23/02/2020

Puissance 4

Alexis Cesaro – Théo Alison

1) Fonctionnement du programme

1.1) Présentation du programme

Le programme est un jeu de type puissance 4 réalisé en C++. Il est basé sur le fichier jeu.c fournis avec l'énoncé.

Le projet a été réalisé avec l'IDE CLION pour la compilation, l'exécution et les tests pour ce compte rendu.

1.2) Compilation et exécution du programme

L'algorithme MCTS

L'algorithme MCTS utilise des récompenses. Ces récompenses ont été définies à 0 ; 0,5 et 1 pour une défaite, un match nul et une victoire respectivement.

La constante C utilisée pour le calcul de la B-valeur est égale à $\sqrt{2}$ (une valeur approximative a été fournis dans le code.

Le programme et ces options

Pou rapporté la possibilité de modifier le comportement du programme, nous avons utilisé la fonction **getopt** et ainsi passer divers paramètres au programme.

Ainsi une commande pour lancer le programme ressemblera à ceci

./Power4 [-param number] [-param number] …

Plusieurs options sont disponibles :

-t <arg>, où arg est un nombre décimal positif non nul. Cette option permet de définir la limite de temps imposée à l'ordinateur pour l'exécution de l'algorithme MCTS.

Note : Dans le cas où le paramètre limitant le nombre d'itération (-i) est ajouté, c'est le paramètre le plus contraignant qui sera retenu.

-i <arg>, où arg est un nombre entier positif non nul. Cette option permet de définir le nombre d'itérations maximal imposé à l'ordinateur pour l'exécution de l'algorithme MCTS. Par défaut, ce paramètre est prioritaire sur la limite de temps.

Note : De la même manière que pour l'option de limitation du temps, si l'option -t de temps est également donnée, le paramètre le plus limitant sera appliqué.

-o <arg>, où arg étant un nombre entier positif non nul. Cette option permet de définir le niveau d'optimisation/amélioration de l'algorithme MCTS. Plusieurs niveaux d'optimisation sont disponibles :

- **0 :** fonctionnement basique de l'algorithme MCTS avec UCB (UCT) (simulations aléatoires)
- **1 :** (***Par défaut*** - Question 3) amélioration des simulations consistant à toujours choisir un coup gagnant lorsque cela est possible.
- **2 :** lorsqu'un coup gagnant est possible, l'algorithme n'est pas utilisé et le coup est joué directement.

-v <arg>, où arg étant un nombre entier positif non nul. Cette option permet de définir le niveau de verbosité du programme. Plusieurs niveaux d'affichage est disponible.

- **0 :** aucun affichage excepté la demande de coup et le plateau de jeu.
- **1 :** (***Par défaut*** - Question 1) affichage pour chaque coup de l'IA le nombre total de simulations réalisées (ou le nombre d'itérations), d'une estimation de la probabilité de victoire pour l'ordinateur en jouant ce coup ainsi que le nombre total de simulations cumulées.
- **2 :** affichage à chaque tour de l'IA du temps passé dans la boucle principale de l'algorithme MCTS.
- **3 :** affichage à chaque tour de l'IA du nombre de simulations réalisées pour chaque coup.
- **4 :** affichage à chaque tour de l'IA de la moyenne des récompenses pour chaque coup.

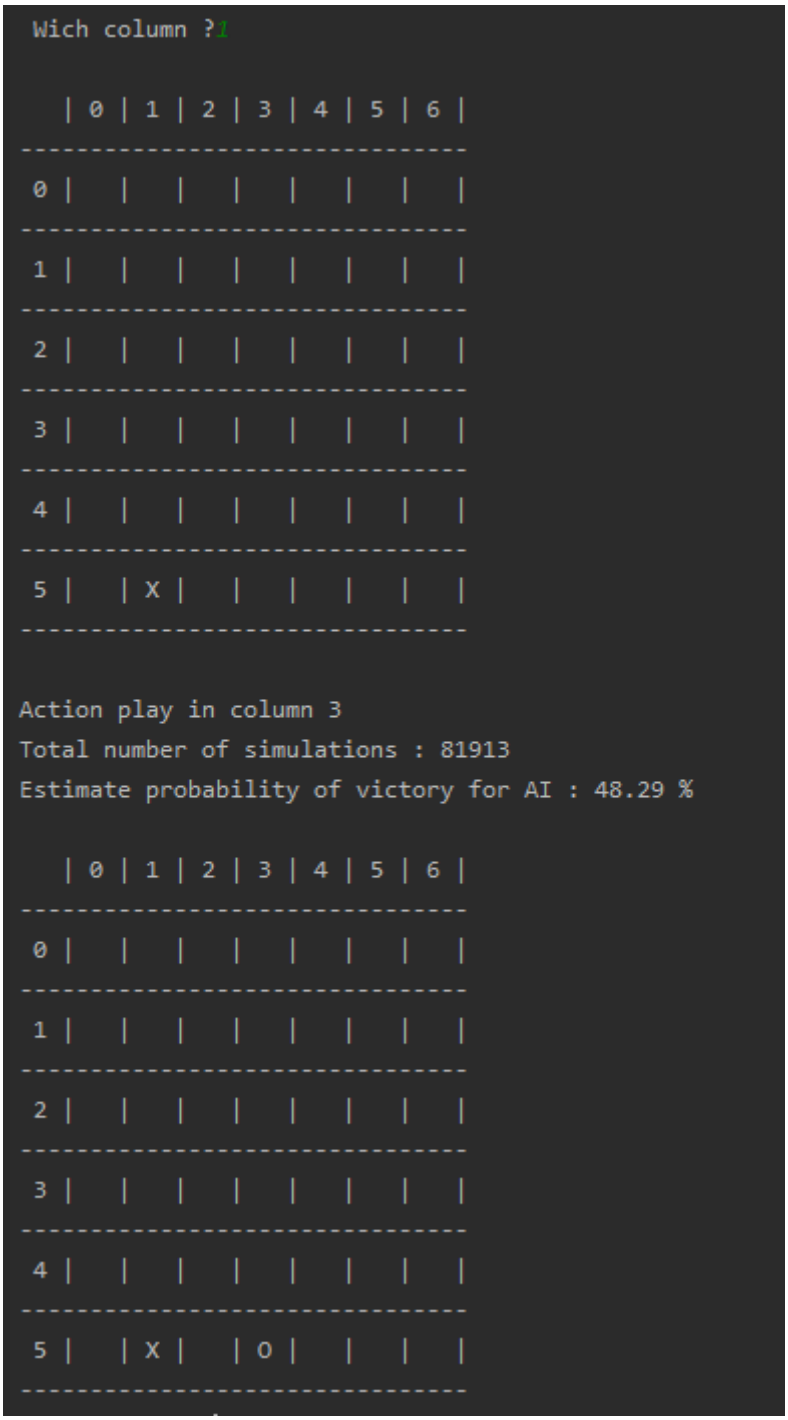
-r, (***Défaut***) option permettant de modifier la méthode de l'algorithme MCTS en utilisant la méthode ROBUSTE. Cette méthode applique l'action avec le plus grand nombre de simulations

-m, option permettant de modifier la méthode de l'algorithme MCTS en utilisant la méthode MAX. Cette méthode applique l'action avec la plus grande moyenne des récompenses.

Ainsi, par défaut, le programme utilise l'algorithme MCTS avec la méthode ROBUSTE en choisissant le meilleur coup possible lorsque le choix est possible (Q3). La limite de temps est de 5s. L'affichage est composé du nombre total de simulations réalisé, de la somme de toutes les simulations réalisées et d'une estimation de la probabilité de victoire**.**

Questions

Question 1



Question 2

Pour cette question, la réponse n'est pas évidente. En effet, il est difficile de savoir la limite (temps / itération) à partir de laquelle l'IA nous battra à tous les coups. En effet cela dépend de notre façon de jouer.

En revanche, En mode par défaut, soit en limitant le temps d'exécution de l'algorithme à 5s et en mode amélioré, il nous a été impossible de battre l'IA.

En laissant l'IA commencé, nous sommes arrivés à gagner une partie en limitant son nombre d'itération à 500. Lorsque le nombre d'itération est supérieurs à 500, il nous a été impossible de gagner.

Nous avons donc ensuite essayé en nous laissant commencé en premier.

De cette manière nous avons réussis à gagner lorsque le nombre d'itération maximum est en dessous de 5000. Au-dessus de cette valeur, il nous a été impossible de gagner.

À la suite de ces différents tests, nous avons essayé en limitant le temps. Lorsque le joueur commence, et en limitant le temps en dessous de 1s il nous a été possible de gagner contre l'IA.

Cependant en laissant l'Ia commencé, il nous a été impossible de gagner en dessous de 0.08 secondes.

Rappelons que ces valeurs peuvent être modifié en fonction du joueurs (bon ou mauvais)

Question 3

On peut tout d'abord noter que du fait de cette amélioration, l'estimation des probabilités de victoire de l'ordinateur (question 1) est biaisée car l'ordinateur choisit toujours les simulations où il peut gagner lorsque c'est possible et non plus au hasard.

En exécutant le programme avec l'amélioration (-o1) et sans (-o0) sur la limite par défaut de 5 secondes, on constate sur le premier coup que le nombre de simulations/itérations réalisées est plus élevé sans l'amélioration (sur notre test : 140 531 avec l'amélioration contre 234 583 sans).
En effet, le fait de chercher les coups gagnants durant les simulations prend plus de temps que de jouer un coup au hasard et on réalise par conséquent moins de simulations mais celles-ci sont plus précises et plus efficaces.
La qualité de jeu de la version avec l'amélioration des simulations est donc supérieur à la version sans du fait que les simulations apportent plus de précisions sur les possibilités de gagner dans un état particulier que les simulations entièrement aléatoires.

Question 4

L'option -O3 défini dans CMakeList permet d'optimiser l'exécutable généré à la compilation afin d'améliorer sa vitesse d'exécution. Il existe différents niveaux d'optimisations et -O3 (si on omet -Ofast qui passe outre des conventions pour encore plus optimiser).

Ainsi sur les différents tests effectués, en jouant toujours de la même manière et en utilisant le critère MAX, nous avons noté un supplément d'environs 100 000 simulations au total effectué par l'IA pour trouver des coups gagnants.

Les options d'optimisations agressives de cette option on pout conséquence d'effectuer plus de calcul dans le même laps de temps ce qui a pour cause d'augmenter le nombre de simulation

Sans l'option -O3 d'optimisation, le programme avec le critère MAX et bloqué à 5s d'exécution (défaut) effectue entre 4 200 000 et 4 300 000

Avec l'option -O3 d'optimisation, le programme avec le critère MAX et bloqué à 5s d'exécution (défaut) effectue entre 4 300 000 et 4 400 000

On peut donc bien remarquer l'augmentation du nombre de simulations total.

Question 5

Lors de multiples essais, nous avons remarqué que quelque soit le critère (max ou robuste) les actions jouées par l'IA restent significativement les mêmes. Cela est dû au faite que le nœud qui représente l'action qui a la moyenne des récompenses le plus élevée est souvent aussi le nœud qui a été le plus visité.

On peut en déduire qu'avec un faible nombre d'itérations, les moyennes des récompenses peuvent soit être surévaluée ou être sous-évaluée et que le nombre de simulations soient plus fiables rendant ainsi le critère robuste plus sûr dans ce genre de cas.

Finalement, après plusieurs parties jouées de la même manière et dans les paramètres par défaut, nous avons remarqué que le nombre total de simulation avec le critère ROBUSTE est de l'ordre de

4 100 000 – 4 300 000 alors qu'en mode MAX, le nombre total de simulation est de l'ordre de 4 200 000 – 4 300 000. (Voir annexe). On remarque donc que le critère MAX effectue moins de simulations.

Question 6

Le Puissance 4 (classique) impliquant un facteur de branchement de 5 en moyenne car le nombre de nœuds diminue plus on se rapproche des feuilles et une profondeur maximale de 6*7 = 42, on aura donc un arbre complet composé de somme(n; 0; 42; 5^n)= 2.8421709430405E+29 nœuds ce qui est vraiment conséquent.

En hypothétisant que 1 nœud est calculé en 1ns

On obtient 2.8421709430405E+29 ns soit 9.012464938611 x 10^+12 années.

En comparaison l'âge de l'univers est de 13,75 milliards d'années

Ainsi un algorithme Min-Max exhaustif (sans limitation de profondeur) n'est pas envisageable dans le cas du Puissance 4.

Annexe

Nombre de simulation entres les critère MAX et ROBUSTE

MAX

```
Total number of simulations : 4317516
```

```
Total number of simulations : 4293747
```

```
Total number of simulations : 4290677
```

```
Total number of simulations : 4210137
```

ROBUSTE

```
Total number of simulations : 4318058
```

```
Total number of simulations : 4101930
```

```
Total number of simulations : 4230915
```

```
Total number of simulations : 4371123
```

Note : Le terme « Total number of simulation » a été remplacé par « Sum of simulation » dans l'affichage du programme pour plus de clarté

Nombre de simulation entres les critère MAX

```
Sum of simulations : 4420739
```

```
Sum of simulations : 4360600
```

```
Sum of simulations : 4291391
```

```
Sum of simulations : 4314928
```