# Rapport Projet IA

Puissance 4

Maéla LACONI
Nada MADAD

# Fonctionnement du programme :

## Compiler le projet : gcc jeu.c -o jeu -lm

## Exécuter le projet : ./jeu

## Fonctionnement du code pour MCTS :

L'algorithme se découpe en 4 étapes principales.

1. **Sélectionner**

On part au tout début à la racine de l'arbre. On parcourt les fils de la racine tant qu'on n'a pas de nœud terminal ou un fils non développé, pour trouver le nœud avec la plus grande B-valeur.

2. **Développer**

Si on est dans le cas où on a un nœud qui est non développé alors on le développe. Puis on prend un nœud au hasard dans la liste des enfants non explorés.

3. **Simuler**

On utilise une marche aléatoire pour simuler la fin de partie

On ajoute à ce niveau là l'amélioration qui consiste à toujours choisir un coup gagnant quand c'est possible.

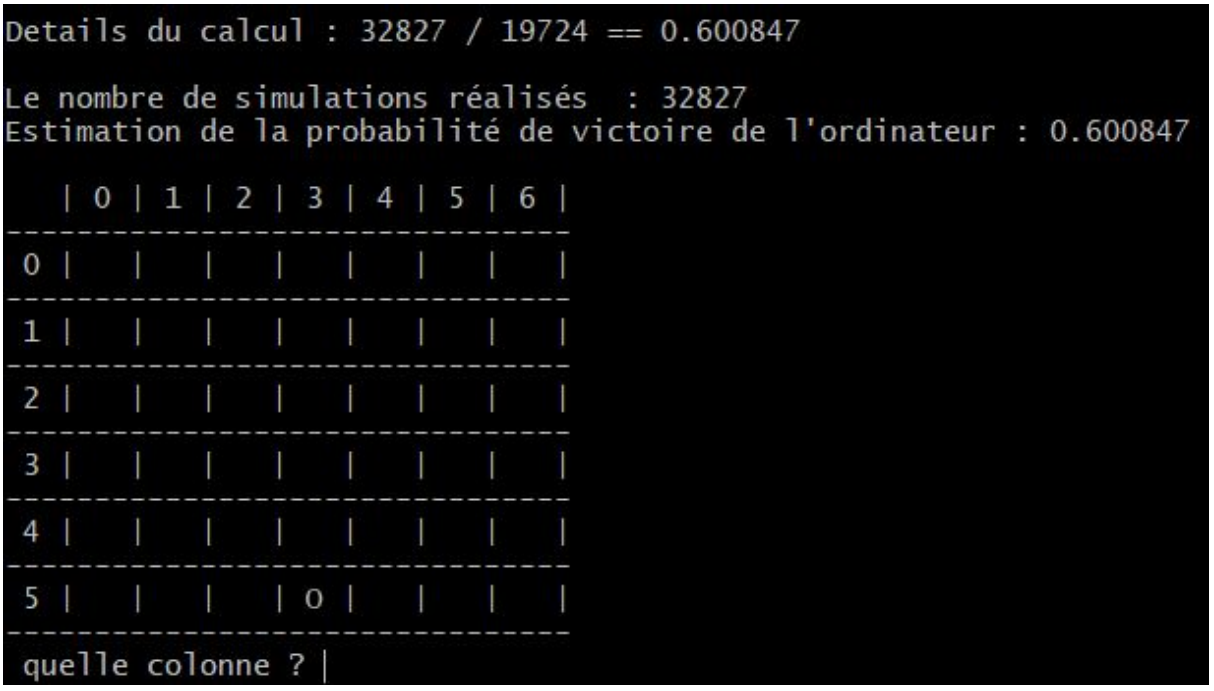
4. **Mettre à jour**

On met à jour tous les nœuds en remontant à la racine en augmentant de 1 leur nombre de simulations. Si l'état mène à une victoire pour l'ordinateur ou un match nul, on augmente également de 1 le nombre de victoires des nœuds.

A la fin de l'algorithme, suivant la stratégie choisie, si on a pris "max" alors on maximise le ratio du nombre de victoires divisé par le nombre de simulation.
Si la stratégie choisie est "robuste", alors on maximise le nombre de stimulations.

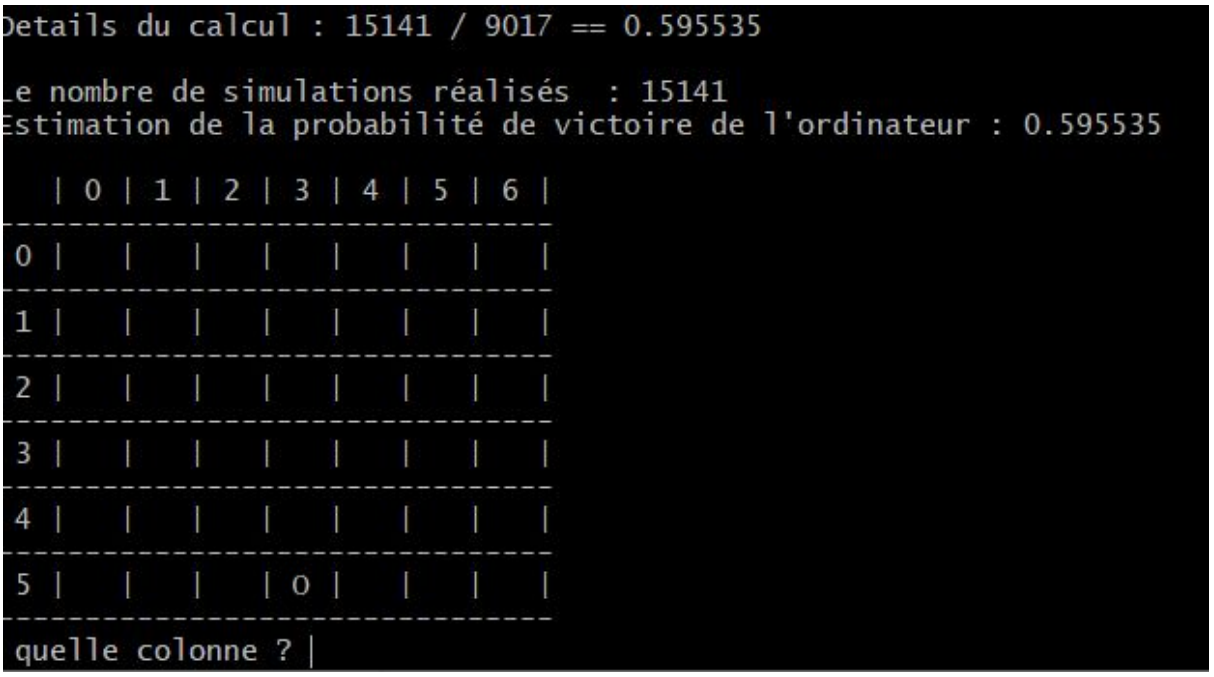
# Réponse aux questions :

1- Le nombre de simulations réalisées pour calculer ce coup et une estimation de la probabilité de victoire pour l'ordinateur:

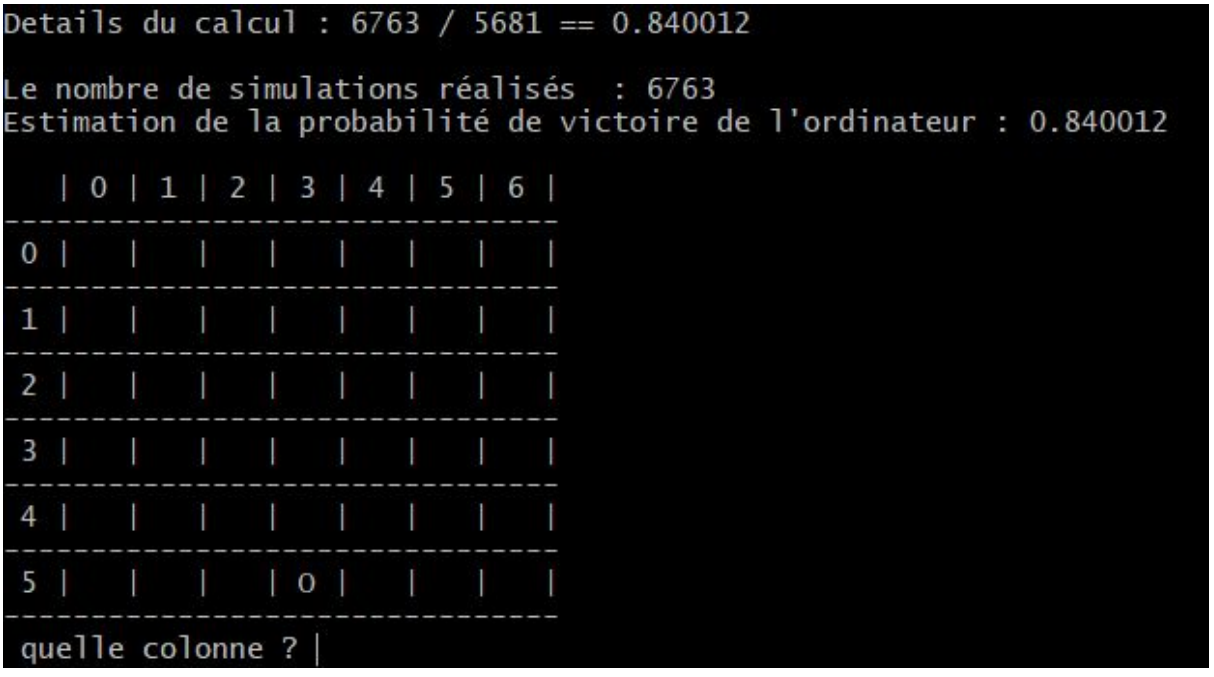



2- L'ordinateur nous bat à tous les coups donc on peut dire qu'à partir de la première seconde, l'ordinateur nous bat déjà.

3- La qualité de jeu de cette nouvelle version avec la précédente est meilleure. On diminue considérablement le nombre de simulation, au moins de moitié.




Ici, lorsque nous jouons sans l'amélioration, on voit que pour ce coup-ci, il y a 15 141 simulations de réalisées.




Ici, on constate pour le même coup joué avec l'amélioration, que l'on fait 6 763 simulations.

On voit donc bien la diminution du nombre de simulations avec la version améliorée.

On peut voir aussi que la probabilité pour l'ordinateur de gagner est passée de 59% à 84% (pour ce coup-ci).

On réduit le parcours en profondeur de l'arbre en coupant certaines branches. Dès qu'un coup gagnant est possible pour l'ordinateur, celui-ci le prend.

4- L'option -O3 est l'une des options qui contrôlent diverses sortes d'optimisations.
Sans aucune option d'optimisation, le but du compilateur est de réduire le coût de la compilation et de faire en sorte que le débogage produise les résultats attendus.
Or l'activation des indicateurs d'optimisation oblige le compilateur à tenter d'améliorer les performances et / ou la taille du code au détriment du temps de compilation et éventuellement de la capacité de déboguer le programme.

Et donc voici les flags que rajoute l'option -O3

```
-falign-functions  -falign-jumps
-falign-labels  -falign-loops
-fcaller-saves
-fcode-hoisting
-fcrossjumping
-fcse-follow-jumps  -fcse-skip-blocks
-fdelete-null-pointer-checks
-fdevirtualize  -fdevirtualize-speculatively
-fexpensive-optimizations
-ffinite-loops
-fgcse  -fgcse-lm
-fhoist-adjacent-loads
-finline-functions
-finline-small-functions
-findirect-inlining
-fipa-bit-cp  -fipa-cp  -fipa-icf
-fipa-ra  -fipa-sra  -fipa-vrp
-fisolate-erroneous-paths-dereference
-flra-remat
-foptimize-sibling-calls
-foptimize-strlen
-fpartial-inlining
-fpeephole2
-freorder-blocks-algorithm=stc
-freorder-blocks-and-partition  -freorder-functions
-frerun-cse-after-loop
-fschedule-insns  -fschedule-insns2
-fsched-interblock  -fsched-spec
-fstore-merging
-fstrict-aliasing
-fthread-jumps
-ftree-builtin-call-dce
-ftree-pre
-ftree-switch-conversion  -ftree-tail-merge
-ftree-vrp
```

```
-fgcse-after-reload
-fipa-cp-clone
-floop-interchange
-floop-unroll-and-jam
-fpeel-loops
-fpredictive-commoning
-fsplit-loops
-fsplit-paths
-ftree-loop-distribution
-ftree-loop-vectorize
-ftree-partial-pre
-ftree-slp-vectorize
-funswitch-loops
-fvect-cost-model
-fvect-cost-model=dynamic
-fversion-loops-for-strides
```

Donc compiler avec  gcc -O3 permet d'utiliser toutes les optimisations d'exécution et de compilation acceptées par gcc. On remarque que le nombre de nœuds simulés est largement plus élevé quand on l'exécute avec l'option.

Avec l'optimisation :

Avec l'option -O3:

```
Details du calcul : 42149 / 27036 == 0.641439

Le nombre de simulations réalisés  : 42149
Estimation de la probabilité de victoire de l'ordinateur : 0.641439
```

Sans l'option -O3 :

```
Details du calcul : 11579 / 7074 == 0.610934

Le nombre de simulations réalisés  : 11579
Estimation de la probabilité de victoire de l'ordinateur : 0.610934
```

Sans optimisation :
Avec -O3 :

```
Details du calcul : 74789 / 29698 == 0.397090

Le nombre de simulations réalisés  : 74789
Estimation de la probabilité de victoire de l'ordinateur : 0.397090
```

Sans -O3 :

```
Details du calcul : 38036 / 14892 == 0.391524

Le nombre de simulations réalisés  : 38036
Estimation de la probabilité de victoire de l'ordinateur : 0.391524
```

5- Comparaison des critères "max" et "robuste"

- Le critère "max"  utilise le nombre de victoires/nombre de simulations.
- Le critère  "robuste" utilise uniquement les simulations.

Si on utilise le critère "max", on constate que l'exploration est plus efficace pour le programme. Cependant, si on utilise le critère "robuste", l'exploration perd son efficacité dans le programme. On remarque aussi que quand on prend le critère "max" l'ordinateur à beaucoup plus de chance pour gagner comparé avec le critère robuste.

6- En implémentant uniquement l'algorithme Minimax, sans optimisation, le temps avant de jouer le premier coup sera très grand.

Une estimation du temps de calcul est $7^{42}$ car pour chaqu'une des 42 (6*7)  cases du tableau, on a 7 coups possibles. Pour détailler cela encore plus, on considère par exemple

un processeur 3GHz donc on aura $3 * 10^9$ opérations par second , équivaut à $31 * 10^6$ opérations en 1 an (365 (jours)*24 (heures)*60 (minutes)*60 secondes).

On a donc une complexité de $7^{42} = 3.11 * 10^{35}$ et $9.3 * 10^{16}$ opérations par année. Si on fait un produit en croix, on en conclut donc qu'il nous faudrait donc $3.34 * 10^{19}$ années.