Intelligence Artificielle

Rapport

Karim Abdi – Reynault Sies

Introduction : Ce document présente les réponses aux questions du projet de l’UE I.A. dans lequel le but est de réaliser une I.A. utilisant l’algorithme MCTS pour le jeu du puissance 4.

**Fonctionnement du programme**

**A partir du GitHub :**

git clone https://github.com/Reynault/Projet\_IA\_Puissance\_4.git

cd Projet\_IA\_Puissance\_4

mkdir build

cd build

cmake ..

make

**A partir du code source fourni :**

cd Projet\_IA\_Puissance\_4

mkdir build

cd build

cmake ..

make

# Question n°1

On obtient pour 5 secondes de recherche, un nombre de simulation entre 300k et 400k.

La probabilité de victoire de l’ordinateur pour chaque coup est très rarement en dessous de 0.5.

Ici, la probabilité de victoire est une estimation basée sur le nombre de simulation et le nombre de victoire du meilleur coup choisi par l’ordinateur. (nb\_victoires / nb\_simulations)

# Question n°2

1 secondes 🡪 Imbattable avec entre 60k et 100k simulations.

# Question n°3

Après avoir implémenter l’amélioration des simulations consistant à toujours choisir un coup gagnant lorsque cela est possible on constate que :

- Le nombre de simulations à diminuer (ralentissement du programme) : cela est due au nombre de tests à faires pour vérifier si les coups son gagnant ou pas.

- Le temps de calcul à augmenter.

# Question n°4

L’argument -O concerne l’optimisation réalisée lors de la compilation du programme. Avec l’argument -03, l’optimisation réalisée par le compilateur est la plus grande et permet d’améliorer la performance du code, et/ou la taille en activant l’ensemble des flags d’optimisation.

Les optimisations les plus importantes sont les suivantes :

* La compilation est plus rapide.
* Effectuer un certain nombre d'optimisations mineures qui sont relativement coûteuses.
* Supprimer les instructions d'extension redondantes.
* Réorganiser les instructions pour éliminer les blocages d'exécution en raison de l'indisponibilité des données requises.

# Question n°5

Pour choisir le coup à jouer en fin d’algorithme on a 2 critères :

-max : choisir le coup qui maximise la récompense **nombre de** **victoires / nombre de simulations**.

-robuste : choisir le coup qui maximise le **nombre de simulation.**

En comparant les deux critères on constate que les deux contraintes ne conduisent pas toujours à des coups différents.

Le critère max parait donner la meilleure performance vue qu’il prend on considération le nombre de victoire par rapport au critère robuste qu’il ignore.

# Question n°6

Estimation du temps moyen nécessaire pour jouer le premier coup avec l’algorithme minmax :

On a 7 colonnes, le nombre moyen de coups moyen a chaque nœud est égale à 7 : donc le facteur de branchement b = 7

On a 42 cases dans le plateau donc on déduit que la profondeur de l’arbre est d = 42.

La recherche d’un coup en développant un arbre de profondeur d nécessite de générer 8^d nœuds : on a donc 7^42 nœuds.

Un processeur avec une fréquence **3GHZ** exécute 3 \* 10^9 instructions / secondes.

Donc pour parcourir tous l’arbre avec un tel processeur il faut : 7^42 / 3 \* 10^9 = 1.04 \* 10 ^ 26 secondes = 3.29 **\* 10 ^ 18 années**

On remarque que le temps de calcul est largement plus grand que l’âge de l’univers, donc il est pas optimale.