# Jeu d'Othello

Polytech' Marseille

Informatique, Réseaux et Multimédia

## Rapport de projet

Amadou KANE - Maxence MOHR

Projet d'algorithmique et programmation

Tuteur : Nicolas DURAND

Année 2012 -2013

# Introduction

Le jeu d'Othello est un jeu de société qui oppose deux joueurs : les noirs et les blancs. Il a été créé en 1971 et est une marque déposée de Lansay. Une variante, le Reversi, est quand à elle libre de droit.

Le but de ce projet est d'appliquer nos connaissances dans le langage C et en algorithmique afin de réaliser non seulement le jeu d'Othello à deux joueurs humains, mais aussi avec un joueur ordinateur ayant plusieurs niveaux de difficultés.

Ce présent document contient toute l'analyse préliminaire de ce projet.

# Sommaire

1. Analyse des boucles nécessaires

2. Algorithmes des boucles
    a. Boucle principale
    b. Boucle d'affichage
    c. Boucle Coup
    d. Boucle Retournement

3. Le  joueur ordinateur

4. Planning prévisionnel établi à la remise des projets

5. Tests

6. Bilan

7. Conclusion

8. Références

Annexe 1 : Planning prévisionnel

# 1. Analyse des boucles nécessaires

Le jeu d'Othello se jouant au tour à tour et à deux joueurs uniquement, nous sommes parvenus à une modélisation par tâche.

Ainsi, une tâche principale gérera le fonctionnement global : Affichage de l'écran titre, sélection du mode de jeu, et appel des autres boucles constituant le jeu.

Une autre tâche s'occupera de l'affichage de l'Othellier, avec des informations sur la partie en cours.

Une troisième tâche cherchera tous les coups possibles pour le joueur qui doit jouer, et vérifiera si son coup est valide ou non.

Ensuite, une tâche effectuera le retournement (si le coup est valide).

Une dernière tâche permettra de compter les pions de chaque joueur. Cela permettra à la fois d'indiquer en cours de partie qui a l'avantage et de donner le score final à la fin de la partie.

De plus, nous avons décidé de modéliser l'othellier à l'aide d'un tableau de dimension 8x8 (comme l'othellier original). Par exemple, à une case vide correspondrait une valeur de cellule 0, un joueur noir 1 et un joueur blanc -1…

Dans la prochaine partie, nous allons détailler et décrire en pseudo-code les différents algorithmes énumérés ici.

# 2. Algorithme des boucles

## a. Boucle principale

Comme expliqué précédemment, cette boucle gère le fonctionnement du jeu de manière globale : Affichage du titre du jeu, choix du nombre de joueurs, initialisation de la partie, demande de l'affichage, et enfin le jeu au tour à tour. Quand aucun n'est possible (les deux joueurs passent leur tour), une boucle compte les pions de chaque joueur et en sort le vainqueur. Enfin, avant de quitter, on affiche le gagnant.

Variables :

- Entiers : type, bloqué, gagnant, joueur
- Tableaux d'entiers : tableau de jeu

```
Boucle principale – Arguments : aucun

Afficher le titre du jeu
Type = Demander à l'utilisateur le mode : 1 ou 2 joueurs
Initialisation (tableau de jeu, bloqué, etc...) : bloqué=tableau de jeu=0
Affichage tableau de jeu

tant que (bloqué!=2)
      bloqué=0
      Si joueur
            bloqué += coup (tableau de jeu, tableau de possibilités,
            joueur, type)
      Affichage tableau de jeu
      joueur = 0

      Si !joueur
            bloqué += coup (tableau de jeu, tableau de
            possibilités, joueur, type)
      Affichage tableau de jeu
      joueur=1
fin tant que


gagnant = fini(tableau)

si gagnant = 1
      afficher les blancs ont gangné
si gagnant = 0
      afficher les noirs ont gagnés
sinon
      afficher match nul
```

## b. Boucle Affichage

Cette fonction a pour but d'afficher l'othellier (avec ses coordonnées : voir schéma) avant que le joueur joue son coup : non seulement l'othellier est affiché, mais aussi le score actuel, et l'invite de saisie du coup.

Variables :

- Entiers : i, j, noirs, blancs
- Tableaux d'entiers : Tableau de jeu

```
Boucle affichage – Arguments : tableau de jeu


Afficher cadre
pour i de 0 à 8
      pour j de 0 à 8
            Afficher tableau[i][j]
            Si case_noire
                  noirs++
            Si case blanche
                  blancs++
      f_pour
f_pour

Afficher cadre
Afficher info du jeu
```

## c. Boucle Coup

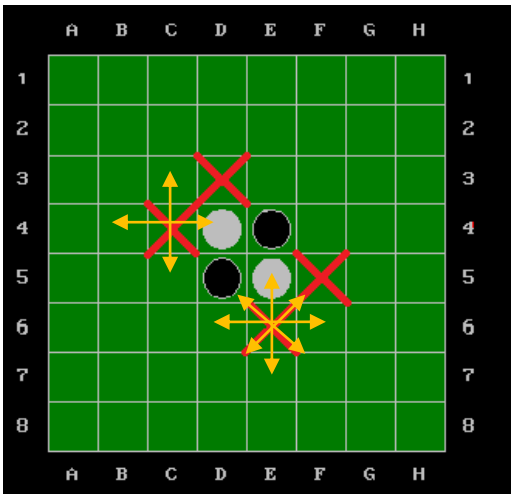



Figure : Coup possible pour les noirs en début de partie

A l'Othello, un coup est possible si le pion que l'on va jouer et un autre de notre couleur encadre des pions de l'autre joueur.

Cette boucle a pour but de chercher un coup possible, et de le stocker. Cela permet de vérifier si le coup saisi par le joueur est valide, mais aussi si un coup est possible ou si le joueur doit passer son tour.

Variables :

- Entiers : i, j, coup_possible, x, y
- Tableaux d'entiers : Tableau de jeu, tableau de possibilités

```
Boucle coup – Arguments : tableau de jeu, tableau de possibilités, type

coup_possible=0
pour i de 0 à 8
      pour j de 0 à 8
            si case_vide(Tableau de jeu[i][j])
                  cherche cases adjacentes pour case joueur
                        si case joueur
                              cherche coup possible pour joueur
                                    si coup possible
                                          coup_possible=1
                                          tableau possibilités[i][j] = 1
      fin pour
fin pour
```

```
si coup_possible
      tant_que(pas_joué)
            Demander coup au joueur => (x,y)
            si tableau possibilités(x,y) = 1
                  tableau(x,y)=1
                  retournement(x,y, tableau de jeu)
                  return 0
            sinon
                  Afficher erreur de saisie
      Fin tant que
Fin si

si !coup_possible
      return 1
```

## d. Boucle de retournement

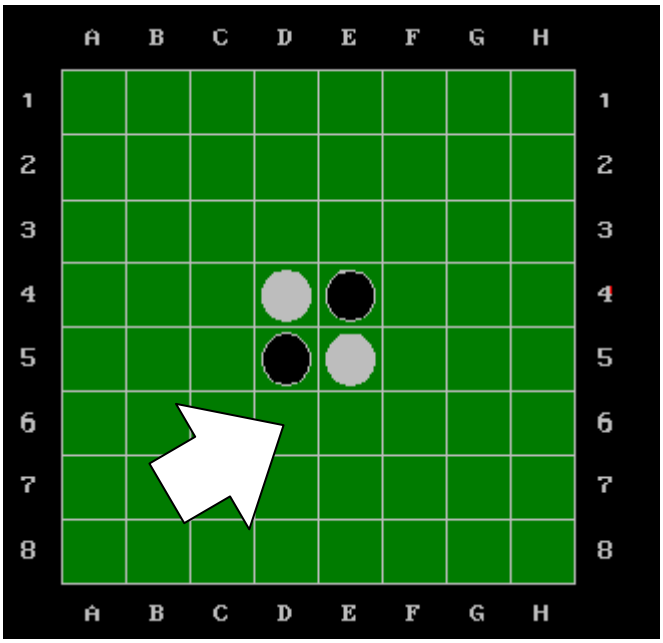



Figure : Les blancs vont jouer en D6

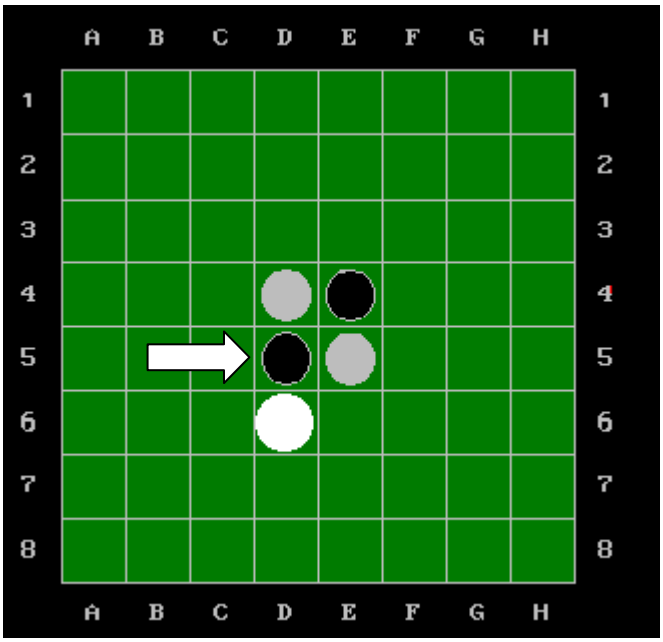



Figure : Le pion est à retourner, il est situé entre un blanc et le blanc tout juste joué…

Cette boucle permet, lorsqu'un coup est possible et joué par le joueur, le retournement des autres pions tel qu'indiqué dans les règles de l'Othello.

Variables :

- Entiers : i, j, case_non_vide, x, y
- Tableaux d'entiers : Tableau de jeu (appelé ici tableau ou T)

```
Boucle de retournement – Arguments : x, y, tableau de jeu


// En haut
i=x-1
tant que (i>=0&&tableau(i,y)!=joueur&&case_non_vide)
      si casevide(i,y)
            case_non_vide=0
      i--
fin_tq

si case_non_vide = 1
      tantque (i!=x)
            si tableau(i,y)=!joueur
                  tableau(i,y)=joueur
            i++
      f_tq


// A gauche
i=y-1
tant que (i>=0&&tableau(x,i)!=joueur&&case_non_vide)
      si casevide(x,i)
            case_non_vide=0
      i--
fin_tq

si case_non_vide = 1
      tantque (i!=y)
            si tableau(x,i)=!joueur
                  tableau(x,i)=joueur
            i++
      f_tq
```

```
// En bas
i=x+1
tant que (i>=7&&tableau(i,y)!=joueur&&case_non_vide)
      si casevide(i,y)
            case_non_vide=0
      i++
fin_tq

si case_non_vide = 1
      tantque (i!=x)
            si tableau(i,y)=!joueur
                  tableau(i,y)=joueur
            i--
      f_tq




// A droite
i=y+1
tant que (i>=7&&tableau(x,i)!=joueur&&case_non_vide)
      si casevide(x,i)
            case_non_vide=0
      i++
fin_tq

si case_non_vide = 1
      tantque (i!=y)
            si tableau(x,i)=!joueur
                  tableau(x,i)=joueur
            i--
      f_tq


//diag haut gauche
i=x-1
j=y-1
tant que (i>=0&&j >=0&&tableau(i,j)!=joueur&&case_non_vide)
      si casevide(i,j)
            case_non_vide=0
      i--
      j--
fin_tq

si case_non_vide = 1
      tantque (i!=x&&j!=y)
            si tableau(i,j)=!joueur
                  tableau(i,j)=joueur
            i++
            j++
      f_tq
```

```
//diag haut droite
i=x-1
j=y+1
tant que (i>=0&&j >=7&&tableau(i,j)!=joueur&&case_non_vide)
      si casevide(i,j)
            case_non_vide=0
      i--
      j++
fin_tq

si case_non_vide = 1
      tantque (i!=x&&j!=y)
            si tableau(i,j)=!joueur
                  tableau(i,j)=joueur
            i++
            j--
      f_tq



//diag bas droite
i=x+1
j=y+1
tant que (i>=7&&j >=7&&tableau(i,j)!=joueur&&case_non_vide)
      si casevide(i,j)
            case_non_vide=0
      i++
      j++
fin_tq

si case_non_vide = 1
      tantque (i!=x&&j!=y)
            si tableau(i,j)=!joueur
                  tableau(i,j)=joueur
            i--
            j--
      f_tq

//diag haut gauche
i=x+1
j=y+1
tant que (i>=7&&j >=7&&tableau(i,j)!=joueur&&case_non_vide)
      si casevide(i,j)
            case_non_vide=0
      i++
      j++
fin_tq

si case_non_vide = 1
      tantque (i!=x&&j!=y)
            si tableau(i,j)=!joueur
                  tableau(i,j)=joueur
            i--
            j--
      f_tq
```

```
//diag bas gauche
i=x+1
j=y-1
tant que (i>=7&&j >=0&&tableau(i,j)!=joueur&&case_non_vide)
      si casevide(i,j)
            case_non_vide=0
      i++
      j--
fin_tq

si case_non_vide = 1
      tantque (i!=x&&j!=y)
            si tableau(i,j)=!joueur
                  tableau(i,j)=joueur
            i--
            j++
      f_tq
```

# 3. Réflexions sur le jeu ordinateur

Comme indiqué par l'énoncé, le joueur IA sera basé sur l'algorithme MiniMax, avec différents niveaux de profondeurs en fonction du niveau de difficulté choisi.

De plus, pour les niveaux les plus difficiles, un système de valeur de cases sera instauré, afin d'établir des priorités de captures de cases (les coins par exemple : une fois pris, ils ne peuvent plus être repris par l'adversaire…)

A priori, l'ajout du joueur ordinateur ne nécessitera pas de grandes modifications sur les codes préexistants, nous étudierons cela plus tard dans le déroulement du projet.

### - Maximisation

Ce mode de jeu est relativement simple : l'ordinateur joue le coup le plus profitable pour lui dans l'immédiat. Cette technique est fortement déconseillée aux joueurs par la fédération française d'Othello, car elle est très dangereuse pour la personne maximisant ses pions. Elle devient très vulnérable aux attaques au coup suivant

Ce mode IA correspond donc à un mode facile.

### - Mini-Max

Dans ce mode de jeu, l'ordinateur ne se cantonne plus qu'à calculer son profit (maximisation), mais voit plusieurs coups en avance tout comme son adversaire, le joueur humain. Le principe étant simple : lorsque l'IA joue, elle va chercher à maximiser son gain, tandis que le tour d'après, le joueur humain a le trait, et va essayer de minimiser le gain de l'IA, puis l'ordinateur réessayera de maximiser ses bénéfices, et ainsi de suite.

Notre fonction MiniMax évaluera donc quelques coups en avance le meilleur coup à jouer à long terme, et non pour ce tour là (maximisation).

Variables :

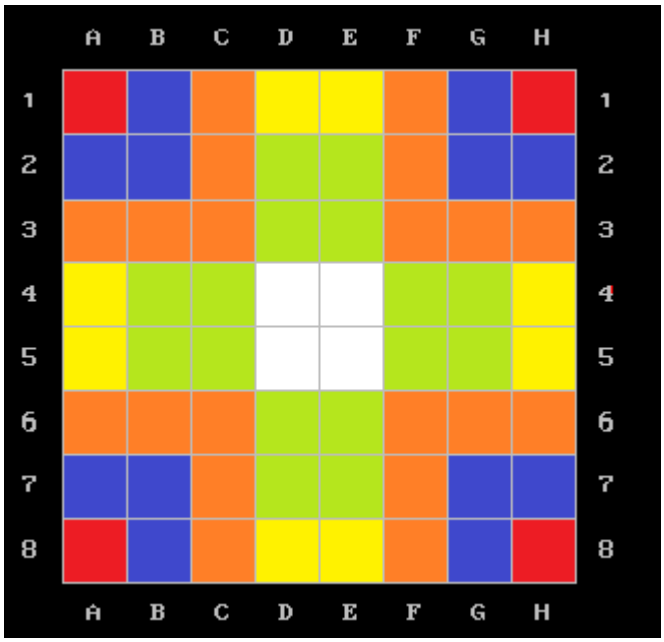
- Entiers : i, j, x, y, val, val1
- Tableaux de jeu et de possibilités

```
Boucle de MiniMax – Arguments : tableau de jeu, tableau de possibilités,
joueur, profondeur

      I = 0
      J = 0
      Val1=-1000
      Copie du tableau de jeu (ctabjeu)
      Copie du tableau de possibilités (ctabpossibl)
      Pour i de 0 à 7
            Pour j de 0 à 7
                  Si case possible
                        Retourne la case jouee sur ctabjeu
                        On prend l'autre joueur (simulation)
                        On recherche les coups possibles
                        Si prof <= L
```

```
        Val = minimax(ctabjeu, ctabpossibl, joueur, prof++)
        Si val > val1
              Val1=val
              X=i
              Y=j
Comparaison des tableaux de jeu
Calcul des valeurs des cases retournées
FIN -> Renvoie valeur, x, y
```

Valeur des cases :




Les coins étant stratégiques, ils ont une priorité haute.

Il ne faut jamais jouer sur les cases bleues sauf impossibilité de faire autrement.

Les bords sont plus prioritaires que le milieu.

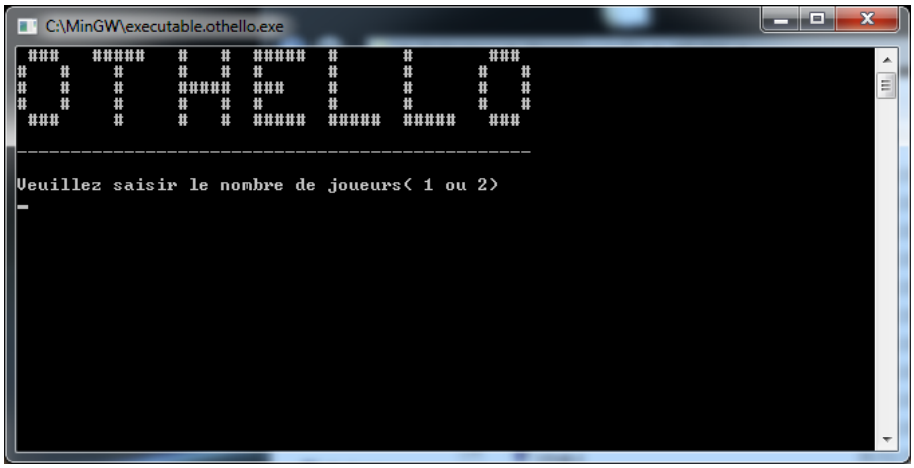
# 4. Planning prévisionnel

Le planning prévisionnel a été établi alors que le projet n'était qu'à l'état d'analyse.

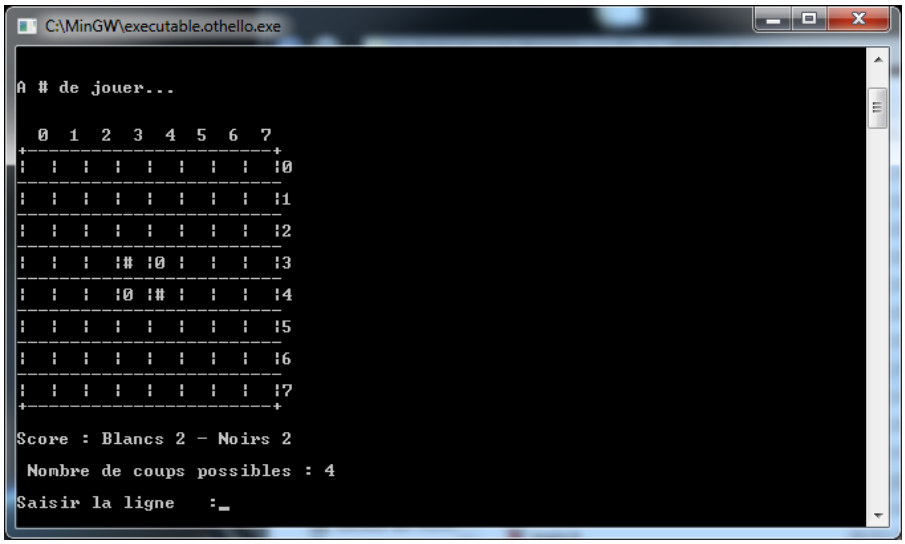
Il est disponible en Annexe 1.

Il contient tout le projet, de la découverte jusqu'à la date butoir imposée.
On peut également y trouver une première répartition des tâches, notamment au niveau de la programmation des différentes boucles.

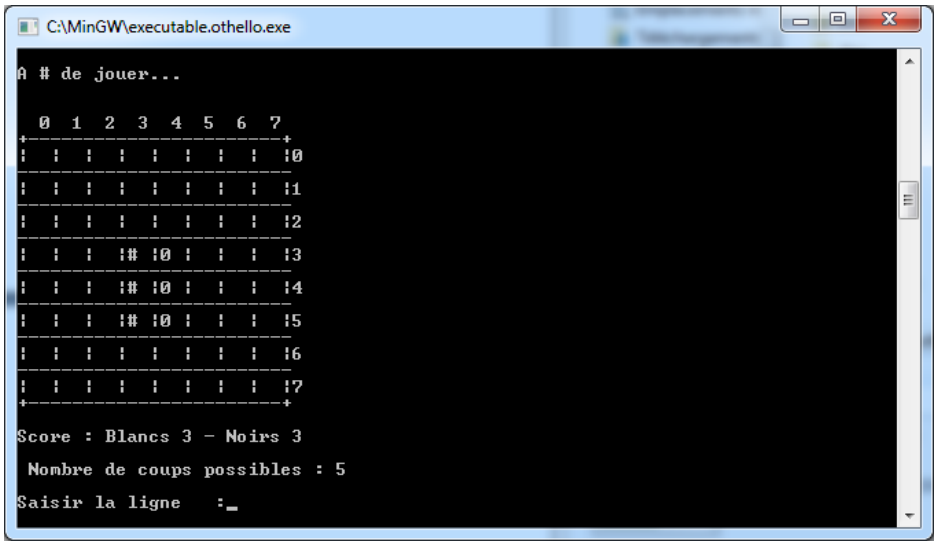
# 5. Tests



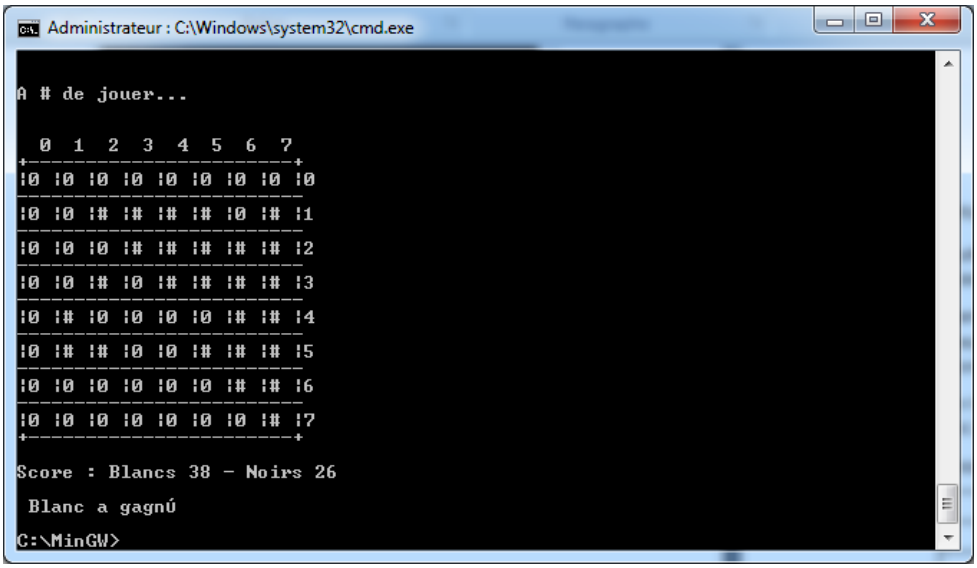

Ecran titre du jeu




En jeu, début de partie




Jeu contre l'ordinateur, après avoir joué en (5 ;3)




Fin du jeu

## Difficultés rencontrées

La première difficulté majeure que nous avons rencontrée a été un dépassement de la valeur d'indice de tableaux, pour les fonctions « coup » et « retournement ».

En effet, en rangeant l'othellier de jeu, par exemple, dans un tableau à deux dimensions, on peut accéder à la valeur d'une case (noir, blanc, vide) en lisant la valeur rangée colonne X, ligne Y (cette valeur est notée T[X][Y] ).

Par convention, un tableau de 3 lignes par 3 colonnes a ses colonnes et ses lignes numérotées de 0 à 2.

Le tableau ressemble à :

| | | |
|---|---|---|
| T[0][0] | T[0][1] | T[0][2] |
| T[1][0] | T[1][1] | T[1][2] |
| T[2][0] | T[2][1] | T[2][2] |

*Figure : Tableau 3 x 3*

Or, du à un manque de conditions, nos fonctions demandaient à accéder à, par exemple, T[0][3], qui n'existe pas. Mais ces coordonnées n'existent pas, malgré tout, elles ne rendent pas un résultat aléatoire, mais la valeur de T[1][0], en raison de sa localisation en mémoire :

| | | | | | | |
|---|---|---|---|---|---|---|
| … | T[0][0] | T[0][1] | T[0][2] | T[1][0] = T[0][3] | T[1][1] = T[0][4] | … |

*Figure : Comment sont rangés les tableaux à plusieurs dimensions dans la mémoire*

Ici, on peut voir que T[0][3] correspond à T[1][0]…

Nous avons eu également beaucoup de mal à implémenter le joueur ordinateur (Mini-Max). En effet, adapter les algorithmes vus en cours à un cas concret n'est pas si évidant.

Enfin, nous avons perdu un peu de temps pendant les Vacances de Noël (Eloignement, connexion Internet, etc…), mais aussi en raison du projet ouverture et des partiels.

# 6. Bilan du projet

Au terme de ce projet, nous pouvons établir un bilan de ce que nous avons réalisé, et de ce que nous n'avons pas eu le temps de faire :

- Analyse : faite, nous connaissions tous les deux le jeu d'Othello avant d'entamer ce projet, il nous a donc été facile de le comprendre.
- Boucle Principale : cette boucle, relativement simple, a été relativement facile à coder.
- Boucle Affichage : tout comme la boucle précédente, elle a été plutôt simple à programmer, tout en cherchant à reproduire un othellier convenable sur une console.
- Boucle Coup : elle a été faite, mais pas évidente à coder en raison d'un problème d'indice de tableau qui a été difficile à retrouver.
- Boucle Retournement : elle a été réalisée, mais nous avons buté sur le même problème que pour la boucle « coup ».
- Boucle IA Maximisation : elle a été réalisée également, en un temps relativement faible.
- Boucle Mini-Max : Non terminée

# 7. Conclusion

Ce projet, basé sur le jeu d'Othello, est très intéressant car il permet d'appliquer à la fois nos connaissances en C et en algorithmique. De plus, le jeu d'Othello est un jeu que nous connaissions tous les deux avant ce projet, ce qui nous a facilité l'analyse et la compréhension du sujet.

Malgré tous nos efforts, il ne nous a pas été possible de le mener à terme, mais il nous a appris beaucoup sur la gestion du temps, les contraintes et les évènements perturbant le planning original.

Néanmoins nous avons pris beaucoup de plaisir à coder ce jeu, et nous en prenons à jouer à cette version de l'Othello !

# 8. Références

Fédération Française d'Othello : http://www.ffothello.org

Images, Règles du jeu

## Logiciels utilisés :

- Kate pour la programmation.
- GCC pour la compilation.
- Microsoft Word 2007 pour le rapport de projet.
- Microsoft PowerPoint 2007 pour la présentation.
- Microsoft Project 2010 pour le planning prévisionnel.
- The GIMP et Microsoft Paint pour les images du rapport.

ANNEXE 1

Jeu d'Othello - Diagramme de Gantt - Projet algorithmique et programmation

Version 1

| N° | Mode Tâche | Nom de la tâche | Durée | Début | Fin |
|---|---|---|---|---|---|
| 1 | | Réflexions sur le projet | 10 jours | Lun 12/11/12 | Ven 23/11/12 |
| 2 | | Rédaction des algorithmes | 1 jour | Ven 23/11/12 | Lun 26/11/12 |
| 3 | | RDV avec tuteur | 0,86 jour | Lun 26/11/12 | Mar 27/11/12 |
| 4 | | Correction rapport d'analyse | 3,14 jours | Mar 27/11/12 | Ven 30/11/12 |
| 5 | | Réflexions sur la séparation des tâches | 3,14 jours | Ven 30/11/12 | Jeu 06/12/12 |
| 6 | | Réalisation fonction affichage | 7 jours | Jeu 06/12/12 | Ven 14/12/12 |
| 7 | | Réalisation fonction coup | 7 jours | Sam 15/12/12 | Lun 24/12/12 |
| 8 | | Réalisation fonction principale | 7 jours | Jeu 06/12/12 | Ven 14/12/12 |
| 9 | | Réalisation fonction retournement | 7 jours | Sam 15/12/12 | Lun 24/12/12 |
| 10 | | Tests indépendents | 2 jours | Mar 25/12/12 | Mer 26/12/12 |
| 11 | | Regroupemment des fonctions | 3 jours | Jeu 27/12/12 | Lun 31/12/12 |
| 12 | | Test commun | 2 jours | Mar 01/01/13 | Mer 02/01/13 |
| 13 | | Réalisation fonction IA | 10 jours | Jeu 03/01/13 | Mer 16/01/13 |
| 14 | | Test indépendant | 4 jours | Jeu 17/01/13 | Mar 22/01/13 |
| 15 | | Regroupemment IA | 1 jour | Mer 23/01/13 | Mer 23/01/13 |
| 16 | | Test commun - Choix scénario | 1 jour | Jeu 24/01/13 | Jeu 24/01/13 |
| 17 | | Rédaction rapport final | 7 jours | Ven 25/01/13 | Lun 04/02/13 |
| 18 | | Oral présentation | | | |