Stratégies d'explorations d'états pour la programmation d'un jeu de réflexion: le puissance 4

Corvisier Jean-Christophe
Clivio Oscar
Projet encadré par M. Renaud Marlet

Objectif:

Implémenter le jeu de puissance 4 ainsi qu'une intelligence artificielle de bonne qualité à difficulté réglable.

Description du jeu :

- Grille de 6 lignes pour 7 colonnes ;
- Chaque joueur insère un jeton à tour de rôle ;
- Un joueur gagne s'il aligne 4 pions en ligne, colonne ou diagonale.

Langages de programmation utilisés:

-Python (module graphique pygame)
-C++ (module graphique imagine ++)

Plan :

-Première approche: l'algorithme Min-Max

-Amélioration du Min-Max: l' algorithme Alpha-Beta

-Développement d'une heuristique d'évaluation de position, résulats.

-Amélioration de l'heuristique par un algorithme génétique

-Conclusion et perspectives

Première approche: Algorithme du Min-Max

Algorithme utilisé dans le cadre d'un jeu à 2 joueurs à information complète et à somme nulle.

Hypothèses:

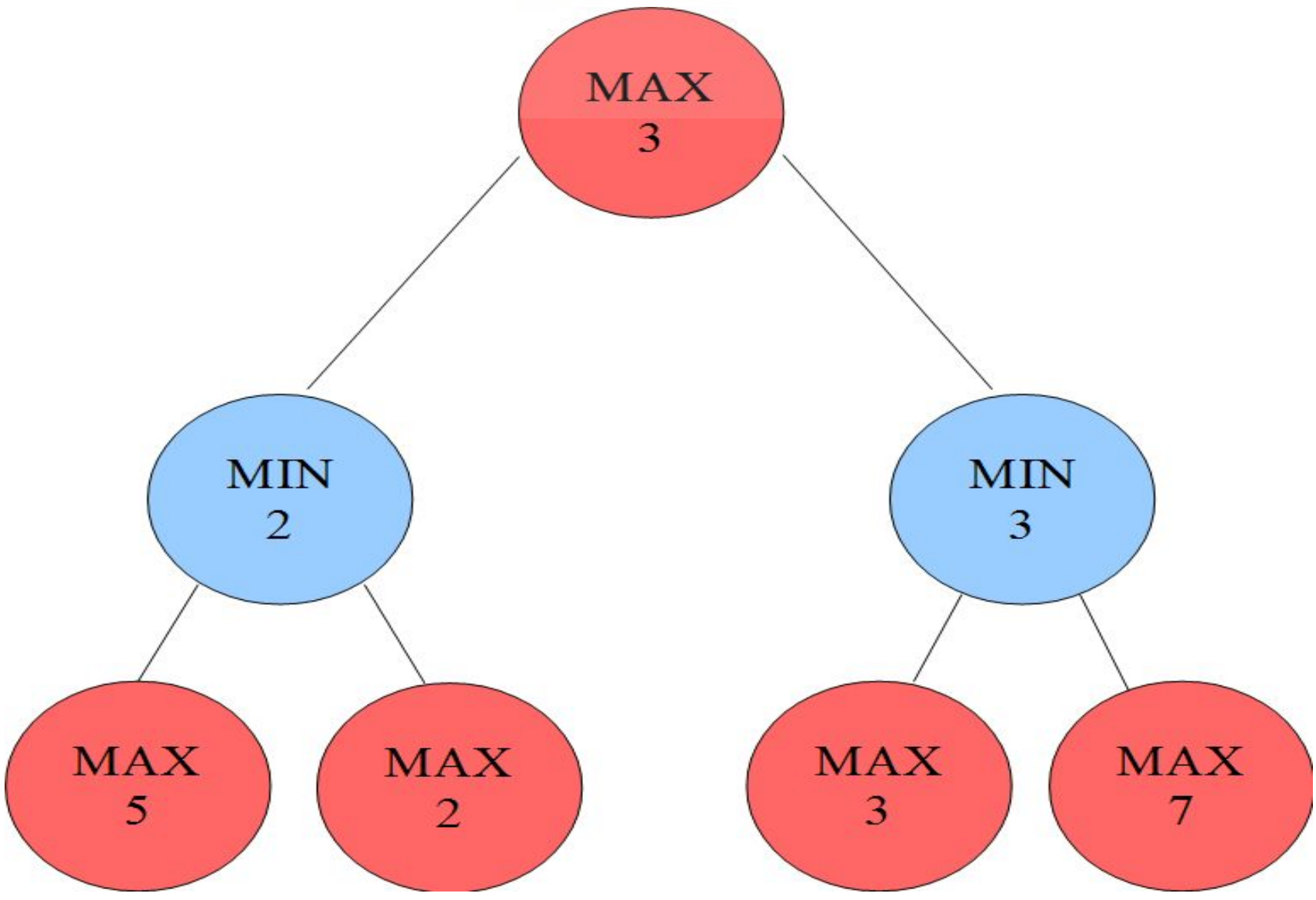
- Chaque position (état) du jeu donne une valeur (évaluation) ;
- Le joueur 1 veut minimiser cette valeur, le joueur 2 la maximiser.

Principe de l'algorithme

Parcours de tout l'arbre des coups possibles par simulation pour trouver le ou les meilleurs coups.

En pratique l'arbre des coups trop grand à explorer ↠ nécessite de fixer une profondeur de réflexion limite.

Schéma de fonctionnement



Amélioration du Min-Max: Algorithme Alpha Beta

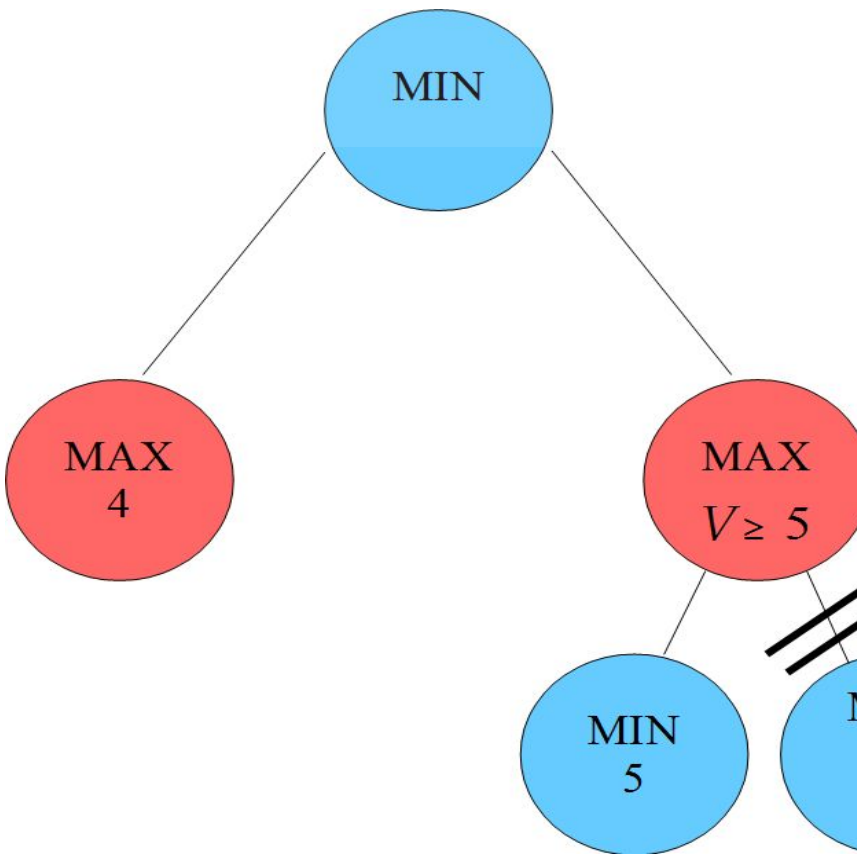
Il va permettre d'améliorer le Min-Max en coupant dans l'arbre de recherche pour ne pas explorer des coups inutiles.

Intérêt: renvoie le même résultat que Mini-Max mais beaucoup plus rapidement ↠ permet d'augmenter la profondeur de calcul pour améliorer la qualité de l ' IA.

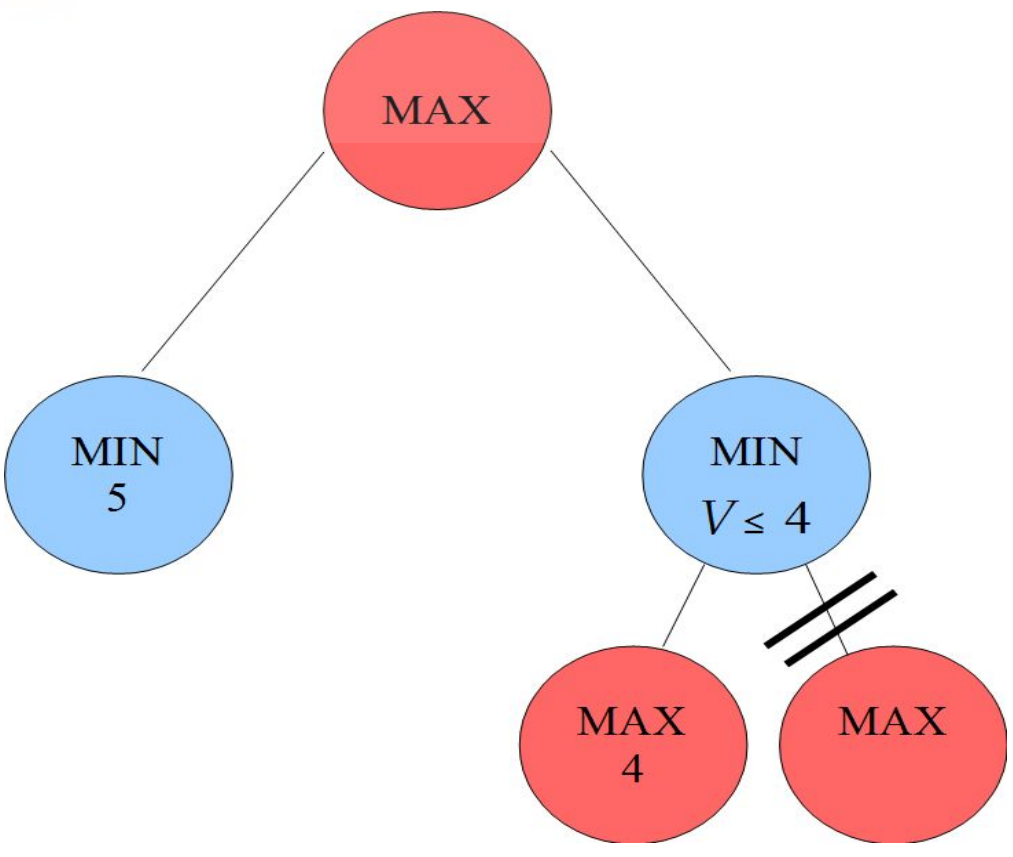
Principe: conservation de bornes (α,β) mis à jour dans les appels récursifs pour "couper" des branches de l'arbre inutile.

Schéma de fonctionnement de l'Alpha-Beta

Coupure "Alpha" (ici α = 4)



Coupure "Beta" (ici β = 5)



Heuristique d'évaluation d'une position

Premiers algorithmes Min-Max et Alpha-Beta:

- Fonction d'évaluation simple: -1 si victoire du joueur 1, 1 si victoire du joueur 2, 0 si partie nulle;

- Résultats: pour une profondeur de recherche raisonnable (4), IA convenable mais avec quelques défauts comme jouer systématiquement sur la dernière colonne en début de partie. En effet, beaucoup de positions ont la même évaluation (0) en début de partie, lors de la recherche avec Min-Max.

<u>Solution :</u> construction d'une heuristique d'évaluation H d'une position pour chaque joueur

Heuristique: fonction qui recherche dans la grille de jeu tous les alignements de 4 cases possibles et calcule pour chaque joueur i le nombre $N_{1,i}$ (resp $N_{2,i}$, $N_{3,i}$) d'alignements contenant exactement un (resp. deux, trois) pion(s) du joueur.

Le score renvoyé pour le joueur i est alors : (en notant P une position)

$$\mathbf{H(p,i) = a\,N_{1,i} + b\,N_{2,i} + c\,N_{3,i}}$$

avec a b c des réels positifs.

En cas de victoire du joueur 1, la fonction H renvoie pour lui un très faible score pour lui et un très grand score pour le joueur 2 , et inversement si le joueur 1 perd.

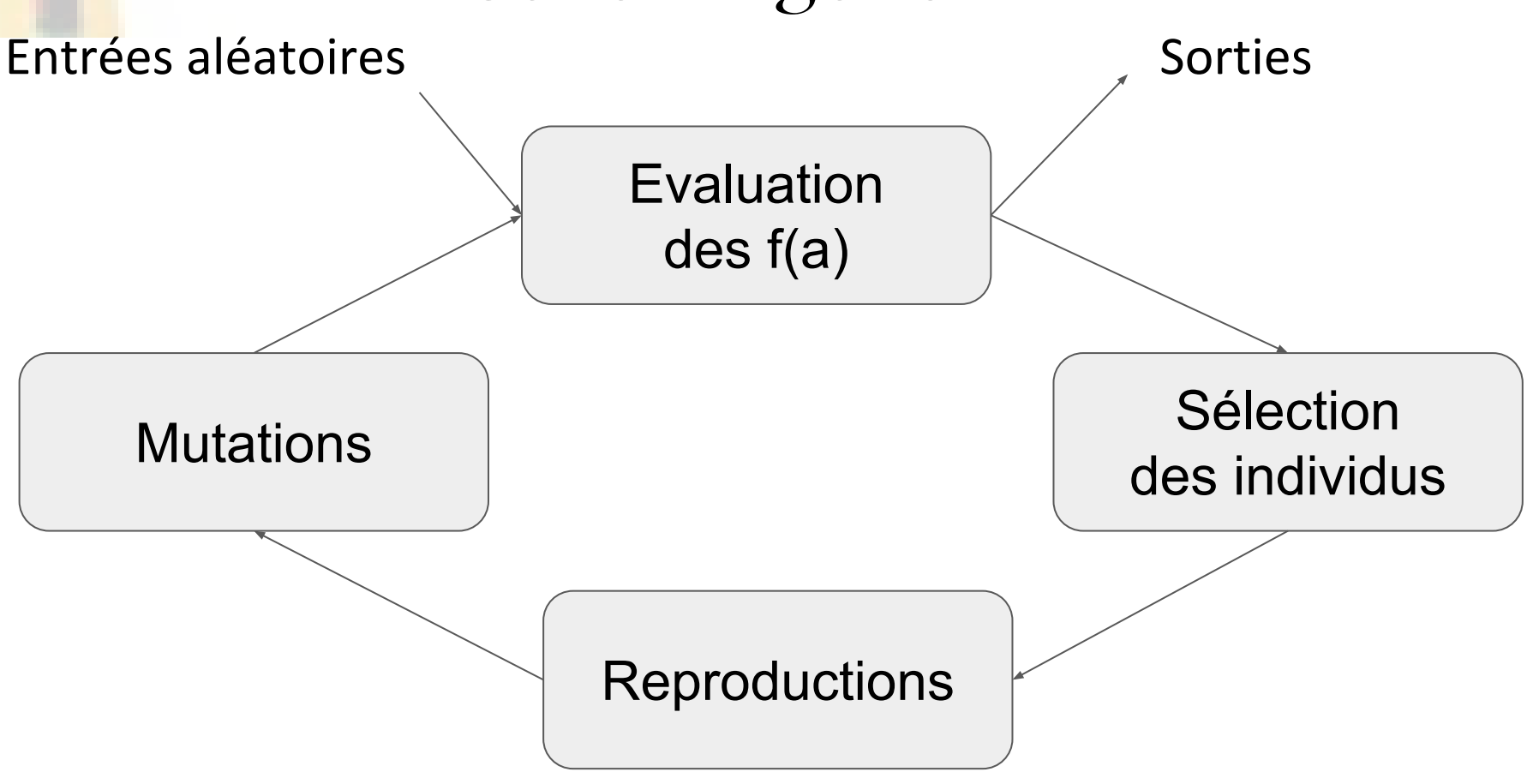
La fonction d'évaluation devient : $\boldsymbol{f(p) = H(p,2) - H(p,1)}$

Test et mesure du temps : en jouant contre 100 ordinateurs jouant au hasard et avec une profondeur de 4, les deux algorithmes renvoient les même résultats et Min-Max dure en moyenne 0.57s, Alpha-Beta 0.20s.

Utilisation d'un algorithme génétique (1/3)

Principe: on code le paramètre (a,b,c) sous la forme d'un <u>génome</u>, c'est-à-dire une séquences de gènes $(g_1, \ldots, g_M)$. On suppose qu'il existe **f : A ↠ R+**, fonction de survie ou *fitness*, à maximiser.

Schéma général



Favoriser les plus aptes tout en maintenant une certaine diversité génétique. Eviter maxima locaux mais non globaux

Hypothèses:

- Population de taille identique à l'évaluation.
- Formation de couples qui se reproduisent avec une probabilité P_r.
- A la mutation, chaque gène mute avec une probabilité P_m.
- Probabilité d'être sélectionné: proportionnelle à f(a).
- Nombre fini de cycles.

<u>Résultat:</u> tout algorithme dont le codage est <u>binaire</u> (g_i dans {0,1}) converge vers un équilibre, qui est l'optimum si et seulement si l'individu le plus apte après une évaluation est conservé à l'identique pendant le processus sélection/reproduction/mutation.

Utilisation d'un algorithme génétique (2/3)

Codage retenu: a, b et c entiers positifs codés sur 10 bits (donc entre 0 et 1023). Génome de 30 gènes binaires.

Evaluation retenue: faire jouer chaque génome contre tous les autres, comme joueur 1 et comme joueur 2. Il détermine ses coups selon ses seuls (a,b,c).

Evaluation d'une partie: 1+g si victoire, 1-g si défaite, 1 sinon, avec $g = 1 - \frac{E-1}{20}$ où E est le nombre d'étapes (joueur 1 puis joueur 2 jouent) nécessaires pour terminer les parties.

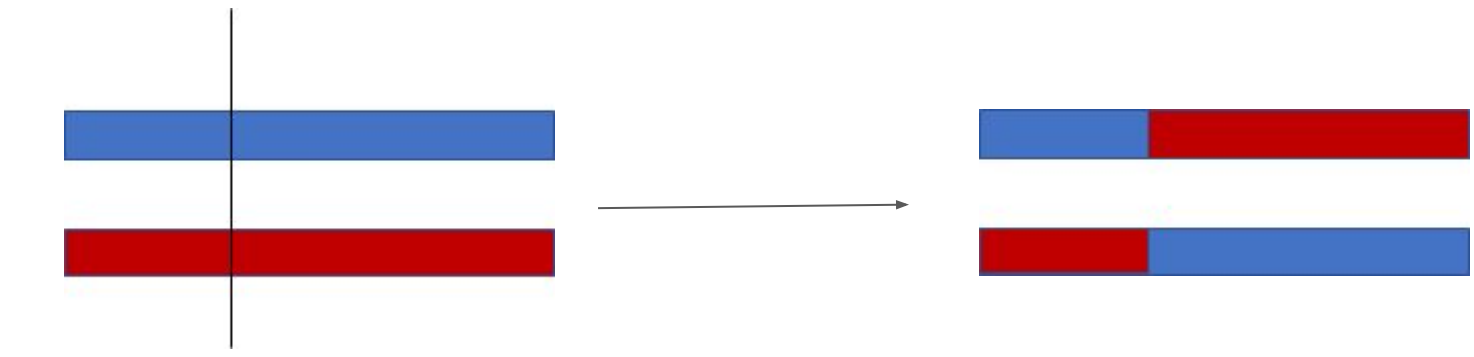
f(a): moyenne des évaluations des parties jouées par a.

Sélection retenue: associer à chaque génome une probabilité, $p(a) = \frac{f(a)}{\sum_{a'} f(a')}$ répéter N sélections d'un individu selon la loi de p.

Sélection par roulette.

Reproduction retenue: dans le nouveau vecteur des génomes, faire des paires des éléments successifs, les faire se reproduire avec $P_r = 0.5$, sauf le couple du génome le plus apte ($P_r = 0$).

Principe de la reproduction:



Indice de coupure tiré au hasard

Mutation retenue: décrite précédemment, $P_m = 0$ pour le génome le plus apte, 0.05 pour les autres.

Utilisation d'un algorithme génétique (3/3)

Trouver mieux que (1,10,100) ?

Profondeur de 3.
Algorithme génétique avec 16 génomes et 100 itérations.
Prendre le meilleur des génomes finaux.
Observations : "début de convergence",
quelques valeurs qui se ressemblent.

↓ 5 fois

Candidats :
(36,197,732), (25,158,542), (15,77,603)
(17,88,444), (15,244,953)

↓

Test : candidat accepté s'il bat (1,10,100)
et s'il bat plus de génomes aléatoires quelconques
et s'il bat plus de génomes aléatoires "humains" (a<b<c)
que (1,10,100) pour 250 tirages dans chaque catégorie

↓

Seul (15,244,953) est validé.
Il est donc "mieux" que (1,10,100).

Conclusion et perspectives

Conclusion :

- L'algorithme Min-Max permet de construire une intelligence artificielle assez efficace. Doser la difficulté avec la profondeur.
- Élagage Alpha-Beta : environ 2.5 fois plus rapide que Mini-Max.
- Algorithme génétique: réelle amélioration de l'IA. Cependant, méthode fastidieuse : compromis entre temps de calcul et pertinence du résultat. Faible robustesse.

Perspectives d'amélioration :

- Changement fonction de *fitness* ? Tests : comparaison surtout avec génomes aléatoires.
- Simulation plus longue pour avoir une meilleure convergence ?
- Arbres de recherche de type Monte-Carlo (MCTS) ?
- Jeu en fait déjà résolu…

Bibliographie :

- *Algorithmes (ou comment pensent les ordinateurs)*, Fédération Française d'Othello : http://www.ffothello.org/informatique/algorithmes/
- *Algorithmes évolutionnistes (et autres heuristiques d'inspirations biologiques),* Fabien Moutarde, Centre de Robotique (CAOR), Mines ParisTech (ENSMP) : http://perso.mines-paristech.fr/fabien.moutarde/ES_Machine Learning/Slides/slides_algosEvolutionnistes.pdf http://perso.mines-paristech.fr/fabien.moutarde/ES_Machine Learning/TP-genetique/TP-GeneticAlgo.html
- *Algorithmes génétiques*, coursgratuits.net : http://informatique.coursgratuits.net/methodes-numeriques/algorithmes-genetiques.php