



ECOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE ET D'ANALYSE DES  
SYSTÈMES - RABAT

---

## Rapport de Projet de programmation : Othello

---

*Réalisé par :*

Mohamed Zouheir SENAN  
Mohamed Amine RHIOUI

*Encadré par :*

Pr.Mahmoud EL HAMLAOUI





## *Remerciements :*

Nous voudrons tout d'abord adresser toute notre gratitude à notre professeur Abdellatif El Faker et Mahmoud EL EL HAMLAOUI, pour sa confiance, sa disponibilité et surtout cette opportunité pour bien maîtriser le langage de programmation C et initier notre carrière par un aussi beau sujet.



# Table des matières

<b>1 Présentation du projet</b>	<b>1</b>
1.1 Sujet . . . . .	1
1.2 Principe du jeu . . . . .	2
1.3 description de problème . . . . .	3
1.4 présentation du rapport . . . . .	3
<b>2 Analyse et conception</b>	<b>4</b>
2.1 Cahier de charges . . . . .	4
2.1.1 livrable1 . . . . .	4
2.1.2 livrable2 . . . . .	4
2.1.3 livrable3 . . . . .	4
2.2 Modélisation . . . . .	4
2.2.1 Conception de la planche d'Othello . . . . .	4
2.2.2 modelisation du joueur . . . . .	5
2.2.3 sauvegarde des données des joueurs . . . . .	5
2.3 Etapes de résolutions . . . . .	5
2.3.1 Algorithmes de logique de jeux . . . . .	5
2.3.1.1 Algorithme des cases possibles d'un jeu . . . . .	5
2.3.1.2 Algorithme de transformations des pions . . . . .	6
2.3.1.3 Algorithme de calculnum . . . . .	6
2.3.1.4 minimax . . . . .	7
2.3.1.5 alphabeta . . . . .	7
2.3.1.6 algorithme de testvalue . . . . .	7
2.3.1.7 jeu Avec des positions aléatoires . . . . .	8
2.3.2 Algorithmes des donnee de utilisateur . . . . .	8
2.3.2.1 Registerement de data . . . . .	8
2.3.2.2 Charge et Sauvegarder de table de jeu . . . . .	8
2.3.2.3 Liste des dix meilleurs scores . . . . .	8
2.3.2.4 Historique des movemennt de jeux . . . . .	8
<b>3 Réalisation du projet</b>	<b>9</b>
3.0.1 Fenêtre d'utilisateur . . . . .	9
3.0.1.1 Description . . . . .	9
3.0.2 les données d'utilisateur . . . . .	10
3.0.2.1 Description . . . . .	10
3.0.3 Fenêtre d'enregistrement . . . . .	10
3.0.3.1 Description . . . . .	10
3.0.4 Fenêtre de Gestion des donnees . . . . .	11
3.0.4.1 Description . . . . .	11
3.0.5 Fenêtre de choix des jeux . . . . .	11
3.0.5.1 Description . . . . .	11
3.0.6 Fenêtre de choix de couleur . . . . .	12
3.0.6.1 Description . . . . .	12
3.0.7 Fenêtre de jeu . . . . .	12
3.0.7.1 Description . . . . .	12
3.0.8 l'historique des mouvements . . . . .	13
3.0.8.1 Description . . . . .	13
3.0.9 dix meilleurs scores . . . . .	13
3.0.9.1 Description . . . . .	13

<b>4 Conclusion</b>	<b>14</b>
4.0.1 Attendu du projet . . . . .	14
4.0.1.1 Livrable 1 : . . . . .	14
4.0.1.2 Livrable 2 : . . . . .	14
4.0.1.3 Livrable 3 : . . . . .	14
4.0.2 les parties réaliser du projet . . . . .	14
4.0.3 Contrainte de temps . . . . .	14
4.0.4 Interface graphique SDL . . . . .	15
4.0.5 Rédaction du rapport . . . . .	15
4.0.6 github . . . . .	15
4.0.7 Amelioration de Projet . . . . .	15



# Chapitre 1

## Présentation du projet

PROJET C s'agit de produire un programme d'environ 500 lignes de code afin de valider les compétences des cours : « Algorithmique », « Technique de programmation » et « Structures de données ». Le programme correspond à 20 heures de travail effectives en langage C. Les étudiants travaillent en binôme et bénéficient des conseils d'un professeur encadrant<sup>1</sup>

### 1.1 Sujet

Le Reversi était créé dans les années 1880, sous l'impulsion des anglais L.Waterman et JW.Mollet, apparaît comme le premier jeu ayant utilisé les principes de retournements de pions, pour mettre à son propre profit les pions capturés à l'adversaire. Jusqu'alors, les prises par substitution(exemple des échecs) ou par saut (exemple des dames) régnait en maîtres, cependant après la réinvention de ce jeu en 1973 par le japonais G.Hasegawa, qu'il sera nommé par la suite Othello, il est maintenant pratiqué dans de très nombreux pays dans le monde. Othello possède la caractéristique de proposer une très grande profondeur stratégique, c'est un jeu de compétition qui a reconnu une très forte notoriété auprès du grand public due notamment à la simplicité de ses règles.



FIGURE 1.1 – Othello Game

## 1.2 Principe du jeu

Un joueur commence la partie<sup>2</sup>. Puis les joueurs jouent à tour de rôle, chacun étant tenu de capturer des pions adverses lors de son mouvement. Si un joueur ne peut pas capturer de pion(s) adverse(s), il est forc   de passer son tour. Si aucun des deux joueurs ne peut jouer, ou si l'othellier ne comporte plus de case vide, la partie s'arrête. Le gagnant en fin de partie est celui qui poss  de le plus de pions.

La capture de pions survient lorsqu'un joueur place un de ses pions     l'extr  mit   d'un alignement de pions adverses contigus et dont l'autre extr  mit   est d  j   occup  e par un de ses propres pions. Les alignements consid  r  s peuvent  tre une colonne, une ligne, ou une diagonale. Si le pion nouvellement plac   vient fermer plusieurs alignements, il capture tous les pions adverses des lignes ainsi ferm  es. La capture se traduit par le retournement des pions captur  s. Ces retournements n'entra  nent pas d'effet de capture en cascade : seul le pion nouvellement pos   est pris en compte. Par exemple, la figure de gauche ci-dessous montre la position de d  part. La figure centrale montre les 4 cases o   Noir peut jouer, gr  ce   la capture d'un pion Blanc. Enfin, la figure de droite montre la position r  sultante si Noir joue en d3. Le pion Blanc d4 a  t   captur   (retourn  ), devenant ainsi un pion Noir.



FIGURE 1.2 – Principe de jeu

## 1.3 description de problème

Le but de ce projet est de programmer un jeu de Othello. Le jeu de Othello est un jeu combinatoire abstrait, sans hasard, avec information complète et parfaite. Deux joueurs, “noir” et “blanc” s’affrontent. Le jeu se joue sur un plateau de 64 cases (8x8) ; chaque joueur joue à tour de rôle en posant une pierre sur une case libre. Si un joueur ne peut poser de pierre alors il doit passer (il ne joue pas et c'est au tour de son adversaire). Si les deux joueurs ne peuvent plus poser de pions alors la partie est finie et le joueur ayant le plus grand nombre de pierres de sa couleur gagne. Un coup est légal si le coup permet de capturer des pierres adverses. Pour capturer des pierres il faut que ces pierres soient encadrées par des pierres adverses.

## 1.4 présentation du rapport

Dans ce rapport on va présenter en détail le problème en donnant les étapes de modélisation et de résolutions sous forme des algorithmes .Après on va présenter les solutions de problème à l'aide de l'interface graphique SDL.

# Chapitre 2

## Analyse et conception

Ce chapitre permet de faire une analyse théorique du jeu Othello. En effet, cette conception est cruciale afin de comprendre la totalité des principes et les coder en se basant sur le cahier de charges fournit.

### 2.1 Cahier de charges

Le cahier de charges présente l'ensemble des instructions et contraintes qui cadrent la réalisation du jeu. Le cahier de charges disponible donne 3 livrables qu'on va élaborer dans cette partie.

#### 2.1.1 livrable1

Livrable 1 : • Recomencer à tout moment en cliquant sur le bouton "recommencer", • Créer et enregistrer les joueurs et leurs caractéristiques (nom et score) sur fichier, • Afficher l'historique des mouvements effectués par les joueurs, • Permettre le chargement d'un jeu sauvegardé auparavant, • Donner la liste des dix meilleurs scores.

#### 2.1.2 livrable2

Jouer contre un non humain. 1. Avec des positions aléatoires, 2. Avec la mise en place d'un algorithme permettant d'optimiser la recherche du meilleur coup, en limitant le nombre de cases visitées. Il est possible d'utiliser l'algorithme min-max avec l'élagage Alpha-Beta

#### 2.1.3 livrable3

Il est également recommandé de considérer : • La sécurité des accès à l'aide de mots de passe, • La réalisation d'une interface de jeu conviviale.

## 2.2 Modélisation

Dans cette partie, nous allons discuter à propos des modèles qu'on a procédé pour résoudre les instructions de cahier de charge.

### 2.2.1 Conception de la planche d'Othello

Dans le projet pour représenter les coups du joueur on a réalisé une **énumération** de couleurs (blanc,noire) et pour la planche on l'a modélisé sous forme d'un tableau de deux dimensions de taille 8x8

```
typedef enum color color;//couleur
enum color
{
    noire,blanc,vide
};
```

FIGURE 2.1 – énumération de couleur

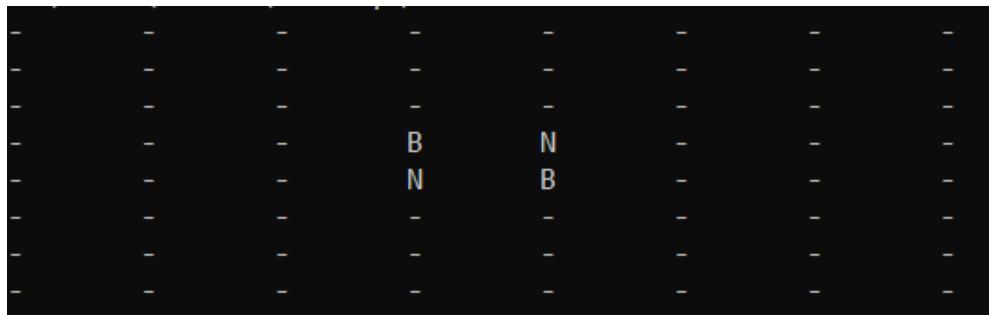


FIGURE 2.2 – ableau qui represente la planche N :noire,B :blanche

### 2.2.2 modelisation du joueur

Le joueur on a choisis de le modeliser par une structure contient trois champs : nom du joueur,le score et la clé d'utilisateur.

```
typedef struct joueur{
    char username[20];
    int score;
    int cle;
}joueur;
```

FIGURE 2.3 – structure du joueur

### 2.2.3 sauvgarde des données des joueurs

Pour enregistrer les données des joueurs on a crée un fichier binair qui contient les données sous forme d'octets qui n'ont donc de sens que pour le logiciel qui les utilise (et non pour les utilisateurs finaux).

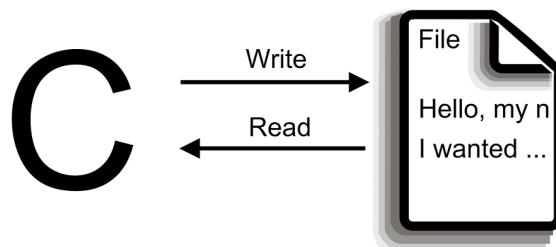


FIGURE 2.4 – fichier des données

## 2.3 Etapes de résolutions

### 2.3.1 Algorithmes de loqique de jeux

#### 2.3.1.1 Algorithme des cases possibles d'un jeu

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.

```

Result{K: tableau d'entiers de taille 2x25 nb et le nombre des cases possible a jouer }
Objet:
i ,j :variables entiers
T: tableau de couleur de taille 8x8
Debut:
Pour i<=0 jusqu'a 7 faire
    Pour j<=0 jusqu'a 7 faire
        Si T[i][j]=pi alors #pi couleur choisi
            chercher les cases possible adjacent pour pi
            ajouter les cases possibles dans K
            nb reçoit le nombres des cases possibles
        FinSi
    Finpour
Finpour
Fin .

```

### 2.3.1.2 Algorithme de transformations des pions

```

Objet:
i , position: variables entieres
debut
pour toute direction faire

    position <-- position du coup
    TantQue(c'est un pion adverse sur la position suivante sur la direction)
        position <-- position suivante sur la direction
        nombre_de_pion_a_transformer <--nombre_de_pion_a_transformer+1
        # On regarde a cote de la position initiale
    Fin Tant que

    Si (position<=8) #la derniere case
        Pour i allant de 1 jusqu'a nombre_de_pion_a_transformer faire
            position initial+i <-- pion allie
        FinPour
    FinSi

FinPour
Fin .

```

### 2.3.1.3 Algorithme de calculnum

Cette algorithme calcul le nombre de pions d'un joueur dans la planche

```

Objet:
n ,i ,j :variables entieres
Debut:
    Pour i<=0 jusqu'a 7 faire
        Pour j<=0 jusqu'a 7 faire
            Si le pion de la case (i,j)=pions de joueur alors
                n<--n+1 # n c'est le nombre de pions
            FinSi
        FinPour
    FinPour
Fin .

```

#### 2.3.1.4 minimax

```
function minimax(node, depth, maximizingPlayer)
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value<---- - infini
        for each child of node do
            value <---- max(value, minimax(child, depth -1, FALSE))
        return value
    else (* minimizing player *)
        value <---- + infini
        for each child of node do
            value <---- min(value, minimax(child, depth -1, TRUE))
        return value
```

#### 2.3.1.5 alphabeta

```
fonction alphabeta(noeud, alpha, beta) alpha est toujours inferieur a beta
    si noeud est une feuille alors
        retourner la valeur de noeud
    sinon si noeud est de type Min alors
        v <---- + infini
        pour tout fils de noeud faire
            v <---- min(v, alphabeta(fils, alpha, beta))
            si alpha >= v alors
                retourner v
            beta <---- Min(beta, v)
    sinon
        v <---- -infini
        pour tout fils de noeud faire
            v <---- max(v, alphabeta(fils, alpha, beta))
            si v >= beta alors
                retourner v
            alpha <---- Max(alpha, v)
    retourner v
```

#### 2.3.1.6 algorithme de testvalue

L'algorithme testvalue retourne la case de la value retourner par alphabeta.

Objets:

i ; variable entiere  
Copie: tableau de couleur de taille 8x8

Debut:

i<----0  
TantQue i<nobre de case possible faire

Copie<---- tableau de la planche  
transformer la Copie en ajoutant la ieme case possible  
on applique alphabeta a la Copie  
si valeur envoyer par alphabeta de tableau=valeur envoyer  
par alphabeta de la copie alors  
 retourner la case  
FinSi  
 i<----i+1  
Fin Tanque

Fin.

### **2.3.1.7 jeu Avec des positions aléatoires**

On a fait appelle le générateur de nombre entier de C pour générer un coup aléatoires  
algo : variable : entier n,K tableau des coup possible,entier nb  
n= GND() modulo nb :  
jeu le coup n qui dans K  
fin

## **2.3.2 Algorithmes des donnee de utilisateur**

### **2.3.2.1 Registerement de data**

Cette partie est une simple application de gestion de fichier dans c, on a utilisé des fonctions des bases des fichiers pour sauvegarder la structure des joueurs dans le fichier .  
algorithme :

```
:1      variable : fichier.bin ,joueur playerA
:2          ecire dans le fichier.bin le joueur player A
fin
```

### **2.3.2.2 Charge et Sauvegarder de table de jeu**

La même que dans le cas de joueur mais cette fois que on a register un Tableaux qui le modele de jeu  
algo de Sauvegarder :

```
variable : fichier.bin ,joueur playerA
ecire dans le fichier.bin le tableaux de jeu player A
fin
```

algorithme de charger :

```
variable : fichier.bin ,joueur playerA
lire le tableau de jeu player A à partir du fichier
fin
```

### **2.3.2.3 Liste des dix meilleurs scores**

Dans cette partir on a cherché dans le fichier les dix meilleurs scores et pour ce là on lira le fichier et on testera sur tous les joueurs

```
variable : fichier.bin ,joueur playerA, joueur T[10]
on a cherché le max de score des jeux et on l'ajoute dans T[0]
Pour i<—1 jusqu'a 10 faire
    chercher le maximum des scores qui n'est dans le tableau T
    ajouter le score trouvé dans le tableau T
```

fin

### **2.3.2.4 Historique des movemennt de jeux**

Pour afficher les mouvements des joueurs on a utilisé une structure nommé move qui contenait trois champs (entier x,entier y,couleur de joueur ) plus une pile contenait un tableau des structure move et un entier pointe sur le dernier mouvement pour afficher les mouvements ,on a parcouru la pile  
algo ajouter un mouvement dans la pile :

C'est tout simplement un empilement de pile

algo affchier les mouvements :

```
variable :pile ,entier n
n=(un entier pointe sur le dernier mouvement)
pour i<—0 allons n fair afficher le move
fin pour
fin
```

# Chapitre 3

## Réalisation du projet

### 3.0.1 Fenêtre d'utilisateur



FIGURE 3.1 – Fenêtre d'inscription

#### 3.0.1.1 Description

Dans cette fenêtre on a deux boutants l'un pour l'inscription et l'autre pour se connecter .

### 3.0.2 les données d'utilisateur



FIGURE 3.2 – Fenêtre de donnée

#### 3.0.2.1 Description

Si l'utilisateur est déjà inscrit il entre son pseudonym et son mot de passe pour se connecter.

### 3.0.3 Fenêtre d'enregistrement

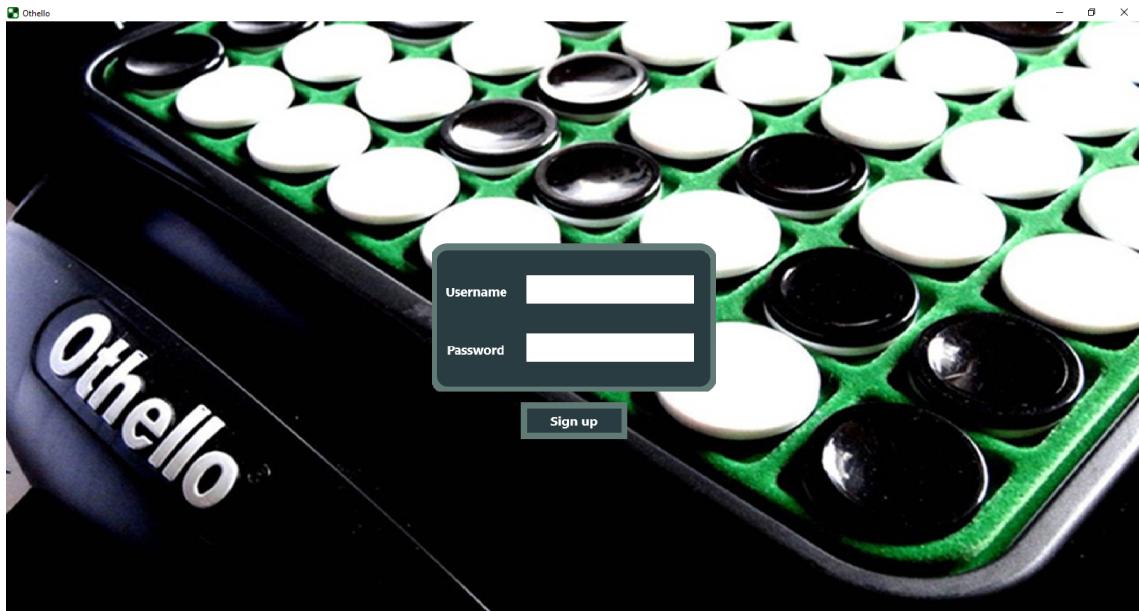


FIGURE 3.3 – Fenêtre d'enregistrement

#### 3.0.3.1 Description

Si l'utilisateur n'est pas encore inscrit il entre son pseudonym et son mot de passe pour s'inscrit.

### 3.0.4 Fenêtre de Gestion des données



FIGURE 3.4 – Fenêtre de Gestion des données

#### 3.0.4.1 Description

Si l'utilisateur entre l'une des données faux On affiche un message pour signaler qu'il doit reentrer son pseudonome et son mot de passe .

### 3.0.5 Fenêtre de choix des jeux

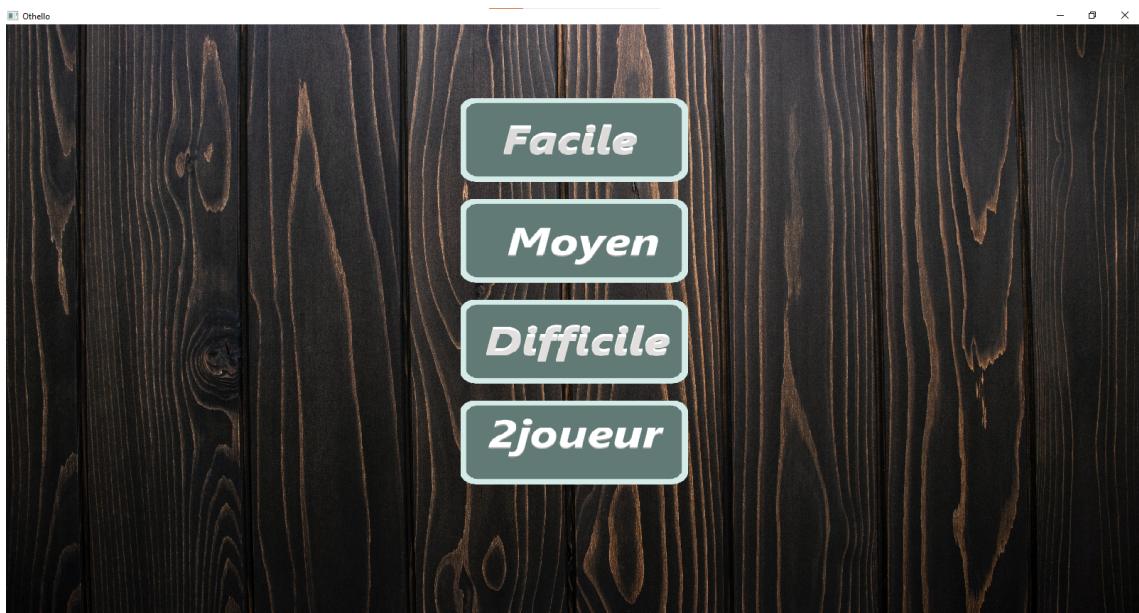


FIGURE 3.5 – choix du jeu

#### 3.0.5.1 Description

l'utilisateur choisi un jeu parmi les choix proposés.

### 3.0.6 Fenêtre de choix de couleur

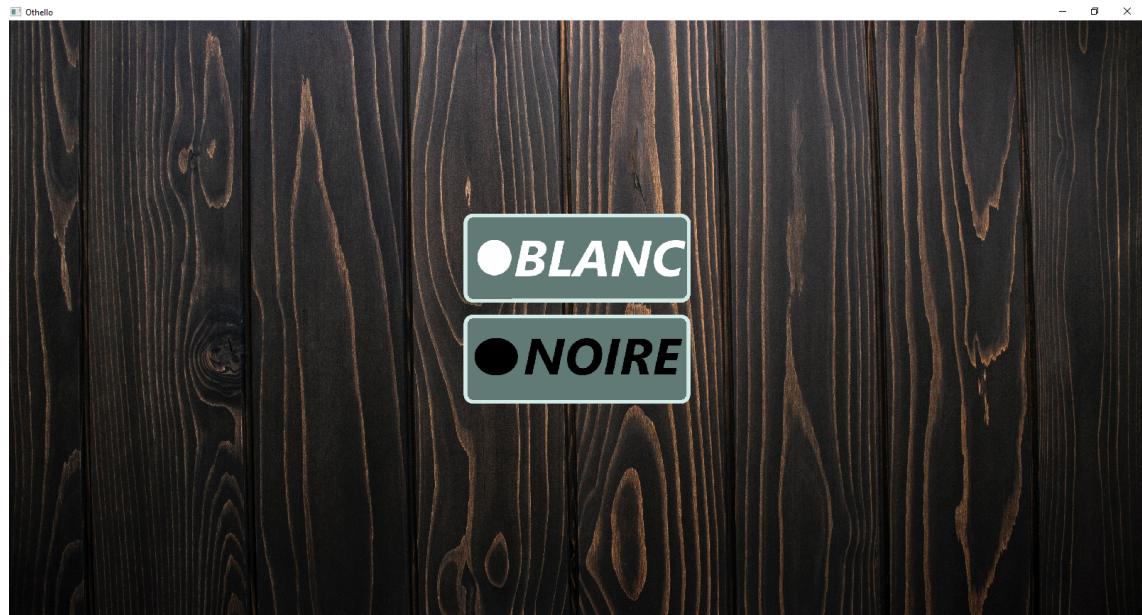


FIGURE 3.6 – Fenêtre de choix de couleur

#### 3.0.6.1 Description

l'utilisateur choisi une couleur parmi les deux couleurs (blanc,noire).

### 3.0.7 Fenêtre de jeu

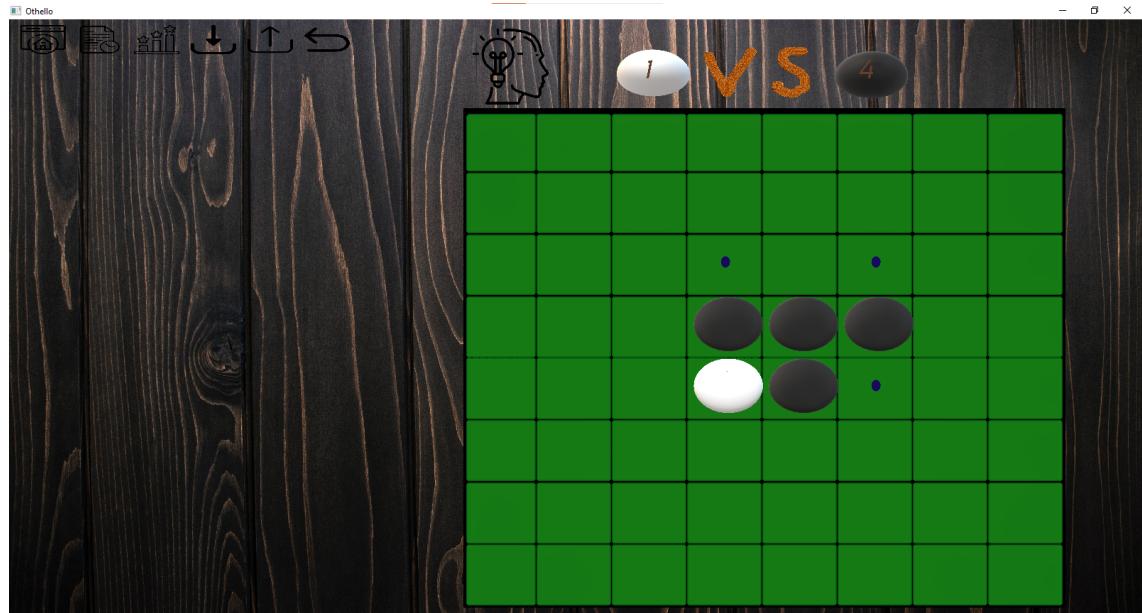


FIGURE 3.7 – Fenêtre de jeu

#### 3.0.7.1 Description

l'utilisateur commence à joué dans le mode choisi.

### 3.0.8 l'historique des mouvements

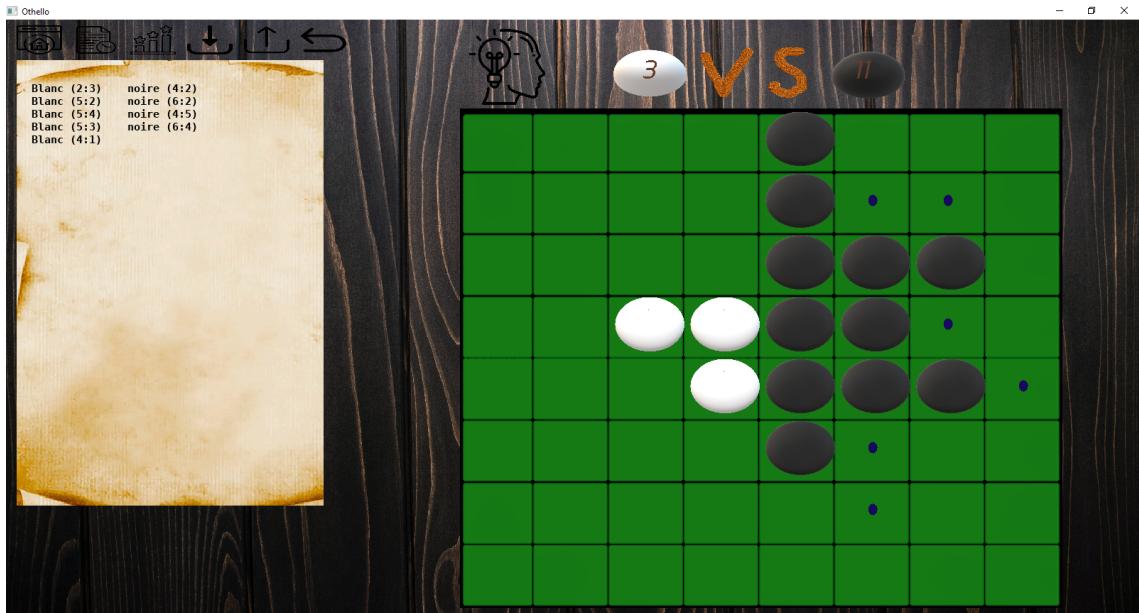


FIGURE 3.8 – historique des mouvements

#### 3.0.8.1 Description

l'utilisateur peut à chaque instant afficher l'historique des mouvements ainsi qu'il peut retourner au mouvement précédent.

### 3.0.9 dix meilleurs scores

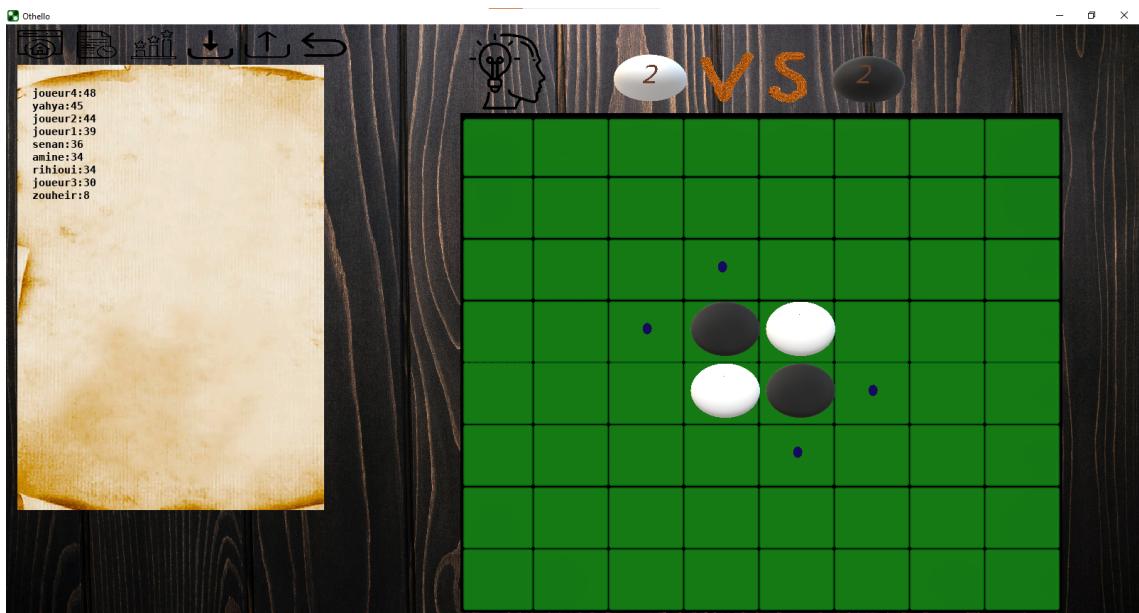


FIGURE 3.9 – top10

#### 3.0.9.1 Description

l'utilisateur peut accéder à la liste des dix meilleurs scores .

# Chapitre 4

## Conclusion

### 4.0.1 Attendu du projet

#### 4.0.1.1 Livrable 1 :

- Recommencer à tout moment en cliquant sur le bouton "recommencer".
- Créer et enregistrer les joueurs et leurs caractéristiques (nom et score) sur fichier.
- Afficher l'historique des mouvements effectués par les joueurs.
- Permettre le chargement d'un jeu sauvegardé auparavant.
- Donner la liste des dix meilleurs scores.

#### 4.0.1.2 Livrable 2 :

Jouer contre un non humain :

1. Avec des positions aléatoires.
2. Avec la mise en place d'un algorithme permettant d'optimiser la recherche du meilleur coup, en limitant le nombre de cases visitées. Il est possible d'utiliser l'algorithme min-max avec l'élagage Alpha-Beta.

#### 4.0.1.3 Livrable 3 :

Il est également recommandé de considérer :

- La sécurité des accès à l'aide de mots de passe.
- La réalisation d'une interface de jeu conviviale.

### 4.0.2 les parties réaliser du projet

Dans ce projet on a traité tous les livrables .Cependant on a reconnu plusieurs difficultés au cours de nos projets ,par ces contraintes on peut citer :

### 4.0.3 Contrainte de temps

La découverte de plusieurs technologies durant ce projet a consommé pas mal de temps. Nous pensons que le délai était suffisant mais trop serré.

#### 4.0.4 Interface graphique SDL

Afin de représenter graphiquement l'affichage du code source, nous avons essayé de manipuler la librairie SDL (Simple DirectMedia Layer). Découvrir la méthode de fonctionnement de cette librairie était un obstacle majeur.



FIGURE 4.1 – Logo de la librairie SDL

#### 4.0.5 Rédaction du rapport

La documentation professionnelle nécessite la manipulation du logiciel de traitement de texte LaTex. Travailler avec ce dernier est inévitable tôt ou tard, donc nous avons voulu exploiter cette opportunité et explorer LaTex.



FIGURE 4.2 – Logo du logiciel LaTex

#### 4.0.6 github

Afin de partager les fichiers concernant le projet on utilisé l'entreprise de développement logiciel et de service Github, le decouvrement de ce logiciel etait un obstacle pour poursuivre notre projet.



FIGURE 4.3 – Logo du logiciel github

pour acceder aux fichiers du projet veuiller cliquer sur le lien ci dessous :  
<https://github.com/zouheir-senan/C-project>

#### 4.0.7 Amelioration de Projet

il y a deux bibliothèques de SDL qui ne sont pas utilisées dans ce projet : SDL-mixer et SDL-net 2.0, l'une pour la gestion de son et l'autre pour faire un jeu online qui pourraient ajouter une valeur pour le projet ,mais à cause du temps on n'a pas pu les faire maintenent , mais dans le future on completera cette version.

# Bibliographie

- [1] <<https://www.libsdl.org>>, 2021.
- [2] <<https://www.overleaf.com>>, 2021.
- [3] <[https://fr.wikipedia.org/wiki/Elagage\\_alpha-beta](https://fr.wikipedia.org/wiki/Elagage_alpha-beta)>, 2021.
- [4] <[https://fr.wikipedia.org/wiki/Algorithme\\_minimax](https://fr.wikipedia.org/wiki/Algorithme_minimax)>, 2021.
- [5] <<https://openclassrooms.com>>, 2021.
- [6] <<https://www.youtube.com/watch?v=yFLa3ln16w0>>, 2021.
- [7] <<https://git-scm.com/docs/git-commit>>, 2021.
- [8] <<https://www.ffothello.org/othello/regles-du-jeu>>, 2021.