Projet Intelligence Artificielle

Othello

Chicha Jordan & Halfon Nessim - Jeudi 3 Mai 2018

Introduction

Dans le cadre de l'UE Intelligence Artificielle il nous a été demander de réaliser une IA s'intégrant dans un jeu tour par tour.
Nous avons décider de choisir comme jeu l'Othello dans lequel nous avons du implémenté une IA utilisant l'algorithme Monte Carlo Tree Search (MCTS) de profondeur 1.
Pour réaliser ce projet nous avons utiliser le langage Python 3.

1. Principe et réalisation

Le principe de l'algorithme Monte Carlo est très simple. Au moment où c'est à l'ordinateur de jouer celui-ci regarde quelles possibilités il a. Admettons qu'il ait devant lui la possibilité de faire 3 coups légaux différents, comment savoir lequel lui octroie le plus de chance de gagner ? Pour ce faire l'ordinateur va partir du premier coup légal et va simuler un nombre donné de partie jusqu'à leurs termes. Apres avoir fait cela, il stockera donc les résultats de ces parties (nombre de victoires et nombre de défaites). Il fera ainsi avec les deux autres coups légaux. Enfin il aura juste à comparer les différents résultats et effectuera le coup dans lequel il a remporté le plus de parties simulées.

Pour coder le module MonteCarloIA nous avons du créer un module permettant simplement de générer un déplacement légal aléatoire.
Le module marche de la façon suivante : nous copions l'objet du plateau de jeu (à l'aide de la fonction deepcopy()) dans l'état où il est au moment où on lance Monte Carlo, cet objet contient le nombre de pions noirs, blancs ainsi que de cases vides. Après cela nous récupérons tous les coups légaux et pour chaque coup légal nous simulons un nombre de parties à l'aide de la méthode jouerPartie() du module Jeu. Cette méthode joue une partie jusqu'à la fin en renvoyant le gagnant. On vérifie ensuite que le gagnant correspond au joueur qui a la main et si c'est le cas on comptabilise une victoire.
À la fin on choisit le coup légal ayant remporter le plus de partie.

2. Resultats

Pour pouvoir mettre en place des analyses nous avons simulé plusieurs cas distincts. Nous avons laisser l'ordinateur jouer contre lui même en laissant tourner deux IA. Ayant un module qui génère des déplacements aléatoires nous avons tester plusieurs cas de figure.

Nous avons joint un fichier PDF réalisé à l'aide d'Excel contenant toutes les analyses effectuées.

Ce que nous pouvons constater de manière générale c'est que plus l'écart de simulations est grand plus le joueur ayant le plus grand nombre de simulation à de chance de gagner la partie. Lorsque deux Monte Carlo ayant les mêmes nombres de simulations s'affrontent le résultat avoisine le 50/50. Cela dit nous avons pu remarquer que le joueur ayant la main de début (le joueur noir) gagne plus souvent que le second (environ 60% de victoires pour le premier à jouer et 40 pour le second).

L'IA présente malgré tout un temps de réponse assez élevé et reste régit par les lois de l'aléatoire, ce qui veut dire que nous ne sommes pas à l'abris de voir gagner une Monte Carlo avec 10 simulations face à une Monte Carlo ayant 10 voir 20 fois plus de simulations. Même si la probabilité que cela arrive est minime, elle est bel et bien existante.

L'algorithme est néanmoins très fonctionnel car contre une IA utilisant des déplacements random celle-ci gagne tout le temps (cela ne veut pas dire que Monte Carlo est infaillible contre une IA utilisant des déplacements random).

Conclusion

Nous pouvons conclure que l'algorithme Monte Carlo de profondeur 1 n'est pas compliqué a implémenter et parait être un bon compromis pour une IA modeste dans des jeux tour par tour.
Cependant il présente quelques avantages et inconvénients.

Pour les avantages nous pouvons noter le fait que Monte Carlo n'exige aucune connaissance stratégique ou technique du domaine où il est utilisé pour prendre des décisions raisonnables. L'algorithme fonctionne sans connaissance du jeu, il lui suffit de connaitre les coups légaux et les conditions de fin. Cela permet de pouvoir exporter cet algorithme facilement sans grosse modification.
Cet algorithme peut être arrêter à tout moment pour renvoyer la meilleure estimation actuelle si l'on souhaite récupérer un résultat immédiatement.

Pour les inconvénients nous pourrions en premier lieu parler du temps d'exécution de cet algorithme, en effet, plus on va simuler de partie plus on aura de chance de gagner (nous pourrions assimiler ça à une recherche dichotomique). De ce fait simuler plusieurs parties jusqu'à leurs fins prend énormément de temps. De plus étant dans une simulation aléatoire il est possible que certains noeuds (parfois les plus avantageux) ne soient pas explorés.

Néanmoins la performance de cet algorithme peut être amélioré en utilisant des techniques intelligentes. Pour commencer il est possible de vérifier qu'une simulation n'ait pas été effectuée, en effet, l'utilisation de l'aléatoire peut engendrer des simulations identiques et donc une utilisation inutile des ressources. On peut aussi mettre en place des conditions de base vérifiables avant de commencer les simulations. Par exemple dans le jeu de l'Othello certaines cases sont plus avantageuses que d'autres comme les coins car ce sont des cases qui ne peuvent être récupérer par l'adversaire nous pourrions donc imaginer qu'une versification préalable sur la case soit effectuée et que si cette dernière se trouve être un coin nous n'ayons pas l'obligation d'utiliser l'algorithme.
Nous pourrions aussi stopper la recherche si par exemple à la première série de simulation nous obtenions une moyenne interessante de victoires.
Pour synthétiser nous pouvons donc associé Monte Carlo aux connaissances du domaine pour éviter des développement de noeuds inutiles.