ÉPREUVE COMMUNE DE TIPE 2012 - Partie D

TITRE :

Jeu de l'awalé

Temps de préparation : ……..……2 h 15 minutes
Temps de présentation devant les examinateurs : …….10 minutes
Dialogue avec les examinateurs : ……..10 minutes

GUIDE POUR LE CANDIDAT :

Le dossier ci-joint comporte au total : 14 pages
Document principal (14 pages, dont celle-ci) ; annexe : 0 page

Travail **suggéré** au candidat :

Le candidat peut, au choix, proposer une synthèse du texte, ou mettre en avant un ou plusieurs algorithmes ou encore une ou plusieurs démonstrations. Dans tous les cas il veillera à bien comprendre ce qu'il expose et cherchera a apporter un éclairage personnel sur le texte.

Les algorithmes pourront être écrits en Caml, en Pascal ou en pseudo-code et devront être complétés en fonction des différentes orientations qui sont données et/ou en fonction des propres idées du candidat.

Attention : s*i le candidat préfère effectuer un autre travail sur le dossier, il lui est* ***expressément recommandé*** *d'en informer le jury avant de commencer l'exposé.*

CONSEILS GENERAUX POUR LA PREPARATION DE L'EPREUVE :

* Lisez le dossier en entier dans un temps raisonnable.
* Réservez du temps pour préparer l'exposé devant les examinateurs.

- Vous pouvez écrire sur le présent dossier, le surligner, le découper … mais tout sera à remettre aux examinateurs en fin d'oral.
- En fin de préparation, rassemblez et ordonnez soigneusement TOUS les documents (transparents, *etc.*) dont vous comptez vous servir pendant l'oral, ainsi que le dossier, les transparents et les brouillons utilisés pendant la préparation. En entrant dans la salle d'oral, vous devez être prêt à débuter votre exposé.
- A la fin de l'oral, vous devez remettre aux examinateurs le présent dossier. Tout ce que vous aurez présenté aux examinateurs pourra être retenu en vue de sa destruction.

IL EST INTERDIT DE SORTIR LE SUJET DU SITE DE L'EPREUVE

1. Awalé : historique et règles

Nous allons nous intéresser à un jeu très ancien dont les origines sont encore mal connues mais semblent associées aux jeux de l'ancienne Égypte. Le jeu d'awalé fait partie des jeux dits de semailles car, originairement, il se joue à l'aide de graines qui sont semées dans deux rangées de six trous creusés dans un plateau, généralement en bois. Deux trous plus grands, appelés greniers, servent de réceptacle pour les gains des joueurs. Ce jeu est très répandu en Afrique, mais à partir du XVII^{ième} siècle on le trouve également en Amérique du Sud, aux Philippines, en Indonésie, en Malaisie, … On le trouve aussi sous le nom d'awélé.

Les règles de ce jeu sont particulièrement simples et s'apprennent en quelques minutes. Il en existe de nombreuses variantes mais nous ferons référence uniquement à la plus répandue. Le jeu se joue à deux, à tour de rôle, l'un des joueurs pouvant être un ordinateur. Il y a initialement quatre graines dans chacune des cases du plateau, soit 48 graines.

A tour de rôle, chaque joueur saisit les graines de l'une de ses six cases – celles du plateau se trouvant devant lui – et les sème une à une dans les cases suivantes dans le sens inverse des aiguilles d'une montre. Après cette distribution si la dernière graine semée tombe dans un trou de l'adversaire comportant déjà une ou deux graines, le joueur récolte les deux ou trois graines de ce trou et les place dans son grenier. Le joueur récupère également les graines des trous précédents de la rangée de l'adversaire qui respectent cette condition, mais ne peut pas récolter le contenu de plus de quatre trous consécutifs. Une règle d'or est que la prise de ces graines est obligatoire. Si avec le nombre de graines présentes dans l'une de ses cases le joueur peut faire un ou plusieurs tours du plateau, alors il devra sauter systématiquement la case de départ.

Un joueur ne peut pas affamer l'adversaire délibérément sauf s'il n'a pas d'autre coup à jouer : ceci signifie que, dans l'esprit de l'awalé il est impératif de jouer un coup qui permette à l'adversaire de rejouer ensuite. Si ce n'est pas possible, la partie s'arrête. Celui qui devait jouer récupère les graines restantes et le joueur adverse reçoit une graine qu'il ajoute à sa propre récolte. La partie s'arrête dès qu'un joueur a gagné au moins 25 graines. Attention ! La partie est nulle si le jeu se trouve dans un processus récurrent ou s'il ne reste plus que six graines ou moins. Si la même configuration réapparaît de façon périodique ou s'il n'est plus possible de jouer, la partie est arrêtée et le joueur ayant le plus de graines est déclaré vainqueur.

2. Introduction à la complexité du jeu de l'awalé

On pourrait croire, du fait de la simplicité des règles que le jeu de l'awalé a un niveau de complexité relativement faible. L'ensemble de toutes les configurations possibles de ce jeu a finalement été trouvé. Ce nombre, très grand, est égal à 889 063 398 406. L'exécution du programme a pris près de 51 heures sur un cluster composé de 144 processeurs très puissants.

Essayons de définir cet ensemble de configurations sur un plateau plus restreint, un petit "Wari"[1] composé seulement de deux rangées de trois cases, avec deux graines par case, soit douze graines au total. Simplifions également les règles pour récolter les graines en utilisant seulement la règle suivante : un joueur ne pourra récolter que si le contenu de la case d'arrivée est composé de deux graines. Cela simplifie le calcul du nombre de configurations possibles car le nombre total de graines restera pair.

On peut trouver ce nombre en considérant que ce problème est équivalent à celui de placer cinq cloisons parmi les douze graines. Pour s'en rendre compte commençons par refaire notre décomposition. Pour simplifier nous considérons notre jeu d'awalé comme une structure à une ligne. Par exemple, la répartition (2, 1, 3, 5, 0, 1) des nombres de graines dans les six cases équivaut à la répartition suivante des douze graines et des cinq cloisons : o o | o | o o o | o o o o o | | o.

Le nombre de possibilités est, dans ce cas, égal à $\binom{17}{5}$ soit 6188.

Il faudrait faire la même chose dans le cas de dix graines, huit graines, et ceci jusqu'à trouver les possibilités de placer deux graines sur le plateau.

Le nombre total de possibilités pour ce mini awalé est égal à : $\sum_{i=0}^{5}\binom{17-2i}{5}$ = 11087.

Ce nombre augmente exponentiellement avec le nombre de cases d'un plateau. Comme on l'a vu auparavant, il est de l'ordre de mille milliards pour un jeu d'awalé classique. Comme le temps de réflexion accordé à chaque joueur est limité, il n'est pas possible, avec les moyens de calcul actuels, de déterminer en un temps raisonnable quel serait le meilleur coup pour une situation donnée.

3. Principe d'évaluation

Nous allons voir différentes versions de l'algorithme du Minimax afin de déterminer une solution, si possible gagnante, dans un temps adapté aux spécificités du jeu de l'awalé. Les algorithmes que nous allons voir dans ce dossier utilisent tous la notion d'arbre, la notion de parcours d'arbre. Un parcours d'arbre est une manière d'ordonner les nœuds d'un arbre afin de les parcourir. On distingue essentiellement deux types de parcours : les parcours en largeur et les parcours en profondeur. Chaque nœud et chaque feuille de l'arbre parcouru correspondra à une configuration du jeu, à une évaluation, à une décision d'un des deux joueurs. Les techniques d'évaluation sont différentes en fonction des parcours choisis.

On peut faire une distinction entre une fonction d'évaluation *stratégique* (appliquée uniquement aux feuilles) qui sera celle qui est considérée dans les algorithmes de ce dossier (fonction `eval`), et une fonction d'évaluation *tactique* (appliquée aux nœuds). Dans le cas de l'awalé, on peut imaginer une fonction d'évaluation stratégique qui évalue les propriétés

[1] Il y a principalement deux versions physiques de jeux d'awalé : la première nommée "Wari" avec deux rangées de cases et la seconde nommée "Solo" avec trois ou quatre rangées de cases.

intrinsèques d'une configuration "terminale" (ses points faibles : des cases avec une ou deux graines ; ses points forts : avoir plus de graines que l'adversaire pour limiter ses choix ; avoir une case avec beaucoup de graines pour pouvoir faire plus d'un tour de plateau et être en mesure de semer chez l'adversaire au premier tour et récolter sa propre semaille au second tour). Mais il est également très important de prendre en compte tout simplement le nombre de graines gagnées, car c'est quand même sur ce critère que l'on détermine le gagnant (contrairement aux échecs où l'on peut perdre beaucoup plus de pièces que l'adversaire, mais quand même mater ce dernier). Il faut donc, pour le cas du jeu de l'awalé, prendre impérativement en compte une fonction d'évaluation tactique qui compte simplement à chaque coup la différence entre le nombre de graines gagnées et le nombre de graines perdues Cette distinction entre évaluation stratégique des feuilles et évaluation tactique des nœuds de calcul des gains immédiats est fondamentale.

4. L'algorithme du Minimax

a) Introduction

L'algorithme du Minimax, élaboré par *John von Neumann* en 1928, est un procédé dédié initialement à la théorie des jeux. Il peut s'appliquer à des jeux comme le Tic-tac-toe, les échecs, le Go, l'awalé …

L'algorithme du Minimax s'applique aux jeux appelés "jeux à deux joueurs de somme nulle". En économie et en sociologie cette notion de jeux à somme nulle pour des personnes ou des entités est importante et s'étend bien au-delà du monde des jeux (par exemple, dans le cadre du jeu de l'awalé, le joueur recevra la valeur 1 s'il est le vainqueur, –1 s'il est le perdant et 0 en cas de partie nulle, la somme étant bien toujours nulle).

L'algorithme du Minimax est utilisé dans le cadre d'une partie entre deux personnes ou d'une personne contre un ordinateur. Par hypothèse une partie comportera un nombre fini de coups et à chaque étape un nombre fini de coups possibles (en conséquence, à chaque étape e, il existe un nombre fini $N(e)$ tel que toute partie commençant à partir de e comporte au plus $N(e)$ coups). Dans ce type de jeu, chaque joueur connaît les gains ou les pertes résultant de ses actions ou de celles de son adversaire ainsi que, le plus souvent, les motivations et les stratégies de ce dernier.

L'idée de l'algorithme de *John von Neumann* est relativement simple. À chacune des étapes de la partie il faut calculer tous les coups possibles en leur définissant une valeur, un gain, prenant en compte les avantages pour le joueur et ceux de son adversaire. Le meilleur choix sera celui qui minimise les gains de son adversaire et maximise ceux du joueur.

Le principe de cet algorithme est de construire un arbre dont la racine est la configuration du jeu à partir de laquelle un joueur X devra décider du coup suivant. Chaque coup potentiel est un nœud fils de cette racine, et c'est l'évaluation des nœuds fils qui permet de choisir le meilleur coup :, le joueur X va choisir le coup qui MAXimise ses gains. A l'inverse, sur chaque nœud fils, *N*, de la racine, l'adversaire,Y, va appliquer une stratégie analogue, mais

opposée en évaluant les nœuds fils de *N* et en choisissant celui qui MINimise les gains du joueur *X*. La répétition récursive de cette stratégie donne lieu à la construction d'un arbre dont les nœuds sont toutes les configurations de jeu possible. En résumé, le joueur *X* maximise les minima de gain calculés par le joueur *Y*, et le joueur *Y*, quand il joue après *X*, minimise les maxima calculés par *X*.

Attention ! Cette manière de procéder n'est pas très réaliste dès que le nombre de coups à prévoir est conséquent. L'arbre généré, dans lequel les nœuds représentent les états du jeu et les arcs, les coups possibles, est souvent trop important pour pouvoir être intégralement parcouru dans un temps raisonnable. La plupart des jeux comme le jeu d'échecs, le jeu de Go, le jeu de l'awalé, le tic-tac-toe excepté, sont dans cette situation.

b) Définitions

Soit E l'ensemble des configurations possibles du jeu. Cela correspond à l'ensemble des feuilles et des nœuds de l'arbre évoqué dans la section précédente. On notera $S(e)$ l'ensemble des configurations que l'on peut obtenir en jouant un coup à partir de la configuration e. Pour le jeu de l'awalé la cardinalité de $S(e)$ sera au plus égale à 6. Une configuration e est dite terminale si $S(e) = \emptyset$.

Soient X et Y les deux joueurs, Y pouvant être un programme exécuté sur un ordinateur. À chaque étape du jeu e est associée la valeur $J(e)$ qui vaut X si le coup précédent a été joué par Y, et Y si le coup précédent a été joué par X. On convient que $J(e_0) = X$ où e_0 est l'état initial de la partie.

Une condition suffisante pour que la partie se termine est $S(e) = \emptyset$.

On définit une fonction d'évaluation stratégique des configurations terminales, *eval*,

On définit également une fonction d'évaluation tactique des configurations non terminales, nommée *calcul*. Cette fonction, appliquée à une configuration non terminale, effectue un calcul entre le gain immédiat du plateau et la valeur retournée par la fonction Minimax sur les fils de cette configuration.

Enfin, l'algorithme 1, ci-dessous, présente la fonction *noeud_choisi*, qui utilise la fonction *Minimax* pour déterminer, en fonction du joueur, quel sera le coup choisi.

```
fonction noeud_choisi (e)
debut
  si  X = J(e) alors
      retourner x' tel x' ∈ S(e) et MiniMax(x') est maximal ;
sinon
      retourner y' tel que y' ∈ S(e) et MiniMax(y') est minimal ;
finfonction
```

Algorithme 1

c) Principe de l'algorithme

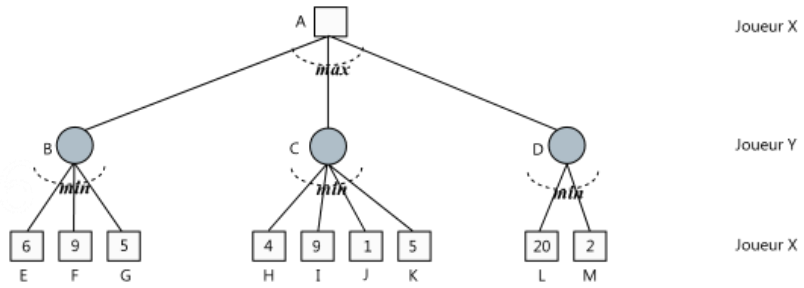

Figure 1

Le schéma de la *Figure 1* représente l'arbre de recherche Minimax pour une configuration donnée. C'est au tour du joueur `X` de jouer, et il doit évaluer la valeur du nœud racine, noté `A`, de cet arbre en choisissant le meilleur coup entre ses nœuds fils notés `B`, `C` et `D` : pour cela, le joueur `X` choisira la valeur maximale des évaluations des nœuds `B`, `C` et `D`. Les valeurs des nœuds `B`, `C` et `D` sont établies par le joueur `Y` qui aura choisi pour chacun d'eux la valeur minimale de tous ses nœuds fils (dans cet exemple, des feuilles de l'arbre). Le joueur `X` essaye de choisir le coup qui va maximiser ses gains alors que le joueur `Y` va à l'inverse essayer de minimiser les gains de `X`.

La plus grande valeur est située au nœud `L` mais ce n'est pas cette valeur qui sera ramenée au nœud `A`. En effet, si le joueur `X` joue le coup `D`, le joueur `Y` jouera le coup `M`, et l'évaluation du nœud `D` sera donc seulement égale à 2. Si le joueur `X` joue le coup `C`, le joueur `Y` jouera le coup `J`, et l'évaluation du nœud `C` sera égale à 1. En revanche, Si le joueur `X` joue le coup `B`, le joueur `Y` jouera le coup `G`, et l'évaluation du nœud `B` sera égale à 5. Du fait de la stratégie du joueur `Y` c'est donc la valeur 5 du nœud `G` qui est remontée au nœud `A` (voir *Figure 2*).

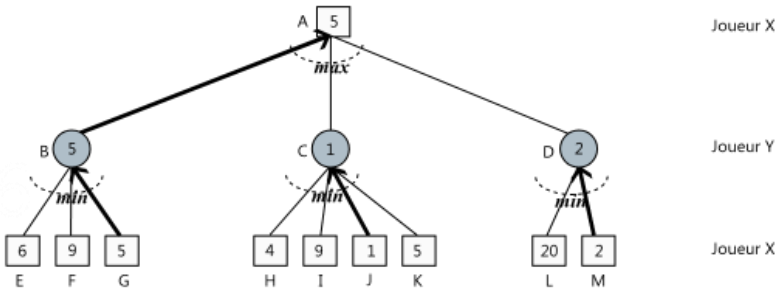

Figure 2

La fonction `Minimax` peut donc être écrite de manière récursive comme ci-dessous :

```
fonction  Minimax(e)          //  Minimax brute : parcours en profondeur
debut
  si e est terminal alors
    retourner eval(e)
  sinon
    si J(e) = X alors       // On calcule la valeur max des successeurs de e
```

```
      inter  ← -∞ ;
      pour s ∈ S(e) faire
        inter = max(inter,  Minimax (s)) ;
      finpour
      retourner calcul(inter, gain(e,X) ;
    sinon                  // On calcule la valeur min des successeurs de e
     inter  ← +∞ ;
      pour s ∈ S(e) faire
        inter = min(inter,  Minimax (s)) ;
      finpour
      retourner calcul(inter, gain(e,Y) ;
    finsi
  finsi
finfonction
```

Algorithme 2

Attention ! La fonction `calcul` n'est pas une évaluation stratégique de la configuration comme la fonction `eval`, mais prend en compte simplement les gains (ou pertes) immédiats liés au coup qui a amené à cette configuration, par exemple, dans le cas du jeu de l'awalé, le nombre de graines gagnées par la configuration `e`.

d) Minimax à profondeur donnée

Ce n'est que pour les jeux très simples que la construction de l'arbre peut être effectuée dans un temps acceptable du point de vue de la durée d'exécution.

La construction de l'arbre sera limitée à une profondeur de recherche donnée *p*, ce qui aura pour conséquence que la fonction *eval* sera appliquée sur des configurations non terminales. Il est clair que plus le déclenchement du calcul de la fonction d'évaluation est loin de la racine, c'est-à-dire donnant la possibilité d'examiner le plus de coups successifs, meilleur sera le résultat. Par contre, il ne sera pas possible de dire avec certitude que le nœud de l'arbre choisi correspond à une configuration gagnante ou à une configuration perdante quelle que soit la valeur attribuée à ce nœud de l'arbre.

```
fonction  Minimax-profix(e, p)     //  Minimax à profondeur fixée
debut
  si e est terminal ou p = 0 alors
      retourner eval(e) ;
  sinon
    si J(e) = X alors    // On calcule la valeur max des successeurs de e
      inter  ← -∞ ;
      pour s ∈ S(e) faire
        inter = max(inter,  Minimax-profix(s , p-1)) ;
      finpour
      retourner calcul(inter, gain(e,X) ;
    sinon   // On calcule la valeur min des successeurs de e
     inter  ← +∞ ;
      pour s ∈ S(e) faire
        inter = min(inter,  Minimax-profix(s, p-1))
      finpour
      retourner calcul(inter, gain(e,Y) ;
    finsi
  finsi
finfonction
```

Algorithme 3

e) L'algorithme NegaMax

Il est possible d'améliorer l'algorithme du Minimax. Sur l'algorithme précédent nous avons deux séquences de fonctions `Minimax` ou `Minimax-profix` qui font quasiment la même chose.

Comme la fonction *`eval`*, appliquée aux descendants d'un nœud, est identique pour les deux joueurs on peut faire le constat que minimiser une valeur revient à maximiser l'opposé de celle-ci.

En conséquence l'algorithme NegaMax n'est pas vraiment un algorithme différent : c'est juste une mise en œuvre qui fusionne les parties calcul du maximum et du minimum.

```
fonction NegaMax(e, p, profondeur)     //  Minimax à profondeur fixée
debut
  si e est terminal ou (profondeur = p) alors
      si profondeur est pair alors retourner eval(e)
      sinon retourner –eval(e)
      finsi
  sinon
      val  ← -∞ ;
      pour s ∈ S(e) faire
        val = max(val, -NegaMax(s , p, profondeur + 1 )) ;
      finpour
      retourner calcul(val, gain(e,J(e))) ;
  finsi
finfonction
```

Algorithme 4

5. Awalé et Minimax

Dans le cas du Minimax la fonction calcul pour l'awalé est relativement simple, elle sera déterminée par la différence entre les graines gagnées par le joueur et celles gagnées par l'adversaire. Par contre il ne faudra pas oublier d'indiquer le numéro du plateau dont est issue la valeur max ou min calculée.

Le schéma de la *`figure 3`* montre un petit exemple mettant en œuvre cet algorithme pour une profondeur égale à deux à partir d'une configuration dans laquelle c'est au tour de *`X`* de jouer. Le joueur *`X`* a choisi les cases de la partie supérieure de l'awalé.

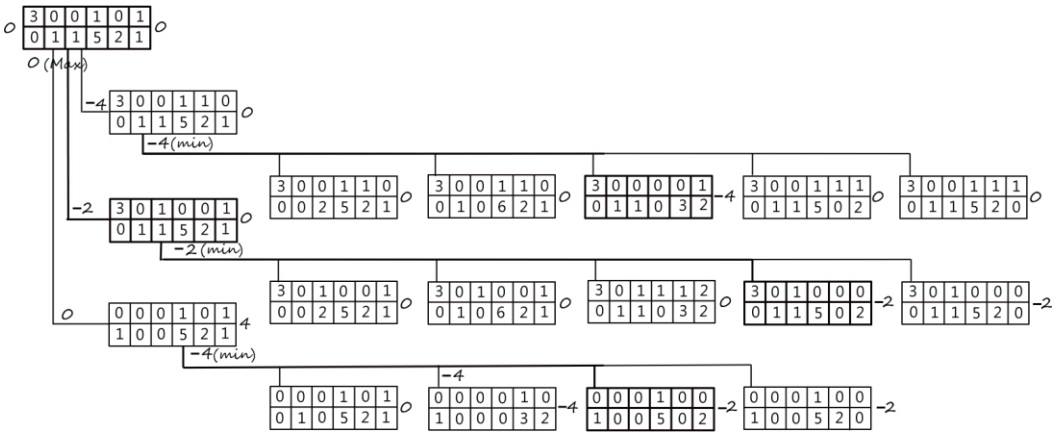

Figure 3 (version Minimax)

La convention des notations pour la `figure 3` est décrite sur le schéma `figure` 4.

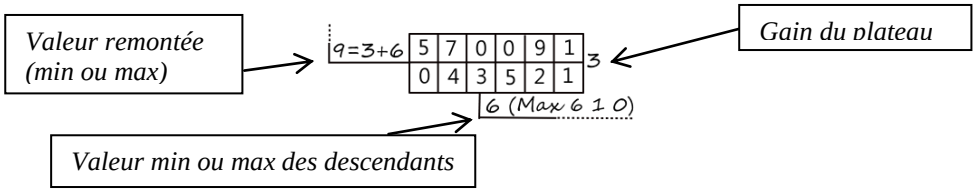

Figure *4*

Remarque : Le jeu de l'awalé se prête particulièrement bien au cas de l'algorithme NegaMax, puisque on peut observer que si *p* désigne le gain du plateau, *d* l'évaluation remontée par les descendants, il suffit que la valeur remontée soit $p - d$ (plutôt que $p + d$) pour qu'à chaque niveau de l'arbre on ne fasse systématiquement que des maximisations. En effet, le changement de signe des évaluations qui caractérise l'algorithme NégaMax est dû au signe "– " qui affecte *d* dans le calcul de la valeur remontée. Il n'est alors plus nécessaire d'affecter de signes positifs ou négatifs les gains de plateaux. Dans l'exemple, les petits-fils du fils du bas vont remonter $d = \max(0, 4, 2, 2) = 4$. L'évaluation du plateau du fils du bas étant égale à $p = 4$, il remontera donc $p - d = 4 - 4 = 0$. Le fils du milieu ayant un gain de plateau nul et ses fils lui remontant $\max(0, 0, 0, 2, 2) = 2$, il remontera $0 - 2 = -2$. Le fils du haut ayant un gain de plateau nul et ses fils lui remontant $\max(0, 0, 4, 0, 0) = 4$, il remontera $0 - 4 = -4$.

6. Complexité

Dans le cas de l'algorithme du Minimax, version basique, le nombre de nœuds et de feuilles est très important. Sa complexité en temps est en $O(b^N)$ ou b est le facteur de branchement, c'est-à-dire le nombre moyen de coups potentiel à chaque configuration et N la profondeur de l'arbre, et sa complexité en espace est en $O(b \times N)$.

Dans la théorie de la complexité des problèmes, les jeux sont en général d'une complexité exponentielle en temps et polynomiale en espace.

Voici un petit tableau donnant les résultats pour le jeu de l'awalé en considérant que pour un nœud de l'arbre le calcul s'effectue en 10^{-6} secondes sur un ordinateur actuel avec un processeur fonctionnant à 2 GHz avec 4 Go de mémoire.

N	5	7	10	15	20
Complexité en temps	0,01s	0,3s	60s	130h	116a

À titre de comparaison, pour le jeu d'échecs, où b est de l'ordre de 35 et N de l'ordre de 100, le nombre 35^{100} est gigantesque (10^{100} représente le nombre d'atomes de l'univers).

En ce qui concerne la complexité en espace si Z désigne la taille mémoire maximale pour stocker les informations relatives à un nœud en cours de traitement, du fait que le parcours de l'arbre des possibilités est effectué en profondeur d'abord, la complexité spatiale est $N \times Z$.

En conclusion, la forme brute de l'algorithme du Minimax ne peut pas s'appliquer ainsi dans la plupart des cas et certainement pas pour le jeu de l'awalé.

7. Élagage Alpha-Beta

Le problème du Minimax réside essentiellement dans la construction de l'arbre qui peut demander un temps très conséquent (plusieurs heures, plusieurs jours, …) même sur les ordinateurs les plus performants d'aujourd'hui.

Le principe de l'élagage Alpha-Beta est de réduire le nombre de nœuds parcourus par l'algorithme Minimax que ce soit pour sa forme brute ou celle limitée à une profondeur p. Nous nommerons Alpha-Beta l'algorithme Minimax avec l'élagage Alpha-Beta.

Lors du parcours de l'arbre, on peut remarquer qu'il est inutile de regarder certains nœuds et que l'on peut alors élaguer certaines branches de l'arbre tout en conservant le même résultat.

On choisit deux valeurs `alpha` et `beta` avec `alpha < beta`. `Alpha` contiendra la valeur minimale que le joueur peut espérer obtenir pour le coup suivant et `beta` la valeur maximale. La règle de l'élagage Alpha-Beta est de ne pas évaluer tous les nœuds dont les valeurs d'évaluation ne seront pas dans l'intervalle [`alpha`, `beta`]. Comme le montrent les algorithmes des fonctions récursives présentées précédemment, l'arbre de toutes les configurations est parcouru en profondeur d'abord. `Alpha` est alors défini comme la plus petite valeur que le joueur en situation `Max` peut obtenir compte tenu des nœuds déjà parcourus . Dans la figure 5, par exemple, après que sa branche de gauche a été parcourue, le nœud racine, de type `Max`, a l'assurance d'obtenir une évaluation au moins égale à `Alpha` = 9. Quand son fils droit, de type `Min`, récupère l'évaluation de son propre fils gauche, égale à 2, il sait qu'en évaluant ses autres fils il obtiendra une évaluation $e \leq 2$ et que, par conséquent, il est inutile de chercher à effectuer ces évaluations car, de toute façon, l'évaluation e étant inférieure `Alpha` = 9, elle ne sera pas retenue par le nœud racine, de type `Max`. De manière symétrique, `beta` est la valeur maximale que le joueur en situation `Min` autorisera `Max` à obtenir. La figure 6 montre que lorsque sa branche de gauche a été parcourue, le nœud racine,

de type *Min*, a l'assurance d'obtenir une évaluation au plus égale à *Beta* = 5. Quand son fils droit, de type *Max*, récupère l'évaluation de son propre fils gauche, égale à 9, il sait qu'en évaluant ses autres fils il obtiendra une évaluation $e \geq 9$ et que, par conséquent, il est inutile de chercher à effectuer ces évaluations car, de toute façon, l'évaluation *e* étant supérieure *Beta* = 5, elle ne sera pas retenue par le nœud racine, de type *Min*.

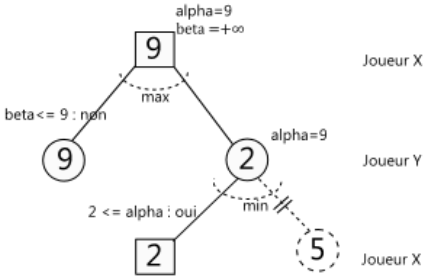

Figure 5 : Coupure Alpha

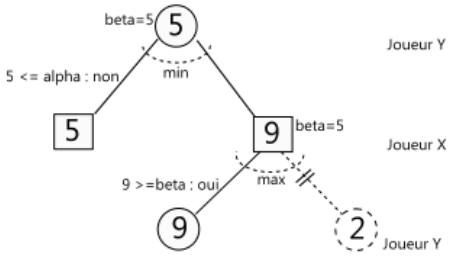

Figure 6 : Coupure Beta

Définition des valeurs *alpha* et *beta*

Si *e* est une configuration de *E* et *S(e)* l'ensemble des configurations jouables à partir de *e* :

- Pour un nœud *Min*, $s \in S(e)$, *alpha* est égale à la plus grande valeur déjà calculée de tous les nœuds *Max* ancêtres de *s*. On ne s'intéressera plus aux nœuds dont la valeur est inférieure ou égale à *alpha*. La valeur *alpha* sera initialisée à $-\infty$.
- Pour un nœud *Max*, $s \in S(e)$, *beta* est égale à la plus petite valeur déjà calculée de tous les nœuds *Min* ancêtres de *s*. On ne s'intéressera plus aux nœuds dont la valeur est supérieure ou égale à *beta*. La valeur beta sera initialisée à $+\infty$.

Algorithme Alpha-Beta avec profondeur :

```
fonction AlphaBeta(e, p, alpha, beta)     // p la profondeur de recherche
debut
  si e est terminal ou p = 0 alors
      retourner eval(e) ;
  sinon
    si J(e) = X alors    // On calcule la valeur max des successeurs de e
      inter ← -∞
      pour s ∈ S(e) faire
        inter = max(inter, AlphaBeta (s , p-1, alpha, beta)) ;
        si calcul(inter, gain(e,X)>= beta alors   // coupure beta
          retourner inter ;
        sinon
          alpha = max (alpha, inter) ;
        finsi
      finpour
      retourner calcul(inter, gain(e,X) ;
    sinon   // On calcule la valeur min des successeurs de e
      inter ← +∞ ;
      pour s ∈ S(e) faire
        inter = min(inter, AlphaBeta (s , p-1, alpha, beta)) ;
        si calcul(inter, gain(e,Y)<= alpha alors    // coupure alpha
          retourner inter ;
        sinon
```

```
          beta = min (beta, inter) ;
        finsi
      finpour
      retourner calcul(inter, gain(e,Y) ;
   finsi
  finsi

finfonction
```

Algorithme 5

Algorithme NegaBeta avec profondeur :

```
fonction NegaBeta(e, p, alpha, beta)
debut
inter, meilleur : info ;
  si e est terminal ou (profondeur = p) alors
      si profondeur est pair alors retourner eval(e)
      sinon retourner –eval(e)
      finsi
  sinon
    meilleur ← -∞ ;
    pour s ∈ S(e) faire
        inter =  - NegaBeta (s , p-1, -beta, -alpha)) ;
        si inter > meilleur alors
           meilleur = inter ;
           si meilleur > alpha alors
              alpha = meilleur ;
              si Alpha >= beta alors
                 retourner meilleur
              finsi
           finsi
       finsi
    finpour
    retourner retourner calcul(meilleur, gain(e,J(e))) ;
  finsi
finfonction
```

Algorithme 6

Les deux algorithmes ci-dessus peuvent être utilisés sans faire référence à une profondeur définie initialement. Dans ce cas on aura un parcours complet de l'arbre à l'exception des branches supprimées à l'aide de la méthode d'élagage Alpha-Beta.

L'algorithme Minimax et la méthode d'élagage Alpha-Beta sont très souvent utilisés pour tous les jeux de réflexion tels que le Go**,** les échecs, l'awalé, l'Othello.

8. Programmation de l'awalé

La fonction `Negamax` peut être adaptée au cas du jeu d'awalé pour donner la fonction `AwaleNegamax` ci-dessous :

```
Constante
   profondeurMax : entier ;
Type
   Configuration : structure {plateau : tableau[12] d'entiers ;
                              gain : entier } ;
fonction AwaleNegamax(e:Configuration, profondeur :entier) : entier
debut
caseDepart, maximum, k, evaluationDescendants, valeurRemontee : entier ;
fils : Configuration ;
   si (profondeur = profondeurMax) alors retourner eval(e) ;
   si (profondeur est pair) alors caseDepart ← 0 sinon caseDepart ← 6 ;
   maximum ← -100 ;
   pour k = caseDepart à casedepart+5 faire
      si e.plateau[k] ≠ 0 alors
         fils ← creerConfigurationSuivante(e, k) ;
```

```
         evaluationDescendants ← AwaleNegamax(fils, profondeur+1) ;
// La ligne suivante correspond à la fonction calcul //
         valeurRemontee ← fils.gain – evaluationDescendants ;
         maximum ← max(valeurRemontee, maximum) ;
      finsi
   finpour
   si maximum = –100 alors retourner eval(e) sinon retourner maximum ;
finfonction
```

Algorithme 7

La fonction `creerConfigurationSuivante` retourne une configuration, c'est-à-dire une structure qui contient le nouveau plateau et le gain effectué. Ce gain doit être calculé en fonction du joueur, et c'est l'argument `k` qui permet de savoir quel joueur a joué.

Attention ! La fonction `calcul` n'est pas implémentée explicitement mais correspond au calcul de la différence du gain du plateau et de l'évaluation des descendants.

Cette fonction retourne, conformément au principe de l'algorithme `Negamax` le maximum de l'évaluation des fils, cette évaluation changeant de signe avec la parité de la hauteur du noeud traité. Ce changement de signe n'est pas explicite, mais apparaît implicitement, d'une part dans le signe "–" qui affecte la variable `evaluationDescendants` dans le calcul de `valeurRemontee` et d'autre part dans le fait que le calcul du gain du plateau est fait à chaque fois en fonction du joueur qui a joué le coup qui a amené à la configuration traitée (`fils.gain` est calculé par la fonction `creerConfigurationSuivante` qui sait quel joueur a joué le coup grâce à l'argument `k`).

Comme cette fonction retourne un entier, elle ne peut pas être directement appelée à partir de la racine, car sinon elle ne retournerait que l'évaluation du meilleur coup sans possibilité de savoir quel est ce coup. La solution consistant à faire retourner à la fonction `AwaleNegamax` une structure qui contiendrait l'évaluation maximale des fils et le numéro de la case du plateau associé à ce fils serait lourde car ce numéro n'est utile que pour le choix du coup, donc uniquement pour la racine, or la fonction est récursive … Il vaut donc mieux utiliser la fonction `noeudChoisi` qui appelle `AwaleNegamax` pour chaque fils de la racine et prend bien soin de mémoriser quelle est la case associée à la configuration fille la plus avantageuse.

```
fonction noeudChoisi (e:Configuration, caseDepart :entier) : entier
debut
caseDepart, maxi, k, kmax, evaluationDescendants, valeurRemontee : entier ;
fils : Configuration ;
   maxi ← –100 ;
   kmax ← –1 ;
   pour k = caseDepart à casedepart+5 faire
      si e.plateau[k] ≠ 0 alors
         fils ← creerConfigurationSuivante(e, k) ;
         evaluationDescendants ← AwaleNegamax(fils, 1) ;
// La ligne suivante correspond à la fonction calcul //
         valeurRemontee ← fils.gain – evaluationDescendants ;
         si valeurRemontee  > maxi alors
            maxi ← valeurRemontee ;
            kmax ← k ;
         finsi
      finsi
   finpour
   si kmax = –1 alors retourner "partie terminée" sinon retourner kmax ;
finfonction
```

Algorithme 8

9. Conclusion

Dans ce document, les différentes versions de l'algorithme du Minimax ont été présentées. L'algorithme NegaMax est le plus performant. Le jeu de l'awalé est un jeu d'une complexité intéressante. Aucun calculateur ne peut aujourd'hui fournir la meilleure solution gagnante en quelques secondes. Le "Cloud Computing" pourra, dans un futur plus ou moins proche, offrir sans doute des possibilités d'atteindre des profondeurs de recherche plus importante dans des temps d'exécution adaptés au jeu de l'awalé. L'algorithme NegaMax offre des résultats très acceptables pour un niveau de profondeur de 10. À cette profondeur tout joueur de niveau moyen, voire bon, est systématiquement battu. Il n'y a que les très bons joueurs qui peuvent espérer battre l'ordinateur avec cet algorithme.

Il est possible également d'emprunter d'autres voies. Plutôt que de tenter de trouver tous les coups possibles il est préférable de regarder s'ils donnent une bonne solution ou non. On utilise alors des méthodes, comme celle dite de Monte-Carlo, qui à l'aide de calcul de probabilités va repérer les meilleures stratégies à suivre. On peut également citer UCT qui est une approche probabiliste de Minimax qui donne d'excellents résultats.