

JEU OTHELLO



المدرسة الحسنية للأشغال العمومية
ECOLE HASSANIA DES TRAVAUX PUBLICS



ZAYD MAZHOUR
SALMI HOUDA
1-SIG

Encadré par :Mme CHERRABI

I-INTRODUCTION :

1-jeu othello :

A-DESCRIPTION :

Othello est un jeu de stratégie à deux joueurs : Noir et Blanc. Il se joue sur un plateau unicolore de 64 cases, 8 sur 8, appelé othellier. Ces joueurs disposent de 64 pions bicolores, noirs d'un côté et blancs de l'autre. Par commodité, chaque joueur a devant lui 32 pions mais ils ne lui appartiennent pas et il doit en donner à son adversaire si celui-ci n'en a plus. Un pion est noir si sa face noire est visible et blanc si sa face blanche est sur le dessus

But du jeu :

Avoir plus de pions de sa couleur que l'adversaire à la fin de la partie. Celle-ci s'achève lorsque aucun des deux joueurs ne peut plus jouer de coup légal. Cela intervient généralement lorsque les 64 cases sont occupées.

2-Historique :

Le jeu de Reversi est né en Angleterre à la fin du 19 siècle. Après avoir rencontré un certain succès, il est tombé pour longtemps dans l'oubli. C'est en 1971 seulement qu'un japonais, Hasegawa, le "ré-inventa" en le rebaptisant «Othello». A tout hasard, Hasegawa déposa le nom, ce qu'il n'aura pas à regretter, le succès étant, cette fois, foudroyant !

La notoriété toute neuve du jeu n'est pas dû aux apports minimes d'Hasegawa mais surtout à la naissance et au développement concomitant des jeux vidéo, puis de la micro-informatique familiale, que le jeu a pu accompagner. La programmation d'une intelligence artificielle digne

JEU OTHELLO

Le jeu d'Othello se déroule sur un othellier de 64 cases. On fait référence aux cases grâce à un système de coordonnées : les lignes sont numérotées de haut en bas, de 1 à 8 ; les colonnes sont étiquetées de gauche à droite, de 'a' à 'h'.

	a	b	c	d	e	f	g	h
1	a1	b1	c1					h1
2	a2							
3	a3							
4								
5								
6								
7								
8	a8							h8

La case en haut à gauche est appelée case a1, celle à sa droite case b1 et ainsi de suite.

	a	b	c	d	e	f	g	h
1		C					C	
2	C	X					X	C
3								
4								
5								
6								
7	C	X					X	C
8		C					C	

Certains types de cases se sont vu, pour des raisons de commodité, attribuer un nom. On parlera de coins (les cases a1, h1, a8 et h8), de cases X, qui sont les cases adjacentes aux coins sur la diagonale, et de cases C qui

sont les autres cases voisines des coins.

Les directions cardinales sont souvent utilisées pour désigner des régions de l'othellier. Par exemple, les cases proches du coin a1 font partie de la région "nord-ouest" tandis que la ligne 8 sera appelée "bord sud".

Il n'est pas nécessaire, pour la lecture de ce livret, de savoir noter une partie. Cependant, si vous désirez lire un commentaire de partie ou conserver vos propres parties, voici comment les noter.

Une partie se note par un diagramme donnant le numéro et la position de chaque coup.

	a	b	c	d	e	f	g	h
1	45	32	19	18	31	24	44	43
2	46	36	9	11	16	15	42	56
3	17	8	3	4	10	22	38	51
4	20	13	5	●	●	6	23	40
5	21	14	7	●	●	1	39	41
6	34	30	12	2	28	29	53	52
7	35	47	33	26	25	37	59	55
8	50	49	48	27	54	60	58	57

Penloup 20-44 Juhem

Le diagramme ci-dessus représente la partie Penloup-Juhem du championnat du monde 1992. Le premier joueur a les noirs, ici Penloup. Le '1' en f5 indique que le premier coup a été joué sur cette case ; puis Blanc a répondu d6, suivi de c3-d3-c4... Normalement, les coups impairs sont noirs et les coups pairs sont blancs mais si l'un des joueurs passe, cela peut changer. Dans cette partie, on verra que le coup 57 est blanc car Noir va passer. D'ailleurs, Noir passe à nouveau : 58 est blanc ; puis Noir joue 59 et Blanc termine avec 60.

C- Les règles du jeu :

Othello est un jeu de stratégie à deux joueurs : Noir et Blanc. Il se joue sur un plateau unicolore de 64 cases, 8 sur 8, appelé othellier.

Ces joueurs disposent de 64 pions bicolores, noirs d'un côté et blancs de l'autre. Par commodité, chaque joueur a devant lui 32 pions mais ils ne lui appartiennent pas et il doit en donner à son adversaire si celui-ci n'en a plus.

Un pion est noir si sa face noire est visible et blanc si sa face blanche est sur le dessus.

But du jeu

Avoir plus de pions de sa couleur que l'adversaire à la fin de la partie. Celle-ci s'achève lorsque aucun des deux joueurs ne peut plus jouer de coup légal. Cela intervient généralement lorsque les 64 cases sont occupées.

Position de départ

Au début de la partie, deux pions noirs sont placés en e4 et d5 et deux pions blancs en d4 et e5 (voir fig. 1).

Noir commence toujours et les deux adversaires jouent ensuite à tour de rôle.

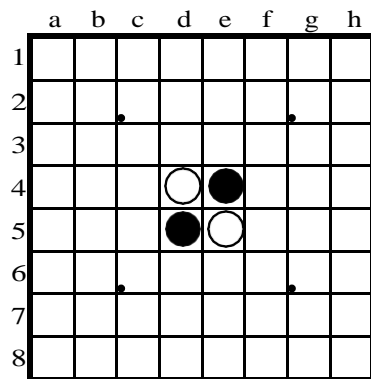


Fig. 1 : la position de départ

La pose d'un pion

A son tour de jeu, le joueur doit poser un pion de sa couleur sur une case vide de

l'othellier, adjacente à un pion adverse. Il doit également, en posant son pion, encadrer un ou plusieurs pions adverses entre le pion qu'il pose et un pion à sa couleur, déjà placé sur l'othellier. Il re-

tourne alors de sa couleur le ou les pions qu'il vient d'encadrer. Les pions ne sont ni retirés de l'othellier, ni déplacés d'une case à l'autre.

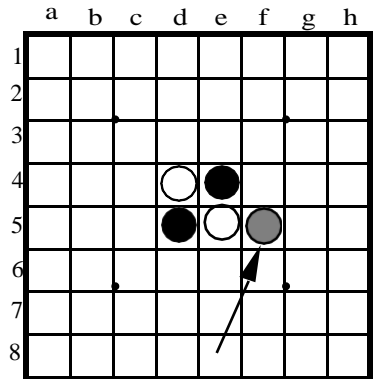


Fig. 2 : Noir joue f5...

Le premier coup de Noir est, par exemple, en f5 (voir figure 2). En jouant f5, il encadre le pion blanc e5 entre le pion qu'il pose et un pion noir déjà présent (ici d5) ;

il retourne alors ce pion (voir figure 3). Noir aurait aussi pu jouer en e6, c4 ou d3. Notons que ces quatre

coups de Noir sont parfaitement symétriques ; Noir n'a donc pas à réfléchir pour choisir son premier coup.

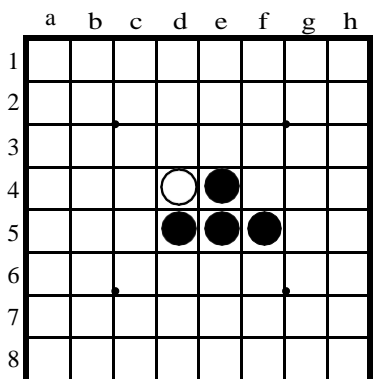


Fig. 3 : et retourne e5 !

C'est maintenant à Blanc de jouer. Il a trois coups possibles (voir figure 4). En effet, il est obligatoire de retourner au moins un pion adverse à chaque coup. Blanc

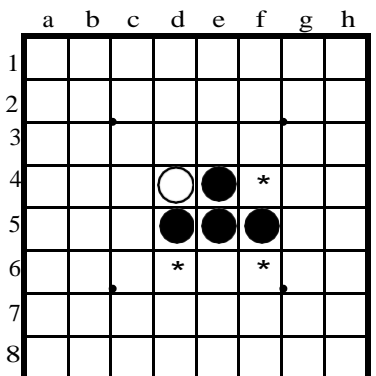


Fig. 4 : Blanc f4, f6 ou d6

JEU OTHELLO

peut donc jouer en f4, f6 ou en d6.

On peut encadrer des pions adverses dans les huit directions.

Par ailleurs, dans chaque direction, plusieurs pions peuvent être encadrés (voir figures 6 et 7). On doit alors tous les retourner.

Le joueur Noir a joué en c6. Il retourne alors les pions b6 (encadré grâce à a6), b5 (encadré grâce à a4), d7 (encadré grâce à e8) c5 et c4 (encadrés grâce à c3). Notons que ni d6, ni e6 ne sont retournés à cause de la case vide en f6.

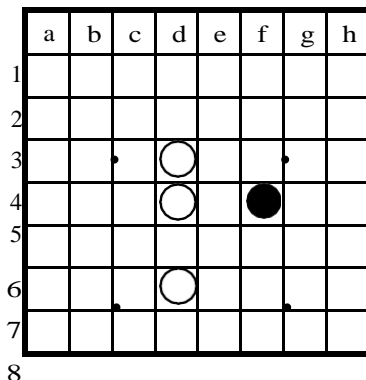


Fig. 5 : si Blanc joue d6.

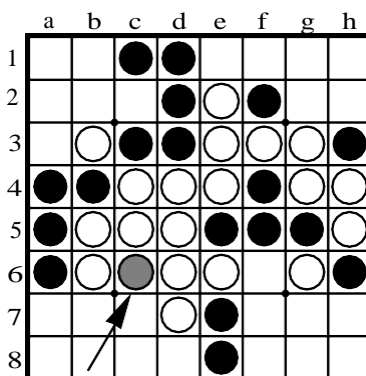


Fig. 6 : Noir joue c6...

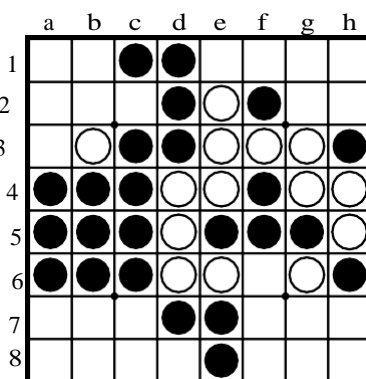


Fig. 7 : et le résultat.

Il n'y a pas de réaction en chaîne : les pions retournés ne peuvent pas servir à en retourner d'autres lors du même tour de jeu. Ainsi, sur la figure 8, Noir joue en a5 : il retourne b5 et c5 qui sont encadrés. Bien que c4 soit alors encadré, il n'est pas retourné (voir figure 9). En effet, il n'est pas encadré entre le pion que l'on vient de poser et un autre pion.

Si, à votre tour de jeu, vous ne pouvez pas poser un pion en retournant un pion adverse suivant les règles, vous devez passer votre

tour et c'est à votre adversaire de jouer. Mais si un retournement est possible, vous ne pouvez vous y soustraire.

Fin de la partie

La partie est terminée lorsque aucun des deux joueurs ne peut plus jouer.

Cela arrive généralement lorsque les 64 cases sont occupées. Mais il se peut qu'il reste des cases vides où personne ne peut jouer : par exemple lorsque tous les pions deviennent d'une même couleur après un retournement. Ou bien comme sur cette position (voir figure 10).

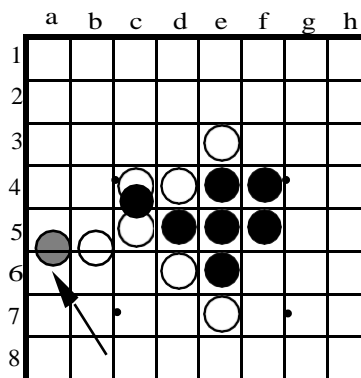


Fig. 8 : Noir joue a5...

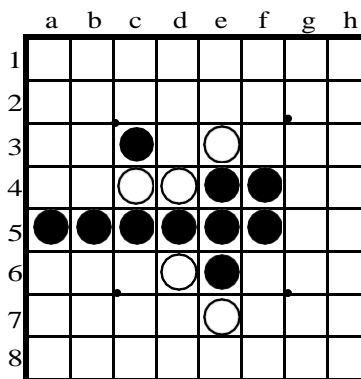


Fig. 9 : c4 reste blanc.

Aucun des deux joueurs ne peut jouer en b1 puisque aucun retournement n'est possible. On compte alors les pions pour déterminer le score. Les cases

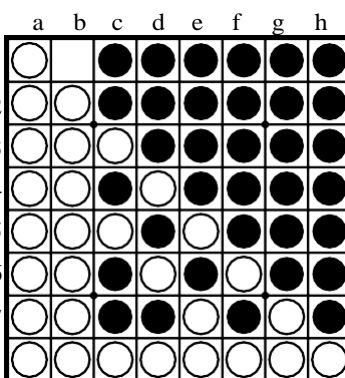
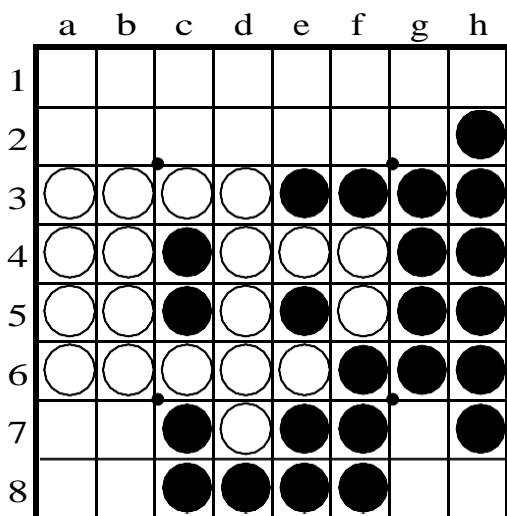


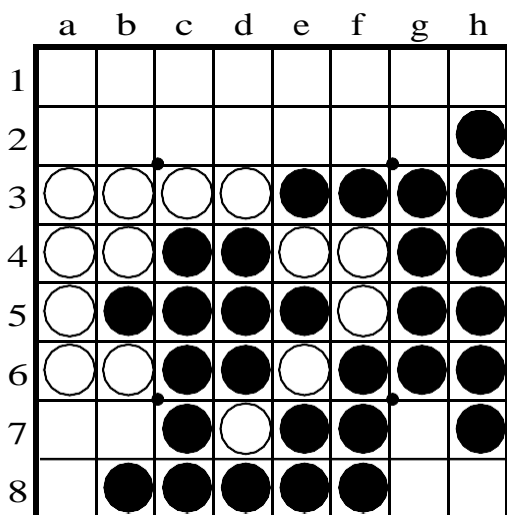
Fig. 10 : la partie est finie !

vides sont attribuées au vainqueur. Dans cette partie, Blanc a 29 pions et Noir 34 et une case vide. Donc Noir gagne 35 à 29.

plausible au sud pour Blanc est c8 et Noir peut alors répondre b8 (voir diagramme 17). On dit que Noir a *gagné un temps* dans la région sud. Blanc doit maintenant jouer le premier au nord.



16. Après c8-d7



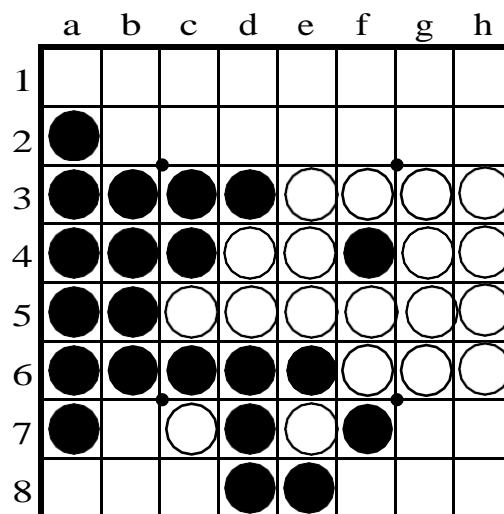
17. Après d7-c8-b8

Une définition rapide du gain d'un temps serait de dire que cela correspond au fait de jouer un coup de plus que l'adversaire dans une région donnée de l'othellier (souvent un bord) et de forcer ainsi l'adversaire à initier le jeu ailleurs (donc à agrandir sa frontière).

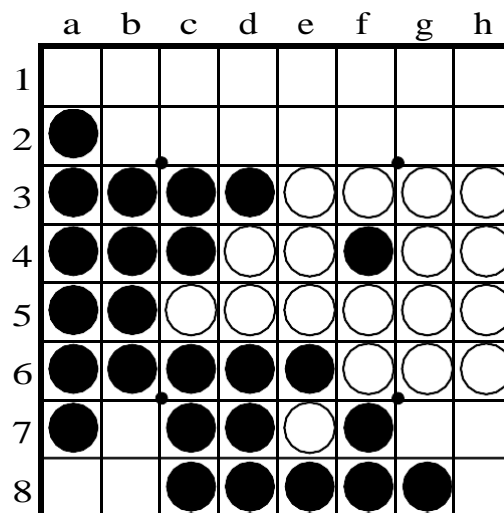
Voici un autre exemple, avec le diagramme 18, de gain de temps sur un bord.

Pour ne pas ouvrir le jeu au nord, Noir veut gagner un temps sur le bord sud. Comment peut-il faire ? Comment choisir entre c8

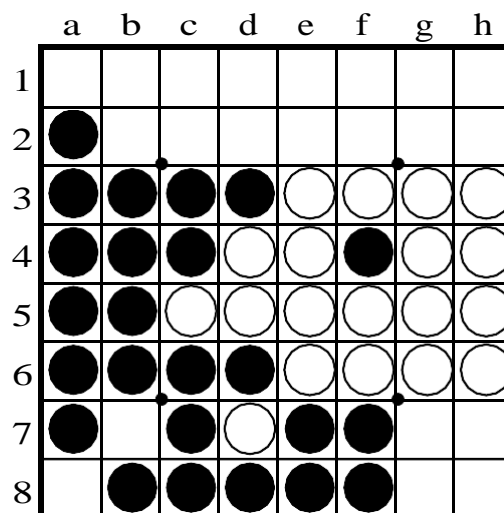
et f8 ? On pourrait croire que ces deux coups sont équivalents avec les deux séquences c8-f8-g8 (voir diagramme 19) et f8-c8-b8 (voir diagramme 20).



18. Noir doit jouer



19. Après c8-f8-g8

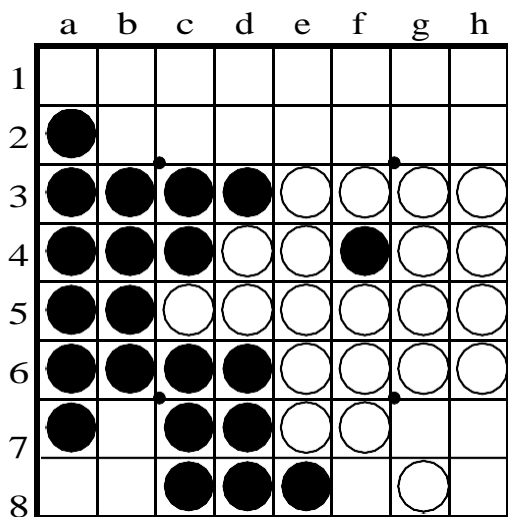


20. Après f8-c8-b8

JEU OTHELLO

Dans les deux cas, Noir gagne le temps qu'il voulait et laisse Blanc ouvrir le premier au nord.

Cependant, si l'on regarde de plus près toutes les réponses de Blanc, on voit que si Noir joue c8, Blanc a un meilleur coup que f8 : il joue g8 ! (voir diagramme 21).



21. Après c8-g8

Alors Noir n'a plus de bon coup au sud (si Noir f8, Blanc reprend le bord avec b8) et doit jouer au nord : il n'a pas gagné le temps voulu.

Dans la position du diagramme 18, Noir devra donc jouer f8 pour gagner un temps.

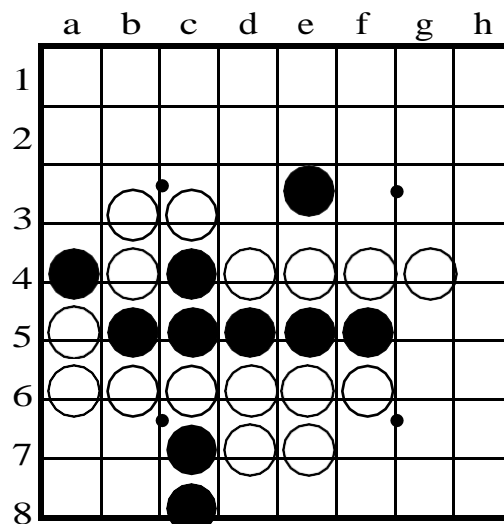
Bien sûr, le gain de plusieurs temps est possible. Inutile de dire que l'adversaire est alors en mauvaise posture puisqu'il va être obligé de jouer plusieurs coups avant que vous n'agrandissiez à votre tour votre frontière.

Voici un exemple dans le diagramme 22.

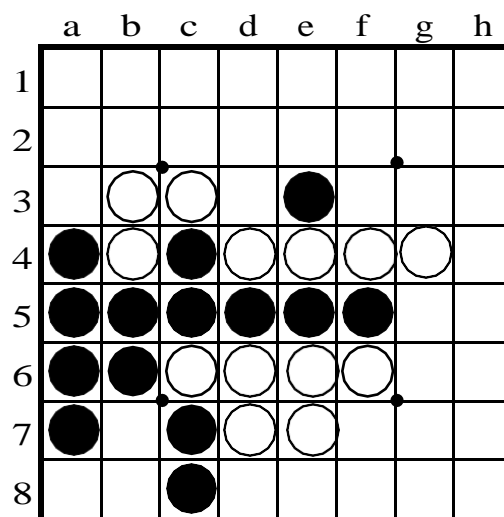
Si Noir joue a7 (voir diagramme 23), Blanc est en très mauvaise posture.

En effet, Blanc a quatre coups raisonnables : g6, f2, e2 et d2, mais il ne pourra pas les jouer tous les quatre. Plus précisément, il ne pourra jouer qu'un coup parmi d2, e2 et f2 car ils retournent tous le même pion noir ; il n'a donc, en fait, que deux coups.

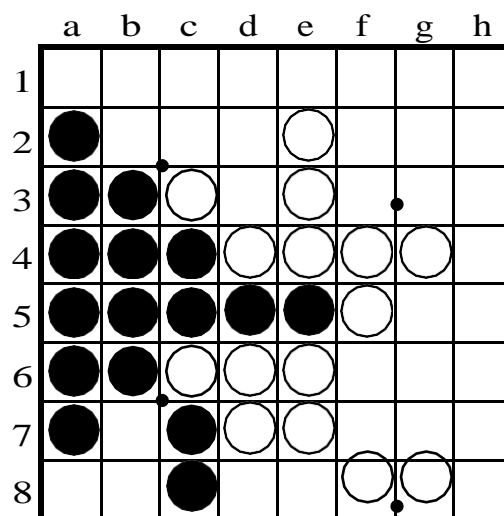
sur le bord ouest : il peut jouer a3 puis a2 ! Et voici la position (diagramme 24) après e2 (par exemple)-a3-g6-a2.



22. Noir doit jouer



23. Après a7



22. Par ailleurs, Noir peut gagner deux temps Après a7-e2-a3-

g6-a2

LE CHEMIN SUIVIE POUR REALISER LE PROJET :

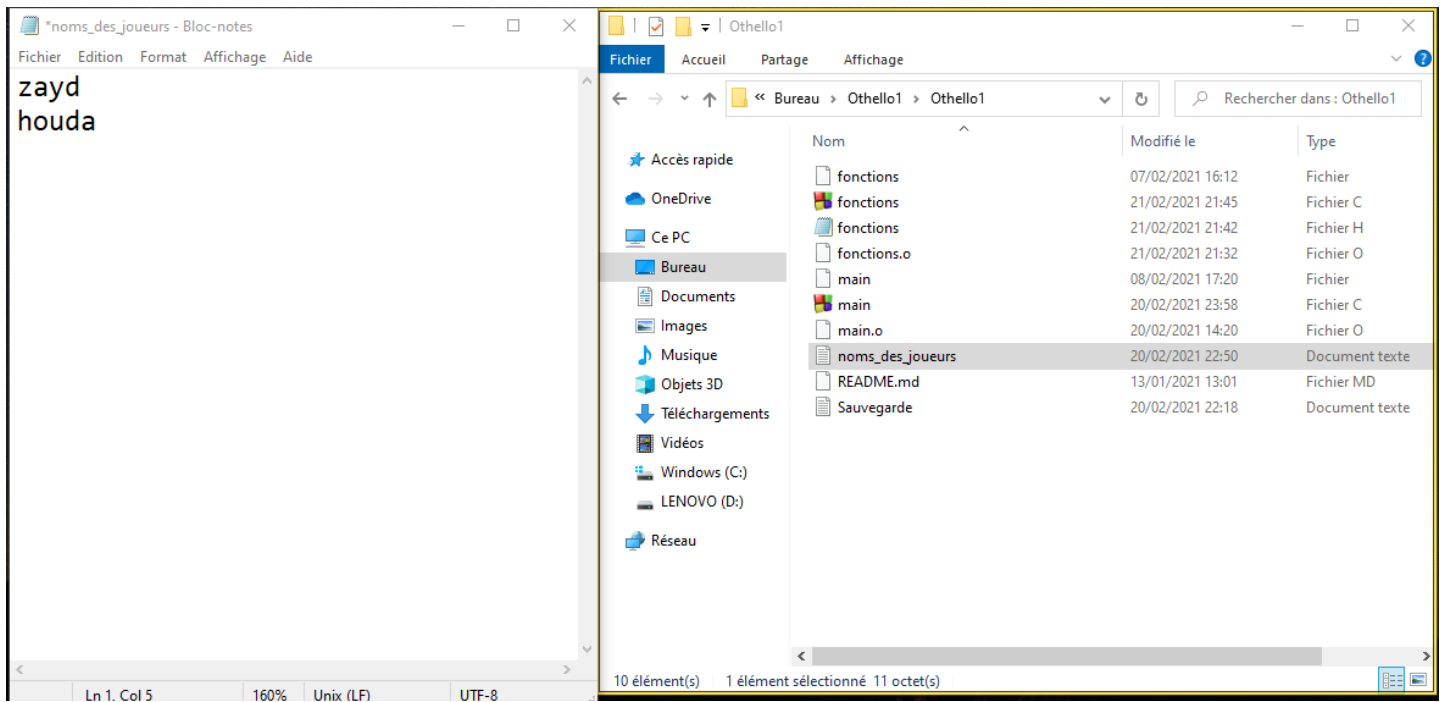
➡ Grille du jeu

➡ Jouer Entres Humains : (J1 Vs J2)

➡ Jouer contre la machine
(intelligence prémitive)

JEU OTHELLO

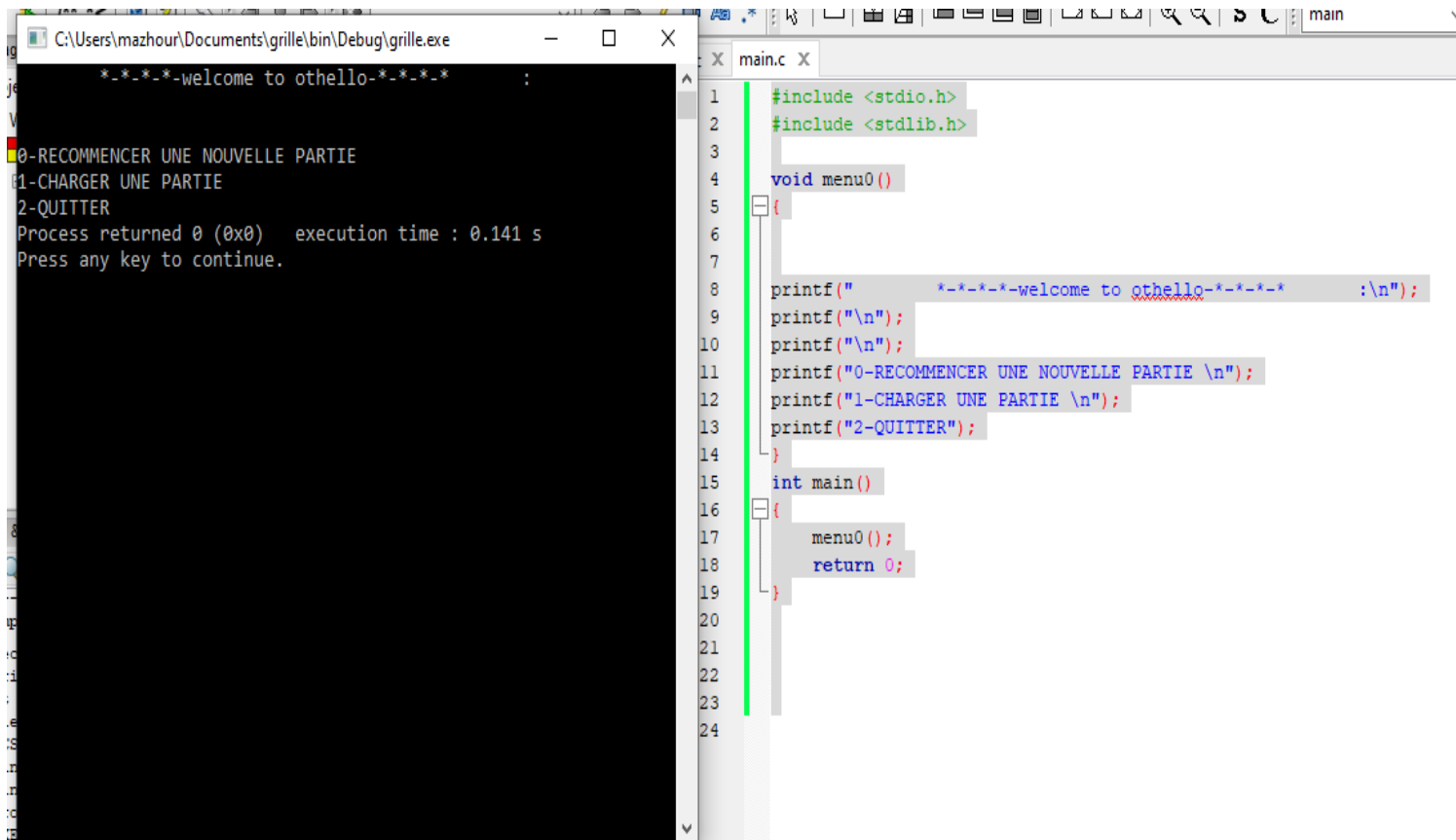
D'abord on a initialiser le Jeu par une fonction qui permet au deux joueurs d'entrer leurs noms, par la suite on a créer une autre fonction qui permet d'enregistrer les nom des joueurs dans un fichier « .txt »,par ailleurs on a définit la fonctionne 'qui commence le premier' retourne le joueur qui va prend le pion noire et commence le jeu.



I- La Grille :

A. Description de la grille :

On a commencé par une fonction qui permet d'afficher le menu principale



The image shows a screenshot of a C program running in a debugger window and its source code in a text editor. The debugger window on the left displays the output of the program, which is a menu for an Othello game. The text editor on the right shows the code for the menu function and the main function.

```
main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void menu0()
5  {
6
7
8      printf("    *-*-*-welcome to othello-*-*-* :\\n");
9      printf("\\n");
10     printf("\\n");
11     printf("0-RECOMMENCER UNE NOUVELLE PARTIE \\n");
12     printf("1-CHARGER UNE PARTIE \\n");
13     printf("2-QUITTER");
14 }
15
16 int main()
17 {
18     menu0();
19     return 0;
20 }
21
22
23
24
```

Output in debugger window:

```
*-*-*-*welcome to othello-*-*-* :
0-RECOMMENCER UNE NOUVELLE PARTIE
1-CHARGER UNE PARTIE
2-QUITTER
Process returned 0 (0x0)   execution time : 0.141 s
Press any key to continue.
```

B-Description des fonctions utilisées :

On a défini deux fonctions permettant d'initialiser le plateau de jeux et affiche le tableau vide dans un premier temps et remplit au milieu après :

JEU OTHELLO

```
60  /* fonction qui initialise le tableau */
61  void init_table(Table T)
62  {
63      int i,j;
64      for (i=0;i<N;i++)
65          for(j=0;j<N;j++)
66              T[i][j]=EMPTY; // On initialise tout le tableau a des espaces vides
67
68      T[3][3]=Noir; //
69      T[4][4]=Noir; // // Initialisation des quatres pions de milieu de plateau
70      T[3][4]=Blanc; //
71      T[4][3]=Blanc; //
72  }
73  /*+++++*/
74  /* fonction qui affiche le tableau */
75  void aff_table(Table T)
76  {
77      int i,j;
78      /* Affichage des chiffres en ligne */
79      printf("\n");
80      for (i=0; i<N; i++)
81          printf(" %d ", i+1);
82
83      /* Definition et affichage du tableau */
84      printf("\n+");
85      for (i=0; i<N; i++)
86          printf("----");
87      printf("\n");
88      for (i=0; i<N; i++) {
89          printf("|");
90          for (j=0; j<N; j++)
91              printf(" %c |", T[i][j]);
92          printf(" %d\n+", 10*(i+1)); // affichage des chiffres en colonne
93          for (j=0; j<N; j++)
94              printf("----");
95          printf("\n");
96      }
```

The screenshot shows a Windows command prompt window titled "C:\Users\mazhour\Documents\les_fonctions_projet_c\main.exe". The output displays an 8x8 grid of the word "vide" (empty) in a monospaced font. Below the grid, it says "Process returned 0 (0x0) execution time : 0.169 s" and "Press any key to continue.". To the right, the source code for "main.c" is visible. It includes headers for `<stdio.h>` and `<stdlib.h>`, defines `EMPTY` as a space character, `N` as 8, and colors for black and white. The `main` function initializes an 8x8 array `T` with `EMPTY` values. It then sets the center four cells: `T[3][3]=Noir;`, `T[4][4]=Noir;`, `T[3][4]=Blanc;`, and `T[4][3]=Blanc;`. Finally, it prints the grid using `printf("vide\t", T[i][j]);` and returns 0.

On a fait un tableau vide dans un premier temps et après on met les quatre pions de centre comme vous voyez :

The screenshot shows the same command prompt window, but the output now displays a board with 'N' and 'B' in the center. The grid is mostly empty, with 'N' at row 3, column 3 and 'B' at row 4, column 4, and vice versa. The execution time is 0.106 s. The source code on the right is similar to the previous one but uses `D` (defined as 8) instead of `N` for the array size and uses `'N'` and `'B'` for the piece characters. The initialization and printing logic remains the same.

La fonction `aff_table` permet d'afficher la grille et démarrer notre jeu :

JEU OTHELLO

```

  1  2  3  4  5  6  7  8
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   | 10
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   | 20
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   | 30
+---+---+---+---+---+---+---+
|   |   |   N | B |   |   |   | 40
+---+---+---+---+---+---+---+
|   |   |   B | N |   |   |   | 50
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   | 60
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   | 70
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   | 80
+---+---+---+---+---+---+---+
C'est votre tour [B]
Entrer 0 pour recommencer !
Veuillez saisir un entier de la forme 'ij' tel que i0 est la ligne et j est la c
olonne associee !
Entrer l'entier :
```

```

  1  2  3  4  5  6  7  8
+---+---+---+---+---+---+---+
|   |   |   |   |   N |   |   | 10
+---+---+---+---+---+---+---+
|   |   N | N | N |   |   |   | 20
+---+---+---+---+---+---+---+
|   |   N | B | B |   |   |   | 30
+---+---+---+---+---+---+---+
|   N | N | B | B | N |   |   | 40
+---+---+---+---+---+---+---+
|   |   N | B |   |   |   |   | 50
+---+---+---+---+---+---+---+
|   |   N | B |   |   |   |   | 60
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   | 70
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   | 80
+---+---+---+---+---+---+---+
NB: Pour l'historique des mouvements vous pouvez remonter en haut
C'est votre tour [B]
Entrer 0 pour recommencer !
Veuillez saisir un entier de la forme 'ij' tel que i0 est la ligne et j est la c
olonne associee !
Entrer l'entier :
```

C-Description des notions des techniques de programmation utilisées :

On a utilisé les fonctions ainsi que la boucle for comme on a vu dans le cours et les tableaux.

II-Entre humains (H VS H) :

A. Description du programme réalisé :

Dans cette partie on va voir H VS H et l'organisation de notre code :

Dans un premier temps, on définit la fonction qui vérifie si les valeurs entrées correspondent à des coups au sein du tableau et une autre qui sert à inverser les coups et enfin on va voir l'ensemble des fonctions qui définit les règles de ce fameux jeu.

B-Description des fonctions utilisées :

La première fonction sert à tester si les valeurs entrées par les deux joueurs correspondent à des coups au sein du tableau en se basant sur la dimension du plateau.

La deuxième va inverser le coup

```

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
/* fonction qui verifie si les valeurs entrees correspondent a des coups au sein du tableau */
Bool case_valide(int ligne,int colonne)
{
    if(ligne >=0 && ligne < N && colonne >=0 && colonne < N ) return true;
    else return false;
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
/* fonction qui inverse le coup ,elle utilisee pour definir les fonctions ci-dessous */
char inverse_coup(char coup)
{
    if(coup==Noir) return Blanc;
    else return Noir;
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
```

JEU OTHELLO

Maintenant on va définir l'ensemble des fonctions qui vérifie si un coup est valide ou non soit (horizontalement, diagonalement, verticalement)

```
/* fonction verifie si le coup est valide verticalement en haut */
Bool valide_verticale_haut(Table T,int ligne,int colonne,char coup) // le parametre coup ici sert
{
    int i=ligne-1,yes=0; // l'entier "yes" permet de verifier si un pion existe
    while(case_valide(i,colonne) && T[i][colonne]==inverse_coup(coup))
    {
        i--; // la recherche de l'extrimite des pions du type "coup"
        yes=1;
    }
    if (case_valide(i, colonne) && T[i][colonne] == coup && yes == 1) return true ;
    return false;
}

/*+++++*/
/* fonction verifie si le coup est valide verticalement en bas */
Bool valide_verticale_bas(Table T,int ligne,int colonne,char coup)
{
    int i=ligne+1,yes=0; // l'entier "yes" permet de verifier si un pion existe
    while(case_valide(i,colonne) && T[i][colonne]==inverse_coup(coup))
    {
        i++; // la recherche de l'extrimite des pions du type "coup"
        yes=1;
    }
    if (case_valide(i,colonne) && T[i][colonne] == coup && yes == 1) return true ;
    return false;
}
/*+++++*/
```



```

/*+++++*/
/* fonction verifie si le coup est valide diagonalement en bas a droite (\) */
Bool valide_diagonale_bas_droit(Table T,int ligne,int colonne,char coup)
{
    int i=ligne+1,j=colonne+1,yes=0;
    while(case_valide(i,j) && T[i][j]==inverse_coup(coup))
    {
        i++;
        j++;
        yes=1;
    }
    if (case_valide(i,j) && T[i][j] == coup && yes == 1) return true ;
    return false;
}

/*+++++*/
/* fonction verifie si le coup est valide diagonalement en bas a gauche (/) */
Bool valide_diagonale_bas_gauche(Table T,int ligne,int colonne,char coup)
{
    int i=ligne+1,j=colonne-1,yes=0;
    while(case_valide(i,j) && T[i][j]==inverse_coup(coup))
    {
        i++;
        j--;
        yes=1;
    }
    if (case_valide(i,j) && T[i][j] == coup && yes == 1) return true ;
    return false;
}

```

JEU OTHELLO

```
/*+++++*/
/* fonction verifie si le coup est valide diagonalement en haut a droite (/) */
Bool valide_diagonale_haut_droit(Table T,int ligne,int colonne,char coup)
{
    int i=ligne-1,j=colonne+1,yes=0;
    while(case_valide(i,j) && T[i][j]==inverse_coup(coup))
    {
        i--;
        j++;
        yes=1;
    }
    if (case_valide(i,j) && T[i][j] == coup && yes == 1) return true ;
    return false;
}

/*+++++*/
/* fonction verifie si le coup est valide diagonalement en haut a gauche (\) */
Bool valide_diagonale_haut_gauche(Table T,int ligne,int colonne,char coup)
{
    int i=ligne-1,j=colonne-1,yes=0;
    while(case_valide(i,j) && T[i][j]==inverse_coup(coup))
    {
        i--;
        j--;
        yes=1;
    }
    if (case_valide(i,j) && T[i][j] == coup && yes == 1) return true ;
    return false;
}

/*+++++*/
/* fonction verifie si le coup est valide horizontalement a droite */
Bool valide_horizontale_droite(Table T,int ligne,int colonne,char coup)
{
    int j=colonne + 1,yes=0; // l'entier "yes" permet de verifier si un pion existe
    while(case_valide(ligne,j) && T[ligne][j]==inverse_coup(coup))
    {
        j++; // la recherche de l'extrimite des pions du type "coup"
        yes=1;
    }
    if (case_valide(ligne,j) && T[ligne][j] == coup && yes == 1) return true ;
    return false;
}

/*+++++*/
/* fonction verifie si le coup est valide horizontalement a gauche */
Bool valide_horizontale_gauche(Table T,int ligne,int colonne,char coup)
{
    int j=colonne - 1,yes=0; // l'entier "yes" permet de verifier si un pion existe
    while(case_valide(ligne,j) && T[ligne][j]==inverse_coup(coup))
    {
        j--; // la recherche de l'extrimite des pions du type "coup"
        yes=1;
    }
    if (case_valide(ligne,j) && T[ligne][j] == coup && yes == 1) return true ;
    return false;
}

/*+++++*/
```

Et après on a rassemblé tous ces fonctionnements dans une seule

```
/*+++++*/
/* définition d'une fonction qui vérifie si un coup est valide. Fonction qui rassemble toutes les fonctions */
Bool coup_valide(Table T,int ligne,int colonne,char coup)
{
    if (!case_valide(ligne, colonne) || T[ligne][colonne]!=EMPTY) return false;
    if(valide_verticale_haut(T,ligne,colonne,coup)
    || valide_verticale_bas(T,ligne,colonne,coup)
    || valide_horizontale_droite(T,ligne,colonne,coup)
    || valide_horizontale_gauche(T,ligne,colonne,coup)
    || valide_diagonale_haut_droit(T,ligne,colonne,coup)
    || valide_diagonale_haut_gauche(T,ligne,colonne,coup)
    || valide_diagonale_bas_droit(T,ligne,colonne,coup)
    || valide_diagonale_bas_gauche(T,ligne,colonne,coup))
    {
        return true;
    }
    return false;
}
/*+++++*/
```

D'une part la fonctionne 'rejouer_ou_non' vérifié si un joueur peut encore jouer ou non :

```
/*+++++*/
/* fonction pour vérifier si un joueur peut jouer ou non . */
Bool rejouer_ou_non(Table T,char coup)
{
    int i,j;
    for(i=0;i<N;i++)
        for(j=0;j<N;j++)
            {if(coup_valide(T,i,j,coup) && T[i][j]==EMPTY) // vérification s'il existe un coup valide
                return true;}
    return false;
}
/*+++++*/
```

D'autre part on a la fonction permettant l'entrée du coup de chaque joueur et affiche la grille :

JEU OTHELLO

```
/*++++++*/
/* fonction permettant l'entree du coup de chaque joueur wt affichage de la grille */
int entrer_son_coup(Table T, char coup) // l'entier k definit le nombre des coups effectues
{
    int n;
    int yes; // pour la verification de recommencement de jeu
    int i,j; // un indice pour enregistrer les valeurs de n dans le table entiers
    int ligne,colonne;
    do
    {
        printf("C'est le tour de joueur : %s [%c]\n",nom_a_afficher(coup)->nom,coup);
        printf("Entrer 0 pour recommencer ! \n");
        printf("Veuillez saisir un entier de la forme 'ij' tel que i0 est la ligne et j est la colonne associee !\n");
        printf("Entrer l'entier : \n"); // on associe a chaque case un entier unique
        scanf("%d",&n); // qui s'agit de la somme de l'entier de la ligne et l'entier de la colonne .
        while(n==0)
        {
            printf("!!!!Est ce que vous voulez vraiment quitter cette partie ? Si oui ,enter encore 0 ! Si non enter un entier quelconque different de 0!!!! \n");
            scanf("%d",&yes);
            if(yes==0)
            {
                init_table(T);
                entrer_noms_joueurs();
                enregistrer_noms_joueurs();
                aff_table(T);
                break;
            }
            else
            {
                aff_table(T);
                printf("C'est le tour de joueur : %s [%c]\n",nom_a_afficher(coup)->nom,coup);
                printf("Veuillez saisir un entier de la forme 'ij' tel que i0 est la ligne et j est la colonne associee !\n");
                printf("Entrer l'entier : \n");
                scanf("%d",&n);
            }
        }
    }
}
```

```

        {
            T[i][j]=coup;
            i--;
            j++;
        } }
    }
    if(valide_diagonale_haut_gauche(T,ligne,colonne,coup)) {
        i=ligne-1;
        j=colonne-1;
        while(case_valide(i,j) && T[i][j]==inverse_coup(coup))
        {   i--;
            j--;
        }
        if(case_valide(i,j) && T[i][j]==coup)
        {   i=ligne-1;
            j=colonne-1;
            while(T[i][j]==inverse_coup(coup))
            {
                T[i][j]=coup;
                i--;
                j--;
            } }
    }
    return n;
//A chaque condition "if" ci-dessus on voit est ce que le coup est valide selon chaque direction pour echanger les pions
}

```

```
        {
            T[i][j]=coup;
            i++;
            j++;
        } }
    }
    if(valide_diagonale_bas_gauche(T,ligne,colonne,coup)) {
        i=ligne+1;
        j=colonne-1;
        while(case_valide(i,j) && T[i][j]==inverse_coup(coup))
        { i++;
          j--;
        }
        if(case_valide(i,j) && T[i][j]==coup)
        { i=ligne+1;
          j=colonne-1;
          while(T[i][j]==inverse_coup(coup))
          {
              T[i][j]=coup;
              i++;
              j--;
          } }
    }
    if(valide_diagonale_haut_droit(T,ligne,colonne,coup)) {
        i=ligne-1;
        j=colonne+1;
        while(case_valide(i,j) && T[i][j]==inverse_coup(coup))
        { i--;
          j++;
        }
        if(case_valide(i,j) && T[i][j]==coup)
        { i=ligne-1;
          j=colonne+1;
          while(T[i][j]==inverse_coup(coup))
          {
              T[i][j]=coup;
```

```

if(valide_horizontale_droite(T, ligne, colonne, coup)) {
    j=colonne+1;
    while(case_valide(ligne, j) && T[ligne][j]==inverse_coup(coup))
        j++;
    if(case_valide(ligne, j) && T[ligne][j]==coup)
    {
        j=colonne+1;
        while(T[ligne][j]==inverse_coup(coup))
        {
            T[ligne][j]=coup;
            j++;
        }
    }
}
if(valide_horizontale_gauche(T, ligne, colonne, coup)) {
    j=colonne-1;
    while(case_valide(ligne, j) && T[ligne][j]==inverse_coup(coup))
        j--;
    if(case_valide(ligne, j) && T[ligne][j]==coup)
    {
        j=colonne-1;
        while(T[ligne][j]==inverse_coup(coup))
        {
            T[ligne][j]=coup;
            j--;
        }
    }
}

if(valide_diagonale_bas_droit(T, ligne, colonne, coup)) {
    i=ligne+1;
    j=colonne+1;
    while(case_valide(i, j) && T[i][j]==inverse_coup(coup))
    {
        i++;
        j++;
    }
    if(case_valide(i, j) && T[i][j]==coup)
    {
        i=ligne+1;
        j=colonne+1;
        while(T[i][j]==inverse_coup(coup))
        {
            T[i][j]=coup;
            i++;
            j++;
        }
    }
}

if(n!=0)
{
    ligne=floor(n/10)-1;          // Ex : la case (4,5) est associe a l'entier 45.
    colonne=n%10-1;
}
}while(!(n>10 && n<90 && n%10!=0) || !coup_valide(T, ligne, colonne, coup));
if(coup_valide(T, ligne, colonne, coup))
    T[ligne][colonne]=coup;
if(valide_verticale_haut(T, ligne, colonne, coup)){
    i=ligne-1;
    while(case_valide(i, colonne) && T[i][colonne]==inverse_coup(coup))
        i--;
    if(case_valide(i, colonne) && T[i][colonne]==coup)
    {
        i=ligne-1;
        while(T[i][colonne]==inverse_coup(coup))
        {
            T[i][colonne]=coup;
            i--;
        }
    }
}
if(valide_verticale_bas(T, ligne, colonne, coup)){
    i=ligne+1;
    while(case_valide(i, colonne) && T[i][colonne]==inverse_coup(coup))
        i++;
    if(case_valide(i, colonne) && T[i][colonne]==coup)
    {
        i=ligne+1;
        while(T[i][colonne]==inverse_coup(coup))
        {
            T[i][colonne]=coup;
            i++;
        }
    }
}
}

```


JEU OTHELLO

Par ailleurs on a défini une fonction qui enregistre le résultat de jeu Dans un fichier ‘.txt’ et ‘partie_terminee’ si une partie est terminée elle affiche le résultats :

```
/*++++++*/
/* la fonction qui sert a enregistrer le resultat d'une partie dans le fichier sauvegarde.txt */
void sauvegarde_result(void)
{
    FILE *fic=fopen("Sauvegarde.txt","a");
    if(fic==NULL)
        exit(1);
    fprintf(fic,"%s (%d) VS %s (%d) ",Joueur1.nom,Joueur1.score,Joueur2.nom,Joueur2.score);
    fprintf(fic,"\n");
    fclose(fic);
}
/*++++++*/
/*++++++*/
/* fonction qui verifie si une partie est terminée ou non et affichage de resultat*/
Bool partie_terminee(Table T)
{
    int i,j, nb_noir=0,nb_blanc=0;
    for(i=0;i<N;i++)
        for(j=0;j<N;j++)
            if(T[i][j]==EMPTY && (rejouer_ou_non(T,Noir) || rejouer_ou_non(T,Blanc) ))
                return false;
            else if(T[i][j]==Blanc) nb_blanc++;
            else if(T[i][j]==Noir) nb_noir++;
    Joueur1.score=nb_blanc;
    Joueur2.score=nb_noir;
    printf("Le jeu est terminée !! \n");
    if(nb_noir<nb_blanc)
        printf("!!!!!! Le joueur : %s [%c] a gagne !!!!!!!\n",Joueur1.nom,Blanc);
    else if(nb_noir>nb_blanc)
        printf("!!!!!! Le joueur : %s [%c] a gagne !!!!!!!\n",Joueur2.nom,Noir);
    else printf("!!!!!! Egalite !!!!!");
    printf("le resultat est %s %d - %d %s \n",Joueur1.nom,Joueur1.score,Joueur2.score,Joueur2.nom );
    return true;
}
/*++++++*/
```

et finalement on a la fonction qui permet de charger une partie en cours :

```
/* 1 fonction permet de charger une partie en cours */
void charger_partie_encours(Table T)
{
    int l;
    char coup=Blanc;
    int ligne,colonne,i,j;
    FILE *fichier=fopen("partie_encours.txt", "r");
    if(fichier==NULL)
    {
        exit(1);
    }
    fscanf(fichier,"%s",Joueur1.nom); /* le fichier partie_encours a comme instruction les deux noms en deux premieres ligne puis des entiers */
    fscanf(fichier,"%s",Joueur2.nom);
    while(1)
    {
        fscanf(fichier,"%d \n",&l);
        ligne=floor(l/10)-1; /* malheureusement on a besoin de repeter toutes les fonctions elementaires pour effectuer les coups */
        colonne=l%10-1; /* c'est due a la construction de entrer_son_coup() qui ne prend pas en parametre l'entier n */
        if(coup_valide(T,ligne,colonne,coup))
            T[ligne][colonne]=coup;
        if(valide_verticale_haut(T,ligne,colonne,coup)){
            i=ligne-1;
            while(case_valide(i,colonne) && T[i][colonne]==inverse_coup(coup))
                i--;
            if(case_valide(i,colonne) && T[i][colonne]==coup)
            {
                i=ligne-1;
                while(T[i][colonne]==inverse_coup(coup))
                {
                    T[i][colonne]=coup;
                    i--;
                }
            }
        }
    }
}
```

JEU OTHELLO

```
/* 1 fonction permet de charger une partie en cours */
void charger_partie_encours(Table T)
{
    int l;
    char coup=Blanc;
    int ligne,colonne,i,j;
    FILE *fichier=fopen("partie_encours.txt","r");
    if(fichier==NULL)
    {
        exit(1);
    }
    fscanf(fichier,"%s",Joueur1.nom); /* le fichier partie_encours a comme instruction les deux noms en deux premieres ligne puis des entiers .
    fscanf(fichier,"%s",Joueur2.nom);
    while(1)
    {
        fscanf(fichier,"%d \n",&l);
        ligne=floor(l/10)-1;          /* malheureusement on a besoin de repeter toutes les fonctions elementaires pour effectuer les coups*
        colonne=l%10-1;              /* c'est due a la construction de entier_son_coup() qui ne prend pas en parametre l'entier n */
        if(coup_valide(T,ligne,colonne,coup))
            T[ligne][colonne]=coup;
        if(valide_verticale_haut(T,ligne,colonne,coup)){
            i=ligne-1;
            while(case_valide(i,colonne) && T[i][colonne]==inverse_coup(coup))
                i--;
            if(case_valide(i,colonne) && T[i][colonne]==coup)
            {
                i=ligne-1;
                while(T[i][colonne]==inverse_coup(coup))
                {
                    T[i][colonne]=coup;
                    i--;
                }
            }
        }
    }
}
```

```

if(valide_diagonale_haut_droit(T,ligne,colonne,coup)) {
    i=ligne-1;
    j=colonne+1;
    while(case_valide(i,j) == T[i][j]==inverse_coup(coup))
    {
        i--;
        j++;
    }
    if(case_valide(i,j) == T[i][j]==coup)
    {
        i=ligne-1;
        j=colonne+1;
        while(T[i][j]==inverse_coup(coup))
        {
            T[i][j]=coup;
            i--;
            j++;
        }
    }
}
if(valide_diagonale_haut_gauche(T,ligne,colonne,coup)) {
    i=ligne-1;
    j=colonne-1;
    while(case_valide(i,j) == T[i][j]==inverse_coup(coup))
    {
        i--;
        j--;
    }
    if(case_valide(i,j) == T[i][j]==coup)
    {
        i=ligne-1;
        j=colonne-1;
        while(T[i][j]==inverse_coup(coup))
        {
            T[i][j]=coup;
            i--;
            j--;
        }
    }
}
coup=inverse_coup(coup);
if(!feof(fichier))
    break;
}
}

```

JEU OTHELLO

```
if(case_valide(ligne,j) && T[ligne][j]==coup)
{
    j=colonne-1;
    while(T[ligne][j]==inverse_coup(coup))
    {
        T[ligne][j]=coup;
        j--;
    }
}

if(valide_diagonale_bas_droit(T,ligne,colonne,coup)) {
    i=ligne+1;
    j=colonne+1;
    while(case_valide(i,j) && T[i][j]==inverse_coup(coup))
    {
        i++;
        j++;
    }
    if(case_valide(i,j) && T[i][j]==coup)
    {
        i=ligne+1;
        j=colonne+1;
        while(T[i][j]==inverse_coup(coup))
        {
            T[i][j]=coup;
            i++;
            j++;
        }
    }
}

if(valide_diagonale_bas_gauche(T,ligne,colonne,coup)) {
    i=ligne+1;
    j=colonne-1;
    while(case_valide(i,j) && T[i][j]==inverse_coup(coup))
    {
        i++;
        j--;
    }
}

if(valide_verticale_bas(T,ligne,colonne,coup)){
    i=ligne+1;
    while(case_valide(i,colonne) && T[i][colonne]==inverse_coup(coup))
        i++;
    if(case_valide(i,colonne) && T[i][colonne]==coup)
    {
        i=ligne+1;
        while(T[i][colonne]==inverse_coup(coup))
        {
            T[i][colonne]=coup;
            i++;
        }
    }
}

if(valide_horizontale_droite(T,ligne,colonne,coup)) {
    j=colonne+1;
    while(case_valide(ligne,j) && T[ligne][j]==inverse_coup(coup))
        j++;
    if(case_valide(ligne,j) && T[ligne][j]==coup)
    {
        j=colonne+1;
        while(T[ligne][j]==inverse_coup(coup))
        {
            T[ligne][j]=coup;
            j++;
        }
    }
}

if(valide_horizontale_gauche(T,ligne,colonne,coup)) {
    j=colonne-1;
    while(case_valide(ligne,j) && T[ligne][j]==inverse_coup(coup))
        j--;
```

```

        if(case_valide(ligne,j) && T[ligne][j]==coup)
        {
            j=colonne-1;
            while(T[ligne][j]==inverse_coup(coup))
            {
                T[ligne][j]=coup;
                j--;
            }
        }

        if(valide_diagonale_bas_droit(T,ligne,colonne,coup)) {
            i=ligne+1;
            j=colonne+1;
            while(case_valide(i,j) && T[i][j]==inverse_coup(coup))
            {
                i++;
                j++;
            }
            if(case_valide(i,j) && T[i][j]==coup)
            {
                i=ligne+1;
                j=colonne+1;
                while(T[i][j]==inverse_coup(coup))
                {
                    T[i][j]=coup;
                    i++;
                    j++;
                }
            }
        }
        if(valide_diagonale_bas_gauche(T,ligne,colonne,coup)) {
            i=ligne+1;
            j=colonne-1;
            while(case_valide(i,j) && T[i][j]==inverse_coup(coup))
            {
                i++;
                j--;
            }
        }
    }
}

```

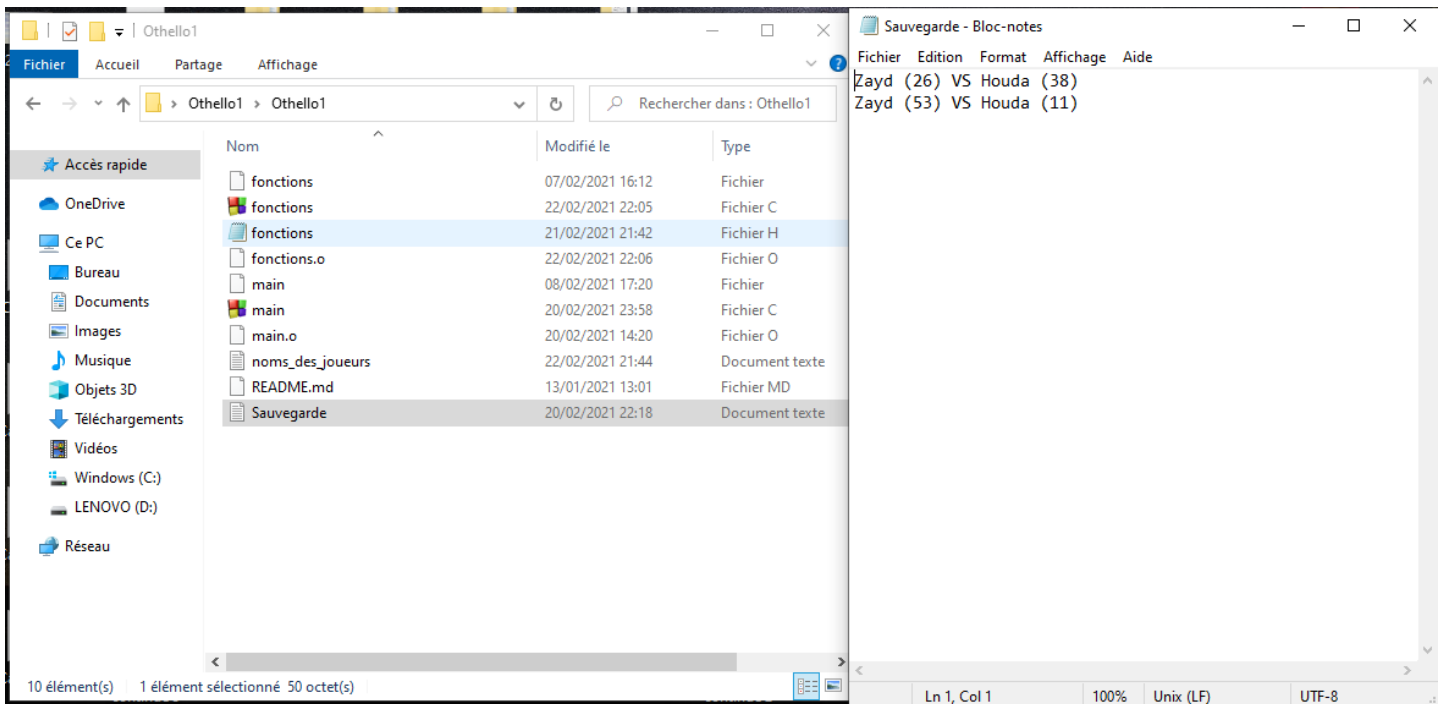
```

if(valide_diagonale_haut_droit(T,ligne,colonne,coup)) {
    i=ligne-1;
    j=colonne+1;
    while(case_valide(i,j) && T[i][j]==inverse_coup(coup))
    {
        i--;
        j++;
    }
    if(case_valide(i,j) && T[i][j]==coup)
    {
        i=ligne-1;
        j=colonne+1;
        while(T[i][j]==inverse_coup(coup))
        {
            T[i][j]=coup;
            i--;
            j++;
        }
    }
}
if(valide_diagonale_haut_gauche(T,ligne,colonne,coup)) {
    i=ligne-1;
    j=colonne-1;
    while(case_valide(i,j) && T[i][j]==inverse_coup(coup))
    {
        i--;
        j--;
    }
    if(case_valide(i,j) && T[i][j]==coup)
    {
        i=ligne-1;
        j=colonne-1;
        while(T[i][j]==inverse_coup(coup))
        {
            T[i][j]=coup;
            i--;
            j--;
        }
    }
}
coup=inverse_coup(coup);
if(feof(fichier))
    break;
}
}

```

le résultat finalement est :

JEU OTHELLO



C-Description des notions des techniques de programmation utilisées :

On a utilisé l'ensemble des techniques avancée dans le langage c soit dans le cours soit en dehors la classe a partire des sites et des vidéos (les boucles (while, if, for, do while), les fichiers, la modularité, les chaines de caractères ...)

1) Avec ordinateur : intelligence primitive A. Description du programme réalisé :

Le programme réalisé est divisé en deux parties :

→ Une partie qui est la fonction **joueur_ordinateur** qui implémente une intelligence simple.

→ Une deuxième partie qui est basée sur les mêmes fonctions de la 1^{ère} partie.

Notre programme est organisé dans les fichiers suivants :

bin	22/02/2021 16:19	Dossier de fichiers	
obj	22/02/2021 16:19	Dossier de fichiers	
fonctions.h	23/02/2021 15:44	Fichier H	2 Ko
joueur_humain.c	23/02/2021 15:45	Fichier C	15 Ko
joueur_ordinateur.c	22/02/2021 22:24	Fichier C	1 Ko
main.c	22/02/2021 22:20	Fichier C	1 Ko
Partie2.cbp	22/02/2021 20:50	Fichier CBP	2 Ko
Partie2.layout	22/02/2021 20:50	Fichier LAYOUT	2 Ko

B. Description des fonctions utilisées :

Les fonctions utilisées sont :

```
Start here X main.c X joueur_humain.c X *joueur_ordinateur.c X *fonctions.h X
34
35 void Joueur_ordinateur(void)
36 void entrer_nom_joueur(void)
37 void enregistrer_nom_joueur(void)
38 void(Qui_commence())
39 void init_table(Table T)
40 void aff_table(Table T)
41 char inverse_coup(char coup)
42 Bool case_valide(int ligne,int colonne)
43 Bool valide_verticale_haut(Table T,int ligne,int colonne,char coup)
44 Bool valide_verticale_bas(Table T,int ligne,int colonne,char coup)
45 Bool valide_horizontale_droite(Table T,int ligne,int colonne,char coup)
46 Bool valide_horizontale_gauche(Table T,int ligne,int colonne,char coup)
47 Bool valide_diagonale_haut_droit(Table T,int ligne,int colonne,char coup)
48 Bool valide_diagonale_haut_gauche(Table T,int ligne,int colonne,char coup)
49 Bool valide_diagonale_bas_droit(Table T,int ligne,int colonne,char coup)
50 Bool valide_diagonale_bas_gauche(Table T,int ligne,int colonne,char coup)
51 Bool coup_valide(Table T,int ligne,int colonne,char coup)
52 Bool rejouer_ou_non(Table T,char coup)
53 int entrer_son_coup(Table T, char coup)
54 Bool partie_terminee(Table T)
55 void sauvegarde_result(void)
```

La fonction : Joueur_ordinateur()

La fonction permet à l'ordinateur de choisir un coup aléatoirement parmi une liste de coups qu'on a stocké dans un tableau et qui lui sont permis dans une situation de jeu donnée.

JEU OTHELLO

```
Start here X main.c X joueur_humain.c X joueur_ordinateur.c X fonctions.h X
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include "fonctions.h"
5
6 void Joueur_ordinateur(void)
7 {
8     // Recherche des cases jouables
9     char* c_possible = malloc(sizeof(char));
10    for (i=0 ; i<8 ; i++)
11    {
12        for (j=0 ; j<8 ; j++)
13        {
14            if (((((((valide_verticale_haut(Table T,int i,int j,char coup)=true || valide_verticale_bas(Table T,int i,int j,char coup)=true) || valide_
15                c_possible=c_possible + T[i][j]
16            }
17        srand(time(NULL));
18        char coup_choisi=c_possible(rand()% sizeof(c_possible));
19        int entrer_son_coup(Table T, char coup_choisi)
20        {
21        }
22    }
```

La fonction : entrer_nom_joueur()

Permet de saisir le nom du joueur.

```
Start here X main.c X joueur_humain.c X joueur_ordinateur.c X fonctions.h X
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include <string.h>
5 #include <math.h>
6 #include "fonctions.h"
7
8 /* la saisie du nom du joueur */
9 void entrer_nom_joueur(void)
10 {
11     char name[30];
12     printf("Donner votre nom: \n");
13     scanf("%s",name);
14 }
```

La fonction : enregistrer_nom_joueur()

```
15 /* la fonction elementaire qui permet d'enregistrer le nom du joueur */
16 void enregistrer_nom_joueur(void)
17 {
18     int i;
19     remove("nom_du_joueur.txt");
20     FILE *fichier=fopen("nom_du_joueur.txt","w");
21     if(fichier==NULL)
22     {
23         exit(1);
24     }
25     fprintf(fichier,"%s\n",Joueur.nom);
26     fclose(fichier);
27 }
```

La fonction : Qui_commence()

Cette fonction permet de faire un tirage au sort pour savoir le joueur qui va commencer.

```

28  /* fonction qui fait un tirage au sort pour voir le joueur qui commence */
29  void(Qui_commence())
30  {
31      srand(time(NULL));
32      int start_first=(rand()% 2)+1;
33      if(start_first==1)
34      {
35          joueur==Noir
36          strcpy(Joueur,Joueur.nom);
37      }
38      else
39      {
40          ordinateur==Noir
41          Joueur_ordinateur();
42      }
43  }
44  }
45  }

```

La fonction : init_table(Table T)

Pour modéliser notre plateau de jeu Othello, nous avons choisi d'utiliser la fonction init_table qui prend comme paramètre la Table T qu'on a créé.

```

46  /* fonction qui initialise le tableau */
47  void init_table(Table T)
48  {
49      int i,j;
50      for (i=0;i<N;i++)
51          for(j=0;j<N;j++)
52              T[i][j]=EMPTY; // On initialise tout le tableau a des espaces vides
53
54
55      T[3][3]=Noir; //
56      T[4][4]=Noir; // // Initialisation des quatres pions de milieu de plateau
57      T[3][4]=Blanc; //
58      T[4][3]=Blanc; //
59  }

```

La fonction : aff_table(Table T)

Cette fonction affiche le tableau initialisé.

```

60  /* fonction qui affiche le tableau */
61  void aff_table(Table T)
62  {
63      int i,j;
64
65      /* Affichage des chiffres en ligne */
66      printf("\n");
67      for (i=0; i<N; i++)
68          printf(" %d ", i+1);
69
70      /* Definition et affichage du tableau */
71
72      printf("\n+");
73      for (i=0; i<N; i++)
74          printf("----");
75      printf("\n");
76      for (i=0; i<N; i++) {
77          printf("|");
78          for (j=0; j<N; j++)
79              printf(" %c |", T[i][j]);
80          printf(" %d\n", 10*(i+1)); // affichage des chiffres en colonne
81          for (j=0; j<N; j++)
82              printf("----");
83          printf("\n");
84      }
85  }

```

La fonction : inverse_coup(char coup)

Cette fonction prend le caractère coup comme argument et renvoie un caractère. Elle permet d'inverser les coups.

JEU OTHELLO

```
84     }  
85 }  
86 /* fonction qui inverse le coup */  
87 char inverse_coup(char coup)  
88 {  
89     if(coup==Noir) return Blanc;  
90     else return Noir;  
91 }  
92 }  
93 }
```

La fonction : case_valide(ligne,collone)

Elle vérifie si la valeur entrée correspond à une case dans le tableau.

```
93  
94 /* fonction qui verifie si les valeurs entrees correspondent a des coups au sein du tableau */  
95 Bool case_valide(int ligne,int colonne)  
96 {  
97     if(ligne >=0 && ligne < N && colonne >=0 && colonne < N ) return true;  
98     else return false;  
99 }  
100 }
```

Les fonctions : valide_verticale_haut(Table T,int ligne,int colonne,char coup) ;
valide_verticale_bas(Table T,int ligne,int colonne,char coup) ; valide_horizontale_droite(Table T,int
ligne,int colonne,char coup) ; valide_horizontale_gauche(Table T,int ligne,int colonne,char coup) ;
valide_diagonale_haut_droit(Table T,int ligne,int colonne,char coup) ;
valide_diagonale_haut_gauche(Table T,int ligne,int colonne,char coup) ;
valide_diagonale_bas_droit(Table T,int ligne,int colonne,char coup) ;
valide_diagonale_bas_gauche(Table T,int ligne,int colonne,char coup) ; coup_valide(Table T,int
ligne,int colonne,char coup)

Ces fonctions vérifient si le coup joué est valide soit verticalement ou horizontalement ou diagonalement.

```
94 /* fonction qui verifie si les valeurs entrees correspondent a des coups au sein du tableau */  
95 Bool case_valide(int ligne,int colonne)  
96 {  
97     if(ligne >=0 && ligne < N && colonne >=0 && colonne < N ) return true;  
98     else return false;  
99 }  
100 /* fonction verifie si le coup est valide verticalement en haut */  
101 Bool valide_verticale_haut(Table T,int ligne,int colonne,char coup) // le parametre coup ici sert a identifier le joueur en question  
102 {  
103     int i=ligne-1,yes=0; // l'entier "yes" permet de verifier si un pion existe  
104     while(case_valide(i,colonne) && T[i][colonne]==inverse_coup(coup))  
105     {  
106         i--; // la recherche de l'extrimite des pions du type "coup"  
107         yes=1;  
108     }  
109     if (case_valide(i,colonne) && T[i][colonne] == coup && yes == 1) return true ;  
110     return false;  
111 }  
112 }
```

```
113  
114 /*+++++*/  
115 /* fonction verifie si le coup est valide verticalement en bas */  
116 Bool valide_verticale_bas(Table T,int ligne,int colonne,char coup)  
117 {  
118     int i=ligne+1,yes=0; // l'entier "yes" permet de verifier si un pion existe  
119     while(case_valide(i,colonne) && T[i][colonne]==inverse_coup(coup))  
120     {  
121         i++; // la recherche de l'extrimite des pions du type "coup"  
122         yes=1;  
123     }  
124     if (case_valide(i,colonne) && T[i][colonne] == coup && yes == 1) return true ;  
125     return false;  
126 }
```

```

126 -;
127 /*+++++++*/
128 /* fonction verifie si le coup est valide horizontalement a droite */
129 Bool valide_horizontale_droite(Table T,int ligne,int colonne,char coup)
130 {
131     int j=colonne + 1,yes=0; // l'entier "yes" permet de verifier si un pion existe
132     while(case_valide(ligne,j) && T[ligne][j]==inverse_coup(coup))
133     {
134         j++; // la recherche de l'extremite des pions du type "coup"
135         yes=1;
136     }
137     if (case_valide(ligne,j) && T[ligne][j] == coup && yes == 1) return true ;
138     return false;
139 }
140 /*+++++++*/

140 /*+++++++*/
141 /* fonction verifie si le coup est valide horizontalement a gauche */
142 Bool valide_horizontale_gauche(Table T,int ligne,int colonne,char coup)
143 {
144     int j=colonne - 1,yes=0; // l'entier "yes" permet de verifier si un pion existe
145     while(case_valide(ligne,j) && T[ligne][j]==inverse_coup(coup))
146     {
147         j--; // la recherche de l'extremite des pions du type "coup"
148         yes=1;
149     }
150     if (case_valide(ligne,j) && T[ligne][j] == coup && yes == 1) return true ;
151     return false;
152 }
153 /*+++++++*/

153 /*+++++++*/
154 /* fonction verifie si le coup est valide diagonalement en haut a droite (/) */
155 Bool valide_diagonale_haut_droit(Table T,int ligne,int colonne,char coup)
156 {
157     int i=ligne-1,j=colonne+1,yes=0;
158     while(case_valide(i,j) && T[i][j]==inverse_coup(coup))
159     {
160         i--;
161         j++;
162         yes=1;
163     }
164     if (case_valide(i,j) && T[i][j] == coup && yes == 1) return true ;
165     return false;
166 }
167

167 /*+++++++*/
168 /* fonction verifie si le coup est valide diagonalement en haut a gauche (\) */
169 Bool valide_diagonale_haut_gauche(Table T,int ligne,int colonne,char coup)
170 {
171     int i=ligne-1,j=colonne-1,yes=0;
172     while(case_valide(i,j) && T[i][j]==inverse_coup(coup))
173     {
174         i--;
175         j--;
176         yes=1;
177     }
178     if (case_valide(i,j) && T[i][j] == coup && yes == 1) return true ;
179     return false;
180 }
181
182

182 /*+++++++*/
183 /* fonction verifie si le coup est valide diagonalement en bas a droite (\) */
184 Bool valide_diagonale_bas_droit(Table T,int ligne,int colonne,char coup)
185 {
186     int i=ligne+1,j=colonne+1,yes=0;
187     while(case_valide(i,j) && T[i][j]==inverse_coup(coup))
188     {
189         i++;
190         j++;
191         yes=1;
192     }
193     if (case_valide(i,j) && T[i][j] == coup && yes == 1) return true ;
194     return false;
195 }
196
197

```

JEU OTHELLO

```
198 /*+++++*/
199 /* fonction verifie si le coup est valide diagonalement en bas a gauche (\) */
200 Bool valide_diagonale_bas_gauche(Table T,int ligne,int colonne,char coup)
201 {
202     int i=ligne+1,j=colonne-1,yes=0;
203     while(case_valide(i,j) && T[i][j]==inverse_coup(coup))
204     {
205         i++;
206         j--;
207         yes=1;
208     }
209     if (case_valide(i,j) && T[i][j] == coup && yes == 1) return true ;
210     return false;
211 }
212
213
214
215
216 /* definition d'une fonction qui verifie si un coup est valide. Fonction qui rassemble toutes les fonctions predefinites ci-dessus */
217 Bool coup_valide(Table T,int ligne,int colonne,char coup)
218 {
219     if (!case_valide(ligne, colonne) || T[ligne][colonne]!=EMPTY) return false;
220     if(valide_verticale_haut(T,ligne,colonne,coup)
221     || valide_verticale_bas(T,ligne,colonne,coup)
222     || valide_horizontale_droite(T,ligne,colonne,coup)
223     || valide_horizontale_gauche(T,ligne,colonne,coup)
224     || valide_diagonale_haut_droit(T,ligne,colonne,coup)
225     || valide_diagonale_haut_gauche(T,ligne,colonne,coup)
226     || valide_diagonale_bas_droit(T,ligne,colonne,coup)
227     || valide_diagonale_bas_gauche(T,ligne,colonne,coup))
228     {
229         return true;
230     }
231     return false;
232 }
233
234
```

La fonction : rejouer_ou_non(Table T,char coup)

Cette fonctions renvoie un boléen et permet de savoir si le joueur peut jouer ou non.

```
235 /* fonction pour verifier si le joueur peut jouer ou non . */
236 Bool rejouer_ou_non(Table T,char coup)
237 {
238     int i,j;
239     for(i=0;i<N;i++)
240     for(j=0;j<N;j++)
241     {if(coup_valide(T,i,j,coup) && T[i][j]==EMPTY) // verification s'il existe un coup valide
242         return true;}
243     return false;
244 }
245
```

La fonction : entrer_son_coup()

Elle permet au joueur d'entrer son coup et affiche la grille.

```
246 /* fonction permettant l'entree du coup de chaque joueur et affiche la grille */
247 int entrer_son_coup(Table T, char coup) // l'entier k definit le nombre des coups effectues
248 {
249     int n;
250     int yes; // pour la verification de recommencement de jeu
251     int i,j; // un indice pour enregistrer les valeurs de n dans la table.entiers
252     int ligne,colonne;
253     do
254     {
255         printf("C'est le tour de JOUEUR : %s [%s]\n",nom_a_afficher(coup)->nom,coup);
256         printf("Entrez 0 pour RESCOMMENCER ! \n");
257         printf("Veuillez saisir un entier de la forme 'ij' tel que i0 est la ligne et j est la colonne associee !\n");
258         printf("Entrez l'entier : \n"); // on associe a chaque case un entier unique
259         scanf("%d",&n); // qui s'agit de la somme de l'entier de la ligne et l'entier de la colonne .
260         while(n==0)
261         {
262             printf("!!!!!!Est ce que vous voulez vraiment quitter cette partie ? Si oui , enter encore 0 ! Si non enter un entier quelconque different de 0!!!! \n");
263             scanf("%d",&yes);
264             if(yes==0)
265             {
266                 init_table(T);
267                 entrer_nom_joueur();
268                 enregistrer_nom_joueur();
269                 aff_table(T);
270                 break;
271             }
272         }
273     }
274 }
```

```

270         break;
271     }
272     else
273     {
274         aff_table(T);
275         printf("C'est le tour de joueur : %s (%c)\n", nom_a_afficher(coup)->nom, coup);
276         printf("Veuillez saisir un entier de la forme 'ij' tel que i() est la ligne et j est la colonne associée !\n");
277         printf("Entrez l'entier : \n");
278         scanf("%d", &n);
279     }
280 }
281 if(n!=0)
282 {
283     ligne=floor(n/10)-1;          // Ex : la case (4,5) est associée à l'entier 45.
284     colonne=n%10-1;
285 }
286 }while(!(n>10 && n<90 && n%10!=0) || !coup_valide(T, ligne, colonne, coup));
287 if(coup_valide(T, ligne, colonne, coup))
288     T[ligne][colonne]=coup;
289 if(valide_verticale_haut(T, ligne, colonne, coup)){
290     i=ligne-1;
291     while(case_valide(i, colonne) && T[i][colonne]==inverse_coup(coup))
292         i--;
293     if(case_valide(i, colonne) && T[i][colonne]==coup)
294     { i=ligne-1;
295       while(T[i][colonne]==inverse_coup(coup))
296       {
297         if(case_valide(i, colonne) && T[i][colonne]==coup)
298         { i=ligne-1;
299           while(T[i][colonne]==inverse_coup(coup))
300           {
301             T[i][colonne]=coup;
302             i--;
303           }
304         }
305       }
306     }
307     if(valide_verticale_bas(T, ligne, colonne, coup)){
308         i=ligne+1;
309         while(case_valide(i, colonne) && T[i][colonne]==inverse_coup(coup))
310             i++;
311         if(case_valide(i, colonne) && T[i][colonne]==coup)
312         { i=ligne+1;
313           while(T[i][colonne]==inverse_coup(coup))
314           {
315             T[i][colonne]=coup;
316             i++;
317           }
318         }
319     }
320     if(valide_horizontale_droite(T, ligne, colonne, coup)) {
321         j=colonne+1;
322         while(case_valide(ligne, j) && T[ligne][j]==inverse_coup(coup))
323             j++;
324         if(case_valide(ligne, j) && T[ligne][j]==coup)
325         { j=colonne+1;
326           while(T[ligne][j]==inverse_coup(coup))
327           {

```


JEU OTHELLO

```
318     { j=colonne+1;
319       while(T[ligne][j]==inverse_coup(coup))
320       {
321         T[ligne][j]=coup;
322         j++;
323       }
324     }
325     if(valide_horizontale_gauche(T, ligne, colonne, coup)) {
326       j=colonne-1;
327       while(case_valide(ligne, j) && T[ligne][j]==inverse_coup(coup))
328       j--;
329       if(case_valide(ligne, j) && T[ligne][j]==coup)
330       { j=colonne-1;
331         while(T[ligne][j]==inverse_coup(coup))
332         {
333           T[ligne][j]=coup;
334           j--;
335         }
336       }
337     }
338     if(valide_diagonale_bas_droit(T, ligne, colonne, coup)) {
339       i=ligne+1;
340       j=colonne+1;
341       while(case_valide(i, j) && T[i][j]==inverse_coup(coup))
342       { i++;
343         j++;
344       }
345       if(case_valide(i, j) && T[i][j]==coup)
```

```
344     }
345     if(case_valide(i, j) && T[i][j]==coup)
346     { i=ligne+1;
347       j=colonne+1;
348       while(T[i][j]==inverse_coup(coup))
349       {
350         T[i][j]=coup;
351         i++;
352         j++;
353       }
354     }
355     if(valide_diagonale_bas_gauche(T, ligne, colonne, coup)) {
356       i=ligne+1;
357       j=colonne-1;
358       while(case_valide(i, j) && T[i][j]==inverse_coup(coup))
359       { i++;
360         j--;
361       }
362       if(case_valide(i, j) && T[i][j]==coup)
363       { i=ligne+1;
364         j=colonne-1;
365         while(T[i][j]==inverse_coup(coup))
366         {
367           T[i][j]=coup;
368           i++;
369           j--;
370         }
371     }
```

```

370         } }
371     }
372     if(valide_diagonale_haut_droit(T,ligne,colonne,coup)) {
373         i=ligne-1;
374         j=colonne+1;
375         while(case_valide(i,j) && T[i][j]==inverse_coup(coup))
376         { i--;
377           j++;
378         }
379         if(case_valide(i,j) && T[i][j]==coup)
380         { i=ligne-1;
381           j=colonne+1;
382           while(T[i][j]==inverse_coup(coup))
383           {
384               T[i][j]=coup;
385               i--;
386               j++;
387           } }
388     }
389     if(valide_diagonale_haut_gauche(T,ligne,colonne,coup)) {
390         i=ligne-1;
391         j=colonne-1;
392         while(case_valide(i,j) && T[i][j]==inverse_coup(coup))
393         { i--;
394           j--;
395         }
396         if(case_valide(i,j) && T[i][j]==coup)
397         { i=ligne-1;
398           j=colonne-1;
399           while(T[i][j]==inverse_coup(coup))
400           {
401               T[i][j]=coup;
402               i--;
403               j--;
404           } }
405     }
406     return n;
407     /* chaque condition "if" ci-dessus on voit est ce que le coup est valide selon chaque direction pour echanger les pions
408  */

```

La fonction : `partie_terminee(Table T)` (vérifie si la partie est terminée)

```

409  /* fonction qui verifie si une partie est terminee ou non et affichage de resultat */
410  Bool partie_terminee(Table T)
411  {
412      int i,j, nb_noir=0,nb_blanc=0;
413      for(i=0;i<N;i++)
414          for(j=0;j<N;j++)
415              if(T[i][j]==EMPTY && (rejouer_ou_non(T,joueur) || rejouer_ou_non(T,ordinateur) ))
416                  return false;
417              else if(T[i][j]==Blanc) nb_blanc++;
418              else if(T[i][j]==Noir) nb_noir++;
419      Joueur.score=nb_blanc;
420      Joueur.score=nb_noir;
421      printf("Le jeu est terminée !! \n");
422      if(nb_noir<nb_blanc)
423          printf("!!!!!! Le joueur : %s [%c] a gagne !!!!!!!\n",Joueur.nom,joueur);
424      else if(nb_noir>nb_blanc)
425          printf("!!!!!! L'ordinateur : [%c] a gagne !!!!!!!\n",ordinateur);
426      else printf("!!!!!! Egalité !!!!!");
427      printf("le resultat est %s %d \n",Joueur.nom,Joueur.score);
428      return true;
429  }

```

JEU OTHELLO

La fonction : sauvegarde_result() qui sauvegarde les résultats.

```
430  /*++++++*/
431  /* la fonction qui sert a enregistrer le resultat d'une partie dans le fichier sauvegarde.txt */
432  void sauvegarde_result(void)
433  {
434      FILE *fic=fopen("Sauvegarde.txt", "a");
435      if(fic==NULL)
436          exit(1);
437      fprintf(fic, "%s (%d) VS %s (%d) ", Joueur.nom, Joueur.score);
438      fprintf(fic, "\n");
439      fclose(fic);
440  }
441
442
```

A. Description des notions des techniques de programmation utilisées :

Les techniques de programmation utilisées sont : les fonctions, les tableaux, les boucles (while, if, for...), les fichiers, la modularité, les chaines de caractères etc.

B. Les fonctionnalités non réalisées :

On a choisi d'utiliser les fonctions de la 1^{ère} partie au lieu de faire une seule fonction joueur_humain().

2) Avec ordinateur : intelligence moyenne

A. Description du programme réalisé :

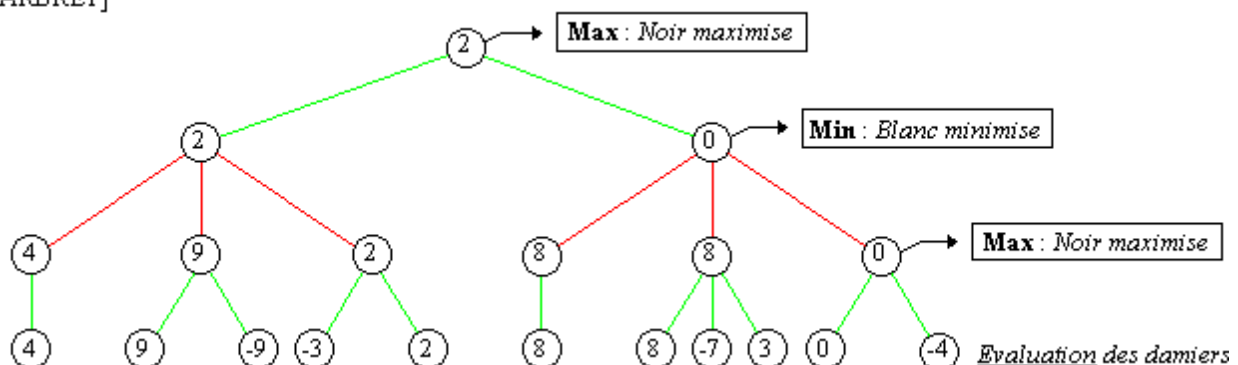
L'idée c'est parcourir par le programme un espace de recherche constitué des positions futures potentiellement atteintes par les joueurs pour anticiper la meilleure séquence à jouer. Puisque on a plusieurs positions possible la machine ira à la position qui vas renvoie le maximum des pions possibles. D'où l'idée d'utiliser les arbres La méthode générale est la suivante : à partir d'une position (racine de l'arbre) on génère tous les coups possibles pour le programme. Puis à partir de ces nouvelles positions on génère toutes les réponses possibles pour l'adversaire. Il est appelé **algorithme minmax**. Son principe est simple : il cherche le meilleur

compromis parmi plusieurs coups jouables, pour un joueur donné.

Autrement dit, il va chercher en profondeur trois par exemple, le meilleur pour les Blancs). Ainsi, quand la machine joue un coup, elle se garantit d'avoir dans le Pème coups à venir (P=profondeur), au moins une note égale à l'un des coups joués à la profondeur P.

L'exemple suivant qui développe un arbre - Min Max, traduit bien ce principe. Nous sommes en profondeur 3 et chaque noeud représente un damier, et donc un jeu différent.

[ARBRE1]



B. Les fonctionnalités non réalisées :

malheureusement on va réécrire les fonctions élémentaires décrites dans la première partie (les règles de jeu) pour jouer.

I. Efforts et difficultés :

1. Efforts :

- ✓ Nous avons réussi à écrire une fonction qui entre un coup pour prédire le meilleur coup possible.
- ✓ On a joué le jeu plusieurs fois pour bien comprendre ce qu'il faut faire.
- ✓ On a élargi notre connaissances dans le langage c plus que celles acquies à la classe pour pouvoir réaliser le projet.
- ✓ Nous avons réussi à réaliser la 1^{ère} et la deuxième partie du jeu en comprenant les différents détails du jeu.

2. Difficultés rencontrés :

Malheureusement on devait réécrire dans la 3^{ème} partie les fonctions élémentaires décrites dans la première partie (les règles de jeu) pour jouer, ainsi plusieurs fonctions en relation avec l'algorithme minmax, ce qui était difficile pour nous.

