphique

Nous avons tout d'abord commencé par créer une interface clickable permettant de jouer au Puissance 4 en utilisant JavaSwing. Nous avons créé une classe abstraite Player dont toutes les classes d'IA vont hériter.



Cette interface permet de choisir le type de chacun des joueur :

- Humain : clickable, c'est un humain qui clique sur la colonne souhaitée
- IA Minimax : c'est l'algorithme Minimax qui va déterminer la colonne et placer le pion sur celle-ci
- IA Alpha-beta : c'est l'algorithme Alpha-beta qui va déterminer la colonne et placer le pion sur celle-ci
- IA MCTS : c'est l'algorithme MonteCarlo Tree Search qui va déterminer la colonne et placer le pion sur celle-ci

Fonctions d'évaluation

La fonction d'évaluation utilisée est la même que celle donnée dans le cours :

Pour chaque joueur, faire la somme des points calculés comme suit :

- 4 pions alignés : 1000 points
- 3 pions et 1 case vide alignés : 50 points
- 2 pions et 2 cases vides alignés : 5 points
- 1 pion et 3 cases vides alignés : 1 point

Faire la différence des scores des deux joueurs.

Cependant nous avons utilisé cette fonction d'évaluation de deux manières distinctes.

- Evaluation seulement pour les coups possibles :

C’est la première fonction d’évaluation que nous avons implémenté, c’est également la plus compliquée des deux. Dans le cas du Puissance 4, nous avons 7 colonnes et le coup joué est forcément dans la première case vide de la colonne. Nous avons donc au maximum 7 coups possibles.

Je regarde donc dans la grille toutes les colonnes non pleines (maximum 7). Pour chacune de ces colonnes, j’appelle une fonction *evaluateOnlyPossibleCell(Grid tableau, int column)* qui me renvoie la différence entre le score de ma couleur et le score de la couleur oppposé pour cette case.

Cette fonction fait appel à plusieurs fonctions qui doivent notamment déterminer les lignes, colonnes et diagonales possibles pour cette case et calculer les valeurs de celles-ci en se basant sur la fonction d’évaluation citée ci-dessus.

- Evaluation de toute la grille :

C’est la deuxième fonction d’évaluation vers laquelle nous nous sommes orientés et c’était la plus simple à implémenter. C’est celle-ci qui sera utilisée dans tous les algorithmes. Dans celle-ci, nous évaluons le score général de la grille en évaluant le score de toutes les lignes, colonnes et diagonales de la grille entière. Contrairement à ce que je pensais, c’est la fonction d’évaluation qui marche le mieux dans notre cas et n’est pas spécialement lente. Cependant, elle peut être beaucoup trop conséquente pour d’autres jeux tels que les échecs ou autres. Mais dans notre cas, c’est cette fonction que l’on utilisera dans nos algorithmes Minimax, Alpha-beta et MCTS.

Comme dit plutôt, dans cette fonction d’évaluation, nous regardons simplement toutes les colonnes, toutes les lignes et toutes les diagonales de la grille, que celles-ci soient déjà remplies ou vides. Cette fonction renverra 3 valeurs :

  - La différence du score dans la grille du joueur Jaune et du score dans la grille du joueur Rouge.
  - Le score dans la grille du joueur Jaune.
  - Le score dans la grille du joueur Rouge.

Algo Minimax

L’algorithme Minimax, est déclaré dans la classe MinimaxIAPlayer.

Il prend en paramètres la grille de jeu actuelle, la profondeur utilisée pour l’algorithme, si le joueur passé doit être maximisé ou pas, et le joueur correspondant (sa couleur dans le jeu actuel).

L'algorithme procède ensuite en explorant tous les mouvements possibles pour le joueur actuel et, pour chaque mouvement, en simulant tous les mouvements possibles de l'adversaire. Il renvoie un score à chaque exploration en suivant la méthode d’évaluation utilisée. Cette exploration continue jusqu'à une certaine profondeur ou jusqu'à ce qu'un état terminal du jeu soit atteint (par exemple, victoire, défaite ou match nul). L’algorithme tour à tour maximise puis minimise le score retourné, dans le but de maximiser le score du joueur actuel et de minimiser le score du joueur adverse.

L’algorithme minimax renvoie un tableau de deux valeurs contenant à la fois la valeur de l’action et le coup correspondant.

Algo Alpha-beta

L’algorithme Alpha-beta, est déclaré dans la classe AlphaBetaIAPlayer. C’est une optimisation de l’algorithme Minimax.

Il prend en paramètres la grille de jeu actuelle, la profondeur utilisée pour l’algorithme, la valeur alpha, la valeur beta, si le joueur passé doit être maximisé ou pas, et le joueur correspondant (sa couleur dans le jeu actuel).

L'algorithme procède de manière similaire à Minimax en explorant tous les mouvements possibles pour le joueur actuel et, pour chaque mouvement, en simulant tous les mouvements possibles de l'adversaire. Cependant, il introduit deux valeurs : alpha (la meilleure valeur trouvée jusqu'à présent pour le maximiseur) et beta (la meilleure valeur trouvée jusqu'à présent pour le minimiseur). Ces valeurs permettent de couper des branches de l'arbre de décision qui ne peuvent pas améliorer le résultat pour le joueur actuel, réduisant ainsi le nombre de nœuds à explorer et améliorant l'efficacité de l'algorithme.

L’exploration continue jusqu'à une certaine profondeur ou jusqu'à ce qu'un état terminal du jeu soit atteint (par exemple, victoire, défaite ou match nul). L’algorithme, à tour de rôle, maximise puis minimise le score retourné tout en effectuant des coupures alpha-beta, dans le but de maximiser le score du joueur actuel et de minimiser le score du joueur adverse tout en optimisant le temps de calcul.

L’algorithme Alpha-Beta renvoie un tableau de deux valeurs contenant à la fois la valeur de l’action et le coup correspondant.

Comparaison des algorithmes

Nous avons tout d’abord commencé par tester les deux fonctions d’évaluation. On se rend rapidement compte que pour une même profondeur c’est la fonction évaluant la grille dans son entièreté qui est beaucoup plus efficace. Il semble en effet que l’autre fonction d’évaluation manque de cohérence à de nombreux moments.

On commence par utiliser les deux algorithmes Minimax et Alpha-beta avec une profondeur de 3.

Voici les résultats pour l’affrontement des IA. Chauqe ligne/colonne contient le nom de l’algo utilisé avec sa profondeur.

	Minimax 3	Minimax 4	Minimax 5	Minimax 6	Minimax 7

Alpha-Beta 3	Celui qui commence perd	Alpha-beta gagne	Minimax gagne	Celui qui commence gagne	Minimax gagne
Alpha-Beta 4	Minimax gagne	Celui qui commence perd	Celui qui commence gagne	Celui qui commence gagne	Minimax gagne
Alpha-Beta 5	Alpha-beta gagne	Celui qui commence gagne	Celui qui commence gagne	Celui qui commence perd	Minimax gagne
Alpha-Beta 6	Celui qui commence gagne	Celui qui commence gagne	Celui qui commence perd	Celui qui commence gagne	Minimax gagne
Alpha-Beta 7	Alpha-Beta gagne	Alpha-Beta gagne	Alpha-Beta gagne	Alpha-Beta gagne	Celui qui commence gagne

On observe dans le tableau ci-dessus que les résultats sont « symétriques » : pour deux mêmes pronfondeurs « inversées » (exemple : Minimax 3/Alpha-Beta 5 et Minimax 5/Alpha-Beta 3), les résultats sont les mêmes. De plus, on observe également pour ces cas de figures que les configurations sur la grille sont similaires.

On observe en outre que de nombreuses fois c'est l'algorithme qui commence qui gagne. Ceci est logique puisque le Puissance 4 est un jeu résolu. Cependant, les résultats restent pour des petites profondeurs relativement incohérents (ou du moins inattendus) et semblent être plus « logiques » lorsque que l'on augmente la profondeur.

Finalement, on voit que pour une profondeur de 7, les deux algorithmes gagnent pour toutes les profondeurs inférieures.

Ainsi, comme on aurait pu le conjecturer, les décisions prises par les deux algorithmes sont les mêmes pour les mêmes cas de figure (résultats « symétriques »).

C'est en fait au niveau des nœuds que cela change.Par exemple, si l'on prend tous les cas de figure ou les deux algorithmes ont la même profondeur, on observe que l'algorithme Alpha-Beta visite en permanence bien moins de nœuds que l'algorithme Minimax.

Par exemple, pour 4 de profondeur :

- Alpha-beta visite 7387 nœuds.
- Minimax visite 24012 nœuds.

Pour une profondeur de 7 :

- Alpha-beta visite 413 670 nœuds.
- Minimax visite 6 671 455 nœuds.

Prenons par exemple une profondeur de 9 pour Alpha-Beta et une profondeur de 7 pour Minimax :

- Alpha-Beta visite 3 286 905 nœuds.
- Minimax visite 7 679 803 nœuds.

Même avec une profondeur 2 fois supplémentaires, Alpha-Beta visite toujours moins de nœuds que Minimax.

Ainsi, on observe que la différence de nœuds visités par les deux algorithmes est conséquente : le nombre de nœuds visité par Alpha-Beta est de manière générale bien moindre au nombre de nœuds visité par Minimax. Ceci permet à Alpha-Beta d'être beaucoup plus rapide et donc de pouvoir augmenter la profondeur de sa recherche bien plus que Minimax sans que cela ne ralentisse le jeu.

Ainsi, il est possible de monter jusqu'à une profondeur de 7 pour Minimax pour avoir un temps d'attente « décent » contre une profondeur de 10 pour Alpha-Beta pour avoir un temps d'attente « décent ». En considérant, que nos algorithmes ne sont surement pas optimaux et qu'il existe des machines bien plus performantes que celles qu'on utilise, on imagine facilement l'avantage que représente l'optimisation de l'algorithme Alpha-Beta vis-à-vis de Minimax.

Algo MonteCarlo

L'algorithme MCTS (Montecarlo TreeSearch), est déclaré dans la classe MonteCarloPlayer.

Il prend en paramètres la grille de jeu actuelle, le joueur correspondant (sa couleur dans le jeu actuel), le nombre maximum d'itération de l'algorithme, et une valeur d'exploration.

Pour ce qui est de l'implémentation, nous avons également suivi celle du cours. À chaque l'algorithme sélectionne un nœud de l'arbre de recherche, étend l'arbre en ajoutant de nouveaux nœuds, simule des parties à partir de ces nœuds pour évaluer leur qualité, et utilise les résultats pour mettre à jour les statistiques des nœuds. Il répète ce processus jusuq'à ce qu'il n'est plus d'itération, puis choisit l'action avec les meilleures statistiques accumulées comme étant la meilleure action à prendre.

L'algorithme MCTS renvoie un entier qui correspond à la meilleure colonne où il faut jouer.

Tests influences MonteCarlo

Nous allons maintenant nous intéresser aux moyennes de la branche maximale de l'arbre, en faisant fluctuer le nombre d'itérations et la valeur d'exploration.

	1	2	5
3000	Entre 15 et 22	Entre 13 et 17	Entre 8 et 11
10000	Entre 17 et 23	Entre 15 et 20	Entre 9 et 13

On observe les deux choses suivantes :

- Plus la valeur d'exploration augmente, plus la taille de la branche maximale a tendance à diminuer. Ceci est logique, car la valeur d'exploration influence le compromis qu'a l'algorithme à explorer de nouvelles actions ou à exploiter des actions connues qui sont

intéressantes. Ainsi, plus la valeur d'exploration augmente, plus celui-ci aura tendance à aller vers d'autres branches, et découvrir de nombreuses solutions. A l'inverse, une valeur basse incitera l'algorithme à persévérer dans sa recherche sur une solution qui a déjà un bon score, le faisant creuser cette piste et allongeant la branche.

- Plus le nombre d'itérations augmente, plus la taille de la branche maximum a tendance a augmenter. Ceci est évident : plus d'itérations signifie une exploration plus conséquente et donc un arbre plus grand.

Comparaison des algorithmes

Nous allons directement comparer ses résultats à Alpha-Beta (étant donné qu'Alpha-Beta et Minimax vont réagir pareil).

Les résultats ci-dessus représentent les tendances qu'ont les affrontements à donner comme résultat après avoir joué 3-4 parties de chaque cas de figure. « Ex-aequo » signifie que l'un comme l'autre gagne autant de partie. Ce sont des approximations étant donné que l'algorithme MonteCarlo ne joue pas toujours de la même manière (pas comme Minimax ou Alpha-Beta). Ils ne déterminent pas les résultats que l'on aura si l'on re-simule les calculs.

	MCTS 1000	MCTS 3000	MCTS 5000	MCTS 10000	MCTS 50000
Alpha-Beta 3	Alpha-Beta gagne	Celui qui commence gagne	ex-aequo	MCTS gagne	MCTS gagne
Alpha-Beta 4	Alpha-Beta gagne	Celui qui commence gagne	MCTS gagne	ex-aequo	MCTS gagne
Alpha-Beta 5	Alpha-Beta gagne	Alpha-Beta gagne	Ex-aequo	MCTS gagne	MCTS gagne / celui qui commence gagne
Alpha-Beta 6	Alpha-Beta gagne	Alpha-Beta gagne	Celui qui commence gagne	MCTS gagne / celui qui commence gagne	MCTS gagne / celui qui commence gagne
Alpha-Beta 7	Alpha-Beta gagne	Alpha-Beta gagne	Alpha-Beta gagne	MCTS gagne / celui qui commence gagne	Ex-aequo

On observe des résultats semblent cohérents :

- Pour peu d'itérations côté MonteCarlo, on aura un Alpha-Beta qui gagnera en permanence.

- Puis, à partir de 5000 itérations, on aura plus d'ex-aequo. Beaucoup de cas où le joueur qui commence remporte la partie.
- Finalement, pour plus d'itérations, on verra que le MCTS aura tendance a remporter la victoire.

Finalement à partir de certaines valeurs, on retrouve le fait que le puissance 4 est un jeu résolu, car c'est souvent le joueur qui commence qui gagne.

L'algorithme MCTS peut aller jusqu'à une valeur de 150 000 itérations pour avoir un temps d'attente convenable. Encore une fois, il doit être possible d'optimiser notre programme ainsi que de le faire tourner sur une machine plus puissante que la notre.

Finalement, on observe que comparé aux deux algorithmes précédents, le MonteCarlo se comporte bien plus comme un humain : là où les parties étaient les mêmes à chaque fois pour l'Alpha-Beta contre le Minimax, ce n'est pas le cas ici, et chaque coup du MCTS peut varier.

Conclusion

En conclusion, l'algorithme Alpha-Beta est bien un optimisation de l'algorithme Minimax, améliorant considérablement l'efficacité de celui-ci en réduisant de manière considérable le nombre de nœuds a visité. C'est via le principe de l'élagage qu'il améliore ceci.

De plus, on a vu que le MCTS est influencé par sa valeur d'exploration ainsi que le nombre d'itérations maximum dont il dispose.

Finalement, on retrouve bien le fait que le Puissance 4 soit un jeu résolu tout au long de notre TP.