RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Supérieure d'Informatique (ESI ex. INI)

CRP
Rapport du TP N°01

Conception et implémentation des améliorations augmentant les performances de la procédure MinMax utilisant les coupures alpha/bêta

Réalisé par :

- BOUAZIZ Sofiane
- OUSSAIDENE Smail

Groupe :

SID 01

Table des matières

1 Introduction

Minimax est un algorithme de prise de décision qui s'applique à la théorie des jeux, généralement utilisé dans les jeux à deux joueurs au tour par tour à somme nulle et à information complète. L'objectif de l'algorithme est de minimiser la perte maximum (c'est-à-dire dans le pire des cas).

Dans cet algorithme, un joueur est appelé le maximisateur, et l'autre joueur est le minimisateur. Si nous attribuons un score d'évaluation au plateau de jeu, un joueur essaie de choisir un état de jeu avec le score maximum, tandis que l'autre choisit un état avec le score minimum.

En d'autres termes, le maximisateur s'efforce d'obtenir le score le plus élevé, tandis que le minimisateur tente d'obtenir le score le plus bas en essayant de contrer les mouvements.

Il est basé sur le concept de jeu à somme nulle. Dans un jeu à somme nulle, le score total d'utilité est divisé entre les joueurs. Une augmentation du score d'un joueur entraîne une diminution du score d'un autre joueur. Ainsi, le score total est toujours égal à zéro. Pour qu'un joueur gagne, l'autre doit perdre. Des exemples de tels jeux sont le poker, les dames, le morpion et dans notre cas les échecs.

Une version améliorée nommée élagage alpha/bêta existe et qui permet d'éviter de parcourir tous les nœuds sans affecter le résultat final. Cependant, cette amélioration n'est pas suffisante, en effet, dans certains cas, la portion élaguée reste négligeable devant la portion parcourue.

Dans le cadre de ce TP, il nous est demandé d'améliorer l'algorithme Minmax avec l'élagage alpha/bêta en qualité et temps d'exécution.

2 Amélioration N°1 : Eviter le problème de l'effet horizon

2.1 Problème posé

Dans les jeux informatiques ou d'autres processus de recherche, un grand arbre de recherche doit être exploré. Il est habituel de fixer une limite de profondeur maximale (hauteur) au-delà de laquelle il est considéré comme non rentable de chercher plus loin.

Aux échecs, les méthodes usuelles d'exploration des arbres de décision par les programmes d'échecs définissent une profondeur maximale et produisent des résultats médiocres, voire catastrophiques, s'il se cache un phénomène important au-delà du dernier choix exploré par l'algorithme, et remettant en question celui-ci.

En effet, une recherche de jeu qui s'arrête à une profondeur fixe pose un problème. Dans une recherche d'échecs, par exemple, le dernier coup des blancs peut-être la dame prend le cavalier, et la fonction d'évaluation rapportera que les blancs sont en haut d'un cavalier. Si la recherche examinait un mouvement plus profond, elle verrait que les noirs ont la réponse le pion prend la reine et se rend compte que les noirs sont en train de gagner.

Un nom pour ce problème est l'effet d'horizon. Une recherche pleine largeur voit tout jusqu'à son horizon, et rien au-delà. Il ira avec plaisir pour les mouvements qui semblent bons à l'horizon, mais qui tournent vite mal, comme la reine prend l'exemple du chevalier ci-dessus.

2.2 Solution proposée

La recherche de quiescence est une recherche supplémentaire, commençant aux nœuds frontières instables de la recherche principale, qui tente de résoudre le problème de l'effet horizon. En principe, les recherches de quiescence devraient inclure tout mouvement susceptible de déstabiliser la fonction d'évaluation, s'il y a un tel mouvement, la position n'est pas stable. Aux échecs, les recherches de quiescence incluent généralement tous les mouvements tactiques, afin que ses échanges tactiques ne perturbent pas l'évaluation.

La recherche de quiescence est un algorithme généralement utilisé pour étendre la recherche aux nœuds instables dans les arbres de jeu minimax dans les programmes informatiques de jeu. C'est une extension de la fonction d'évaluation pour différer l'évaluation jusqu'à ce que la position soit suffisamment stable pour être évaluée.

Les joueurs humains ont généralement suffisamment d'intuition pour décider s'ils doivent abandonner un mouvement de mauvaise apparence ou rechercher un mouvement prometteur à une grande profondeur. Une recherche de quiescence tente d'imiter ce comportement en demandant à un ordinateur de rechercher les positions « instables » à une plus grande profondeur que les positions « stables » pour s'assurer qu'il n'y a pas de pièges cachés et pour obtenir une meilleure estimation de sa valeur.

Pour notre solution, nous avons implémenté la recherche de quiescence adaptée au jeu d'échecs. Nous allons présenter dans ce qui suit les différentes fonctions et procédures que nous avons implémentées.

2.3 Implémentation

2.3.1 configuration_echec

Cette fonction permet de vérifier si le roi du joueur 'mode' de la configuration 'conf' est menacé par le joueur adversaire. Si c'était le cas donc cette configuration représente une configuration d'échec pour le joueur 'mode'.

```
int configuration_echec(int mode, struct config *conf) {
     int i,j;

     if ( mode == MAX ) {
           i = conf->xrB;
           j = conf->yrB;
           return caseMenaceePar( MIN, i, j, conf);
     }
     else {
           i = conf->xrN;
           j = conf->yrN;
           return caseMenaceePar( MAX, i, j, conf);
     }
} // fin de configuration_echec
```

2.3.2 conf_stable

Une configuration 'conf' du joueur 'mode' est instable si :

- Le joueur ‘mode’ peut capturer une pièce du joueur adversaire à partir de la configuration ‘conf’

Ou bien

- La configuration ‘conf ‘ est une configuration d’échec pour le joueur ‘mode’

Cette fonction permet de vérifier si la configuration 'conf' du joueur 'mode' est stable ou pas.

```
int conf_stable(int mode, struct config *conf){
    //Parcour l'échiquier pour trouver les positions des pièces (i,j)
    int i,j;
    for (i=0; i<8; i++){
            for (j=0; j<8; j++) {
                if ( mode == MAX  && conf->mat[i][j] < 0 ){
                   // Verefier si le joueur MAX peut capturer une piece du
                      joueur MIN
                    if ( caseMenaceePar( MAX, i, j, conf ) ) return 0;
                }
                else {
                    if ( mode == MIN  && conf->mat[i][j] > 0 ){
                        // Verefier si le joueur MIN peut capturer une piece du
                           joueur MAX
                         if ( caseMenaceePar( MIN, i, j, conf ) ) return 0;
                    }
                }
            }
    }

    return (!configuration_echec(mode, conf));
} // fin de conf_stable
```

Remarque

Au début, nous avons utilisé ces deux critères pour définir une configuration instable :

- Le joueur ‘mode’ peut capturer une pièce du joueur adversaire à partir de la configuration ‘conf’

Ou bien

- La configuration ‘conf ‘ du joueur ‘mode’ est une configuration menacée par le joueur adversaire (au moins une pièce du joueur ‘mode’ de la configuration ‘conf’ est menacée par le joueur adversaire)

Mais en implémentant cette solution, On a remarqué que :

- Un grand surplus a été ajouté au temps d’exécution
- La recherche de l’arbre dans certains cas devient trop longue (presque tous les nœuds frontières de l’arbre deviennent instables) et le programme s’arrête brusquement

Pour remédier à ce problème nous avons restreint la deuxième condition d’instabilité des configurations sur seulement les configurations d’échec (Le roi du joueur ‘mode’ de la configuration ‘conf’ est menacé par le joueur adversaire). Après cette modification, le programme ne s’arrête plus brusquement et la recherche de l’arbre ne devient pas trop longue.

2.3.3 nb_pieces_config

Cette fonction permet de retourner le nombre de pièce du joueur 'mode' pour la configuration 'conf'.

```
int nb_pieces_config(int mode, struct config *conf)
{
    int i,j, nb = 0;

    //Parcour l'echiquer pour trouver les positions des pieces (i,j)
    for (i=0; i<8; i++){
            for (j=0; j<8; j++) {
                if ( mode == MAX  && conf->mat[i][j] > 0 ){
                        nb++;
                }
                else {
                    if ( mode == MIN  && conf->mat[i][j] < 0 ){
                         nb++;
                    }
                }
            }
    }
    return nb;
} //fin nb_pieces_config
```

2.3.4 configuration_capture

Cette fonction permet de vérifier si la configuration 'conf' du joueur 'mode' a subi une capture par le joueur adversaire en comparant le nombre de pièces de la configuration père du joueur ‘mode’ avec le nombre de pièces de la configuration actuelle ‘conf’ du même joueur.

```
int configuration_capture(int mode, struct config *conf_pere, struct config
*conf)
{
  return ( nb_pieces_config(mode, conf_pere) !=  nb_pieces_config(mode, conf) );
} //fin configuration_capture
```

2.3.5 generer_succ_tactiques

Pour la recherche de quiescence, si on utilise la procédure generer_succ pour générer les successeurs d’une configuration instable, ça serait trop coûteux en temps d’exécution et en espace mémoire. De ce fait, pour une configuration instable nous avons implémenté la procédure generer_succ_tacttiques qui nous permet de générer les successeurs tactiques d’une configuration instable.

Un successeur est tactique s’il :

- A permet au joueur ‘mode’ de capturer une pièce du joueur adversaire (cas de capture)

Ou bien

- A permet au joueur ‘mode’ de sortir d’une configuration d’échec. C’est-à-dire que la configuration ‘conf ‘ (père de ce successeur) était une configuration échec (Le roi du joueur ‘mode’ de la configuration ‘conf’ est menacé par le joueur

adversaire). et le choix de ce successeur nous a permis de sortir de cette situation d'échec (Sortie d'échec).

Cette fonction permet ainsi de générer les successeurs tactiques de la configuration 'conf' dans le tableau T et elle retourne aussi dans n le nombre de configurations filles générées.

```
void generer_succ_tactiques( struct config *conf, int mode, struct config T[],
int *n )
{
    struct config S[100];
    int m = 0;
    *n = 0;

    // generer les sucesseur de la configuration conf
    generer_succ( conf, mode, S, &m );

    for (int k=0; k < m; k++) {
          if ( ( !dejaVisitee(&S[k]) ) ) {
                if ( mode == MAX ) {
                     /* Si ce sucesseur permet au joueur MAX capturer une
                        configuration du joueur MIN
                        ou bien
                        la configuration 'conf' est une configuration d'echec
                        (Sortie d'echec)
                    */
                     if (configuration_capture(MIN, conf, &S[k]) ||
                         configuration_echec(MAX, conf)) {
                        copier(&S[k], &T[*n]);
                        *n = *n + 1;
                     }
                }
                else {
                    /* Si ce sucesseur permet au joueur MIN capturer une
                       configuration du joueur MAX
                       ou bien
                       la configuration 'conf' est une configuration d'echec
                       (Sortie d'echec)
                    */
                    if (configuration_capture(MAX, conf, &S[k]) ||
                        configuration_echec(MIN, conf)) {
                        copier(&S[k], &T[*n]);
                        *n = *n + 1;
                     }
                }
          }
    }
} // fin de generer_succ_tactiques
```

2.3.6 recherche_quiescence

Cette fonction permet d'implémenter l'algorithme de recherche de quiescence. Elle permettra donc d'étendre la recherche aux configurations instables dans les arbres de minimax. C'est une extension de la fonction d'évaluation pour différer l'évaluation jusqu'à ce que la position soit suffisamment stable pour être évaluée.

```
int recherche_quiescence( struct config *conf, int mode, int alpha, int beta,
int largeur, int numFctEst )
{
    int n, i, score, score2;
    struct config T[100];

    config_actuelle conf_act;

    // Si la configuration 'conf' est une feuille
    if ( feuille(conf, &score) ){
        return score;
    }

    // Si la configuration 'conf' est une configuration stable
    if ( conf_stable( mode, conf) ) {
               return Est[numFctEst]( conf );
    }

    // Pour le cas d'une configuration 'conf' instable
    if ( mode == MAX ) {

       // Generer les sucesseurs tactique de cette configuration instable
           generer_succ_tactiques( conf, MAX, T, &n );

           if ( largeur != +INFINI ) {
              for (i=0; i<n; i++)
                T[i].val = Est[numFctEst]( &T[i] );

              if ( largeur < n ) n = largeur;   // pour limiter la largeur
              d'exploration
           }

           if( n !=0 ){
            conf_act.conf = conf;
            conf_act.mode = mode;
            sort(T, n, sizeof(struct config), confcmp321_prom, (void *)
                 &conf_act);
           }

           score = alpha;

           for ( i=0; i<n; i++ ) {
                score2 = recherche_quiescence( &T[i], MIN, score, beta, largeur,
                                               numFctEst);
                if (score2 > score) score = score2;
                if (score >= beta) {
                        // Coupe Beta
                        nbBeta++;       // compteur de coupes beta
                        return score;   // evaluation tronquee ***
                }
           }
       }
    else  { // mode == MIN

       // Generer les sucesseurs tactique de cette configuration instable
       generer_succ_tactiques( conf, MIN, T, &n );

           if ( largeur != +INFINI ) {
                for (i=0; i<n; i++)
                     T[i].val = Est[numFctEst]( &T[i] );

                 if ( largeur < n ) n = largeur;        // pour limiter la
                 largeur d'exploration
           }
```

```
        if( n !=0 )
        {
             conf_act.conf = conf;
             conf_act.mode = mode;
             sort(T, n, sizeof(struct config), confcmp123_prom, (void *)
                  &conf_act);
        }

        score = beta;

        for ( i=0; i<n; i++ ) {

         score2 = recherche_quiescence( &T[i], MAX, alpha, score, largeur,
                                        numFctEst );
                if (score2 < score) score = score2;
                if (score <= alpha) {
                        // Coupe Alpha
                        nbAlpha++;      // compteur de coupes alpha
                        return score;    // evaluation tronquee ***
                }
           }
         }
         if ( score == +INFINI ) score = +100;
         if ( score == -INFINI ) score = -100;

         return score;

} // fin de recherche_quiescence
```

Remarque

Nous avons utilisé dans cette fonction, la procédure ***sort*** au lieu de ***qsort***. En effet, cette version de la fonction représente la version finale, elle comprend ainsi toutes les améliorations proposées et implémentées pour la question 1 et 2. Cette fonction sera plus expliquée dans la section *3.3.5 Fonction complète*.

2.3.7 minmax_ab

Pour la fonction minmax_ab nous l'avons modifié pour tenir compte de l'effet horizon. Nous avons donc rajouté la condition suivante :

```
if ( niv == 0 ){
   recherche_quiescence(conf, mode, alpha, beta, largeur, numFctEst);
}
```

2.4 Tests

Dans le but de tester notre solution, nous avons implémenté la recherche de quiescence dans la fonction estm7 et utilisé la fonction estm1 pour évaluer un nœud final (un nœud feuille ou bien un nœud stable), ainsi nous pourrons la comparer avec la fonction estm1 qui n'implémente pas la recherche de quiescence et donc elle ne tient pas en compte l'effet horizon.

Dans ce qui suit nous allons recenser les résultats des différentes parties d'échecs ou nous avons mis la fonction estm7 (joueur blanc) contre estm1 (joueur noir) pour tester l'efficacité de notre solution.

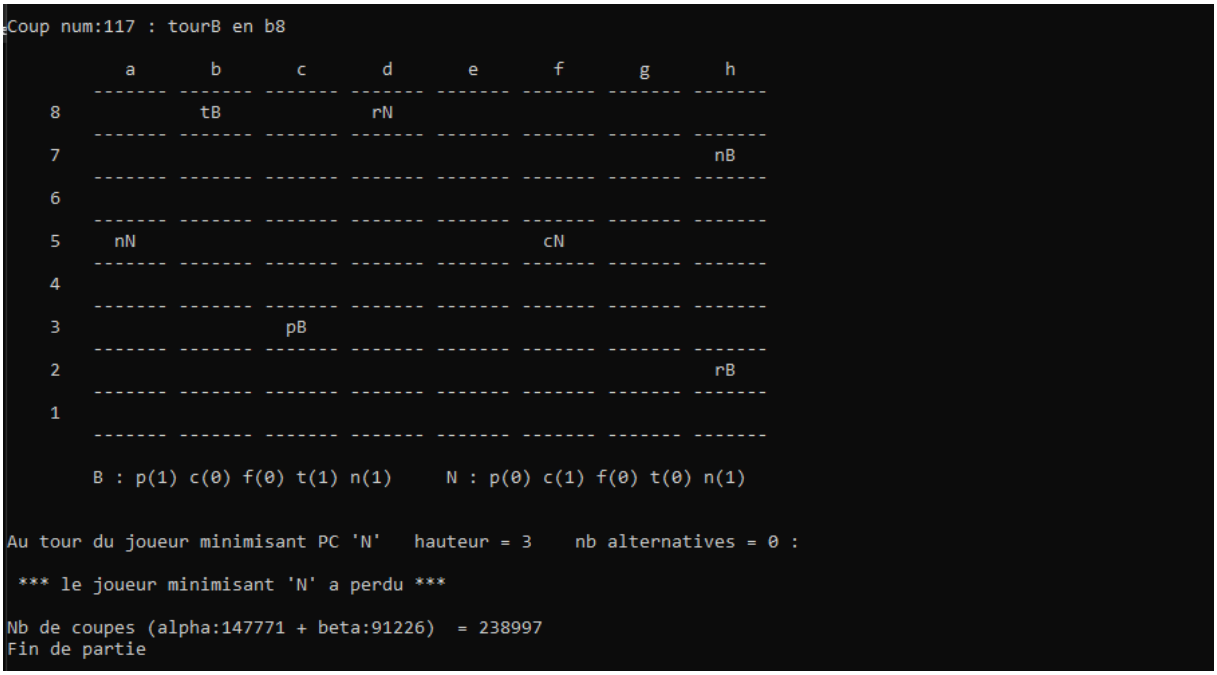
Profondeur \ Largeur	2	3	4	5	6	7
2	B gagnant	B gagnant	B gagnant	B gagnant	B gagnant	B gagnant
3	N gagnant	B gagnant	B gagnant	B gagnant	B gagnant	B gagnant
4	B gagnant	B gagnant	B gagnant	B gagnant	B gagnant	B gagnant
5	B gagnant	B gagnant	B gagnant	B gagnant	B gagnant	Prend trop de temps
6	B gagnant	B gagnant	B gagnant	B gagnant	B gagnant	B gagnant
7	B gagnant	B gagnant	B gagnant	B gagnant	Prend trop de temps	B gagnant

Nous constatons que **34** des parties jouées (36) arrivent à une fin contre 2 parties qui prennent prend trop de temps, et que le joueur utilisant l'estimation 07 gagne dans **97%** des parties finies.

Nous justifiant cela par le fait que la recherche de quiescence nous permis de donner une meilleure estimation à un nœud.

Ci-dessous quelques captures d'écran des résultats :

Profondeur = 3 et largeur = 3 :



Profondeur = 6 et largeur = 6 :

```
Coup num: 97 : reineB en d4

          a       b       c       d       e       f       g       h
    ------- ------- ------- ------- ------- ------- ------- -------
    8     tB
    ------- ------- ------- ------- ------- ------- ------- -------
    7
    ------- ------- ------- ------- ------- ------- ------- -------
    6                                                     pN
    ------- ------- ------- ------- ------- ------- ------- -------
    5             pN
    ------- ------- ------- ------- ------- ------- ------- -------
    4                             nB
    ------- ------- ------- ------- ------- ------- ------- -------
    3             fB      rN      pN      rB
    ------- ------- ------- ------- ------- ------- ------- -------
    2                     pB
    ------- ------- ------- ------- ------- ------- ------- -------
    1
    ------- ------- ------- ------- ------- ------- ------- -------

        B : p(1) c(0) f(1) t(1) n(1)     N : p(3) c(0) f(0) t(0) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 6    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:678667 + beta:2285989)  = 2964656
Fin de partie
```

Profondeur = 7 et largeur = 7 :

```
Coup num: 89 : tourB en e7

          a       b       c       d       e       f       g       h
    ------- ------- ------- ------- ------- ------- ------- -------
    8                                     tN
    ------- ------- ------- ------- ------- ------- ------- -------
    7                             tB                      pN
    ------- ------- ------- ------- ------- ------- ------- -------
    6                             rN      cN
    ------- ------- ------- ------- ------- ------- ------- -------
    5                                     cB
    ------- ------- ------- ------- ------- ------- ------- -------
    4                             pB                      pB
    ------- ------- ------- ------- ------- ------- ------- -------
    3                     rB              pB
    ------- ------- ------- ------- ------- ------- ------- -------
    2                     pB
    ------- ------- ------- ------- ------- ------- ------- -------
    1
    ------- ------- ------- ------- ------- ------- ------- -------

        B : p(4) c(1) f(0) t(1) n(0)     N : p(1) c(1) f(0) t(1) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 7    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:1021976 + beta:812728)  = 1834704
Fin de partie
```

Remarque

On remarque que plus on augmente les paramètres d'entrées (profondeur et largeur) le nombre de coups déminue, ce qui affirme que les coups sont de plus en plus calculés (de plus en plus intelligents). De plus on remarque que le temps d'exécution augmente en prenant en compte l'effet horizon, cette inconvenant sera contourné dans *l'amélioration 02*.

Profondeur = 3 et largeur = 2 :

```
Coup num:206 : reineN en h6

          a       b       c       d       e       f       g       h
       ------- ------- ------- ------- ------- ------- ------- -------
    8                                          nN      cN      rN
       ------- ------- ------- ------- ------- ------- ------- -------
    7
       ------- ------- ------- ------- ------- ------- ------- -------
    6                                                          nN
       ------- ------- ------- ------- ------- ------- ------- -------
    5
       ------- ------- ------- ------- ------- ------- ------- -------
    4
       ------- ------- ------- ------- ------- ------- ------- -------
    3
       ------- ------- ------- ------- ------- ------- ------- -------
    2
       ------- ------- ------- ------- ------- ------- ------- -------
    1                                                  rB
       ------- ------- ------- ------- ------- ------- ------- -------

       B : p(0) c(0) f(0) t(0) n(0)     N : p(0) c(1) f(0) t(0) n(2)


Au tour du joueur maximisant PC 'B'   hauteur = 3    nb alternatives = 0 :

 *** le joueur maximisant 'B' a perdu ***

Nb de coupes (alpha:4122 + beta:3471)  = 7593
Fin de partie
```

Remarque

On remarque que la partie a pris 206 coups, ce qui montre que la partie a été serrée, de plus les blancs ont perdu avec qu'un roi en fin de partie « bare king » (les noirs n'ont pas gagné facilement).

3 Amélioration N°2 : Ordonnancement des successeurs d'une configuration donnée

3.1 Problème posé

L'avantage de l'élagage alpha-bêta réside dans le fait que les branches de l'arbre de recherche peuvent être éliminées. De cette façon, le temps de recherche peut être limité au sous-arbre le plus prometteur et une recherche plus profonde peut être effectuée dans le même temps.

La quantité d'élagage et de ce fait la taille de l'arbre de recherche dépend fortement de l'ordre d'évaluation des différents coups possibles, il est donc logique de les ordonner de manière que les coups les plus prometteurs soient recherchés en premier. Ceci est particulièrement important à la racine lorsqu'une coupure évite de chercher dans une grande partie de l'arbre.

En effet avec un facteur de branchement[1] (moyen ou constant) de b, et une profondeur de recherche de d plis, le nombre maximal de positions de nœuds feuilles évaluées (lorsque l'ordre de déplacement est pessimal) est de $O(b^d)$ identique à une recherche minimax simple. Cependant, si l'ordre des déplacements pour la recherche est optimal (ce qui signifie que les meilleurs déplacements sont toujours recherchés en premier), le nombre de positions de nœuds feuilles évaluées est d'environ $O(b^{d/2})$. Dans ce cas, le facteur de branchement effectif est réduit à sa racine carrée, ou, de manière équivalente, la recherche peut aller deux fois plus loin avec la même quantité de calcul.

Il est important de souligner que l'algorithme alpha-bêta avec ordonnancement des déplacements possibles produit le même résultat que l'algorithme minimax simple mais, dans de nombreux cas, il est plus rapide, car il ne cherche pas dans les branches non pertinentes.

3.2 Solution proposée

Afin d'ordonner les différents coups possibles, nous avons fait face à deux méthodes :

1. À l'aide d'une connaissance du domaine, construire une hiérarchie des coups possibles, dans notre cas du jeu d'échecs une capture est plus prometteuse qu'un simple déplacement par exemple.
2. Utiliser iterative deepening, afin d'utiliser les meilleurs coups de l'itération précédente pour ordonner les coups de l'itération actuelle.

Dans ce qui nous allons se focaliser sur la première méthode.

Dans le but de comparer entre deux configurations, nous avons dû dessiner une hiérarchie ordonnée du plus prometteur au moins prometteur comme suit :

1. Capture du roi du joueur adversaire (échec et mat)
2. Capture d'une pièce autre que le roi du joueur adversaire
3. Menace du roi du joueur adversaire (échec)
4. Transformation d'un pion du joueur courant vers
    a. Une reine
    b. Une toure
    c. Un cavalier
    d. Un fou

[1] Le facteur de branchement est le nombre de fils à chaque nœud

3.3 Implémentation

Sur la version initiale du code, une fonction de comparaison est implémentée, mais celle-ci prend en considération la valeur d'estimation de la configuration seulement :

```
int confcmp123(const void *a, const void *b)       // pour l'ordre croissant
{
    int x = ((struct config *)a)->val, y = ((struct config *)b)->val;
    if ( x < y )
         return -1;
    if ( x == y )
         return 0;
    return 1;
}  // fin confcmp123
```

3.3.1 Capture du roi

Pour savoir si le roi a été capturé dans une configuration, nous avons implémenté une fonction intitulée *configuration_capture_roi*, où nous avons récupéré la position du roi et vérifié si elle est différente de (-1,-1)

```
int configuration_capture_roi(int mode, struct config *conf) {
    if (mode == MAX){
        return ((conf->xrB == -1) && (conf->yrB == - 1));
    }
    else {
        return ((conf->xrN == -1) && (conf->yrN == - 1));
    }
}
```

Cette fonction sera utilisée comme suit dans le cas d'ordonnancement des successeurs d'un nœud maximisant en ordre croissant :

```
// Cas 1 : roi du mode adversaire est capturé //
int roi_capture_A = configuration_capture_roi(MIN, conf1);
int roi_capture_B = configuration_capture_roi(MIN, conf2);
if (!roi_capture_A && roi_capture_B) return 1;
if (roi_capture_A && !roi_capture_B) return -1;
```

Si le roi du joueur adversaire de chaque configuration est capturé ou le contraire (le roi du joueur adversaire de chaque configuration n'est pas capturé), alors nous devons vérifier les conditions qui suivent.

3.3.2 Capture d'une pièce autre que roi

Pour savoir si une pièce a été capturée dans une configuration, nous avons implémenté une fonction intitulée *configuration_capture* présentée précédemment dans la section *2.3.4 configuration_capture.*

Cette fonction sera utilisée comme suit dans le cas d'ordonnancement des successeurs d'un nœud maximisant en ordre croissant :

```
// Cas 2 : une piece du mode adversaire est capturée //
int conf_capture_A = configuration_capture(MIN, conf, conf1);
int conf_capture_B = configuration_capture(MIN, conf, conf2);
if (!conf_capture_A && conf_capture_B) return 1;
if (conf_capture_A && !conf_capture_B) return -1;
```

Si aucune pièce du joueur adversaire n'a été capturée dans les deux configurations, ou au moins une dans chacune alors nous devons vérifier les conditions qui suivent.

3.3.3 Menace du roi

Pour savoir si le roi est menacé dans une configuration (échec), nous avons implémenté une fonction intitulée *configuration_echec* présentée précédemment.

Cette fonction sera utilisée comme suit dans le cas d'ordonnancement des successeurs d'un nœud maximisant en ordre croissant :

```
// Cas 3 : le roi du adversaire est menacé //
int roi_menace_A = configuration_echec(MIN, conf1);
int roi_menace_B = configuration_echec(MIN, conf2);
if (!roi_menace_A && roi_menace_B) return 1;
if (roi_menace_A && !roi_menace_B) return -1;
```

Si le roi du joueur adversaire de chaque configuration est menacé ou le contraire, alors nous devons vérifier la condition qui suit.

3.3.4 Transformation d'un pion

Pour savoir si un pion s'est transformé vers une autre pièce (reine, toure, cavalier ou fou) dans une configuration, nous avons implémenté une fonction intitulée *configuration_pion_transforme*, où on vérifie si le nombre de pion a déminué de 1, dans ce cas il faut vérifier de plus si le nombre d'une des autres pièces a augmenté de 1. Pour cela nous avons aussi implémenté la fonction *nb_config_piece* qui calcule le nombre d'occurrence d'un type de pièce.

```
int nb_config_piece(int mode, struct config *conf, char piece)
{
    int i,j, nb = 0;

    //Parcour l'echiquer pour trouver les positions des pieces (i,j)
    for (i=0; i<8; i++){
            for (j=0; j<8; j++) {
                if ( mode == MAX  && conf->mat[i][j] > 0
                     && conf->mat[i][j] == piece){
                        nb++;
                }
                else {
                    if   (  mode   ==   MIN     &&   conf->mat[i][j]   <   0
                          && (-conf->mat[i][j]) == piece){
                         nb++;
                    }
                }
            }
    }
    return nb;
}
```

```
int configuration_pion_transforme(int mode, struct config *conf_pere,
struct config *conf){
    if(nb_config_piece(mode, conf_pere, 'p') == (nb_config_piece(mode,
conf, 'p') + 1 ))
    {
        if( nb_config_piece(mode, conf, 'n') == ( nb_config_piece(mode,
conf_pere, 'n') + 1) ) return 4;
        if( nb_config_piece(mode, conf, 't') == ( nb_config_piece(mode,
conf_pere, 't') + 1) ) return 3;
        if( nb_config_piece(mode, conf, 'c') == ( nb_config_piece(mode,
conf_pere, 'c') + 1) ) return 2;
        if( nb_config_piece(mode, conf, 'f') == ( nb_config_piece(mode,
conf_pere, 'f') + 1) ) return 1;
    }
    return 0;
}
```

Cette fonction sera utilisée comme suit dans le cas d'ordonnancement des successeurs d'un nœud maximisant en ordre croissant :

```
// Cas 4 : la transformation d'un pion //
int pion_transform_A = configuration_pion_transforme(MAX, conf, conf1);
int pion_transform_B = configuration_pion_transforme(MAX, conf, conf2);
if (pion_transform_A < pion_transform_B) return 1;
if (pion_transform_A > pion_transform_B) return -1;
```

Si aucune transformation du pion du joueur courant ou la même s'est survenue dans les deux configurations on passera à la comparaison de la valeur d'estimation :

```
return confcmp321(a,b);
```

3.3.5 Fonction complète

```
#ifdef _WIN32
    int confcmp321_prom(void * arg, const void *a, const void *b)
#elif linux
    int confcmp321_prom(const void *a, const void *b, void * arg)
#endif // _WIN32
{
    struct config *conf1 = (struct config *)a, *conf2 = (struct config
*)b;
    config_actuelle *conf =  (config_actuelle *)arg;
    if (conf->mode == MAX ){

        // Cas 1 : roi du mode adversaire est capturé //
        int roi_capture_A = configuration_capture_roi(MIN, conf1);
        int roi_capture_B = configuration_capture_roi(MIN, conf2);
        if (!roi_capture_A && roi_capture_B) return 1;
        if (roi_capture_A && !roi_capture_B) return -1;

        // Cas 2 : une piece du mode adversaire est capturée //
        int conf_capture_A = configuration_capture(MIN, conf, conf1);
        int conf_capture_B = configuration_capture(MIN, conf, conf2);
        if (!conf_capture_A && conf_capture_B) return 1;
        if (conf_capture_A && !conf_capture_B) return -1;


        // Cas 3 : le roi du adversaire est menacé //
        int roi_menace_A = configuration_echec(MIN, conf1);
        int roi_menace_B = configuration_echec(MIN, conf2);
        if (!roi_menace_A && roi_menace_B) return 1;
        if (roi_menace_A && !roi_menace_B) return -1;

        // Cas 4 : la transformation d'un pion //
        int pion_transform_A = configuration_pion_transforme(MAX, conf,
        conf1);
        int pion_transform_B = configuration_pion_transforme(MAX, conf,
        conf2);
        if (pion_transform_A < pion_transform_B) return 1;
        if (pion_transform_A > pion_transform_B) return -1;

        return confcmp321(a,b);
    }
    else{
        // Cas 1 : roi du mode adversaire est capturé //
        int roi_capture_A = configuration_capture_roi(MAX, conf1);
        int roi_capture_B = configuration_capture_roi(MAX, conf2);
        if (!roi_capture_A && roi_capture_B) return 1;
        if (roi_capture_A && !roi_capture_B) return -1;

        // Cas 2 : une piece du mode adversaire est capturé //
        int conf_capture_A = configuration_capture(MAX, conf, conf1);
        int conf_capture_B = configuration_capture(MAX, conf, conf2);
        if (!conf_capture_A && conf_capture_B) return 1;
        if (conf_capture_A && !conf_capture_B) return -1;
```

```
        // Cas 3 : le roi du adversaire est menacé //
        int roi_menace_A = configuration_echec(MAX, conf1);
        int roi_menace_B = configuration_echec(MAX, conf2);
        if (!roi_menace_A && roi_menace_B) return 1;
        if (roi_menace_A && !roi_menace_B) return -1;

        // Cas 4 : la transformation d'un pion //
        int pion_transform_A = configuration_pion_transforme(MIN, conf,
            conf1);
        int pion_transform_B = configuration_pion_transforme(MIN, conf,
            conf2);
        if (pion_transform_A < pion_transform_B) return 1;
        if (pion_transform_A > pion_transform_B) return -1;

        /*
        // Cas 5 : une piece du mode adversaire est menacée //
        int conf_menace_A = configuration_menancee(MAX, conf1);
        int conf_menace_B = configuration_menancee(MAX, conf2);
        if (!conf_menace_A && conf_menace_B) return 1;
        if (conf_menace_A && !conf_menace_B) return -1;
        */

        return confcmp321(a,b);
    }
} //fin confcmp321_prom
```

Vous pouvez remarquer qu'il y a un paramètre de plus *arg*, ceci est nécessaire car les comparaisons faites nécessitent la configuration du père et le mode, de ce fait nous avons défini une structure *config_actuelle* comme suit :

```
typedef struct {
  struct config *conf;
  int mode;
}config_actuelle;
```

Mais l'ajout de ce paramètre va à l'encontre de la définition de la fonction qsort, alors nous avons dû utiliser une autre fonction qui accepte un comparateur avec un argument de plus. Mais cette fonction n'est pas uniformisée entre les différents environnements :

- Sur Windows : la fonction est intitulée qsort_s et prend en paramètre un comparateur à trois entrées dont le premier est l'argument supplémentaire
- Sur Linux : la fonction est intitulée qsort_r et prend en paramètre un comparateur à trois entrées dont le dernier est l'argument supplémentaire

Ainsi, nous avons dû utiliser les directive *#ifdef _WIN32* et *#ifdef linux comme suit*

```
#ifdef _WIN32
    int confcmp321_prom(void * arg, const void *a, const void *b);
    int confcmp123_prom(void * arg, const void *a, const void *b);
    #define sort qsort_s
#elif linux
    int confcmp321_prom(const void *a, const void *b, void * arg);
    int confcmp123_prom(const void *a, const void *b, void * arg);
    #define sort qsort_r
#endif // _WIN32
```

Enfin la fonction sort sera utilisé comme ci-dessous :

```
config_actuelle conf_act;
conf_act.conf = &conf;
conf_act.mode = tour;
sort(T, n, sizeof(struct config), confcmp321_prom, (void *) &conf_act);
```

Cette portion de code ou une similaire est implémentée six fois dans le fichier, deux fois dans la fonction main pour le cas tour = MAX et tour = MIN, et deux fois dans chacune des fonctions minmax et la recherche de quiescence pour le cas mode = MAX et mod = MIN.

Remarque

La hiérarchie présentée plus haut n'est que la version finale, en effet nous avons passé par plusieurs versions comme celle-là :

1. Capture du roi du joueur adversaire (échec et mat)
2. Capture d'une pièce autre que le roi du joueur adversaire
    a. Une reine
    b. Une toure
    c. Un cavalier
    d. Un fou
    e. Un pion
3. Menace du roi du joueur adversaire (échec)
4. Transformation d'un pion du joueur courant vers
    a. Une reine
5. Menace d'une pièce du joueur adversaire autre que roi
    a. Une reine
    b. Une toure
    c. Un cavalier
    d. Un fou
    e. Un pion

Mais nous avons remarqué que ces extensions n'ont pas amélioré le temps d'exécution, ce qui nous a poussés à éliminer les comparaisons inutiles.

3.4 Tests

Dans ce qui suit, on va comparer le temps d'exécution de différentes parties d'échecs où nous avons mis le joueur 'MAX' utilisant l'estimation 7 (qui prend en compte l'effet horizon et donc elle ajoute un surplus de temps) contre le joueur 'MIN' utilisant estimation 1 avant et après les améliorations apporté dans la question 2.

Pour rappel, afin de tester notre solution nous avons implémenté la recherche de quiescence dans la fonction estm7 et utilisé la fonction estm1 pour évaluer un nœud final (un nœud feuille ou bien un nœud stable), ainsi nous pourrons la comparer avec la fonction estm1 qui n'implémente pas la recherche de quiescence et donc elle ne tient pas en compte l'effet horizon.

Il faut s'attendre à ce que le temps d'exécution après les améliorations apportées dans la question 2 sera largement plus petit que sans ces dernières.

On va illustrer ci-dessus, plusieurs tests en modifient la hauteur et la largeur dans chacun d'entre eux.

N° Test	Hauteur	Largeur	Explication
01	2	2	Temps d'exécution sans les améliorations de la question 2 : **9,512 s** Temps d'exécution avec les améliorations de la question 2 : **3,986 s** On remarque que le temps d'exécution avec l'implémentation des améliorations de la question 2 est largement plus petit que sans l'implementation des améliorations de la question 2. On remarque aussi que le nombre de coups générés avec l'implémentation des améliorations de la question 2 (**27 coups**) est plus petit que celui obtenue en question 1 (**61 coups**), ceci peut justifier le fait d'avoir plus de coupures alpha bêta dans la question 1 que la question 2. On peut aussi justifier le fait d'avoir plus de coupures alpha bêta dans la question 1 que dans la question 2 par la l'impact de ses coupures. En effet, des coupures qui occurrent dans un niveau x sont plus conséquentes que des coupures qui occurrent dans un niveau x + i. i > 1.

Avant l'implémentation des améliorations de la question 2 :

```
Coup num: 61 : reineB en g8

          a       b       c       d       e       f       g       h
    ------- ------- ------- ------- ------- ------- ------- -------
  8                                   rN              nB
    ------- ------- ------- ------- ------- ------- ------- -------
  7                                                           tB
    ------- ------- ------- ------- ------- ------- ------- -------
  6                   cN
    ------- ------- ------- ------- ------- ------- ------- -------
  5
    ------- ------- ------- ------- ------- ------- ------- -------
  4
    ------- ------- ------- ------- ------- ------- ------- -------
  3                   cB                      pN
    ------- ------- ------- ------- ------- ------- ------- -------
  2                   pB                      cB
    ------- ------- ------- ------- ------- ------- ------- -------
  1                                           rB
    ------- ------- ------- ------- ------- ------- ------- -------

       B : p(1) c(2) f(0) t(1) n(1)     N : p(1) c(1) f(0) t(0) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 2    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:3505 + beta:4361)  = 7866
Fin de partie

Process returned 0 (0x0)   execution time : 9.512 s
```

Avec l'implémentation des améliorations de la question 2 :

```
Coup num: 27 : reineB en c8

          a       b       c       d       e       f       g       h
    ------- ------- ------- ------- ------- ------- ------- -------
  8                   nB              rN      fN      cN      tN
    ------- ------- ------- ------- ------- ------- ------- -------
  7                                   pN      pN      pN      pN
    ------- ------- ------- ------- ------- ------- ------- -------
  6                         pN
    ------- ------- ------- ------- ------- ------- ------- -------
  5
    ------- ------- ------- ------- ------- ------- ------- -------
  4                         fB
    ------- ------- ------- ------- ------- ------- ------- -------
  3
    ------- ------- ------- ------- ------- ------- ------- -------
  2    cB                   pB      pB      pB      pB      pB
    ------- ------- ------- ------- ------- ------- ------- -------
  1                                   rB      fB              tB
    ------- ------- ------- ------- ------- ------- ------- -------

       B : p(5) c(1) f(2) t(1) n(1)     N : p(5) c(1) f(1) t(1) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 2    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:1329 + beta:912)  = 2241
Fin de partie

Process returned 0 (0x0)   execution time : 3.968 s
```

N° Test	Hauteur	Largeur	Explication
02	2	3	Temps d'exécution sans les améliorations de la question 2 : **66,980 s** Temps d'exécution avec les améliorations de la question 2 : **11,159 s** On remarque que le temps d'exécution avec l'implémentation des améliorations de la question 2 est largement plus petit que sans l'implementation des améliorations de la question 2. On remarque aussi que le nombre de coups générés avec l'implémentation des améliorations de la question 2 (**61 coups**) est plus petit que celui obtenue en question 1 (**71 coups**), ceci peut justifier le fait d'avoir plus de coupures alpha bêta dans la question 1 que la question 2. On peut aussi justifier le fait d'avoir plus de coupures alpha bêta dans la question 1 que dans la question 2 par la l'impact de ses coupures. En effet, des coupures qui occurrent dans un niveau x sont plus conséquentes que des coupures qui occurrent dans un niveau x + i. i > 1.

Avant l'implémentation des améliorations de la question 2 :

```
Coup num: 71 : reineB en f8

         a       b       c       d       e       f       g       h
    ------- ------- ------- ------- ------- ------- ------- -------
  8                                         nB      cN      tN
    ------- ------- ------- ------- ------- ------- ------- -------
  7                         cB                      rN
    ------- ------- ------- ------- ------- ------- ------- -------
  6
    ------- ------- ------- ------- ------- ------- ------- -------
  5                                                         pN
    ------- ------- ------- ------- ------- ------- ------- -------
  4                                 fB                      pB
    ------- ------- ------- ------- ------- ------- ------- -------
  3                                             pB
    ------- ------- ------- ------- ------- ------- ------- -------
  2                 pB              rB      pB      tB
    ------- ------- ------- ------- ------- ------- ------- -------
  1
    ------- ------- ------- ------- ------- ------- ------- -------

       B : p(4) c(1) f(1) t(1) n(1)     N : p(1) c(1) f(0) t(1) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 2    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:432409 + beta:299569)  = 731978
Fin de partie

Process returned 0 (0x0)   execution time : 66.980 s
```

Avec l'implémentation des améliorations de la question 2 :

```
Coup num: 61 : tourB en h8

          a       b       c       d       e       f       g       h
    ------- ------- ------- ------- ------- ------- ------- -------
  8                           rN                              tB
    ------- ------- ------- ------- ------- ------- ------- -------
  7
    ------- ------- ------- ------- ------- ------- ------- -------
  6                                                           pN
    ------- ------- ------- ------- ------- ------- ------- -------
  5                           cB
    ------- ------- ------- ------- ------- ------- ------- -------
  4                                                           pB
    ------- ------- ------- ------- ------- ------- ------- -------
  3
    ------- ------- ------- ------- ------- ------- ------- -------
  2                           pB      rB      pB      pB
    ------- ------- ------- ------- ------- ------- ------- -------
  1                                       fB              tB
    ------- ------- ------- ------- ------- ------- ------- -------

       B : p(4) c(1) f(1) t(2) n(0)     N : p(1) c(0) f(0) t(0) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 2    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:7324 + beta:9273)  = 16597
Fin de partie

Process returned 0 (0x0)   execution time : 11.159 s
```

N° Test	Hauteur	Largeur	Explication
03	2	5	Temps d'exécution sans les améliorations de la question 2 : **197,674 s** Temps d'exécution avec les améliorations de la question 2 : **42,594 s** On remarque que le temps d'exécution avec l'implémentation des améliorations de la question 2 est largement plus petit que sans l'implementation des améliorations de la question 2. On peut justifier le fait d'avoir plus de coupures alpha bêta dans la question 1 que dans la question 2 par la l'impact de ses coupures. En effet, des coupures qui occurrent dans un niveau x sont plus conséquentes que des coupures qui occurrent dans un niveau x + i. i > 1.

Avant l'implémentation des améliorations de la question 2 :

```
Coup num: 51 : reineB en f7

          a       b       c       d       e       f       g       h
       ------- ------- ------- ------- ------- ------- ------- -------
    8             cN              nN      rN              fB      tN
       ------- ------- ------- ------- ------- ------- ------- -------
    7                                                 nB
       ------- ------- ------- ------- ------- ------- ------- -------
    6
       ------- ------- ------- ------- ------- ------- ------- -------
    5
       ------- ------- ------- ------- ------- ------- ------- -------
    4             pN              pN
       ------- ------- ------- ------- ------- ------- ------- -------
    3                             pB              cB              pN
       ------- ------- ------- ------- ------- ------- ------- -------
    2                     pB              rB      pB
       ------- ------- ------- ------- ------- ------- ------- -------
    1                                                             tB
       ------- ------- ------- ------- ------- ------- ------- -------

       B : p(3) c(1) f(1) t(1) n(1)     N : p(3) c(1) f(0) t(1) n(1)


Au tour du joueur minimisant PC 'N'   hauteur = 2    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:982288 + beta:2459168)  = 3441456
Fin de partie

Process returned 0 (0x0)   execution time : 197.674 s
```

Avec l'implémentation des améliorations de la question 2 :

```
Coup num: 79 : tourB en a5

          a       b       c       d       e       f       g       h
       ------- ------- ------- ------- ------- ------- ------- -------
    8
       ------- ------- ------- ------- ------- ------- ------- -------
    7             tN                                              pN
       ------- ------- ------- ------- ------- ------- ------- -------
    6
       ------- ------- ------- ------- ------- ------- ------- -------
    5     tB
       ------- ------- ------- ------- ------- ------- ------- -------
    4             pN                      pB
       ------- ------- ------- ------- ------- ------- ------- -------
    3     rN
       ------- ------- ------- ------- ------- ------- ------- -------
    2     pB                      pB              pB      pB
       ------- ------- ------- ------- ------- ------- ------- -------
    1             tB                      rB      fB      cB
       ------- ------- ------- ------- ------- ------- ------- -------

       B : p(5) c(1) f(1) t(2) n(0)     N : p(2) c(0) f(0) t(1) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 2    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:110169 + beta:458548)  = 568717
Fin de partie

Process returned 0 (0x0)   execution time : 42.594 s
```

N° Test	Hauteur	Largeur	Explication
04	3	2	Temps d'exécution sans les améliorations de la question 2 : **12,820 s** Temps d'exécution avec les améliorations de la question 2 : **4,379 s** On remarque que le temps d'exécution avec l'implémentation des améliorations de la question 2 est largement plus petit que sans l'implementation des améliorations de la question 2. On remarque aussi que le nombre de coups générés avec l'implémentation des améliorations de la question 2 **(27 coups)** est plus petit que celui obtenue en question 1 **(206 coups),** ceci peut justifier le fait d'avoir plus de coupures alpha bêta dans la question 1 que la question 2. Pour ce cas, nous remarquons que **le joueur maximisant B a perdu** contre le joueur minimisant N avant l'implémentation des améliorations de la question 2, même si la fonction d'estimation du joueur maximisant (fonction estimation 7) tient au compte de l'effet horizon. Par contre, avec l'implémentation des améliorations de la question 2, **le joueur maximisant B** a gagné contre le joueur minimisant N. On peut conclure que l'ordonnancement des successeurs du plus prometteur au moins prometteur permet aussi d'améliorer la qualité du jeu.

Avant l'implémentation des améliorations de la question 2 :

```
Coup num:206 : reineN en h6

         a       b       c       d       e       f       g       h
    ------- ------- ------- ------- ------- ------- ------- -------
    8                                       nN      cN      rN
    ------- ------- ------- ------- ------- ------- ------- -------
    7
    ------- ------- ------- ------- ------- ------- ------- -------
    6                                                       nN
    ------- ------- ------- ------- ------- ------- ------- -------
    5
    ------- ------- ------- ------- ------- ------- ------- -------
    4
    ------- ------- ------- ------- ------- ------- ------- -------
    3
    ------- ------- ------- ------- ------- ------- ------- -------
    2
    ------- ------- ------- ------- ------- ------- ------- -------
    1                                               rB
    ------- ------- ------- ------- ------- ------- ------- -------

       B : p(0) c(0) f(0) t(0) n(0)     N : p(0) c(1) f(0) t(0) n(2)


Au tour du joueur maximisant PC 'B'   hauteur = 3    nb alternatives = 0 :

 *** le joueur maximisant 'B' a perdu ***

Nb de coupes (alpha:4122 + beta:3471)  = 7593
Fin de partie

Process returned 0 (0x0)   execution time : 12.820 s
```

Avec l'implémentation des améliorations de la question 2 :

```
Coup num: 27 : reineB en c8

         a       b       c       d       e       f       g       h
    ------- ------- ------- ------- ------- ------- ------- -------
    8                   nB              rN      fN      cN      tN
    ------- ------- ------- ------- ------- ------- ------- -------
    7                                   pN      pN      pN      pN
    ------- ------- ------- ------- ------- ------- ------- -------
    6                           pN
    ------- ------- ------- ------- ------- ------- ------- -------
    5
    ------- ------- ------- ------- ------- ------- ------- -------
    4                           fB
    ------- ------- ------- ------- ------- ------- ------- -------
    3
    ------- ------- ------- ------- ------- ------- ------- -------
    2    cB                     pB      pB      pB      pB      pB
    ------- ------- ------- ------- ------- ------- ------- -------
    1                                   rB      fB              tB
    ------- ------- ------- ------- ------- ------- ------- -------

       B : p(5) c(1) f(2) t(1) n(1)     N : p(5) c(1) f(1) t(1) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 3    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:1369 + beta:950)  = 2319
Fin de partie

Process returned 0 (0x0)   execution time : 4.379 s
```

N° Test	Hauteur	Largeur	Explication
05	3	3	Temps d'exécution sans les améliorations de la question 2 : **24,853 s** Temps d'exécution avec les améliorations de la question 2 : **7,275 s** On remarque que le temps d'exécution avec l'implémentation des améliorations de la question 2 est largement plus petit que sans l'implementation des améliorations de la question 2. On peut justifier le fait d'avoir plus de coupures alpha bêta dans la question 1 que dans la question 2 par la l'impact de ses coupures. En effet, des coupures qui occurrent dans un niveau x sont plus conséquentes que des coupures qui occurrent dans un niveau x + i. i > 1.

Avant l'implémentation des améliorations de la question 2 :

```
Coup num:117 : tourB en b8

         a       b       c       d       e       f       g       h
       ------- ------- ------- ------- ------- ------- ------- -------
   8           tB              rN
       ------- ------- ------- ------- ------- ------- ------- -------
   7                                                       nB
       ------- ------- ------- ------- ------- ------- ------- -------
   6
       ------- ------- ------- ------- ------- ------- ------- -------
   5    nN                                 cN
       ------- ------- ------- ------- ------- ------- ------- -------
   4
       ------- ------- ------- ------- ------- ------- ------- -------
   3                  pB
       ------- ------- ------- ------- ------- ------- ------- -------
   2                                                       rB
       ------- ------- ------- ------- ------- ------- ------- -------
   1
       ------- ------- ------- ------- ------- ------- ------- -------

       B : p(1) c(0) f(0) t(1) n(1)     N : p(0) c(1) f(0) t(0) n(1)


Au tour du joueur minimisant PC 'N'   hauteur = 3    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:147771 + beta:91226)  = 238997
Fin de partie

Process returned 0 (0x0)   execution time : 24.853 s
```

Avec l'implémentation des améliorations de la question 2 :

```
Coup num:127 : reineB en b8

         a       b       c       d       e       f       g       h
       ------- ------- ------- ------- ------- ------- ------- -------
   8           nB                                              rN
       ------- ------- ------- ------- ------- ------- ------- -------
   7
       ------- ------- ------- ------- ------- ------- ------- -------
   6
       ------- ------- ------- ------- ------- ------- ------- -------
   5
       ------- ------- ------- ------- ------- ------- ------- -------
   4                                                           pN
       ------- ------- ------- ------- ------- ------- ------- -------
   3                  pB      cB      pB      pB
       ------- ------- ------- ------- ------- ------- ------- -------
   2                                              rB
       ------- ------- ------- ------- ------- ------- ------- -------
   1
       ------- ------- ------- ------- ------- ------- ------- -------

       B : p(3) c(1) f(0) t(0) n(1)     N : p(1) c(0) f(0) t(0) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 3    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:6959 + beta:9440)  = 16399
Fin de partie

Process returned 0 (0x0)   execution time : 7.275 s
```

N° Test	Hauteur	Largeur	Explication
06	3	4	Temps d'exécution sans les améliorations de la question 2 : **107,213 s** Temps d'exécution avec les améliorations de la question 2 : **11,317 s** On remarque que le temps d'exécution avec l'implémentation des améliorations de la question 2 est largement plus petit que sans l'implementation des améliorations de la question 2. On peut justifier le fait d'avoir plus de coupures alpha bêta dans la question 1 que dans la question 2 par la l'impact de ses coupures. En effet, des coupures qui occurrent dans un niveau x sont plus conséquentes que des coupures qui occurrent dans un niveau x + i. i > 1.

Avant l'implémentation des améliorations de la question 2 :

```
Coup num: 57 : reineB en f7

         a       b       c       d       e       f       g       h
       ------- ------- ------- ------- ------- ------- ------- -------
   8                           nN              fN      cN      tN
       ------- ------- ------- ------- ------- ------- ------- -------
   7                                   rN      nB              pN
       ------- ------- ------- ------- ------- ------- ------- -------
   6
       ------- ------- ------- ------- ------- ------- ------- -------
   5                                   cB
       ------- ------- ------- ------- ------- ------- ------- -------
   4            pN                     cB
       ------- ------- ------- ------- ------- ------- ------- -------
   3
       ------- ------- ------- ------- ------- ------- ------- -------
   2     tB      tB      pB      pB      cN      pB      pB      rB
       ------- ------- ------- ------- ------- ------- ------- -------
   1
       ------- ------- ------- ------- ------- ------- ------- -------

      B : p(4) c(2) f(0) t(2) n(1)     N : p(2) c(2) f(1) t(1) n(1)


Au tour du joueur minimisant PC 'N'   hauteur = 3    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:1190744 + beta:557891)  = 1748635
Fin de partie

Process returned 0 (0x0)   execution time : 107.213 s
```

Avec l'implémentation des améliorations de la question 2 :

```
Coup num: 67 : reineB en d8

         a       b       c       d       e       f       g       h
       ------- ------- ------- ------- ------- ------- ------- -------
   8                           nB                      cN
       ------- ------- ------- ------- ------- ------- ------- -------
   7                                   rN      cB      fN      pN
       ------- ------- ------- ------- ------- ------- ------- -------
   6
       ------- ------- ------- ------- ------- ------- ------- -------
   5                           fB
       ------- ------- ------- ------- ------- ------- ------- -------
   4
       ------- ------- ------- ------- ------- ------- ------- -------
   3     pB              pB                      pB
       ------- ------- ------- ------- ------- ------- ------- -------
   2                           pB              pB              pB
       ------- ------- ------- ------- ------- ------- ------- -------
   1                   fB              rB
       ------- ------- ------- ------- ------- ------- ------- -------

      B : p(6) c(1) f(2) t(0) n(1)     N : p(1) c(1) f(1) t(0) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 3    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:39982 + beta:36050)  = 76032
Fin de partie

Process returned 0 (0x0)   execution time : 11.317 s
```

N° Test	Hauteur	Largeur	Explication
07	3	5	Temps d'exécution sans les améliorations de la question 2 : **85,885 s** Temps d'exécution avec les améliorations de la question 2 : **12,968 s** On remarque que le temps d'exécution avec l'implémentation des améliorations de la question 2 est largement plus petit que sans l'implementation des améliorations de la question 2. On peut justifier le fait d'avoir plus de coupures alpha bêta dans la question 1 que dans la question 2 par la l'impact de ses coupures. En effet, des coupures qui occurrent dans un niveau x sont plus conséquentes que des coupures qui occurrent dans un niveau x + i. i > 1.

Avant l'implémentation des améliorations de la question 2 :

```
Coup num: 73 : tourB en b5

          a       b       c       d       e       f       g       h
       ------- ------- ------- ------- ------- ------- ------- -------
    8
       ------- ------- ------- ------- ------- ------- ------- -------
    7                                           nB
       ------- ------- ------- ------- ------- ------- ------- -------
    6                   fB                      cN      tB
       ------- ------- ------- ------- ------- ------- ------- -------
    5           tB                              rN
       ------- ------- ------- ------- ------- ------- ------- -------
    4                                           pN              pN
       ------- ------- ------- ------- ------- ------- ------- -------
    3                   pB      rB              cB
       ------- ------- ------- ------- ------- ------- ------- -------
    2                                           pB
       ------- ------- ------- ------- ------- ------- ------- -------
    1           cB                                                  nN
       ------- ------- ------- ------- ------- ------- ------- -------

       B : p(2) c(2) f(1) t(2) n(1)     N : p(2) c(1) f(0) t(0) n(1)


Au tour du joueur minimisant PC 'N'   hauteur = 3    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:1028082 + beta:282534)  = 1310616
Fin de partie

Process returned 0 (0x0)   execution time : 85.885 s
```

Avec l'implémentation des améliorations de la question 2 :

```
Coup num:115 : reineB en b5

          a       b       c       d       e       f       g       h
       ------- ------- ------- ------- ------- ------- ------- -------
    8
       ------- ------- ------- ------- ------- ------- ------- -------
    7
       ------- ------- ------- ------- ------- ------- ------- -------
    6
       ------- ------- ------- ------- ------- ------- ------- -------
    5     rN      nB                                      pN      pN
       ------- ------- ------- ------- ------- ------- ------- -------
    4                                   pB
       ------- ------- ------- ------- ------- ------- ------- -------
    3                                                           pB
       ------- ------- ------- ------- ------- ------- ------- -------
    2                   pB      pB      fB      pB
       ------- ------- ------- ------- ------- ------- ------- -------
    1                                   rB              tB
       ------- ------- ------- ------- ------- ------- ------- -------

       B : p(5) c(0) f(1) t(1) n(1)     N : p(2) c(0) f(0) t(0) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 3    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:20374 + beta:25288)  = 45662
Fin de partie

Process returned 0 (0x0)   execution time : 12.968 s
```

N° Test	Hauteur	Largeur	Explication
08	4	2	Temps d'exécution sans les améliorations de la question 2 : **8,392 s** Temps d'exécution avec les améliorations de la question 2 : **6,756 s** On remarque que le temps d'exécution avec l'implémentation des améliorations de la question 2 est largement plus petit que sans l'implementation des améliorations de la question 2. On remarque aussi que le nombre de coups générés avec l'implémentation des améliorations de la question 2 (**37 coups**) est plus petit que celui obtenue en question 1 (**75 coups**). On remarque aussi que le nombre de coupures alpha bêta avec l'implémentation des améliorations de la question 2 est plus grand que sans l''implementation des améliorations de la question 2.

Avant l'implémentation des améliorations de la question 2 :

```
Coup num: 75 : fouB en d6

         a       b       c       d       e       f       g       h
    ------- ------- ------- ------- ------- ------- ------- -------
    8            rN
    ------- ------- ------- ------- ------- ------- ------- -------
    7
    ------- ------- ------- ------- ------- ------- ------- -------
    6    nB                     fB                             cN
    ------- ------- ------- ------- ------- ------- ------- -------
    5                                                  pN      fN
    ------- ------- ------- ------- ------- ------- ------- -------
    4
    ------- ------- ------- ------- ------- ------- ------- -------
    3            pN
    ------- ------- ------- ------- ------- ------- ------- -------
    2            tB                             rB              tB
    ------- ------- ------- ------- ------- ------- ------- -------
    1
    ------- ------- ------- ------- ------- ------- ------- -------

       B : p(0) c(0) f(1) t(2) n(1)     N : p(2) c(1) f(1) t(0) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 4    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:6330 + beta:5135)  = 11465
Fin de partie

Process returned 0 (0x0)   execution time : 8.392 s
```

Avec l'implémentation des améliorations de la question 2 :

```
Coup num: 37 : fouB en b5

         a       b       c       d       e       f       g       h
    ------- ------- ------- ------- ------- ------- ------- -------
    8            tB              nN      rN      fN      cN      tN
    ------- ------- ------- ------- ------- ------- ------- -------
    7                                    pN      pN      pN      pN
    ------- ------- ------- ------- ------- ------- ------- -------
    6
    ------- ------- ------- ------- ------- ------- ------- -------
    5            fB
    ------- ------- ------- ------- ------- ------- ------- -------
    4                                    pN                      cB
    ------- ------- ------- ------- ------- ------- ------- -------
    3                                                    pB
    ------- ------- ------- ------- ------- ------- ------- -------
    2                    pB
    ------- ------- ------- ------- ------- ------- ------- -------
    1                    fB      nB      rB                      fN
    ------- ------- ------- ------- ------- ------- ------- -------

       B : p(2) c(1) f(2) t(1) n(1)     N : p(5) c(1) f(2) t(1) n(1)


Au tour du joueur minimisant PC 'N'   hauteur = 4    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:17364 + beta:10426)  = 27790
Fin de partie

Process returned 0 (0x0)   execution time : 6.756 s
```

N° Test	Hauteur	Largeur	Explication
09	4	3	Temps d'exécution sans les améliorations de la question 2 : **16,062 s** Temps d'exécution avec les améliorations de la question 2 : **8,972 s** On remarque que le temps d'exécution avec l'implémentation des améliorations de la question 2 est largement plus petit que sans l'implementation des améliorations de la question 2. On peut justifier le fait d'avoir plus de coupures alpha bêta dans la question 1 que dans la question 2 par la l'impact de ses coupures. En effet, des coupures qui occurrent dans un niveau x sont plus conséquentes que des coupures qui occurrent dans un niveau x + i. i > 1.

Avant l'implémentation des améliorations de la question 2 :

```
Coup num: 53 : reineB en e7

         a       b       c       d       e       f       g       h
 ------- ------- ------- ------- ------- ------- ------- -------
    8            cN      fN              rN                      tN
 ------- ------- ------- ------- ------- ------- ------- -------
    7                                    nB      pN              pN
 ------- ------- ------- ------- ------- ------- ------- -------
    6                                            fB
 ------- ------- ------- ------- ------- ------- ------- -------
    5
 ------- ------- ------- ------- ------- ------- ------- -------
    4            pN
 ------- ------- ------- ------- ------- ------- ------- -------
    3                            pB      pN      pB
 ------- ------- ------- ------- ------- ------- ------- -------
    2            tN
 ------- ------- ------- ------- ------- ------- ------- -------
    1                                            rB      cB
 ------- ------- ------- ------- ------- ------- ------- -------

       B : p(2) c(1) f(1) t(0) n(1)     N : p(4) c(1) f(1) t(2) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 4    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:95025 + beta:57820)  = 152845
Fin de partie

Process returned 0 (0x0)   execution time : 16.062 s
```

Avec l'implémentation des améliorations de la question 2 :

```
Coup num:103 : tourB en g7

         a       b       c       d       e       f       g       h
 ------- ------- ------- ------- ------- ------- ------- -------
    8     rN                              tN
 ------- ------- ------- ------- ------- ------- ------- -------
    7                                                    tB
 ------- ------- ------- ------- ------- ------- ------- -------
    6
 ------- ------- ------- ------- ------- ------- ------- -------
    5                            fB                              pB
 ------- ------- ------- ------- ------- ------- ------- -------
    4
 ------- ------- ------- ------- ------- ------- ------- -------
    3
 ------- ------- ------- ------- ------- ------- ------- -------
    2                    pB      pB                      rB
 ------- ------- ------- ------- ------- ------- ------- -------
    1
 ------- ------- ------- ------- ------- ------- ------- -------

       B : p(3) c(0) f(1) t(1) n(0)     N : p(0) c(0) f(0) t(1) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 4    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:10142 + beta:17006)  = 27148
Fin de partie

Process returned 0 (0x0)   execution time : 8.972 s
```

N° Test	Hauteur	Largeur	Explication
10	4	4	Temps d'exécution sans les améliorations de la question 2 : **10,360 s** Temps d'exécution avec les améliorations de la question 2 : **10,789 s** On remarque que le temps d'exécution avec l'implémentation des améliorations de la question 2 est largement plus petit que sans l'implementation des améliorations de la question 2. On peut justifier cela, par le fait que les successeurs peuvent être déjà ordonné du plus prometteur au moins prometteur pour une combinaison donnée (hauteur et largeur), et donc l'ordonnancement de ces successeurs va ajouter un surplus en temps d'exécution.

Avant l'implémentation des améliorations de la question 2 :

```
Coup num: 69 : reineB en e8

          a       b       c       d       e       f       g       h
    ------- ------- ------- ------- ------- ------- ------- -------
    8                                   nB              cN      tN
    ------- ------- ------- ------- ------- ------- ------- -------
    7                                           rN      pN      pN
    ------- ------- ------- ------- ------- ------- ------- -------
    6                                           pN
    ------- ------- ------- ------- ------- ------- ------- -------
    5                   fN
    ------- ------- ------- ------- ------- ------- ------- -------
    4
    ------- ------- ------- ------- ------- ------- ------- -------
    3                                   tB
    ------- ------- ------- ------- ------- ------- ------- -------
    2                   pB      pN      pB      pB      pB
    ------- ------- ------- ------- ------- ------- ------- -------
    1                           rB              fB
    ------- ------- ------- ------- ------- ------- ------- -------

        B : p(4) c(0) f(1) t(1) n(1)     N : p(4) c(1) f(1) t(1) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 4    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:36328 + beta:25755)  = 62083
Fin de partie

Process returned 0 (0x0)   execution time : 10.360 s
```

Avec l'implémentation des améliorations de la question 2 :

```
Au tour du joueur maximisant PC 'B'   hauteur = 4    nb alternatives = 24 :
.v choix=1 (score:100)

Coup num:143 : reineB en f6

          a       b       c       d       e       f       g       h
    ------- ------- ------- ------- ------- ------- ------- -------
    8
    ------- ------- ------- ------- ------- ------- ------- -------
    7                                           rN
    ------- ------- ------- ------- ------- ------- ------- -------
    6                                           nB
    ------- ------- ------- ------- ------- ------- ------- -------
    5     pB                            pB              pN
    ------- ------- ------- ------- ------- ------- ------- -------
    4                                   pB              rB
    ------- ------- ------- ------- ------- ------- ------- -------
    3
    ------- ------- ------- ------- ------- ------- ------- -------
    2
    ------- ------- ------- ------- ------- ------- ------- -------
    1
    ------- ------- ------- ------- ------- ------- ------- -------

        B : p(3) c(0) f(0) t(0) n(1)     N : p(1) c(0) f(0) t(0) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 4    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:12946 + beta:31976)  = 44922
Fin de partie

Process returned 0 (0x0)   execution time : 10.789 s
```

N° Test	Hauteur	Largeur	Explication
11	5	5	Temps d'exécution sans les améliorations de la question 2 : **706.733** s Temps d'exécution avec les améliorations de la question 2 : **12,318** s On remarque que temps d'exécution avec l'implémentation des améliorations de la question 2 est **largement** plus petit que sans l'implementation des améliorations de la question 2. (plus de **57** fois plus rapide) On remarque aussi que le nombre de coups générés avec l'implémentation des améliorations de la question 2 (**51 coups**) est plus petit que celui obtenue en question 1 (**181 coups**), ceci peut justifier le fait d'avoir plus de coupures alpha bêta dans la question 1 que la question 2. On peut aussi justifier le fait d'avoir plus de coupures alpha bêta dans la question 1 que dans la question 2 par la l'impact de ses coupures. En effet, des coupures qui occurrent dans un niveau x sont plus conséquentes que des coupures qui occurrent dans un niveau x + i. i > 1.

Avant l'implémentation des améliorations de la question 2 :

```
Coup num:181 : reineB en b7

          a       b       c       d       e       f       g       h
       ------- ------- ------- ------- ------- ------- ------- -------
    8
       ------- ------- ------- ------- ------- ------- ------- -------
    7             nB
       ------- ------- ------- ------- ------- ------- ------- -------
    6                     rN                                      tN
       ------- ------- ------- ------- ------- ------- ------- -------
    5             tB      pB              pN                      pN
       ------- ------- ------- ------- ------- ------- ------- -------
    4                                     pB
       ------- ------- ------- ------- ------- ------- ------- -------
    3
       ------- ------- ------- ------- ------- ------- ------- -------
    2
       ------- ------- ------- ------- ------- ------- ------- -------
    1                                                             rB
       ------- ------- ------- ------- ------- ------- ------- -------

       B : p(2) c(0) f(0) t(1) n(1)     N : p(2) c(0) f(0) t(1) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 5    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:962147 + beta:8247114)  = 9209261
Fin de partie

Process returned 0 (0x0)   execution time : 706.733 s
```

Avec l'implémentation des améliorations de la question 2 :

```
Coup num: 51 : reineB en b5

          a       b       c       d       e       f       g       h
       ------- ------- ------- ------- ------- ------- ------- -------
    8                                     rN      fN      cN      tN
       ------- ------- ------- ------- ------- ------- ------- -------
    7                                     pN      pN      pN      pN
       ------- ------- ------- ------- ------- ------- ------- -------
    6
       ------- ------- ------- ------- ------- ------- ------- -------
    5             nB
       ------- ------- ------- ------- ------- ------- ------- -------
    4                             pN
       ------- ------- ------- ------- ------- ------- ------- -------
    3                                                             pB
       ------- ------- ------- ------- ------- ------- ------- -------
    2             pB              pB      pB      pB              pB
       ------- ------- ------- ------- ------- ------- ------- -------
    1             cB      fB      rB              fB      tB
       ------- ------- ------- ------- ------- ------- ------- -------

       B : p(6) c(1) f(2) t(1) n(1)     N : p(5) c(1) f(1) t(1) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 5    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:11791 + beta:79664)  = 91455
Fin de partie

Process returned 0 (0x0)   execution time : 12.318 s
```

N° Test	Hauteur	Largeur	Explication
12	6	6	Temps d'exécution sans les améliorations de la question 2 : **136.662 s** Temps d'exécution avec les améliorations de la question 2 : **9,620 s** On remarque que temps d'exécution avec l'implémentation des améliorations de la question 2 est **largement** plus petit que sans l'implementation des améliorations de la question 2. On remarque aussi que le nombre de coups générés avec l'implémentation des améliorations de la question 2 (**97 coups**) est plus petit que celui obtenue en question 1 (**23 coups**), ceci peut justifier le fait d'avoir plus de coupures alpha bêta dans la question 1 que la question 2. On peut aussi justifier le fait d'avoir plus de coupures alpha bêta dans la question 1 que dans la question 2 par la l'impact de ses coupures. En effet, des coupures qui occurrent dans un niveau x sont plus conséquentes que des coupures qui occurrent dans un niveau x + i. i > 1.

Avant l'implémentation des améliorations de la question 2 :

```
Coup num: 97 : reineB en d4

         a       b       c       d       e       f       g       h
       ------- ------- ------- ------- ------- ------- ------- -------
   8     tB
       ------- ------- ------- ------- ------- ------- ------- -------
   7
       ------- ------- ------- ------- ------- ------- ------- -------
   6                                                       pN
       ------- ------- ------- ------- ------- ------- ------- -------
   5             pN
       ------- ------- ------- ------- ------- ------- ------- -------
   4                             nB
       ------- ------- ------- ------- ------- ------- ------- -------
   3             fB      rN      pN      rB
       ------- ------- ------- ------- ------- ------- ------- -------
   2                     pB
       ------- ------- ------- ------- ------- ------- ------- -------
   1
       ------- ------- ------- ------- ------- ------- ------- -------

      B : p(1) c(0) f(1) t(1) n(1)     N : p(3) c(0) f(0) t(0) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 6    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:678667 + beta:2285989)  = 2964656
Fin de partie

Process returned 0 (0x0)   execution time : 136.662 s
```

Avec l'implémentation des améliorations de la question 2 :

```
Coup num: 23 : reineB en c8

         a       b       c       d       e       f       g       h
       ------- ------- ------- ------- ------- ------- ------- -------
   8     tN      cN      nB              rN      fN      cN      tN
       ------- ------- ------- ------- ------- ------- ------- -------
   7                                     pN      pN      pN      pN
       ------- ------- ------- ------- ------- ------- ------- -------
   6
       ------- ------- ------- ------- ------- ------- ------- -------
   5                             pN
       ------- ------- ------- ------- ------- ------- ------- -------
   4
       ------- ------- ------- ------- ------- ------- ------- -------
   3     pB
       ------- ------- ------- ------- ------- ------- ------- -------
   2                             pB      pB      pB      pB      pB
       ------- ------- ------- ------- ------- ------- ------- -------
   1     tB              fB              rB      fB              tB
       ------- ------- ------- ------- ------- ------- ------- -------

      B : p(6) c(0) f(2) t(2) n(1)     N : p(5) c(2) f(1) t(2) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 6    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:16054 + beta:58018)  = 74072
Fin de partie

Process returned 0 (0x0)   execution time : 9.620 s
```

N° Test	Hauteur	Largeur	Explication
13	7	7	Temps d'exécution sans les améliorations de la question 2 : **80,036 s** Temps d'exécution avec les améliorations de la question 2 : **51,083 s** On remarque que le temps d'exécution avec l'implémentation des améliorations de la question 2 est largement plus petit que sans l'implementation des améliorations de la question 2. On remarque aussi que le nombre de coups générés avec l'implémentation des améliorations de la question 2 (**77 coups**) est plus petit que celui obtenue en question 1 (**89 coups**), ceci peut justifier le fait d'avoir plus de coupures alpha bêta dans la question 1 que la question 2. On peut aussi justifier le fait d'avoir plus de coupures alpha bêta dans la question 1 que dans la question 2 par la l'impact de ses coupures. En effet, des coupures qui occurrent dans un niveau x sont plus conséquentes que des coupures qui occurrent dans un niveau x + i. i > 1.

Avant l'implémentation des améliorations de la question 2 :

```
Coup num: 89 : tourB en e7

         a       b       c       d       e       f       g       h
    ------- ------- ------- ------- ------- ------- ------- -------
    8                                       tN
    ------- ------- ------- ------- ------- ------- ------- -------
    7                               tB                      pN
    ------- ------- ------- ------- ------- ------- ------- -------
    6                               rN      cN
    ------- ------- ------- ------- ------- ------- ------- -------
    5                                       cB
    ------- ------- ------- ------- ------- ------- ------- -------
    4                               pB                      pB
    ------- ------- ------- ------- ------- ------- ------- -------
    3                       rB              pB
    ------- ------- ------- ------- ------- ------- ------- -------
    2                       pB
    ------- ------- ------- ------- ------- ------- ------- -------
    1
    ------- ------- ------- ------- ------- ------- ------- -------

       B : p(4) c(1) f(0) t(1) n(0)     N : p(1) c(1) f(0) t(1) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 7    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:1021976 + beta:812728)  = 1834704
Fin de partie

Process returned 0 (0x0)   execution time : 80.036 s
```

Avec l'implémentation des améliorations de la question 2 :

```
Coup num: 77 : reineB en a3

         a       b       c       d       e       f       g       h
    ------- ------- ------- ------- ------- ------- ------- -------
    8           tN
    ------- ------- ------- ------- ------- ------- ------- -------
    7                                       pN              pN
    ------- ------- ------- ------- ------- ------- ------- -------
    6                               pN
    ------- ------- ------- ------- ------- ------- ------- -------
    5
    ------- ------- ------- ------- ------- ------- ------- -------
    4    rN                     pB
    ------- ------- ------- ------- ------- ------- ------- -------
    3    nB                                                 pB
    ------- ------- ------- ------- ------- ------- ------- -------
    2                           pB      fB      pB              pB
    ------- ------- ------- ------- ------- ------- ------- -------
    1           cB                      rB
    ------- ------- ------- ------- ------- ------- ------- -------

       B : p(5) c(1) f(1) t(0) n(1)     N : p(3) c(0) f(0) t(1) n(0)


Au tour du joueur minimisant PC 'N'   hauteur = 7    nb alternatives = 0 :

 *** le joueur minimisant 'N' a perdu ***

Nb de coupes (alpha:305187 + beta:446128)  = 751315
Fin de partie

Process returned 0 (0x0)   execution time : 51.083 s
```

Pour résumer on a :

Hauteur	Largeur	Temps d'exécution avant l'implémentation des améliorations de la question 2 (s)	Temps d'exécution après l'implémentation des améliorations de la question 2 (s)
2	2	**9,512**	**3,986**
2	3	**66,980**	**11,159**
2	5	**197,674**	**42,594**
3	2	**12,820**	**4,379**
3	3	**24,853**	**7,275**
3	4	**107,213**	**107,213s**
3	5	**85,885**	**12,968**
4	2	**8,392**	**6,756**
4	3	**16,062**	**8,972**
4	4	**10,360**	**10,789**
5	5	**706.733**	**12,318**
6	6	**136.662**	**9,620**
7	7	**80,036**	**51,083**

On remarque, que dans la plupart des tests le temps d'exécution après l'implémentation des améliorations de la question 2 est largement plus petit qu'au temps d'exécution avant l'implémentation des améliorations de la question 2.

Remarque

Pour le cas particulier où la hauteur = 3 et la largeur = 2, nous remarquons que :

- Avant l'implémentation des améliorations de la question 2, **le joueur maximisant B a perdu** contre le joueur minimisant N même si la fonction d'estimation du joueur maximisant (fonction estimation 7) tient au compte l'effet horizon.
- Avec l'implémentation des améliorations de la question 2 2 **le joueur maximisant B a gagné** contre le joueur minimisant N

On peut conclure, que l'ordonnancement des successeurs du plus prometteur au moins prometteur permet non seulement de réduire largement le temps d'exécution, mais aussi d'améliorer la qualité de jeu.

4 Conclusion

Afin d’améliorer la performance de la procédure MinMax utilisant les coupes alpha/bêta et répondre à la problématique posée, nous avons réalisé dans ce TP une solution composée de deux optimisations principales qui améliorent la qualité de jeu d’échec et son temps d’exécution. Dans la première partie, on a implémenté la recherche de quiescence afin de remédier au problème de l’effet horizon et tenir compte du critère de stabilité de la configuration. Puis dans la deuxième partie, on a amélioré le choix de l’ordre d’évaluation des différents coups possibles, en triant les configurations filles dans l’ordre de la plus prometteuse à la moins prometteuse en se basant sur le type de ses configurations.

À travers ce rapport, nous avons présenté tout le travail réalisé dans ce TP dans le cadre du module CRP (Complexité & Résolution de Problèmes). À la fin de chaque amélioration, nous avons présenté différents tests pour illustrer l’effet de l’amélioration implémentée et montrer l’efficacité de notre solution.

Pour conclure, L'optimisation de l'alpha-béta est encore un sujet de recherche très active. De nombreuses autres optimisations sont sans doute possibles pour améliorer la performance de la procédure MinMax utilisant les coupes alpha/bêta. Nous avons implémenté dans ce TP, deux optimisations principales et nous avons obtenu de meilleures performances en ce qui concerne la qualité du jeu d'échecs et aussi son temps d'exécution.

5 Annexe : Iterative Deepning

5.1 Solution proposée

La plupart des programmes de jeu basés sur l'alpha-bêta mettent en œuvre cette méthode en utilisant une technique appelée approfondissement itératif. Cela fonctionne en effectuant d'abord une recherche à 1 pli, puis en réorganisant les mouvements en fonction des résultats. L'arbre est ensuite exploré à nouveau jusqu'à une profondeur de 2 plis, et ainsi de suite. L'idée est que le meilleur mouvement pour une recherche de (d-1)-plis est susceptible d'être aussi le meilleur pour une recherche de d-plis.

Il est clair que cela peut entraîner des recherches répétées au cours du pli précédent, mais les chances d'une recherche améliorée par la suite signifient que le coût en vaut la peine. En général, cette méthode aboutit à la recherche du meilleur coup en premier dans plus de 90% des cas aux échecs.

L'algorithme alpha-bêta recherche l'arbre avec une fenêtre de recherche initiale de $[-\infty, +\infty]$. Cependant, en général, les valeurs extrêmes ne se produisent pas et il y aura un mouvement quelque part qui s'inscrira dans une plage plus étroite. Par conséquent, si nous commençons avec une fenêtre plus petite, nous pouvons élaguer les branches au début de notre recherche avant de trouver notre meilleur chemin.

Une méthode, appelée recherche par aspiration peut être combinée avec iterative deepening, en centrant la fenêtre sur la valeur retournée par l'itération (d-1) et fixe la fenêtre à plus ou moins un intervalle raisonnable (δ).

5.2 Implémentation

On a commencé par définir la fenêtre comme une constante globale

```
#define valFenetre  50
```

Puis on a remplacé l'appel de minmax (dans la fonction main) pour chaque successeur par une boucle while qui calcule minmax en variant la hauteur de 1 jusqu'à la hauteur souhaitée et misant à jour les valeurs d'alpha et bêta en fonction de la valeur retournée par l'itération précédente.

```
for (i=0; i<n; i++) {

    alpha = score;
    beta = +INFINI;

    h = 1;
    while (h <= hauteur) {
        val = minmax_ab( &T[i], MIN, h, alpha, beta, largeur, estMax );
        if ((val <= alpha) || (val >= beta)) {
            alpha = -INFINI;    //Nous sommes tombés en dehors de la
fenêtre,
            beta = +INFINI;     //alors réessayez avec une fenêtre pleine
largeur (et la même profondeur)
        }
        else {
            alpha = val - valFenetre;  // Configurer la fenêtre pour la
prochaine itération.
            beta = val + valFenetre;
            h++;
        }
    }
```

```
    cout = val;
    printf("."); fflush(stdout);
    // printf(" %4d", cout); fflush(stdout);
    if ( cout > score ) {  // Choisir le meilleur coup (c-a-d le plus
grand score)
          score = cout;
          j = i;
    }
    if ( cout == 100 ) {
            printf("v"); fflush(stdout);
            break;
    }
}
```

Remarque

Après avoir implémenté Iterative deepening nous n'avons pas remarqué une amélioration considérable dans le temps, au contraire pour certaines entrées de profondeur et largeur le temps a largement augmenté ce qui nous a amené à le retirer.

6 Références

Amélioration N°1 : Eviter le problème de l'effet horizon

- Programmation des jeux, Jean-Marc Alliot : http://www.alliot.fr/COURS/JEUX/jeux2.pdf?fbclid=IwAR2GORA89oH_6iZkObzyjKZl1xdO3fYJ3xZudanr3sg6ue6mhrGrNsEwS5M
- Des Optimisations de l'Alpha-Béta, Tristan Cazenave : https://www.lamsade.dauphine.fr/~cazenave/papers/berder00.pdf
- Chapitre 15 Programmer un jeu d'échecs, Didier Müller : https://www.apprendre-en-ligne.net/pj/echecs/chapitre15.pdf?fbclid=IwAR18NrMJ_2OJra1Yu1PSaEzXwjouKBF2X4_HTOAFfNLlmCJXG2ZvPFGty6I
- CS440/ECE448 Lecture 9: Minimax Search, by Svetlana Lazebnik 9/2016 Modified by Mark Hasegawa-Johnson 9/2017 : https://courses.engr.illinois.edu/ece448/fa2017/ece448fa2017lecture9.pdf?fbclid=IwAR13qCRUhLjrvUFGWQoRNW05ZpXGpGCf3XAaA-DHa6xqjmYnKtbv2QIQja0

Amélioration N°2 : Ordonnancement des successeurs d'une configuration donnée

- Javatpoint, Alpha-Beta Pruning : https://www.javatpoint.com/ai-alpha-beta-pruning?fbclid=IwAR0l_afrjwCR4veI9LrY4zjmVOISPFIJmmTJAdmbNgrCyYGxicdqhAEZxWI
- 1library, Alpha-beta Pruning : https://1library.net/article/alpha-pruning-investigations-playing-chess-endgames-reinforcement-learning.7qvn92lz?fbclid=IwAR2TciaLKRGklc_sW83rsqthrs90EPb7UiXxCMow-IWM5FezQi1qmiNuaRQ

Iterative Deepning

- Iterative Deepening and Move Ordering, Jonathan Schaeffer : https://webdocs.cs.ualberta.ca/~jonathan/PREVIOUS/Courses/657/Notes/5.IDandMO.pdf
- Aspiration Windows : https://web.archive.org/web/20071031095918/http://www.brucemo.com/compchess/programming/aspiration.htm
- Transposition Table, History Heuristic, and other Search Enhancements, Tsan-sheng Hsu : https://homepage.iis.sinica.edu.tw/~tshsu/tcg/2017/slides/slide8.pdf