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.

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 1 seconde l’ordinateur me bat à tout les coups, nous avons essayé de faire jouer l’IA contre elle-même et mettre 1 seconde de réflexion ou 2, conduit au même choix du coup optimal selon elle.

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

Cela est dû au fait qu’on réduit la profondeur de l’arbre de 1 au minimum.

Et donc on réduit la probabilité de 1/7 de choisir un état gagnant par simulation à 1 sur un nœud qui à 7 fils. Nous voyons donc bien qu’à partir d’une profondeur de 6 (là ou il y a la première possibilité d’avoir un nœud gagnant). On peut potentiellement faire les pires choix possibles et atteindre une profondeur de 6\*6 (étant la profondeur restante pour arriver à un match nul à partir de la profondeur 6)

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.

-O3

Optimize yet more. -O3 turns on all optimizations specified by -O2 and also turns on the following optimization flags:

-fgcse-after-reload

-fipa-cp-clone

-floop-interchange

-floop-unroll-and-jam

-fpeel-loops

-fpredictive-commoning

-fsplit-paths

-ftree-loop-distribute-patterns

-ftree-loop-distribution

-ftree-loop-vectorize

-ftree-partial-pre

-ftree-slp-vectorize

-funswitch-loops

-fvect-cost-model

-fversion-loops-for-strides

-O2 turns on all optimization flags specified by -O. It also turns on the following optimization flags:

-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

With -O, the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time.

-O turns on the following optimization flags:

-fauto-inc-dec

-fbranch-count-reg

-fcombine-stack-adjustments

-fcompare-elim

-fcprop-registers

-fdce

-fdefer-pop

-fdelayed-branch

-fdse

-fforward-propagate

-fguess-branch-probability

-fif-conversion

-fif-conversion2

-finline-functions-called-once

-fipa-profile

-fipa-pure-const

-fipa-reference

-fipa-reference-addressable

-fmerge-constants

-fmove-loop-invariants

-fomit-frame-pointer

-freorder-blocks

-fshrink-wrap

-fshrink-wrap-separate

-fsplit-wide-types

-fssa-backprop

-fssa-phiopt

-ftree-bit-ccp

-ftree-ccp

-ftree-ch

-ftree-coalesce-vars

-ftree-copy-prop

-ftree-dce

-ftree-dominator-opts

-ftree-dse

-ftree-forwprop

-ftree-fre

-ftree-phiprop

-ftree-pta

-ftree-scev-cprop

-ftree-sink

-ftree-slsr

-ftree-sra

-ftree-ter

-funit-at-a-time

CA DOUBLE LE NOMBRE DE NOEUDS GENERER POUR 5S A TESTER POUR PLUS DE TEMPS MAIS JE PENSE PAS QUE CA AUGMENTE DE FAÇON EXPONENTIELLE VOYONS P != NP

Toutes ces options double le nombre de nœuds visité.

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 ?

=> robuste : maximise le nombre de simulations

=> max : maximise le ratio victoire/nb\_simu

=> à tester mais le critère robuste semble nul a chier

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).