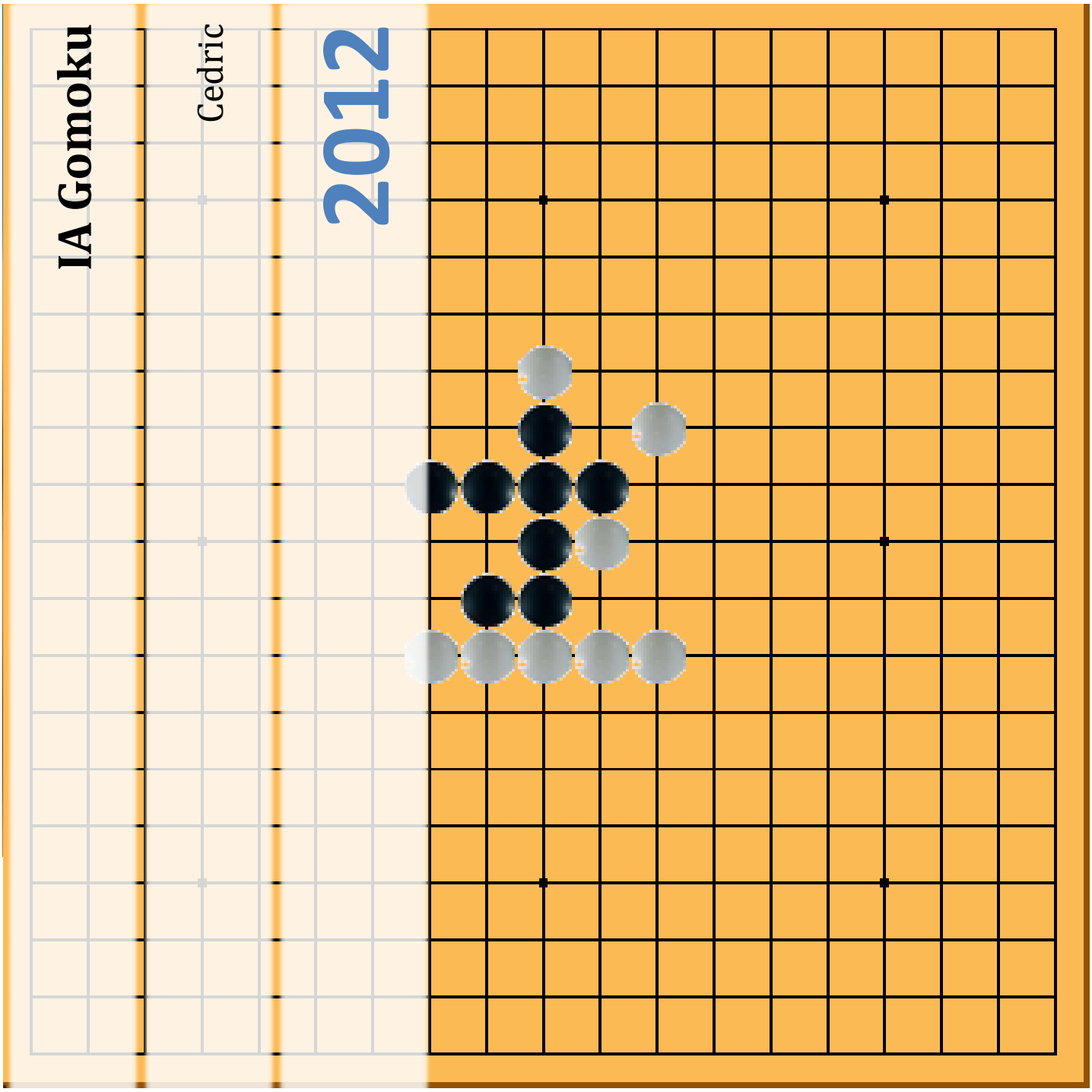


Rapport gomoku ninuki

Epitech 2012
Projet Gomoku

Table des matières

1. Représentation des connaissances

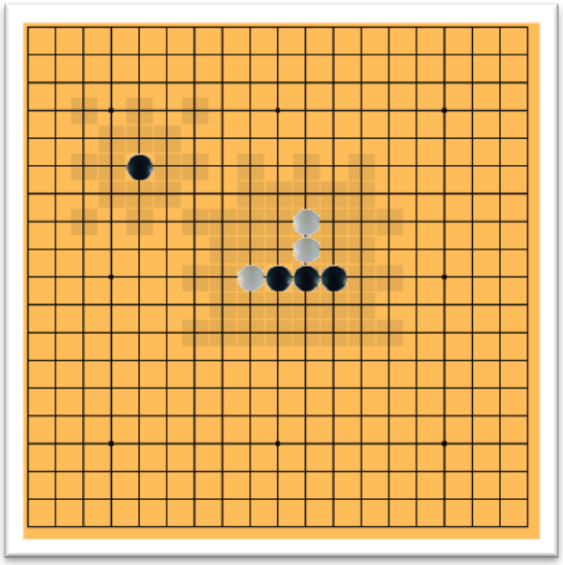
La grille est un tableau de 361 cases. Une case est un int64 et contient les informations suivantes :

- La couleur de la case : Vide, Noir, Blanche
- Un tableau de huit index codés sur 5 bits, les index sont des patterns.
  On peut retrouver diverses infos relatives à un pattern grâce à un autre tableau de struct.
- L'influence de la case blanche et noir

Par exemple pour un pattern « o_oo » on sait qu'il contient 3 cases de la même couleur, avec au moins un trou au milieu.

2. Sélection des coups

Toute case vide située à au moins deux case d'une case pleine est susceptible d'être jouée.

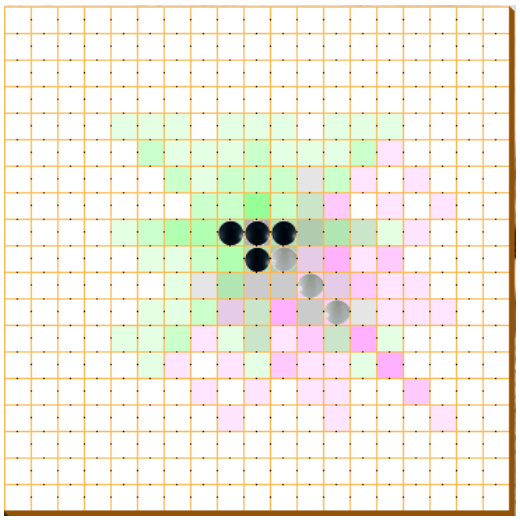


a. Trie des coups

Les performances d'un alphabeta sont intrinsèquement liées à l'élagage de l'arbre. Afin de d'améliorer cet élagage on peut trier les coups joués afin de jouer les meilleurs en premier.
Il faut cependant une méthode rapide pour trier les coups.
Nous introduisons l'influence Mapping.

b. Influence Mapping

L'influence Mapping est une technique utilisée dans certains jeux (RTS, FPS ect …). Elle va permette de savoir rapidement si une case est plus influencé par les noirs ou les blancs.

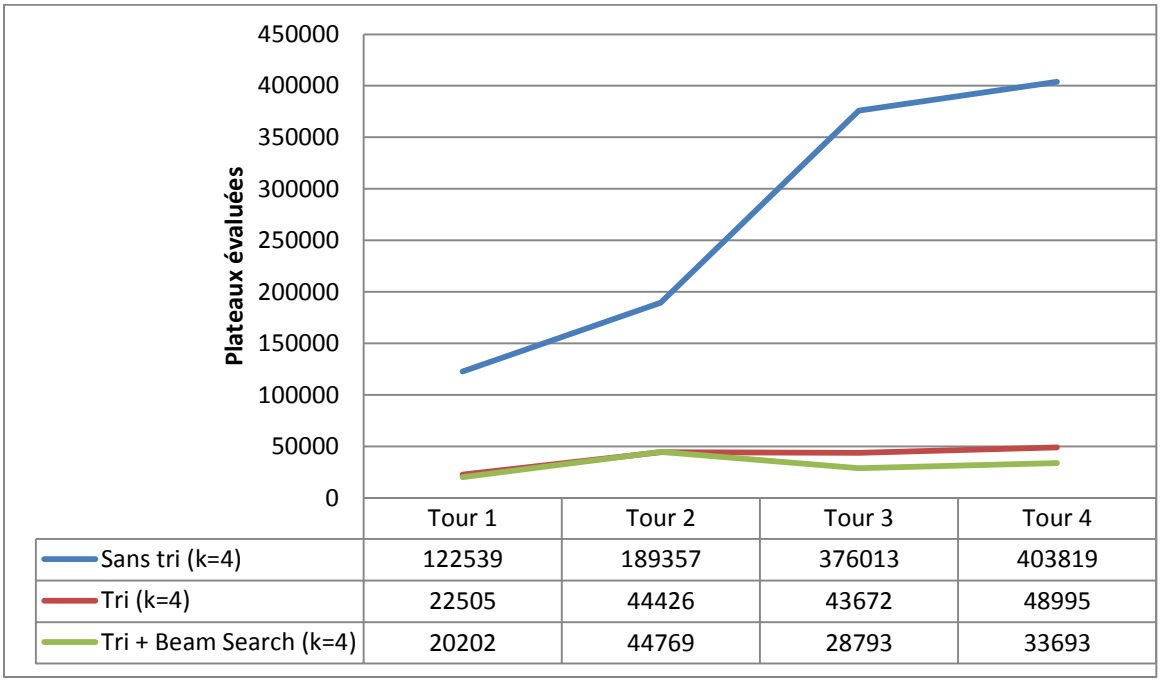


Le problème de cette technique dans notre cas est qu'elle n'est pas assez précise. Elle est utile pour faire de la **conquête.** Le surplus de mise à jour à faire et la maigre utilité que nous lui avons trouvée

Cependant elle est une bonne alliée quand il s'agit de trié les coups jouer. Les coups ayant la plus haute influence sont plus susceptibles d'être joué.

c. Beam Search

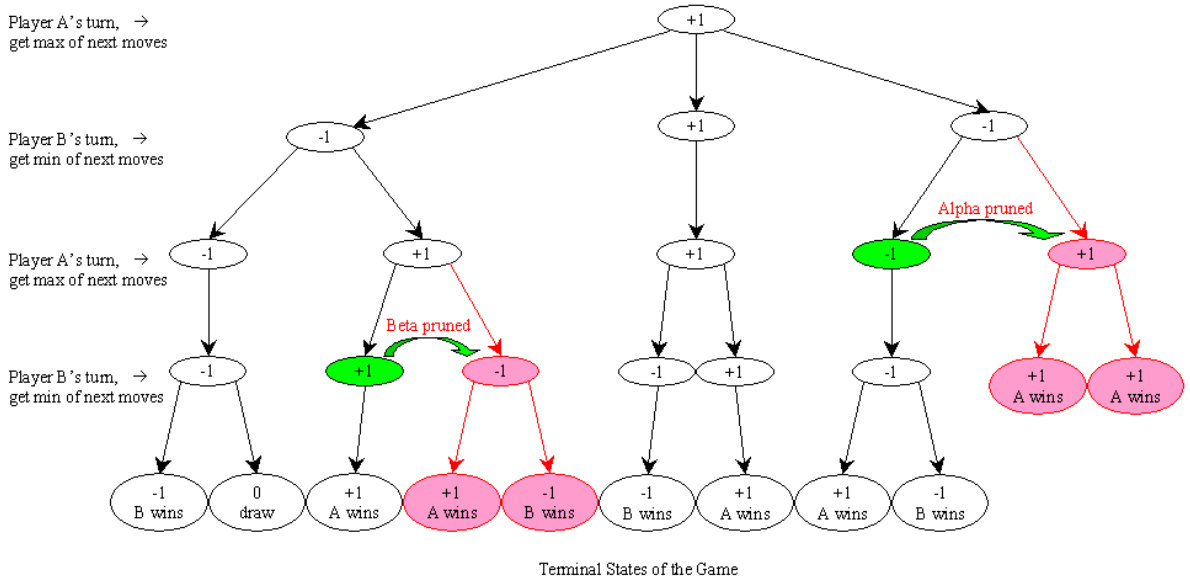
Maintenant que nous évaluons les nœuds qui sont le plus susceptibles d'être prometteur en premiers, nous pouvons nous dispenser des fils en fin de liste afin de « caper » la complexité de l'AlphaBeta $O(b^k)$ à $O(40^k)$



	Tour 1	Tour 2	Tour 3	Tour 4
Sans tri (k=4)	122539	189357	376013	403819
Tri (k=4)	22505	44426	43672	48995
Tri + Beam Search (k=4)	20202	44769	28793	33693

3. AlphaBeta

Une fois les coups possibles générés, on peut créer les fils pour l'alpha beta.
Petit rappel sur l'alpha beta :



The state in green is where one can determine that the states in pink can be pruned.
Note that the values of the children of the green states must be calculated before the value of the green state itself can be determined..

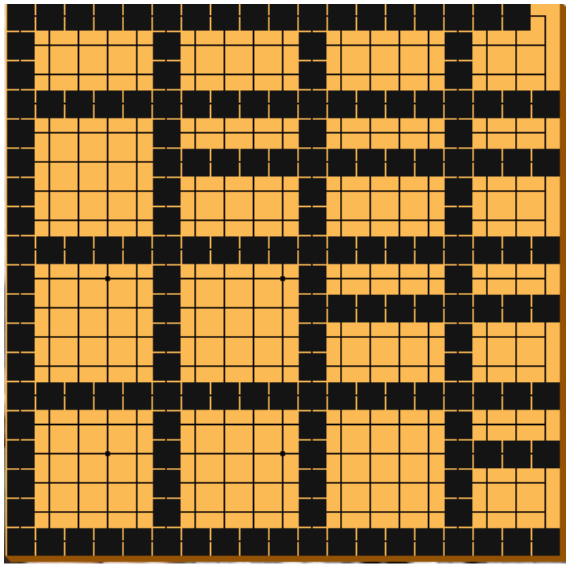
L'alphabeta est donc une variante du MinMax permettant de ne pas évaluer les nodes inutiles.
Nous ajoutons à cela un calcul multithread afin d'accélérer la génération et l'évaluation de l'arbre.

4. Heuristique

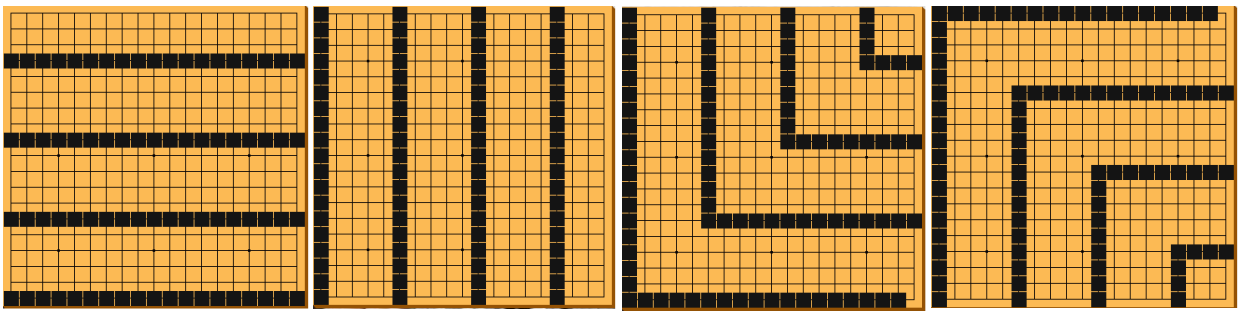
L'heuristique est la fonction permettant d'évaluer le plateau. Le gomoku étant un zero-sum game on a : h(noir) = -h(blanc).

a. Cases évaluée.

Dans un premier temps, nous définissons une méthode pour parcourir la grille et n'évaluer que les cases utiles.



En noir les case évaluées, nous parcourons la grille en hauteur, largeur et sur deux diagonal :



b. Valeur d'un alignement

Nous devons définir une valeur pour un alignement donné. Nous définissons deux variable A et B tel que :

A = nombre de case alignées sans case vide au milieu
B = Nombre case pleine – A

Nous devons trouver une fonction F tel que : F(a, b) < F(a, b+1) < F(a+1, b) < F(a + 1, B + 1)
Nous définissons F(a, b) = 5^(a+b) + 5^a

Ensuite un alignement peut être étendu d'un côté ou de l'autre s'il ne peut être étendu que d'un côté nous lui impliquons un malus car il suffit d'un seul coup pour le bloquer. Ainsi un trois non libre à la même valeur qu'un deux libre.

c. Score d'une map

1. Addition/soustration

La première solution envisagée est d'additionner au score de la map si l'alignement est de la bonne couleur et de le soustraire s'il est de la mauvaise.

5. Pistes d'amélioration

a. Hash des maps

Au gomoku plusieurs séries de coups peuvent mener à la même map. Si A B C produise un état P alors il est probable que C B A produise le même état avec le même score.

On peut donc garder en cache le score d'une map afin de le réutiliser plus tard, évitant l'évaluation. Cependant nous n'avons pas pu déterminer une fonction de hash efficace.