Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes - RABAT

Rapport de Projet de programmation : OTHELLO

Réalisé par :

BENHAMMOU Nouhayla
BESSA Hamza

Encadré par :

Pr. EL HAMLAOUI Mahmoud

Année Scolaire 2020/2021

Remerciements :

Nous voudrons tout d'abord adresser toute notre gratitude à notre professeur Mahmoud 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.

Nous désirons aussi remercier tous ceux qui contribuaient à la réussite de ce projet, notament ROCHDI FAILALI et NAJI YOUNES pour leurs conseils et leurs connaissances qu'ils nous ont partagés.

Table des matières

1 Introduction

1.1 OTHELLO : Le jeu

Le jeu d' 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.

FIGURE 1 – othello

1.2 OTHELLO : Principe du jeu

Othello se joue à 2, sur un plateau unicolore de 64 cases (8 sur 8), avec des pions bicolores, noirs d'un côté et blancs de l'autre.Le but du jeu est d'avoir plus de pions de sa couleur que l'adversaire à la fin de la partie, celle-ci s'achevant lorsque aucun des deux joueurs ne peut plus jouer de coup légal, généralement lorsque les 64 cases sont occupées
.Au début de la partie, la position de départ est indiquée ci-contre. Les noirs commencent.Chacun à son tour, les joueurs vont poser un pion de leur couleur sur une case vide, adjacente à un pion adverse.
Chaque pion posé doit obligatoirement encadrer un ou plusieurs pions adverses avec un autre pion de sa couleur, déjà placé.Il retourne alors le ou les pions adverse(s) qu'il vient d'encadrer.
Les pions ne sont ni retirés de l'othellier, ni déplacés d'une case à l'autre.On peut encadrer des pions adverses dans les huit directions et plusieurs pions peuvent être encadrés dans chaque direction.Par exemple, le joueur Noir a joué en c6. Il retourne alors les pions b6, b5, d7, c5 et c4.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.Si un joueur ne possède aucun coup permettant le retournement de pions adverses, celui-ci passe son tour et c'est à l'adversaire de jouer.

2 Contexte générale

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 6 instructions qu'on va élaborer dans cette partie.

2.1.1 Les besoins fonctionnels :

Après une étude détaillée du système, cette partie est réservée à la description des exigences fonctionnelles des différents acteurs de l'application.
Cette partie est concerne les deux premiers :
Un joueur peut sans l'existence de l'internet :
•S'identifier.
• Choisir entre jouer contre un humain ou la machine ;
Si le joueur a choisis « Contre l'ordinateur », il devra :
• Choisir le niveau de difficulté (soit facile, soit difficile) ;
Si le joueur a choisis « Contre un humain », il peut :
• Commencer dès le début, s'il est la première fois qu'il va jouer en Parcours .
• Recommencer la partie s'il ne désire pas continuer la partie en cours.
• Quitter le jeu à n'importe quel moment.
• Accès rapide à la rubrique recommencer et quitter.
• 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 Les besoins non fonctionnels :

Les besoins non fonctionnels décrivent toutes les contraintes techniques, ergonomiques et esthétiques auxquelles est soumis le système pour sa réalisation et pour son bon fonctionnement.
Cette partie concerne le troisième livrable. Et en ce qui concerne notre application, nous avons dégagé les besoins suivants :
• La disponibilité :L'application doit être disponible pour être utilisé par n'importe quel utilisateur.
• La sécurité de l'accès aux informations critiques :Nous devons prendre en considération la confidentialité des données des joueurs surtout au niveau de remplissage du formulaire d'authentification.
• La convivialité de l'interface graphique : L'application doit fournir une interface conviviale et simple pour tout type.
d'utilisateur.
• La fiabilité :Les données fournies par l'application doivent être fiables.
• La performance :La possibilité de retourner au menu principal de l'application à partir de n'importe quelle fenêtre de celle-ci.En l'occurence le système doit réagir dans un délai précis, quel que soit l'action de l'utilisateur.
• Design simple et conviviale .
• Interfaces graphiques ergonomiques .
• Langue utilisée : français.

2.2 Problématique

Le jeu d'OTHELLO recèle certaines disfonctionalitées par rapport au temps d'execution ,notamment dans le jeu contre la machine. La réalisation de ce projet par le biais de certains algorithmes qui vont accélerer la recherche du meilleur coup par la machine . Ceci contribuera par conséquent dans l'optimisation du jeu et donc une application plus facile à manipuler

2.3 Objectifs

Toute analyse faite, il s'avère donc que la conception du jeu comporte deux parties.La premiere conerne le jeu contre un humain et dans laquelle plusieurs options sont permises à savoir recommencer le jeu , le quitter ou encore sauvegarder son historique .
La deuxième concerne le jeu contre la machine et aura recourt à des algorithmes de l'intelligence artificielle. La seconde partie recèle non seulement les memes fonctionalitées que la première mais aussi une option qui permet à l'utilisateur de choisir le niveau de difficultée du jeu .
Les deux parties seront initiées par une fiche dont l'utilisateur remplirera ses informations personnelles notament le nom et le mot de passe .

3 Analyse théorique

3.1 Algorithmes et relations mathématiques

3.1.1 Algorithmes pour livrable 1

Pour les algorithmes , nous avons préferer de prendre des captures d'éran pour minimiser l'espace qu'elles vont occuper dans le rapport vu leurs longeurs .En l'occurence , nous n'allons introduire que les algorithmes les plus importants du livrable.

A.Coup Valide
Cet algorithme permet de détecter si un coup choisis par l'utilisateur est valide ou pas . Ceci nécessiste une analyse des cases vides situées à une position appartenant aux terroitoire des cases du joueur en question.

```
-algo de la fonction coup_valide-

FONCTION coup_valide(m,lig,col,joueur) : structure t_matrice
                                         m[8][8] : tableau de taille 8*8 de type caratere
                                         fin-structure
                                         lig,col,joueur:entiers
objet :
        i,j,ok:entiers
        ca,cj: carateres
debut
  si(joueur= 1)alors
        cj<- NOIR
        ca<-BLANC
   sinon
        cj <- BLANC
        ca <- NOIR
  fin_si

    si (!case_existe(lig, col) ou m[lig][col] != VIDE) alors retourner 0
   fin_si

    /* Vertical vers le bas */
    i <- lig - 1
    ok <- 0
   tant-que (case_existe(i, col) et m[i][col]= ca) faire
        i<-i-1
        ok<- 1
    fin_tant-que.

    si(case_existe(i, col) et m[i][col] = cj et ok = 1) alors retourner 1
   fin_si.

    /* Vertical vers le haut */
    i <- lig + 1
    ok<-0
    tant-que(case_existe(i, col) et m[i][col] = ca) faire
        i<-i+1
        ok<- 1
    fin_tant-que.
    si (case_existe(i, col) et m[i][col] = cj et ok = 1) faire retourner 1
   fin_si.

    /* Horizontal vers la gauche */
    j <- col - 1
    ok <-0
    tant-que (case_existe(lig, j) et m[lig][j] = ca) faire
        j<-j-1
        ok<-1
    fin_tant-que.
    si (case_existe(lig, j) et m[lig][j] = cj et ok = 1) retourner 1
   fin_si.

    /* Horizontal vers la droite */
    j <- col + 1
    ok <- 0
    tant-que (case_existe(lig, j) et m[lig][j] = ca) faire
        j<-j+1
        ok<- 1
    fin_tant-que.

    si (case_existe(lig, j) et m[lig][j] = cj et ok = 1) retourner 1
    fin_si.

    /* Diagonal / vers le bas */
    i <- lig - 1
    j <- col - 1
    ok <- 0
    tant-que (case_existe(i, j) et m[i][j] = ca) faire
        i<-i-1
        j<-j-1
        ok<- 1
    fin_tant-que.

    si (case_existe(i, j) et m[i][j] = cj et ok = 1) retourner 1
    fin_si.

    /* Diagonal / vers le haut */
    i<- lig + 1
    j <- col + 1
    ok <- 0
    tant-que (case_existe(i, j) et m[i][j] = ca) faire
        i<-i+1
        j<-j+1
        ok <- 1
    fin_tant-que.
    si (case_existe(i, j) et m[i][j] = cj et ok = 1) retourner 1

    /* Diagonal \ vers le bas */
    i <- lig - 1
    j <- col + 1
    ok <- 0
    tant-que (case_existe(i, j) et m[i][j] = ca) faire
        i<-i-1
        j<-j+1
```

```
            j<-j+1
            ok <- 1
        fin_tant-que.

        si (case_existe(i, j) et m[i][j] = cj et ok = 1) retourner 1
       fin_si.

        /* Diagonal \ vers le haut */
        i <- lig + 1
        j <- col - 1
        ok <- 0
        tant-que (case_existe(i, j) et m[i][j] = ca) faire
            i<-i+1
            j<-j-1
            ok <- 1
        fin_tant-que.
        si (case_existe(i, j) et m[i][j] = cj et ok = 1) retourner 1
        fin_si.

        retourner 0
    fin.
```

B.Choisir coup

Cet algorithme permet à l'utilisateur de sélectionner un coup valide parmis tous les coups possibles . Elle exploite la fonction coup valide.

```
fonction choisir_coup(m,lig,col,joueur,nom1,nom2,n) : structure t_matrice
                                m[8][8] : tableau de taille 8*8 de type caratere
                                fin-structure
                                lig,col,joueur,n:entiers
objets:
 c:variable qui peut etre entier ou caracetre.
debut:
si(joueur=1)alors
        ecrire ("C'est au tour du joueur",nom1," de jouer(noir)")
    sinon
         ecrire("C'est au tour du joueur",nom2," de jouer(blanc)")
fin_si.
   ecrire("Choisissez une ligne:")
   lire(c)
    si(c='q')alors sortir du programme //quitter le jeu a tout moment
    sinon si(c='r') alors  // recommencer le jeu a tout moment
       si(n=1)alors demarerLeJeu(nom1,nom2,nomF,n)
       sinon si(n=2)alors demarerLeJeuVsMachine(nom1,nom2,nomF,n)
      sortir du programme.
       fin_sinon si.
      fin_si.
    fin_sinon si.
    fin_si.
   lig=c
    ecrire("Choisissez une colonne:")
    lire(c)
    col=c
    tant_que(!coup_valide (m, lig, col, joueur)) faire
        ecrire("Ce coup n'est pas valide")
        ecrire("Choisissez une ligne:")
        lire(c)
        lig=c
       ecrire("Choisissez une colonne:")
       lire(c)
       col=c
  fin_tant-que.
fin.
```

C.Case existe

Cet algorithme détecte les cases de l'othellier et permet de savoir si la case entrée par l'utilisateur est valable ou pas.

```
FONCTION case_existe(lig,col) : lig,col:entiers

DEBUT
retourner ((col >= 0) et (col < N) et (lig >= 0) et (lig < N))
fin.
```

D.Partie terminée

Cet algorithme concerne la fin de la partie . Il s'agit donc d'annoncer le gagnant lorsque toutes les cases de l'othellier sont occupées . Il fait appel à plusieurs fonctions citées dans le programme.

```
fonction partie_terminee (m,nom1,nom2) : structure t_matrice
                                         m[8][8] : tableau de taille 8*8 de type caratere
                                         fin-structure
                                         nom1,nom2:chaines de caracteres
objets:
  i, j, nb_noir, nb_blanc:entiers

    nb_noir<-0
    nb_blanc<- 0
     pour i<-0 jusqu'a i<N  faire
        pour j<-0 jusqu'a j<N faire
            si(m[i][j] =VIDE et (peut_jouer(m, 1) ou peut_jouer(m, 2))) alors // VIDE ET peut_jouer sont deja definie
                retourner 0
              sinon
                si (m[i][j] = NOIR) alors nb_noir++
                sinon si (m[i][j] = BLANC) alors nb_blanc++
                fin_sinon-si.

          fin_si.
        fin_pour.
     fin_pour.

    si (nb_noir > nb_blanc) alors
        ecrire ("Le joueur",nom1,"a gagne !!!")
          score[0]<-nb_noir  // le tableau score est de type entier et de taille 2
          score[1]<-nb_blanc //  il est considerer comme une varible globale

    sinon si (nb_blanc > nb_noir) alors
        ecrire ("Le joueur",nom2," a gagne !!!")
       score[0]<-nb_noir
       score[1]<-nb_blanc

    sinon
        ecrire("Les deux joueurs sont a egalite")
        score[0]<-nb_noir
        score[1]<-nb_noir
    fin_sinon
    fin_sinon-si.
   fin_si.
```

3.1.2 Algorithmes pour livrable 2

A.RANDOM

Il s'agit du premier niveau du jeu , à savoir le niveau EASY . L'algorithme Random permet alors de tirer une case valide aléatoirement parmi tous les cas possibles.

B.MINMAX avec elagge ALPHA-BETA

Il s'agit d'une strategie independante du jeu auquel on veut jouer tant que ce jeu est :

1. **à deux joueurs**
2. **à une information complete**
3. **à coups asynchrones (nombre fini et limite)**
4. **à deux joueurs**

Beaucoup de jeux de plateau correspondent à ces critères :
echecs, go, othello, morpion, etc.

***Principe :**
Le joueur qui commence (joueur MAX) :
cherche à trouver, parmi toutes les situations à sa disposition, une situation qui lui permet de maximiser ses gains.
L'autre joueur (joueur MIN) :
doit trouver, à partir de toutes les situations qui conduisent à la victoire du permier joueur, la situation qui minimise les gains de ce joueur
Minimax = minimiser la perte maximum
A partir d'une position donnée, il existe une arborescence de coups jusqu'à la victoire (V), au nul

(N) ou à la défaite (D). A titre d'exemple, que penser de cette situation ? (cercle représente le joueur MAX, carré représente le joueur MIN).

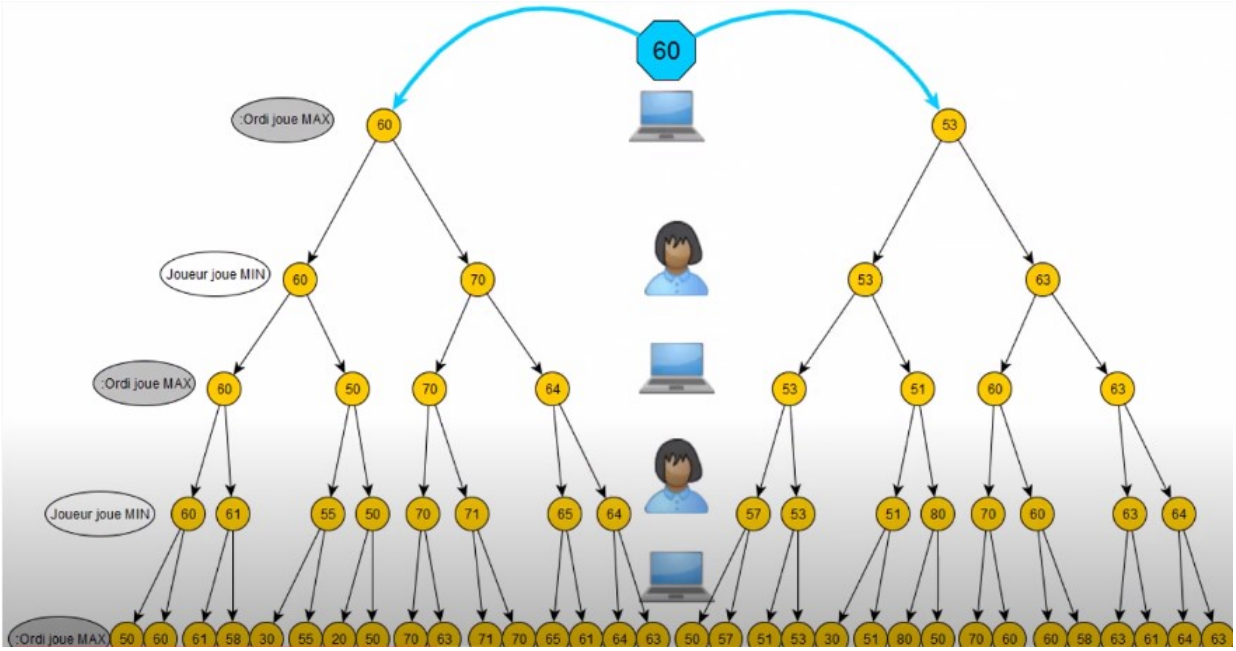

FIGURE 2 – Arborescence de l'algorithme MINMAX

En réalité, il est impossible de développer entièrement l'arbre du jeu et de dire si une feuille correspond à une position gagnante ou à une position perdante (à cause d'une complexité combinatoire) Dans ce cas, il est nécessaire de disposer d'une fonction d'évaluation (heuristique), capable d'estimer le plus précisément possible la qualité d'une position. on définit alors une profondeur de recherche (horizon de l'IA) les feuilles de l'arbre sont associées à une valeur numérique donnée par cette fonction d'évaluation.

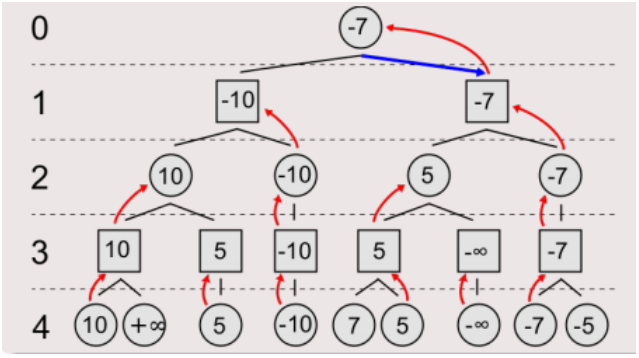

FIGURE 3 – Arborescence de l'algorithme MINMAX dans le scénario réel

1.nœud : configuration actuelle du plateau du jeu
2.profondeur : profondeur actuelle
3.evalMax : si vrai alors joueur MAX sinon joueur MIN
4.retour : valeur du nœud
Ses Conditions d'arret sont :
1.fin de jeu (victoire, nul ou d´efaite)

2.ou profondeur = 0 (on atteint l'horizon d'IA)Il s'agit donc d'une fonction récursive sur la profondeur dont les paramètres sont :

1.nœud : configuration actuelle du plateau du jeu
2.profondeur : profondeur actuelle
3.evalMax : si vrai alors joueur MAX sinon joueur MIN
4.retour : valeur du nœud
Ses Conditions d'arret sont :
1.fin de jeu (victoire, nul ou d´efaite)
2.ou profondeur = 0 (on atteint l'horizon d'IA)

```
Fonction MinMax(noeud : Plateau, profondeur : Entier, evalMax : Booleen) : Entier
Début
  Si profondeur = 0 ou victoire(noeud) ou defaite(noeud) ou nul(noeud) Alors
    retourner evaluation(noeud)
  Sinon {on est sur un noeud interne}
    Si evalMax Alors
      retourner max_{f∈fils}(MinMax(f, profondeur − 1, faux))
    Sinon {on évalue le joueur adversaire}
      retourner min_{f∈fils}(MinMax(f, profondeur − 1, vrai))
    FinSi
  FinSi
Fin
```

FIGURE 4 – Pseudo code de l'algorithme MINMAX

MinMax peut etre optimisé en enlevant certaines branches qui, selon le fonctionnement de l'algorithme, n'ont pas à etre explorées .Cette optimisation peut se faire par le biais de l'algorithme ALPHA-BETA et ceci comme suit :
1.Associer à chaque nœud, en plus de sa valeur, 2 autres quantit´es :
A.alpha : approximation par défaut = score minimum du joueur MAX.
B.beta : approximation par excès = score maximum du joueur MIN
2.Le couple (alpha,beta) avec alpha < beta est appelé fenetre de valeur au d´ebut, alpha est initialisée à , beta à + quand alpha >= beta élaguer le nœud correspond.

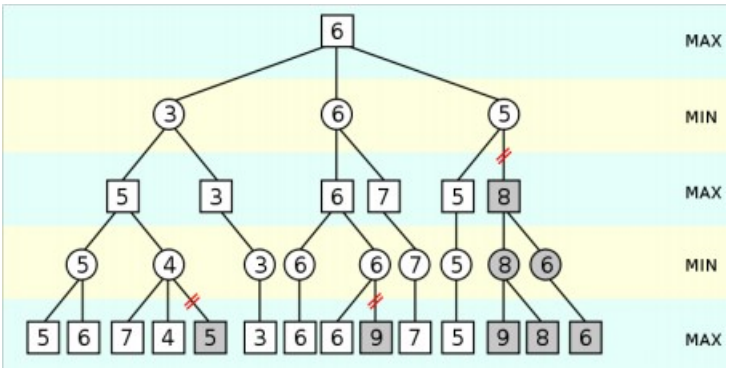

FIGURE 5 – Schéma d'une coupure

On peut donc optimiser l'algorithme MINMAX avec un elagage ALPHA-BETA et ceci en appliquant l'algorithme dans la figure ci dessuous :

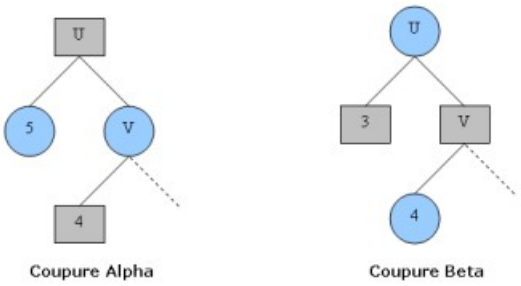

FIGURE 6 – Schéma d'une coupure alpha beta

AlphaBeta

```
Fonction AlphaBeta(noeud : Plateau, profondeur : Entier,
    alpha : Entier, beta : Entier,  evalMax : Booleen) :
    Entier
Variables score : Entier
Debut
  Si profondeur = 0 ou victoire(noeud) ou defaite(noeud) ou
      nul(noeud) Alors
    retourner evaluation(noeud)
  Sinon
    Si evalMax Alors //joueur MAX
      Pour chaque coup de coupJouables(noeud)
        score = AlphaBeta(applique(coup,noeud),profondeur-1,
            alpha,beta,faux)
        Si score > alpha Alors alpha = score FinSi
        // on a trouvé un meilleur coup
        Si alpha ≥ beta Alors retourner alpha FinSi
        // coupe beta
      FinPour
      retourner alpha // c'est le meilleur coup
...
    Sinon // joueur MIN
      Pour chaque coup de coupJouables(noeud)
        score = AlphaBeta(applique(coup,noeud),profondeur-1,
            alpha,beta,vrai)
        Si score < beta Alors beta = score FinSi
        // l'adversaire a trouvé un pire coup
        Si alpha ≥ beta Alors retourner beta  FinSi
        // coupe alpha
      FinPour
      retourner beta // meilleur coup pour l'adversaire
    FinSi
  FinSi
Fin
```

FIGURE 7 – Algorithme ALPHA-BETA

3.2 Diagrammes globaux

3.2.1 Diagramme du fonctionnement global

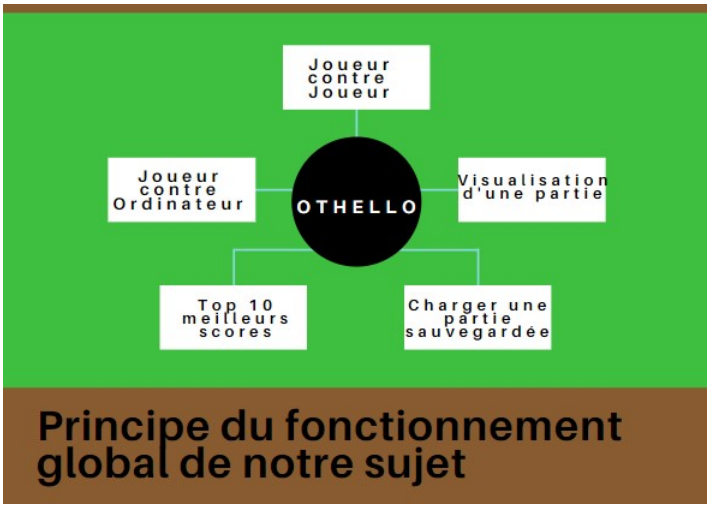

FIGURE 8 – Schéma du principe du fonctionnement de notre projet

3.2.2 bête à cornes d'OTHELLO

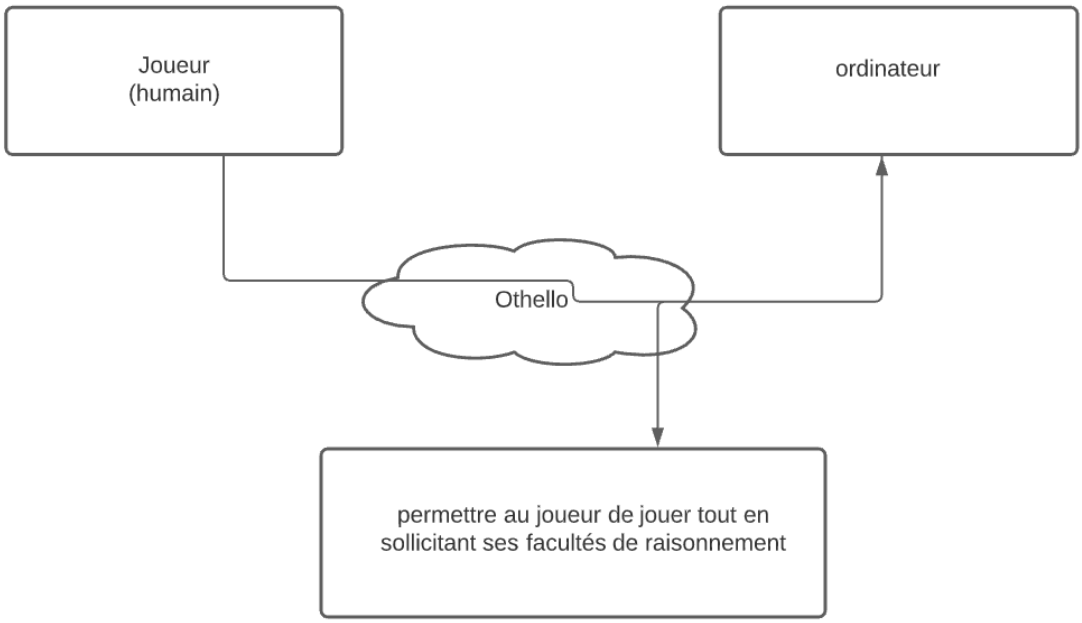

FIGURE 9 – Bete à corne du jeu Othllo

3.2.3 Organigramme du Projet

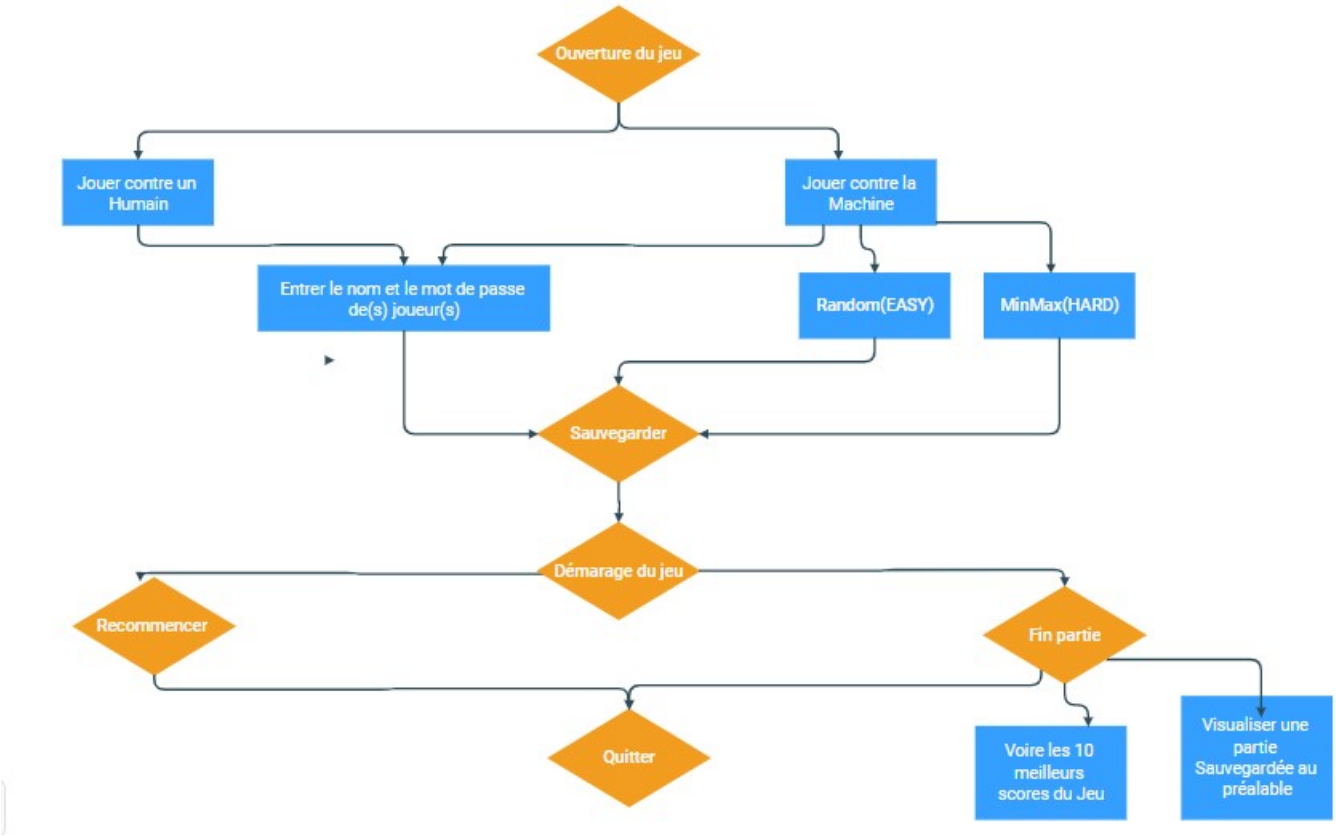

FIGURE 10 – Organigramme du Projet

4 Réalisation et résultats

4.1 Les directives

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include "jeu.h"
#include "jeu1vs1.h"
#include "jeu1vsMachine.h"
#include "securite.h"
```

4.2 Fonctions du code source

Chaque instruction necéssite une ou plusieurs fonctions. Dans cette partie nous allons élaborer chacune des fonctions utilisés dans le code source.
Dans une première étape nous avons initialisé notre code par les directives suivantes :

Dans une seconde étape nous avons défini plusieurs fonctions nécessaires pour le fonctionnement

des deux livrables .
les fonctions suivantes sont dans le fichier : jeu1vs1.c ,et leurs prototypes sur le fichier : jeu1vs1.h

‘

4.2.1 Initialisation

init_matrice (t_matrice m)

Permet d’initialiser la matrice (le plateau de jeu)

4.2.2 Affichage matrice

void afficher_matrice (t_matrice m)

Permet d’afficher le plateau de jeu (la matrice) sous la forme suivante :

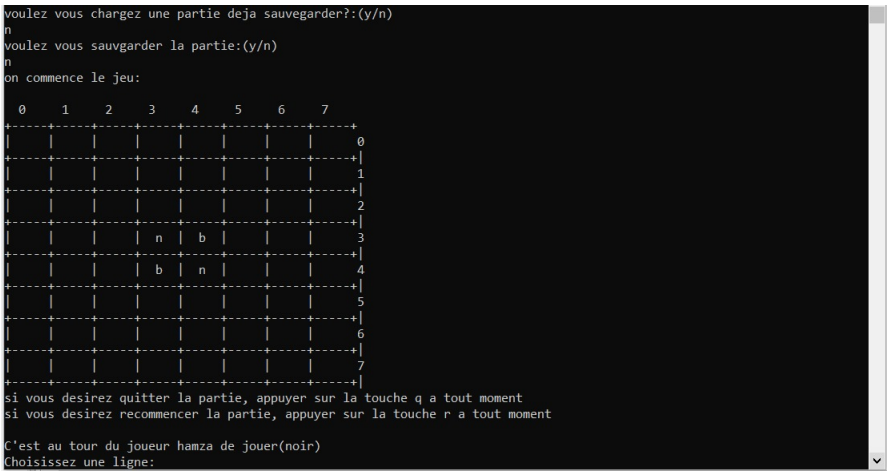

FIGURE 11

4.2.3 Existence d’une case

int case_existe (int lig, int col)

C’est un fonction qui détermine si une case existe ou non (si l’indice ne dépasse pas la taille du tableau N), renvoie 1 si la case existe 0 sinon.

4.2.4 Validation du coup

int coup_valide (t_matrice m, int lig, int col, int joueur)

C’est une fonction qui cherche a chaque fois si le coup joue est valide, renvoie 1 si le coup est valide et 0 sinon

4.2.5 Tour joueur

int peut‘_jouer (t_matrice m, int joueur)

C’est une fonction qui détermine si un joueur peut encore jouer, renvoie 1 si il peut sinon 0

4.2.6 Joueur qui suit

int joueur_suivant (int joueur)

Elle Renvoie le numéro du joueur suivant (qui varie entre 1 et 2)

4.2.7 Choix du coup

void choisir_coup (t_matrice m, int *lig, int *col, int joueur,char *nom1,char *nom2,char *nomF,int n)

Cette fonction par laquelle on choisit le coup qu'on va jouer (on choisit d'abord la ligne puis la colonne)

4.2.8 Fin de la partie

int partie_terminee (t_matrice m,char *nom1,char *nom2)

C'est une fonction qui indique la fin de la partie, renvoie 1 si la partie est finie sinon 0.

4.2.9 Jouer un coup

void jouer_coup (t_matrice m, int lig, int col, int joueur)

C'est une fonction responsable de faire des changements sur le plateau (lorsque les noirs deviennent blanches et vise versa)

4.2.10 Demarage

void demarerLeJeu(char *nom1,char *nom2,char *nomF,int n)

cette fonction est le moteur du jeu 1vs1 , elle reçoit tout ces fonctions pour lancer le jeu sur console

les fonctions suivantes sont dans le fichier : sauvgarde.c ,et leurs prototypes sur le fichier : sauvgarde.h

4.2.11 Sauvegarde

void ChargePartieSauvegarder(char *nom1,char *nom2,char *nomF,int n)

Cest une fonction qui charge un jeu déjà sauvegarder (on sauvegarde que si le joueur veut sauvegarder la partie : le joueur choisie le nom du fichier ou il veut sauvegarder sa partie et lorsque il veut charger sa partie il n'a qu'a donner le nom du fichier).Afin de pouvoir sauvegarder une partie en cours de même que l'on pourra en rappeler une précédemment sauvée, il faut faire usage au méthodes de stockages des fichiers. En effet, on propose la sauvegarde de chaque planche modifée au cours du jeu. On revient à ce fichier pour rappeler la dernière partie. On propose également donner le choix au joueur s'il veut reprendre une partie précedemment jouée ou commencer une nouvelle.

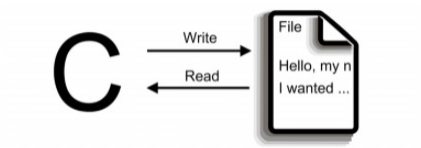

FIGURE 12 – Sauvegarde

4.2.12 Visualisation

void visualiser (char *nomF)

cette fonction permet de visualiser l'historique des mouvements effectuer au cours de la partie ou bien de voir les top 10 meilleures scores

4.2.13 Top 10 meilleurs scores

void top10scores()

cette fonction qui crée un fichier pour stocker les top10scores

Les fonctions suivantes sont dans le fichier : securite.c ,et leurs prototypes sur le fichier : securite.h

4.2.14 Vérification

void FichierSignUp(char *nom,int i)

c'est une fonction qui vérifie si le joueur a déjà un compte

4.2.15 Nouveau compte ?

void FichierSignIn(char *nom,int i)

c'est une fonction qui permet au joueur s'il est nouveau de créer un nouveau compte

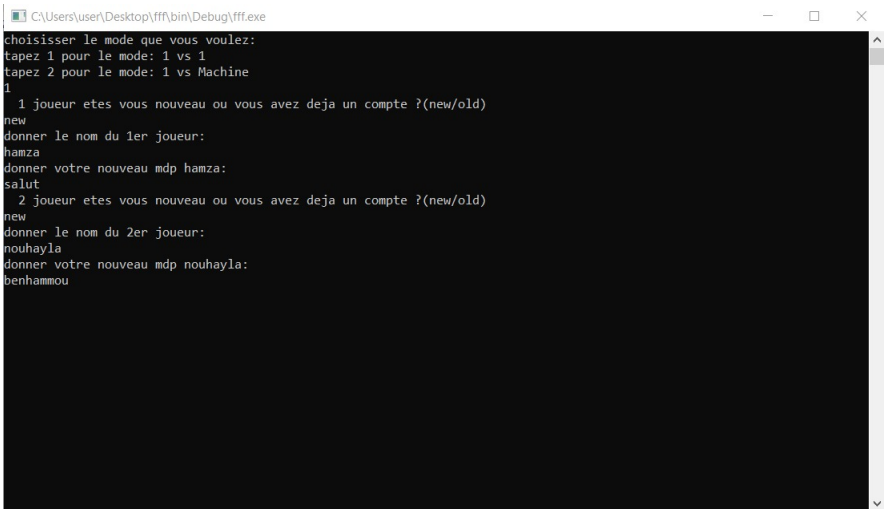

FIGURE 13

4.3 Fonctions du deuxieme livrable :

les fonctions suivantes sont dans le fichier : jeu1vsMachine.c ,et leurs prototypes sur le fichier : jeu1vsMachine.h

4.3.1 Random

void choisir_coupMachine (t_matrice m,int *lig,int *col,int joueur,char *nomF)

C'est une fonction qui permet à l'ordi de choisir son coup d'une manière aléatoire

4.3.2 Demarage

void demarerLeJeuVsMachine(char *nom1,char *nom2,char *nomF,int n)

Afin de représenter graphiquement l'affichage du code source, nous avons essayé de manipuler la librairie SDL2(Simple Directmedia Layer). Découvrir la méthode de fonctionnement de cette librairie était un obstacle majeur.

FIGURE 14 – Logo de la librairie SDL

4.4.1 Queslques apperçus

Nous avons aussi tenter de travailler avec la librairie GTK assocoéeà glade afin de réaliser l'interface .

Il faut noter que nous avons fait de notre mieux afin de finir l'interface graphique de notre projet. Néomoins , nous avons réaliser quelques interfaces et lier ces dernières pour avoir un bon fonctionnement du jeu. Il est vrai que nous voulions accompagner ce projet d'une interface graphique achevée et bien accomplie , tout de meme c'était une opportunité qui nous a permis l'apprrentissage des atouts de réalisation des interfaces , et nous avons hate de les exploiter durant les projets à venir.

FIGURE 15 – logo de la bibliothèque GTK

4.5 Rédaction Rapport

La documentation professionelle 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 16 – latex

4.6 Difficultés rencontrées

4.6.1 Rédaction et organisation du code

Ecrire un clean code est un défi pour tous les développeurs. Nous avons essayé de garantir le maximum de la clareté. En effet, nous avons commenté le code d'une façon pertinente. De plus, nous avons divisé le code en plusieurs fichiers " .c " et des fichiers " .h ".

4.6.2 Contrainte du 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.

5 Conclusion

PROJET C s'agit de produire un programme afin de valider les compétences des cours : « Algorithmique », « Technique de programmation » et « Structures de données ». Le programme correspond à 3 semaines de travail effectives en langage C.
La conception du jeu "OTHELLO"est alors une bonne application pour excercer les conaissances acquises dans les trois cours et nécessite davantage l'emprunt de quelques techniques dans le domaine de l'intelligence artificielle ainsi que l'exploitation des bobliothèques SDL et GTK .
A travers le document courant, nous vous avons donc prodigué le fruit d'un bon semestre de labeur mutuel et recherche continue ornés par encadrement instructif et professionnel, que nous espérons être à la hauteur de vos estimations.

6 Bibliographie

1. **http ://www.lecomptoirdesjeux.com/regle-reversi.htm**
2. **https ://www.youtube.com/watch ?v=f30Ry1WOe**$_Q t = 632s$
3. **https ://openclassrooms.com/fr/courses/19980-apprenez-a-programmer-en-c/17117-installation-de-la-sdl**
4. **https ://www.gtk.org/**
5. **https ://glade.gnome.org/**
6. **https ://www.ffothello.org/informatique/algorithmes/**
7. **https ://www.overleaf.com/project**