FILIERE MP : ENS (PARIS) – ENS LYON – ENS CACHAN

PAGE DE GARDE DU RAPPORT DE TIPE 2013

NOM : MARTY Prénoms : Olivier Michel

Lycée : Clemenceau

Classe : MP*

Ville : Nantes

Concours auxquels vous êtes admissible dans la banque inter-ENS :

(Mettre une croix très visible dans la ou les case(s) vous concernant)

ENS Cachan	MP - option MP		MP - option MPI	
	Informatique	X		
ENS Lyon	MP - option MP		MP - option MPI	
	Informatique - option M	X	Informatique - option P	
ENS (Paris)	MP - option MP		MP - option MPI	
	Informatique	X		

Matière dominante du TIPE : Informatique
(mathématiques, informatique ou physique)

Titre du TIPE : Jeu du puissance 4 et intelligence artificielle

Nombre de pages (à porter dans les cases ci-dessous) :

Texte	T	5	Illustrations	I	3	Bibliographie	B	1

Résumé imprimé (6 lignes) :

Le jeu du puissance 4 se joue à deux, on peut donc écrire un programme afin de jouer contre un ordinateur. L'algorithme mini maxi souvent utilisé pour les jeux à deux joueurs convient, mais il est lent, comment l'améliorer ?

Les élagages permettent de couper des branches de l'arbre exploré en laissant invariant le résultat de l'algorithme. L'élagage alpha beta en est un, et j'ai inventé un nouvel élagage, dit du coup gagnant, dont l'idée repose sur un comportement humain. Cet élagage peut être optimisé en temps en ne l'appliquant pas systématiquement, et j'ai cherché à partir d'un modèle simplifié des arbres l'optimum.

A Nantes, le 30/05/2013
Signature du (de la) candidat(e)

Signature du professeur responsable de la classe préparatoire dans la discipline

Cachet de l'établissement

Table des matières

1 Présentations générales

1.1 Le jeu

Le puissance 4 est un jeu à deux joueurs, qui se déroule sur un plateau de 6 lignes et 7 colonnes. À tour de rôle, les joueurs déposent un pion de leur couleur dans la colonne libre de leur choix, et celui-ci tombe dans la case libre la plus basse.

Le vainqueur est celui qui réussi à aligner quatre pions de sa couleur, verticalement, horizontalement, ou en diagonale. La partie peut être nulle si aucun n'y parvient avant que le plateau ne soit rempli.

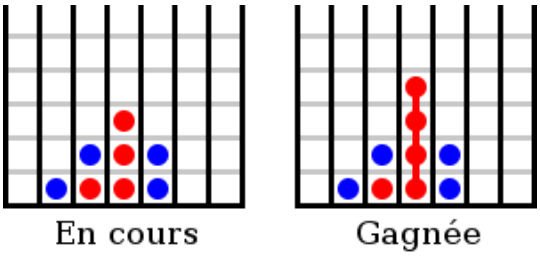

FIGURE 2 – Plateaux de jeu.

Dans toute la suite, le programme est représenté en bleu et l'adversaire en rouge.

1.2 L'algorithme mini maxi

Pour jouer contre l'ordinateur, il faut écrire un programme qui choisit dans quelle colonne placer ses pions. Pour cela, on peut utiliser l'algorithme mini maxi, qui repose sur cette observation : supposons qu'un joueur connaisse le gain que lui procure chacun des coups qui s'offre à lui, il va alors jouer le coup qui maximise son gain. S'il considère les gains de son adversaire, il joue le coup qui le minimise.

L'algorithme construit donc l'arbre des plateaux accessibles, les branches correspondant au choix d'une colonne par le joueur dont ce serait le tour, et attribue des notes aux feuilles qui quantifient son gain s'il atteint cette feuille. Puis il étiquette les nœuds internes depuis les feuilles vers la racine en suivant la règle précédente : si le nœud correspond à un choix du programme, on étiquette le nœud par le maximum des étiquettes de ses fils, et sinon par le minimum.

De cette manière, l'étiquette d'un nœud est celle du plateau que l'on atteindra, tout en prenant en compte la défense de l'adversaire.

Lorsque tous le nœuds ont été étiquetés, le programme joue le coup qui a l'étiquette maximale.

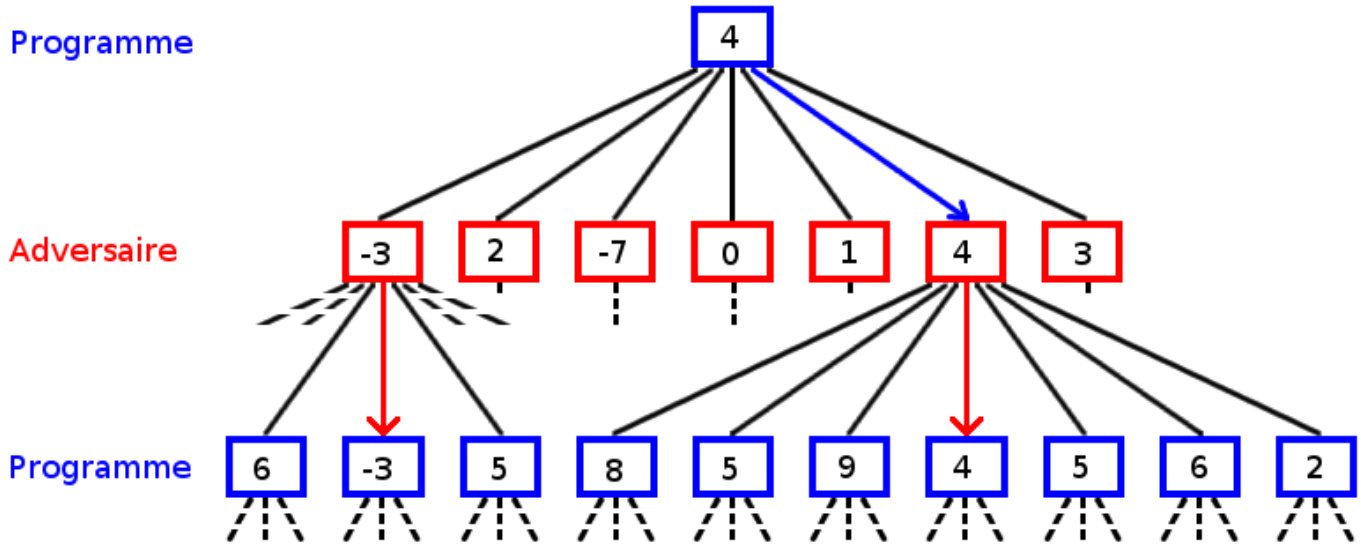

FIGURE 3 – Partie de l'arbre des coups possibles : l'adversaire jouerait le pire coup, le programme le meilleur. Les notes du bas proviennent des feuilles, non représentées.

Afin de noter les plateaux, l'algorithme nécessite une fonction d'évaluation, qui dépend du jeu et dont les critères sont décidés par le programmeur. Dans deux cas, le programme donnera une réponse optimale :

— la fonction d'évaluation est parfaite : il suffit alors de construire uniquement un arbre de hauteur 1, et de jouer le meilleur coup ;

— les feuilles correspondent uniquement à des parties finies : l'arbre est de taille maximale. On attribue les notes 1 aux parties gagnées, 0 aux nulles, et -1 aux perdues. En jouant dans un coup étiqueté par la note 1, le programme est sûr de gagner.

Au puissance 4, c'est une contrainte de temps qui empêche de construire l'arbre total : en effet il reste au maximum $7 \times 6 = 42$ coups à jouer, donc si aucune partie n'est finie avant le dernier coup, il y aurait 7^{42} feuilles, et alors, pour un temps d'exécution de la fonction d'évaluation de l'ordre de $1ns$, il faudrait déjà 10 milliards de milliards d'années de calcul juste pour les feuilles...

Cet algorithme suppose que l'adversaire joue bien : en jouant dans une autre colonne, si l'adversaire se trompe, on peut parfois obtenir un meilleur gain.

1.3 Fonction d'évaluation

Afin de pouvoir travailler, j'ai codé une fonction d'évaluation assez simple, mais qui donne de bon résultats :

— trois pions alignés valent 4 si l'une des extrémités est libre, de façon à se qu'on puisse aligner quatre pions au total ;

— deux pions alignés valent 1 si les cases autour sont libres de façons à se qu'on puisse aligner quatre pions au total.

Les alignements de pions du programme sont comptés positivement, et ceux de l'adversaire négativement, de sorte que la fonction d'évaluation soit symétrique.

2 Améliorations de l'algorithme

L'algorithme mini maxi fonctionne, mais en pratique la hauteur de l'arbre doit être très limitée car il est lent. On cherche à l'améliorer.

2.1 L'élagage alpha beta

2.1.1 Principe

Dans certains cas, il est possible d'économiser le parcours de sous-arbres sans changer le résultat de l'algorithme. Sachant que le parcours de l'arbre se fait en profondeur, à l'aide d'une fonction récursive, considérons les portions d'arbre suivantes :

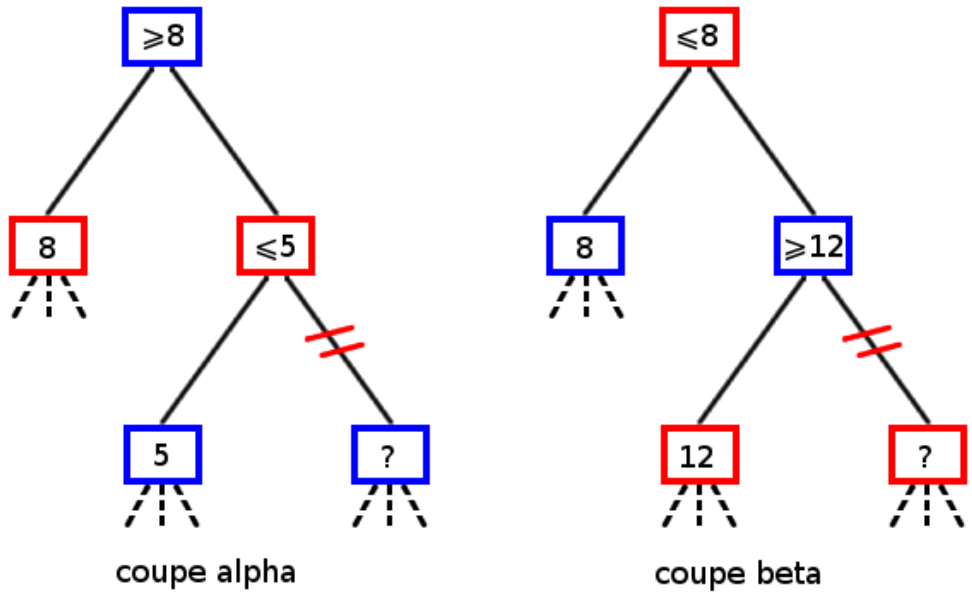

FIGURE 4 – Exemple de coupe alpha et beta

Dans la figure 4 à gauche, le sous-arbre gauche a été exploré et sa racine est étiquetée par 8. Le sous-arbre droit n'est pas complètement exploré, mais on sait que son étiquette sera inférieure à 5. On a $\max(8, \min(5, \ldots)) = 8$ donc on connaît la note de la racine sans avoir à explorer le sous-arbre gauche : on peut effectuer une coupe, dite alpha.

La figure de droite illustre la même idée, symétrique par rapport aux joueurs. On a $\min(\max(12, \ldots), 8) = 8$, on peut donc effectuer une coupe, dite beta.

2.1.2 Résultats

Pour un arbre de hauteur 10, l'élagage alpha beta supprime 80% des nœuds internes et 82% des feuilles (une représentation graphique du nombre de feuilles et présentée plus loin). De plus, si l'algorithme essaye en premier lieu les meilleurs coups, le nombre de coupes augmente. On peut donc obtenir d'encore meilleurs résultats, pour un coût de calcul supplémentaire très faible.

2.2 L'approfondissement itératif

2.2.1 Principe

Le principe de l'algorithme mini maxi est d'éloigner l'horizon du programme pour lui permettre de mieux évaluer chaque coup. On en déduit que plus la hauteur de l'arbre est grande, plus la réponse va être pertinente.

Cependant, explorer un arbre plus grand prend du temps, et ce temps dépend de l'ordinateur et de sa charge. C'est donc difficile de fixer une hauteur.

On peut donc fixer un temps maximum pour répondre, pendant lequel on explore des arbres de plus en plus grands, jusqu'à ce que ce temps soit atteint, et alors on donne le coup désigné par le dernier arbre que l'on a entièrement parcouru.

De cette manière, on répond avec la meilleure pertinence possible dans le temps imparti.

De plus, si l'on note p la hauteur de l'arbre, le nombre de feuille et environ 7^p, donc si l'on atteint la profondeur p_{max}, la complexité, en nombre de feuilles, de cet algorithme est en

$$O\left(\sum_{p=1}^{p_{max}} 7^p\right) = O\left(7^{p_{max}}\right)$$

La complexité est donc la même que l'algorithme mini maxi avec un arbre de hauteur p_{max} : la plus grande partie du temps de calcul est consacrée à la réponse que l'on donne.

2.2.2 Profondeur atteinte au cours de la partie

Le graphique suivant montre la profondeur moyenne atteinte au cours de parties de puissance 4 par cet algorithme, ainsi que la profondeur maximale possible : au-delà le plateau est rempli.

En abscisse se trouve le nombre de coup déjà joués.

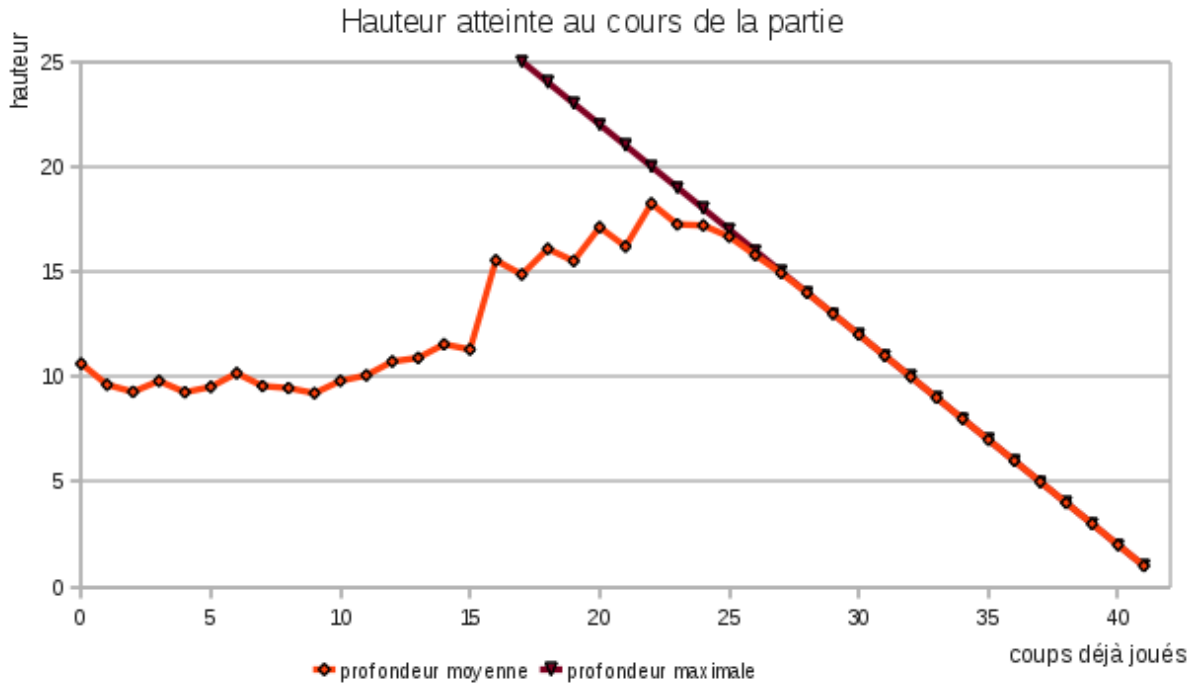

FIGURE 5 – Hauteur maximale de l'arbre pendant la partie.

Interprétations :

— au début de la partie, quand le plateau est vide, l'arbre va être le plus long à explorer car aucune colonne n'est remplie : il y a souvent 7 coups possibles ;

— plus tard, des colonnes se remplissent assez souvent, et donc les arbres sont moins gros : on a le temps d'explorer des arbres plus hauts ;

— à la fin de partie, le nombre de coups prévisibles à l'avance se réduit car le plateau est rempli : la profondeur maximale est plafonnée.

L'approfondissement itératif permet aussi de varier la profondeur au cours de la partie.

3 Mon nouvel élagage

Pour encore accélérer l'algorithme, j'ai cherché une autre façon d'élaguer l'arbre.

3.1 Motivation et principe

Lorsqu'un humain joue, et qu'il peut gagner en un seul coup, il ne cherche pas à savoir si jouer dans les autres colonnes vaut le coup. Il en va de même quand il peut perdre au prochain coup : il assure tout de suite sa défense.

L'algorithme mini maxi peut, lui, très bien explorer les six premiers coups, avant de se rendre compte qu'il gagne avec le septième.

On peut donc améliorer l'algorithme en regardant d'abord si l'on peut gagner, puis si l'on peut perdre, et le cas échéant jouer ces coups. Ces tests peuvent en fait être effectués tout au long de l'arbre, et si un coup gagnant ou perdant est trouvé, seul un sous-arbre sera coupé.

Si le coup gagnant ou perdant apparaît dès la racine, il faut le jouer, sinon on est au sein de l'arbre et :

— si c'est un coup gagnant, on renvoie directement la note d'une partie gagnée : $EVAL\_MAX - (p+1)$ (où $EVAL\_MAX$ est une constante, et p est la profondeur du nœud (on gagne à la profondeur p+1), pour que gagner rapidement soit mieux vu que gagner dans longtemps) ;

— si c'est un coup perdant, on explore uniquement ce coup.

3.2 Invariance du résultat

Cet élagage coupe des sous-arbres, montrons que le résultat de l'algorithme mini maxi reste inchangé.

On suppose qu'au niveau d'un nœud, c'est le programme qui doit jouer, son étiquette est donc le maximum des étiquettes de ses fils. Il y a quatre situations différentes :

— on trouve un coup gagnant : voir figure 6

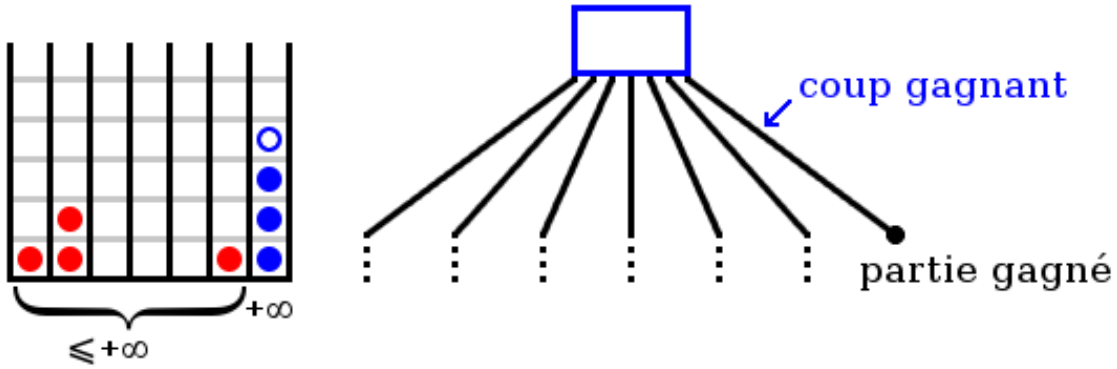

FIGURE 6 – Situation de coup gagnant. Le pion creux correspond au coup qu'il faut jouer.

Le début d'un sous-arbre est représenté, et en jouant tout à droite, le programme peut gagner.
La note que doit avoir la case bleue est la note maximum de ses fils. À droite, cette note est $EVAL\_MAX - (p+1)$, et dans les autres colonnes la note lui sera inférieure ou égale, donc le résultat est inchangé.

— on trouve un coup perdant : voir figure 7
Si le programme ne joue pas tout à droite, l'adversaire va gagner. Les six premiers coups ont donc la note $-(EVAL\_MAX - (p+1))$, alors que le dernier, à droite, ne peut avoir une pire note : le résultat est inchangé.

— le programme est piégé, l'adversaire va pouvoir gagner à deux endroits différents : voir figure 8
Le programme ne va donc explorer que le premier coup perdant qu'il trouve. L'adversaire a alors un coup gagnant donc la note obtenue est $-(EVAL\_MAX - (p+1))$. Quelque soit le coup joué par le programme l'adversaire gagnera, donc n'explorer qu'un seul coup ne change pas le résultat.

— on ne trouve ni l'un, ni l'autre, et alors on effectue les opérations habituelles de mini maxi.

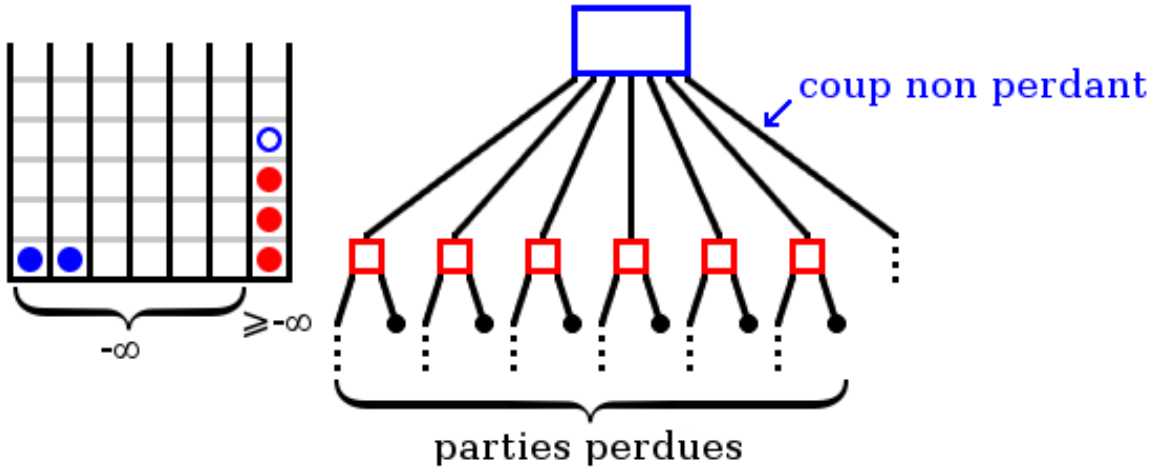

FIGURE 7 – Situation de coup perdant.

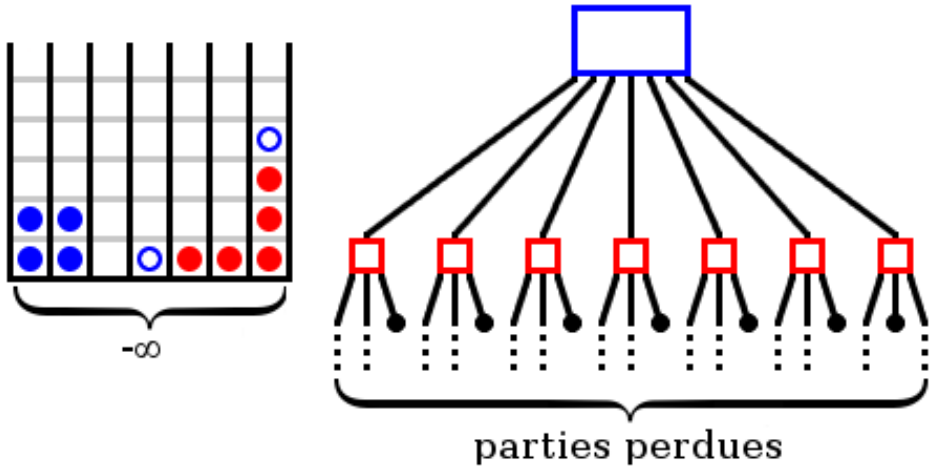

FIGURE 8 – Situation avec deux coups perdants.

Par symétrie de l'algorithme, le même raisonnement s'applique si c'est à l'adversaire de jouer, en conclusion :

L'élagage du coup gagnant laisse invariant le résultat de l'algorithme.

3.3 Performances

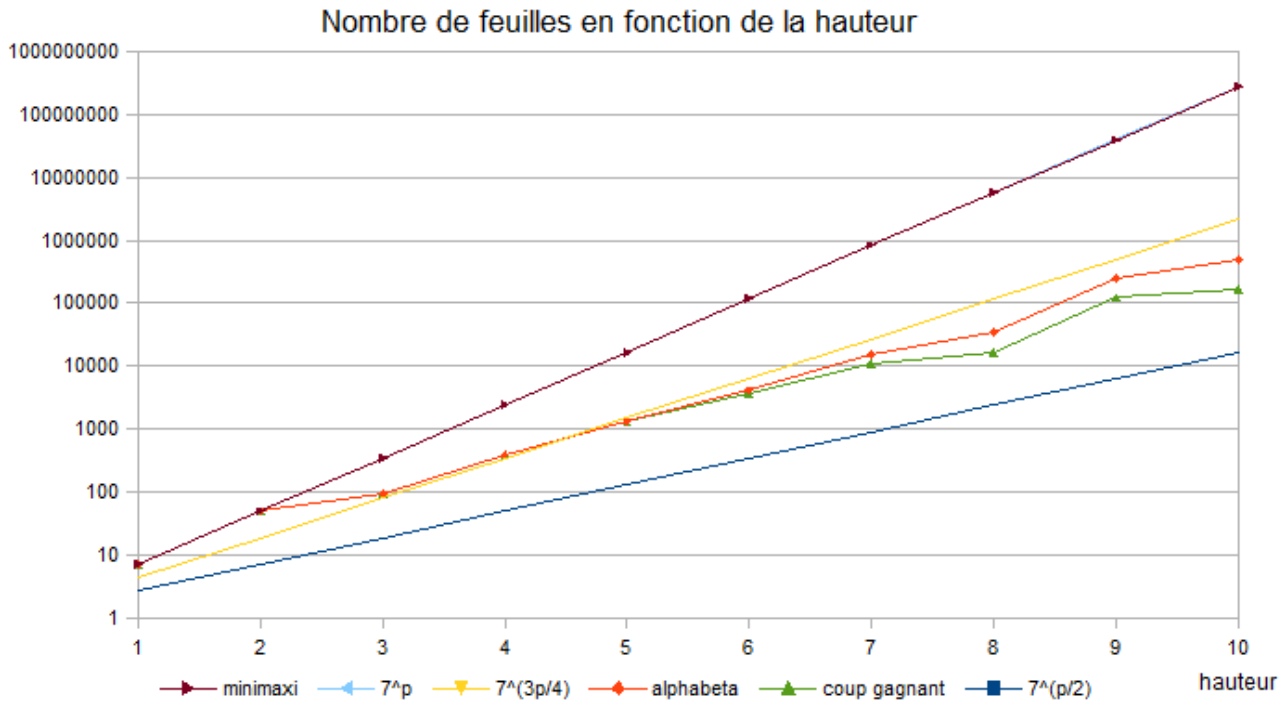

FIGURE 9 – Comparaison des algorithmes.

L'élagage alpha beta permet de diminuer nettement le nombre de feuilles de l'arbre : il est de l'ordre de 3.8^p où p est la hauteur de l'arbre, contre 7^p pour mini maxi.

L'élagage du coup gagnant améliore encore ces chiffres : pour un arbre de hauteur 10, l'élagage coupe deux tiers des feuilles.

3.4 Optimisation

Cet élagage permet effectivement de couper des sous-arbres, mais il demande des opérations en plus : tester à chaque nœud interne jusqu'à trois fois chaque coup légal : s'il est gagnant, perdant, puis continuer mini maxi.

Le temps de calcul supplémentaire est constant (pour le même nombre de coups légaux), alors que le temps économisé dépend de la taille des sous-arbres coupés. Ainsi en exécutant l'élagage du coup gagnant uniquement pour les nœuds assez loin des feuilles, dont les sous-arbres sont grands, on peut peut-être accélérer l'algorithme.

On note q la distance minimale entre les feuilles (les plus profondes) et les nœuds pour laquelle on applique l'élagage du coup gagnant : pour $q = 1$, l'élagage est effectué pour tous les nœuds internes, pour $q = 2$, il l'est sur tous sauf le dernier étage de nœuds internes, etc.

Le graphique suivant montre le rapport entre le temps d'exécution de l'algorithme alpha beta avec l'élagage et sans, en fonction q.

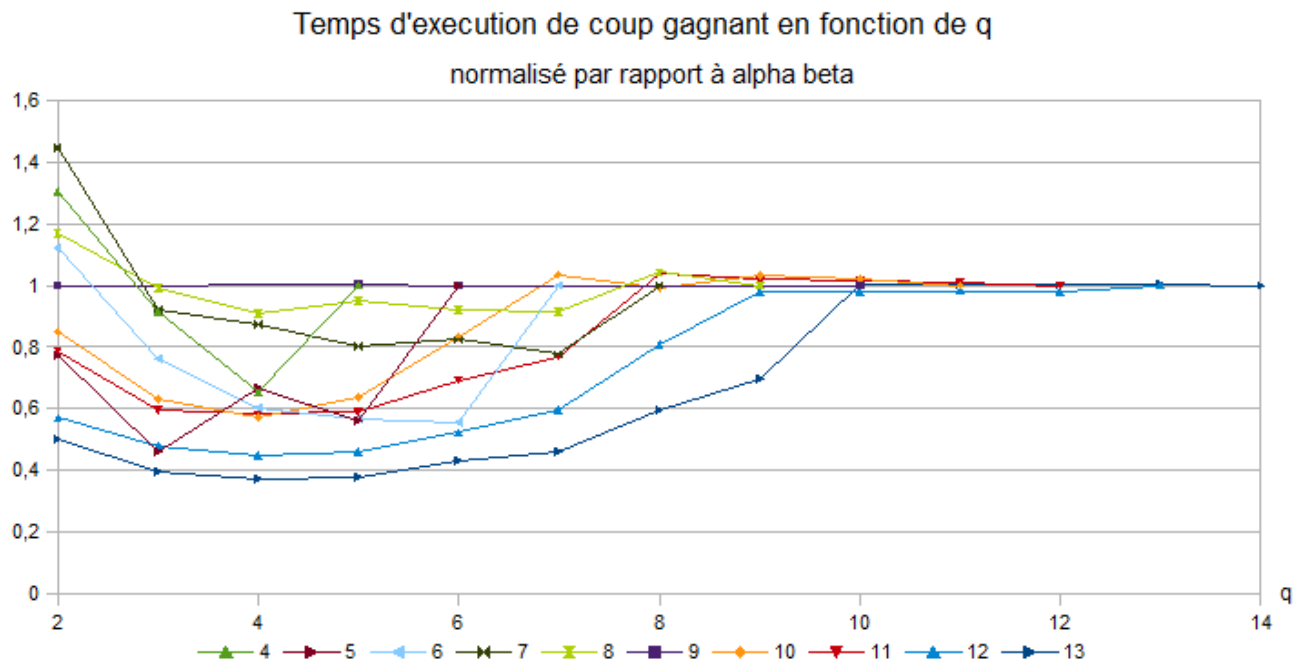

FIGURE 10 – Temps pour différentes hauteurs.

On peut voir que :
— même si l'élagage du coup gagnant supprime beaucoup de feuilles, l'appliquer systématiquement ralenti l'algorithme si la taille de l'arbre n'est pas suffisante ;
— on a bien un minimum, qui semble ne pas dépendre de la hauteur de l'arbre, pour $q = 4$.

4 Calcul du temps d'exécution de l'algorithme

Afin de voir si le minimum ne dépend pas de la profondeur, on cherche une expression du temps de calcul de l'algorithme.

4.1 Modèle simplifié de l'arbre

On fait les hypothèses simplificatrices suivantes :
— les nœuds internes ont exactement k fils (k peut ne pas être entier en le considérant comme une moyenne) ;
— la probabilité de ne pas avoir de coupe par l'élagage du coup gagnant est constante, notée α ;
— les temps de calcul décrits ci-dessous sont constants.

4.2 Temps de calcul

On note :
— $q \in \mathbb{N}^*$ la profondeur jusqu'à laquelle on applique l'élagage du coup gagnant (si $q = 1$, on effectue l'élagage sur tous les nœuds internes) ;
— τ le temps de calcul au sein d'un nœud (sans le calcul des sous-arbres) ;
— τ' le temps de calcul au sein d'une feuille ;
— t le temps supplémentaire, par nœud, engendré par coup gagnant.

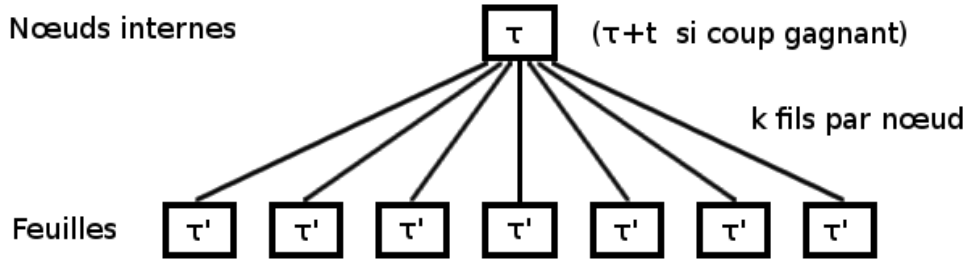

FIGURE 11 – Temps d'exécution des différentes parties.

On note enfin T_p le temps de calcul d'un arbre de hauteur $p \in \mathbb{N}$. On a alors la relation de récurrence

$$T_p = \begin{cases} \tau' & \text{si } p = 0 \\ \tau + k \cdot T_{p+1} & \text{si } 0 < p < q \\ \tau + t + \alpha \cdot k \cdot T_{p+1} & \text{si } p \geq q \end{cases}$$

car $\alpha \cdot k \cdot T_{p+1}$ est l'espérance du temps de calcul des sous-arbres, si l'élagage du coup gagnant est effectué.

Expérimentalement, on va voir que $k \neq 1$ et $\alpha k \neq 1$, donc la suite $(T_p)_{p\in\mathbb{N}}$ est arithmético-géométrique entre les rangs 0 et q-1, d'où, d'après le terme général d'une telle suite (cf annexe A),

$$T_{q-1} = k^{q-1}\left(T_0 - \frac{\tau}{1-k}\right) + \frac{\tau}{1-k}$$

De plus, la suite $(T_p)_{p\geq q-1}$ est également arithmético-géométrique, donc pour $p \geq q-1$,

$$T_p = (\alpha k)^{p-(q-1)}\left(T_{q-1} - \frac{t+\tau}{1-\alpha k}\right) + \frac{t+\tau}{1-\alpha k}$$

On en déduit, pour $p \geq q-1$,

$$T_p = (\alpha k)^{p-q+1}\left(k^{q-1}\left(\tau' - \frac{\tau}{1-k}\right) + \frac{\tau}{1-k} - \frac{t+\tau}{1-\alpha k}\right) + \frac{t+\tau}{1-\alpha k}$$

ou encore, en posant

$$\boxed{\lambda = \tau' - \frac{\tau}{1-k}} \qquad \boxed{\mu = \frac{\tau}{1-k} - \frac{t+\tau}{1-\alpha k}} \qquad \boxed{t_1 = \frac{t+\tau}{1-\alpha k}}$$

on a

$$\boxed{T_p = (\alpha k)^p\, \alpha^{-q+1}\left(\lambda + \mu k^{-q+1}\right) + t_1}$$

Pour le temps de calcul sans l'élagage du coup gagnant, il faut $p = q-1$, d'où, avec

$$\boxed{t_2 = \mu + t_1 = \frac{\tau}{1-k}}$$

le temps de calcul pour alpha beta est :

$$\boxed{T_p^{\alpha\beta} = \lambda k^p + t_2}$$

On pose, pour p donné, $f(q) = \frac{T_p}{T_p^{\alpha\beta}}$

Soit

$$f(q) = \frac{(\alpha k)^p\, \alpha^{-q+1}\left(\lambda + \mu k^{-q+1}\right) + t_1}{\lambda k^p + t_2}$$

Dès que p n'est pas trop petit, on peut négliger t_1 et t_2 devant les autres termes, et alors

$$f(q) \simeq \alpha^{p-q+1}\left(1 + \frac{\mu}{\lambda}k^{-q+1}\right)$$

On voit alors qu'augmenter la hauteur de 1 multiplie le rapport $\frac{T_p}{T_p^{\alpha\beta}}$ par $\alpha < 1$, donc

$$\boxed{\text{L'élagage du coup gagnant est d'autant plus efficace que l'arbre est haut}}$$

On cherche à présent le minimum de f. Si on considère q comme une variable réelle, f est dérivable et

$$f'(q) = \frac{(\alpha k)^p\, \alpha^{-q+1}}{T_p^{\alpha\beta}}\left(\ln(\alpha)\left(\lambda + \mu k^{-q+1}\right) + \mu \ln(k) k^{-q+1}\right)$$

Et donc

$$\begin{aligned} f'(q) = 0 &\Leftrightarrow \ln(\alpha)\left(\lambda + \mu k^{-q+1}\right) + \mu \ln(k) k^{-q+1} = 0 \\ &\Leftrightarrow \ln(\alpha)\left(\lambda + \mu k^{-q+1}\right) = -\mu \ln(k) k^{-q+1} \\ &\Leftrightarrow k^{-q+1}\left(\mu + \mu \frac{\ln(k)}{ln(\alpha)}\right) = -\lambda \\ &\Leftrightarrow -q+1 = \frac{1}{ln(k)} \ln\left(\frac{-\lambda}{\mu + \mu\frac{\ln(k)}{ln(\alpha)}}\right) \end{aligned}$$

Soit

$$\boxed{f'(q_{min}) = 0 \Leftrightarrow q_{min} = \frac{1}{ln(k)} \ln\left(\frac{-\mu}{\lambda}\frac{\ln(\alpha k)}{\ln(\alpha)}\right) + 1}$$

On trouve alors que cette valeur ne dépend pas de la profondeur.

4.3 Mesure des temps moyens

Afin de pouvoir confronter le modèle précédent aux expériences, il faut mesurer le temps d'exécution moyen de chaque parties de l'algorithme.

Au niveau d'une feuille, l'algorithme exécute uniquement la fonction d'évaluation. J'ai donc exécuté cette fonction sur un grand nombre de plateaux, et mesuré le temps d'exécution, pour obtenir le temps moyen d'exécution :

$$\boxed{\tau' \simeq 3,5\mu s}$$

Ensuite, pour les temps moyens d'exécution au niveau des nœuds internes, il faut une moyenne qui corresponde à la réalité, donc mesurer le temps d'exécution de l'algorithme sans appeler la fonction d'évaluation (le temps d'exécution au niveau des feuilles est alors nul) ne convient pas car les coupes alpha et beta ne correspondent plus à un cas réel, ainsi que les élagages du coup gagnant.

Pour avoir une meilleure estimation, j'ai compté le nombre de nœuds internes N et de feuilles F parcourus pendant l'algorithme, et j'ai mesuré le temps total d'exécution T, ceci avec et sans l'élagage du coup gagnant. Alors le temps consacré aux nœuds internes est $T - F \times \tau'$, donc le temps moyen par nœud est $\frac{T-F\tau'}{N}$, soit

$$\boxed{\tau \simeq 11\mu s} \quad \text{et} \quad \boxed{t \simeq 35 - 11\mu s \simeq 24\mu s}$$

4.4 Embranchement et probabilité

Si l'on note F le nombre de feuilles d'un arbre de hauteur p, et si l'on néglige les feuilles qui ne sont pas de profondeur maximale, le coefficient k d'embranchement, supposé constant, vérifie $F = k^p$. En mesurant donc le nombre de feuilles on trouve

$$k_{mini\ maxi} \simeq 7 \qquad k_{\alpha\beta} \simeq 3,8 \qquad k_{coup\ gagnant} \simeq 3,5$$

On prend le maximum, pour surestimer le temps :

$$\boxed{k = max(k_{\alpha\beta}, k_{coup\ gagnant}) = 3,8}$$

Trois remarques s'imposent :

- $k_{mini\ maxi}$ n'est pas rigoureusement égal à 7 car à partir d'un certain nombre de coup, des colonnes se remplissent et donc le coefficient d'embranchement diminue ;
- il serait possible de mener les calculs de T_p avec les deux valeurs légèrement différentes de $k_{\alpha\beta}$ et $k_{coup\ gagnant}$, mais on verra plus loin que l'hypothèse de constance de α est grossière, donc la précision sur k améliorerait peu le résultat ;
- on a
$$k_{mini\ maxi}^{1/2} \leq k_{\alpha\beta} \leq k_{mini\ maxi}^{3/4}$$
comme c'est écrit dans la littérature, la plus faible valeur correspondant au cas où les coups sont essayés du meilleur au moins bon, ce qui maximise les coupes alpha beta.

Il reste à estimer la probabilité de ne pas avoir une coupe avec l'élagage du coup gagnant. Pour cela, j'ai mesuré le nombre de coupes effectuées et le nombre de nœuds internes parcourus, et le calcul donne

$$\boxed{\alpha \simeq 0,9}$$

Mais cette valeur varie beaucoup selon la hauteur du nœud courant.

4.5 Calcul numérique

On peut maintenant calculer

$$\boxed{\lambda \simeq 7,4\mu s} \qquad \boxed{\mu \simeq 10\mu s} \qquad \boxed{t_1 \simeq -14\mu s} \qquad \boxed{t_2 \simeq -3,9\mu s} \qquad \boxed{q_{min} \simeq 3}$$

Et on a

$$\begin{aligned} f(1) &= 0,843 \\ f(2) &= 0.532 \\ f(3) &= 0,473 \\ f(4) &= 0,491 \\ f(5) &= 0,535 \end{aligned}$$

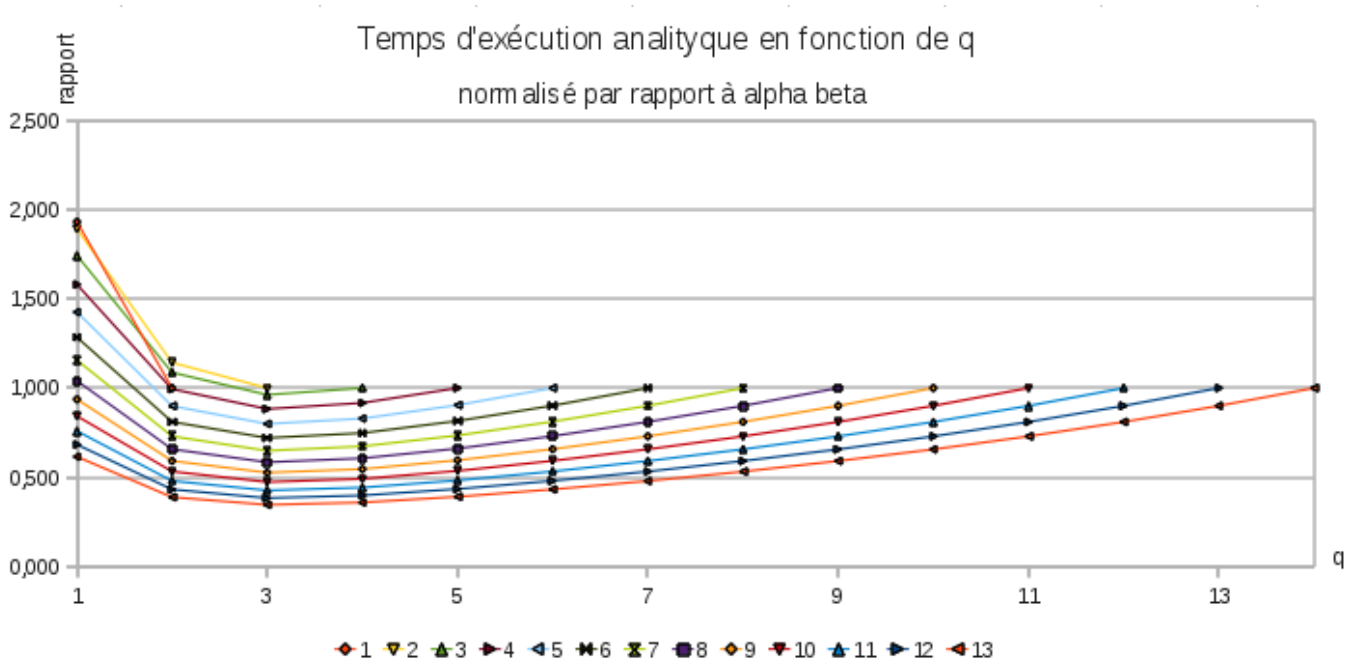

FIGURE 12 – f(q) pour différentes hauteurs

On atteint le temps optimal pour $q = 3$

Les moitiés gauche des courbes ont la même allure que celles obtenues expérimentalement pour les grandes hauteurs, et on trouve le même rapport minimal, mais en $q = 3$ contre $q = 4$ pour les mesures. Ceci peut être dû aux imprécisions sur les constantes.

La fin des courbes ne se ressemblent pas à cause d'une erreur : pour les mesures expérimentales j'ai appliqué l'algorithme sur un plateau vide, il est donc impossible d'avoir des coups gagnants ou perdants avant le quatrième coups. Ceci peut également expliquer le décalage du minimum : on a effectué moins de coupes au total, donc on gagne moins de temps et il faut moins appliquer l'élagage du coup gagnant pour aller plus vite.

5 Conclusion

Ce nouvel élagage permet d'accélérer un peu l'algorithme mini maxi avec l'élagage alpha beta, et est surtout performant sur les grands arbres. De plus, le temps minimum de ce nouvel algorithme ne dépend que de paramètres fixés, ce qui est intéressant pour chercher le programme le plus rapide.

D'autres améliorations de cet algorithme sont connues, mais la plupart modifient son résultat, alors que l'élagage du coup gagnant le laisse invariant.

6 Bibliographie

— Tristan Cazenave, L'intelligence artificielle : une approche ludique ; Eyrolles (2011)
— Stuart J. Russell and Peter Norvig, Artificial Intelligence : A Modern Approach ; Prentice Hall (1995)
— Georges ROBIN, Jeu, intelligence artificielle et apprentissage - Application à Puissance 4 ; http ://geo-rem.perso.neuf.fr/ (mars 2013)

A Annexe : suite arithmético-géométrique

A.1 Théorème

Soient $(a,b) \in \mathbb{R} \setminus \{1\} \times \mathbb{R}$, et $u_0 \in \mathbb{R}$
On pose

$$\forall n \in \mathbb{N}, u_{n+1} = au_n + b$$

Alors cette suite récurrente arithmético-géométrique admet comme terme général

$$\boxed{\forall n \in \mathbb{N}, u_n = a^n \left(u_0 - \frac{b}{1-a} \right) + \frac{b}{1-a}}$$

A.2 Démonstration

On pose $\forall n \in \mathbb{N}$, $v_n = u_n - \frac{b}{1-a}$
On a alors

$$\begin{aligned} v_{n+1} &= au_n + b - \frac{b}{1-a} \\ &= au_n + \frac{b(1-a)-b}{1-a} \\ &= a\left(u_n - \frac{b}{1-a} \right) \\ &= av_n \end{aligned}$$

Donc la suite $(v_n)_{n\in\mathbb{N}}$ est géométrique, et donc $\forall n \in \mathbb{N}, v_n = a^n v_0$.
On en déduit

$$\forall n \in \mathbb{N}, u_n = a^n \left(u_0 - \frac{b}{1-a} \right) + \frac{b}{1-a}$$

B Annexe : extrait de code source

N.B. : La classe jeu est abstraite pour pouvoir utiliser la fonction minimaxi avec d'autres jeux, il faut renseigner les fonctions evaluer(), jouer(), dejouer(), coups_legaux(), et coup_suivant().

```
int jeu::minimaxi(T_etiquette joueur, int &coup_max, int profondeur, int alpha = - EVAL_MAX - 1, int beta = EVAL_MAX + 1)
{
   T_coup le_coup_gagnant(coup_zero), coup(coup_zero);
   T_etiquette g(VIDE);
   int eval, eval_max(-EVAL_MAX - 1);

   // permier cas terminal : profondeur maximale
   if(profondeur == pmax)
      return joueur*evaluer(); // joueur vaut +/- 1


    T_liste_coups coups = coups_legaux(); // liste des coups legaux

   // cas terminal : plus de coups legaux : match nul
   if(!coup_suivant(coups, coup))
      return 0;

   // s'il faut effectuer l'elagage du coup gagnant :
   if(profondeur < coup_gagnant_max + 1)
   {
      // cherche un coup gagnant
      le_coup_gagnant = coup_gagnant(coups, joueur);
      if(le_coup_gagnant != coup_zero)
      {
         eval = EVAL_MAX - profondeur - 1; // -1 car on gagne a la profondeur d'apres

         if(profondeur == 0) // on veut savoir le meilleur coup
            coup_max = le_coup_gagnant;

         return eval;
      }

      // cherche un coup perdant
      le_coup_gagnant = coup_gagnant(coups, adversaire(joueur));
      if(le_coup_gagnant != coup_zero) // on ne va regarder que celui-la
      {
         if(profondeur == 0) //il faut forcement jouer ici (sinon on ne testera que ce coup)
         {
            coup_max = le_coup_gagnant;
            return 0;
         }
      }
   }

   // pour chaque coup legal
   for(coup = coup_zero; coup_suivant(coups, coup);)
   {
      // il faut regarder ce coup : le coup perdant s'il existe, sinon tous
      if(le_coup_gagnant == coup_zero || coup == le_coup_gagnant)
      {
         // on joue virtuellement
         jouer(coup, joueur);


         // cas terminal :
         if(profondeur >= coup_gagnant_max + 1) // on n'effectue pas coup gagnant, il faut verifier si quelqu'un gagne
         {
            g = gagnant(coup); // VIDE, joueur, ou NUL (pas l'adversaire)
            if(g == joueur) // on gagne
               eval = EVAL_MAX - profondeur;

            else if(g == NUL)
               eval = 0;
         }

         if(g == VIDE) // si effectuer_coup_gagnant : toujours vrai, sinon depend de gagnant()
         {
            // on appelle recursivement cette fonction, pour savoir la note du coup teste
            eval = -minimaxi(adversaire(joueur), coup_max, profondeur + 1, -beta, -alpha);
         }
```

```
71
72          // on dejoue le coup pour ne pas modifier le plateau
73          dejouer(coup);
74
75          // on cherche la note maximale
76          if(eval > eval_max) // meilleur note que precedemment
77          {
78              // on enregistre la note maximale
79              eval_max = eval;
80
81              if(profondeur == 0) // il faut renseigner le meilleur coup
82              {
83                  coup_max = coup;
84              }
85
86              // coupes alpha/beta :
87              else if(eval_max > alpha) // il ne faut pas changer alpha pour la profondeur 0,
                      ce n'est pas la valeur qui nous interesse, mais les coups qui l'atteignent
88              {
89                  alpha = eval_max;
90                  if(alpha >= beta) // elagage alpha/beta : si le maximum (pour l'instant) est
                          plus grand que ce que mini a deja trouve, c'est pas la peine de
                          continuer...
91                      return eval_max;
92              }
93          }
94      }
95  }
96
97  return eval_max;
98 }
```