Rapport de projet

Intelligence Artificielle

Dylan Licho
Nicolas Kleinhentz

2019 - 2020
Université de Lorraine

Fonctionnement du programme

Compilation

La compilation, à partir du terminal, se fait avec la commande *make*, en effet le Makefile permet la création du fichier executable et la commande *make clean* permet de supprimer le fichier exécutable.

Execution

Le programme s'execute avec la commande *./puissance temps* avec le temps étant le temps à accorder au MCTS-UCT. Une option d'affichage de debug existe en utilisant la commande *./puissance temps 1*. Cet affichage étant uniquement du debug il n'est en rien esthétique.

Fonctionnement

Le programme commence par demander le critère voulu ainsi que le choix du premier joueur. Ensuite le choix du coup par l'utilisateur se fait en choisir la colonne dans laquelle il veut joueur. Pour le coup de l'ordinateur l'algorithme du MCTS-UCT permettra de jouer le coup le plus adapté avec une certaine limite de temps pour le calcul.

Pour le MCTS-UCT de notre programme, il fonctionne comme suit :

- Selection : l'algorithme parcours, à partir de la racine, les fils ayant la plus grande B valeur jusqu'à arriver à un noeud terminal (*nb_enfants < 1*) ou un dont tous les fils n'ont pas été développés. Notre fonction *tousFilsDeveloppes(Noeud*)* compte les fils développés du noeud (*nb_simus > 0*) et compare le compte avec le nombre de fils du noeud en renvoyant *1* (*true*) s'il sont égaux et *0* (*false*) sinon.
- Développement : on cherche à développer le noeud choisi par l'étape précédente, s'il n'est pas final (fonction *estFinale(Noeud*)* qui renvoie *1* (*true*) si la partie est finale/terminée et *0* (*false*) sinon), nous avons ici deux cas. D'une part, si le noeud choisi n'a pas encore de fils, on créé tous ses fils puis on choisi aléatoirement l'un deux pour la prochaine étape. D'autre part, si le noeud a déjà ses fils créés, on choisi aléatoirement l'un d'eux n'étant pas développé pour la prochaine étape.
- Simulation : la simulation consiste à itérer sur une état temporaire en y appliquant des coups aléatoires jusqu'à ce que la partie se finisse (par une victoire ou un match nul). Une optimisation consistant à choisir un coup spécifique est implémentée si ce coup entraîne la victoire de l'ordinateur (demandée par la question 3).
- Mise à jour : on parcours, du noeud actuel (ayant donc été choisi à l'étape de développement) jusqu'à la racine de l'arbre, tout en incrémentant le nombre de simulations de chaque noeud de 1 et le nombre de victoires de 1 si l'ordinateur est gagnant de la simulation.

On fini par choisir le meilleur coup par rapport au critère choisi et ce coup sera donc celui joué par l'ordinateur.

Questions

1. Affichez à chaque coup de l'ordinateur, le nombre de simulations réalisées pour calculer ce coup et une estimation de la probabilité de victoire pour l'ordinateur.

Voir execution du programme.

2. Testez différentes limites de temps pour l'ordinateur et comparez les résultats obtenus. A partir de quel temps de calcul l'ordinateur vous bat à tous les coups ?

A partir de 14 secondes, l'ordinateur change de stratégie et à partir de 16 secondes, il devient imbattable.

3. Implémentez l'amélioration des simulations consistant à toujours choisir un coup gagnant lorsque cela est possible. Comparez la qualité de jeu de cette nouvelle version avec la précédente et expliquez à quoi cela est dû.

La qualité des coups joués par l'ordinateur est meilleure avec l'optimisation. En effet l'optimisation permet à l'IA d'allouer plus de temps de calcul pour l'exploration car la simulation est raccourcie au possible.

4. Si vous travaillez en C, quelle est l'utilité ici de compiler avec gcc -O3 plutôt qu'avec les options par défaut ? Donnez des exemples illustratifs.

L'option -03 permet d'utiliser toutes les optimizations d'exécution et de compilation supportées par gcc. Cela nous permettra d'avoir une IA qui ira parcourir l'arbre des états de jeu bien plus profondément que sans l'optimisation, et on aura donc, supposément, une meilleure IA pour un même temps de calcul. Le nombres de simulations double par rapport à l'execution sans l'option.

5. Comparez les critères "max" et "robuste" pour choisir le coup à jouer en fin d'algorithme. Conduisent-ils souvent à des coups différents ? Lequel paraît donner la meilleure performance ?

Les critères "max" et "robuste" devraient conduire à des coups quelques fois différent. En effet le critère « max » est axé sur le ratio victoires sur simulations tandis que le critère « robuste » est axé sur les simulations uniquement. L'exploration en utilisant le critère « max » est le plus efficace pour notre programme. L'exploration en utilisant le critère « robuste » n'est pas efficace dans notre programme mais ceci pourrait être dû a une mauvaise compréhension du MCTS-UCT ou d'une erreur de programmation.

6. Donnez une estimation du temps de calcul nécessaire pour jouer le premier coup avec l'algorithme Minimax (sans alpha-beta ni limitation de profondeur).

En estimant le facteur de branchement moyen à 6, et la profondeur moyenne à 21 (42 étant le nombres de cases du plateau, on estime le remplissage à la moitié en moyenne), on a donc une complexité de temps de calcul de l'ordre de $O(6^{21})$ donc $3*10^{35}$. L'âge de l'univers étant $14*10^{9}$. On considère un processeur 3GHz donc $3*10^{9}$ opérations par secondes. En une année on a 365*24*60*60 secondes donc $31*10^{6}$. On peux donc executer $3*10^{9} * 31*10^{6} = 9,3*10^{16}$ opérations par années. On a donc une complexité de $3*10^{35}$ et $9,3*10^{16}$ opérations par année, ce qui équivaut à $3,22*10^{18}$ et donc pas envisageable en terme de calcul.