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

Rapport de Projet de programmation : OTHELLO

Réalisé par :

Somiya FOURTASSI
Jihane KARIB

Encadré par :

Pr : M. El HAMLAOUI

Année Scolaire 2020/2021

Avant d'entamer notre rapport, nous tenons énormément à remercier notre encadrant du projet, Professeur M. El Hamlaoui, pour ses efforts déployés à nous orienter, pour les conseils précieux dont il nous a fait part, sans oublier les efforts de nos professeurs des modules de « Structures de données » : Monsieur Abdellatif EL FAKER, « Algorithmique » : Monsieur Ahmed ETTALBI, et de « Techniques de programmation » : Monsieur Hatim Guermah pour le savoir qu'ils nous ont transmis tout au long le semestre.

Table des matières

Contexte Général

1 Présentation de projet

1.1 Projet C

Dans le cadre de notre formation ingénieur en informatique, un projet de programmation demeure nécessaire afin de cristalliser nos connaissances en algorithmique, techniques de programmation, structures de données et théorie des graphes.

Il s'agit de coder un programme en C qui répondra au cahier de charges proposé par l'encadrant. En ce qui nous concerne, le programme souhaité est celui d'un jeu de table : Othello suivant des normes précises.

1.2 Historique du jeu et étymologie

Au 19ème siècle, le jeu Réversi a été inventé en Angleterre. 140 ans après, ce jeu a été réinventé sous le nom d'Othello en Japan par Goro Hasegawa qui est devenu par la suite un grand classique. Le nom Othello a été suggéré par le père de Goro par analogie avec les rebondissements de la pièce de Shakespeare.

1.3 Règles du jeu

Othello est un jeu de stratégie et de tactique à deux joueurs : Noir et Blanc, chacun d'eux possède 32 pions bicolores mais ils ne sont pas propres à lui puisqu'il peut en donner à son adversaire si celui-ci n'en a plus. Othello se joue sur un plateau carré unicolore de 64 cases nommé othellier. Les règles du jeu comprennent :

— C'est le noir qui déclenche le jeu !
— L'état initial du plateau est décrit dans la figure suivante :

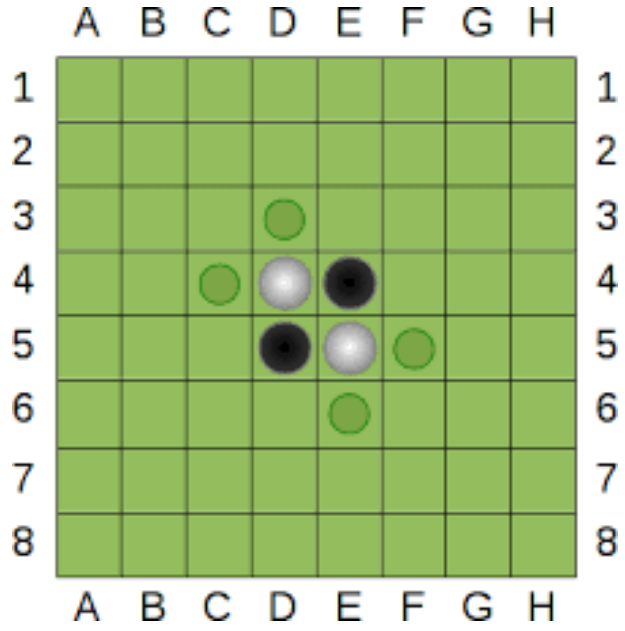


FIGURE 1 – Etat initial de la table du jeu.

— Un cout est dit légal lorsque le joueur peu se positionner de façon à entourer des pions de son adversaire soit horizontalement, verticalement, ou de façon oblique dans les deux sens !
— La couleur des pions entourés de l'adversaire se change !

— Le jeu se joue à tour de rôle jusqu'à ce que le plateau est plein ou aucun cout n'est légal pour les deux joueurs !
— Le gagnant et celui qui possède le grand nombre des pions de sa couleur !

2 Cahier de charges et problématique

Il s'agit de programmer un code en langage C du jeu Othello, en assurant certains attendus bien précises par notre encadrant Mr. El Hamlaoui. Ces derniers sont répartis en 3 livrables. Le 1er sert à disposer le jeu de différents outils organisationnels, le 2eme offre la possibilité de jouer contre un non humain , et le 3eme renforce les performances graphiques et confidentiels du jeu.

3 Objectifs

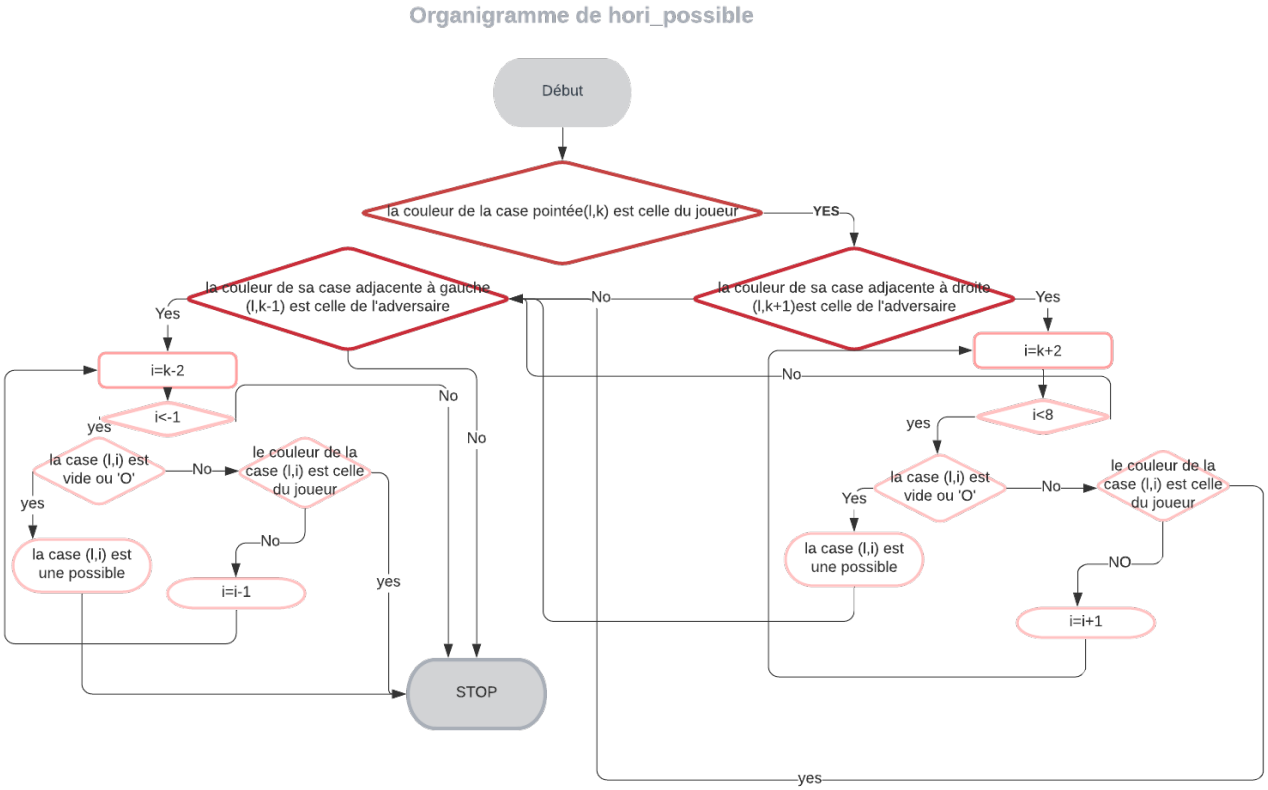
Il nous est proposé en 1er lieu de :
— Permettre au joueur de recommencer à tout moment souhaité.
— Créer un fichier ou les caractéristiques du joueur (nom et score) sont enregistrés.
— Afficher l'historique des mouvements effectués par les joueurs.
— Permettre au joueur de charger un jeu sauvegardé auparavant
— Donner la liste des dix meilleurs scores.
Puis il est question d'organiser le même jeu contre un non humain avec :
— des positions aléatoires.
— Un algorithme d'optimisation de la recherche du meilleur coup en limitant le nombre de cases visités.
Et finalement on souhaite :
— Renforcer la sécurité des accès à l'aide de mots de passe.
— Réaliser une interface graphique conviviale.

Analyse et Concept

1 Code du jeu

1.1 Header : PossibleCase.h

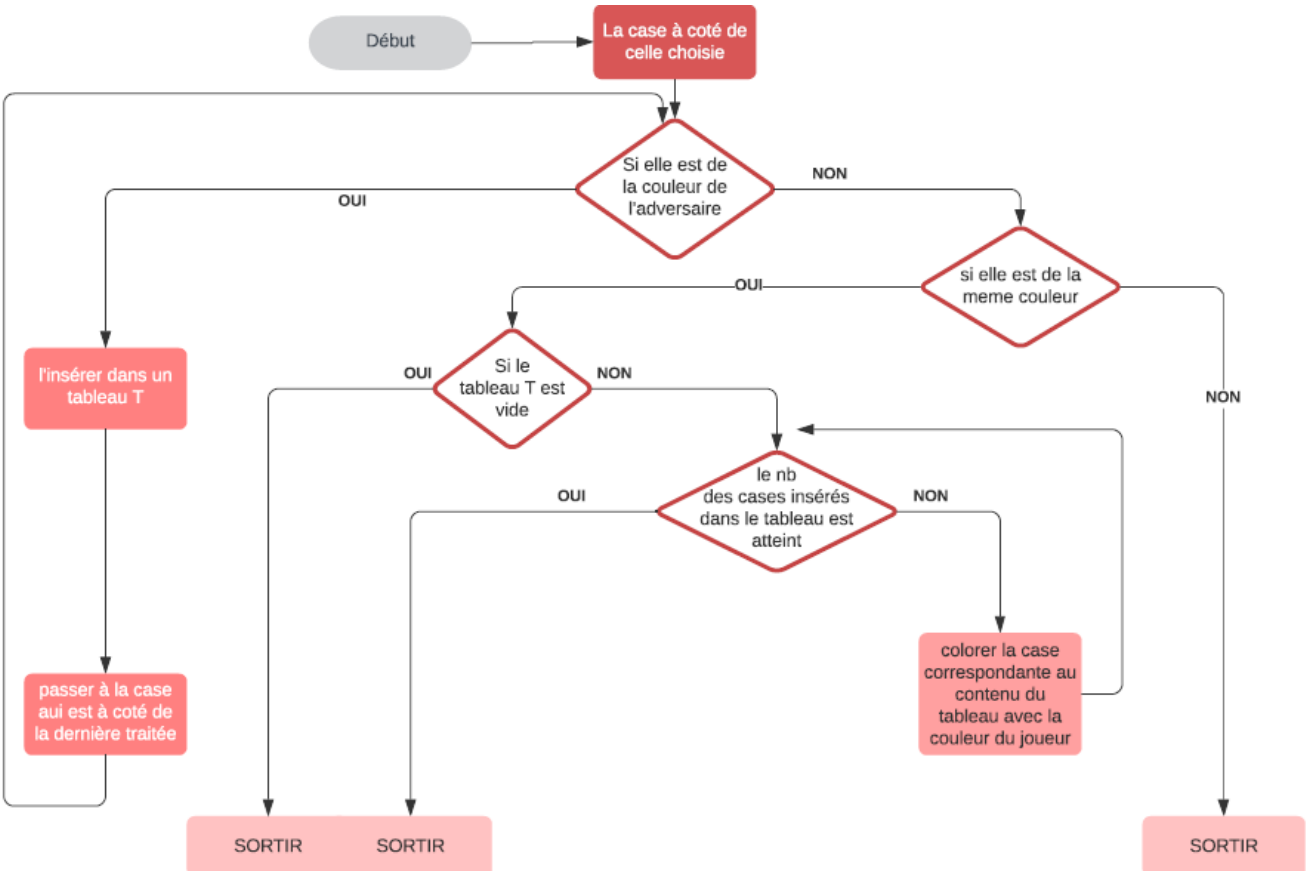
Ce "header" comprend les prototypes de 5 fonctions qui permettent la recherche des cases possibles au joueur et l'affectation d'un 'O' à ces dernières. Cette recherche fonctionne dans les 4 sens : horizontal, vertical , diagonal, et diagonal inverse. Le principe est décrit comme suit :



1.2 Header : ColorChange.h

Ce "header" comprend les prototypes de 4 fonctions qui permettent le changement de la couleurs des pions encadrés par celle du joueur actuel . A partir de la case choisie, le traitement se fera dans les 4 sens : horizontal, vertical , diagonal, et diagonal inverse.Le principe est décrit comme suit :

]



1.3 Header : Others.h

Ce "header" comprend les prototypes de 3 différentes fonctions :

— afficher_plat : cette fonction permet l'affichage du plateau en jouant sur la console.

— pawn_calculator : celle-ci calcule le nombre des pions noirs et blancs à chaque tour.

— plein : cette dernière vérifie si toutes les cases sont remplies.

1.4 La fonction play.c

C'est la fonction principale du programme. Elle permet la gestion du jeu en se basant sur les fonctions décrites précédemment. Le principe es décrit comme suit :

Algorithm 1: Algorithme de la fonction play

```
Initialisation de la matrice avec des cases vides sauf les 4 cases centrales qui seront remplies comme est
  décrit dans la documentation du jeu;
do
    if aucun des joueurs n'a de cases possibles then
        afficher la table et calculer les pions de chacun des joueurs;
        SORTIR;
    end
    Recherche des cases possibles pour le joueur actuel;
    if Ce joueur n'a pas de coups légaux then
        Passer à la prochaine itération;
    end
    Afficher la table et Calculer les pions;
    do
        Lecture de la case choisie par le joueur actuel;
        if C'est une case non possible then
            Avertir le joueur de ceci;
        end
    while le joueur n'a pas choisi une case légale;
    Remplir la case choisie avec la couleur correspondante;
    Remettre les cases possibles à vide;
    Le changement de la couleurs des pions encadrés par celle du joueur actuel;
    Incrémenter la variable caractérisant l'itération actuelle;
while la table n'est pas entièrement remplie;
Déclaration du gagnant;
```

2 Livrable1

- Recommencer : avant de donner au jouer actuelle la main pour choisir une case , on lui propose l'option 'recommencer'. Si notre proposition est validée, retourner à la fonction play de nouveau.
- Stocker : Créer et enregistrer les joueurs et leurs caractéristiques (nom et score) sur fichier. Le principe est décrit comme suit :

Algorithm 2: Algorithme de la fonction Stocker

```
Ouverture du fichier texte titré "BaseDeDonnees" (ou stockés le nom et le score des joueurs précédents);
Création d'un nouveau fichier texte local titré "bdd";
while la fin du fichier n'est pas atteinte do
    Lecture de la ligne atteinte par le curseur;
    if le nom lu ne correspond pas à celui du joueur then
        Ajouter cette ligne lue dans le fichier "bdd";
    else
        Calculer le nouveau score du joueur ( ancien score + score actuel);
        Ajouter Son nom et son nouveau score dans "bdd";
    end
end
if le joueur n'est pas trouvé dans "BaseDeDonnees" then
    lui ajouter;
end
Fermeture des deux fichiers;
Suppression du fichier "BaseDeDonnees";
Changement du nom de "bdd" avec "BaseDeDonnees";
```

- PrintMoves : Mise à jour et Affichage de l'historique des mouvements effectués par les joueurs au cours du jeu.
- Permettre le chargement d'un jeu sauvegardé auparavant

— ***Upload*** : le sauvegarde d'une partie Complète ou non. Le principe est décrit comme suit :

Algorithm 3: Algorithme de la fonction Upload

```
Ouverture du fichier texte nommé "gamepart";
Lecture d'un nom caractéristique de la partie Sauvegarder en vérifiant l'unicité du nom entré;
Ecriture de ce nom dans le fichier "gamepart";
Ecriture des mouvements effectués;
Fermeture du fichier;
```

— ***Reload*** : le chargement d'une partie sauvegardée. Le principe est décrit comme suit :

Algorithm 4: Algorithme de la fonction Reload

```
Ouverture du fichier texte nommé "gamepart";
while la fin du fichier n'est pas atteinte do
    Lecture d'un nom du fichier;
    if Ce nom lu correspond au nom de la partie désiré then
        Lecture des coordonnées d'une case;
        Mise à jour du curseur;
        Sortir;
    end
end
Fermeture du fichier;
```

— topten : Donner la liste des dix meilleurs scores. Le principe est décrit comme suit :

Algorithm 5: Algorithme de la fonction Topten

```
Ouverture du fichier texte titré "BaseDeDonnees" (ou stockés le nom et le score des joueurs précédents);
Création d'un nœud contenant comme info nom et score de la 1ere ligne lue, et pointant sur NULL;
Insertion de ce nœud dans une liste initialement vide;
while la fin du fichier n'est pas atteinte do
    Création d'un nœud contenant comme info nom et score de la ligne suivante, et pointant sur NULL;
    Insertion, en ordre décroissant en terme de scores, de ce nœud dans la liste;
end
Fermeture du fichier;
Affichage des 10 premiers nœuds de la liste;
```

3 Livrable2

PlayAI : Jouer contre un non humain, le principe est décrit comme suit :

Algorithm 6: Algorithme de la fonction play$_AI$

```
1  Affectation aléatoire d'une couleur au joueur;
2  Initialisation de la matrice avec des cases vides sauf les 4 cases centrales qui seront remplies comme est
     décrit dans la documentation du jeu;
3  do
4      if aucun des joueurs(humain ou non humain) n'a de cases possibles then
5          afficher la table et calculer les pions de chacun des joueurs;
6          SORTIR;
7      end
8      Recherche des cases possibles pour le joueur actuel;
9      if il n y'a pas de coups légaux then
10         Passer à la prochaine itération;
11     end
12     if c'est le tour du joueur humain then
13         Afficher la table et Calculer les pions;
14         do
15             Lecture de la case choisie par le joueur actuel;
16             if C'est une case non possible then
17                 Avertir le joueur de ceci;
18             end
19         while le joueur n'a pas choisi une case légale;
20     else
21         if le jeu est de niveau facile then
22             Choix aléatoire d'une case parmi celles possibles;
23         else
24             La mise en place de l'algorithme "min-max ave l'élagage Alpha-Beta" permettant
                 d'optimiser la recherche du meilleur coup, en limitant le nombre de cases visitées;
25         end
26     end
27     Affichage de l'historique des mouvements joués;
28     Remplir la case choisie avec la couleur correspondante;
29     Remettre les cases possibles à vide;
30     Le changement de la couleurs des pions encadrés par celle du joueur actuel;
31     Incrémenter la variable caractérisant l'itération actuelle;
32 while la table n'est pas entièrement remplie;
33 Déclaration du gagnant;
34 Mise à jour du fichier "BaseDeDonnees" en tenant compte du score et du nom du joueur;
```

4 Livrable3

- Sécurité des accès :
  - ***login*** : Il s'agit de certifier l'identité du joueur en lui autorisant 3 chances pour entrer son mot de passe.
  - ***sign_ up*** : Vérifier l'unicité du nom d'utilisateur, et lui permettre la création de son propre mot de passe et sa confirmation.
- ***La réalisation d'une interface de jeu*** : Il s'agit d'un ensemble de fonctions permettant une visualisation améliorée de la procédure du jeu. (voir réalisation et concept)

Réalisation et Résultats

1 Outils utilisés

La réalisation de ce projet a été soutenu par divers outils :

— ***Éditeurs de code*** : L'édition de la majorité du code a été faite à l'aide de ATOM et VSCODE. Ensuite lors de la manipulation de l'interface graphique, nous avons eu recours à CODE : :BLOCKS.

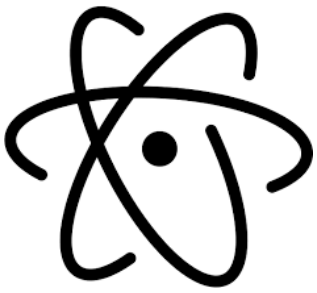


FIGURE 2 – LOGO de ATOM

FIGURE 3 – LOGO de Visual Studio Code

FIGURE 4 – LOGO de Code : :Blocks

— ***Outils graphiques*** : La réalisation de l'interface graphique du jeu a été établie en nous appuyant sur l'ensemble des fonctions de la librairie GTK en utilisant le concepteur GLADE qui prend en charge toute la partie de gestion/génération de l'interface .

FIGURE 5 – LOGO de GTK

Le choix de la librairie GTK a été fait vu la possibilité de manipulation du contenu textuel et la gestion des boutons de menu contrairement à SDL.

— ***Outils de communication*** : Vue la pandémie COVID19, un travail collectif organisé a été difficile d'être réalisée voire impossible sans les outils suivants : * la plateforme GITHUB qui a permis l'organisation de l'évolution du code.(voir lien repository en bibliographie)
* Le logiciel Discord avec lequel, nous avons pu écrire toutes les phases du code ensemble sans séparation des taches, ce qui a permis une diversité des idées et une compréhension de la totalité du programme.

— ***Éditeur de rapport*** : LATEX nous a permis une édition professionnelle du rapport.

2 Fonctions du code source

2.1 Librairies et déclarations

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define true 1
#define false 0
typedef char matrice[8][8];
typedef struct { int Wh; int Bl;} couple;
typedef struct { char username[30]; int score;} playerID;
typedef struct {int x ; int y ; int cur; } cordonnee;
typedef struct _joueur{ playerID *info; struct _joueur *next;} joueur;
typedef joueur *joueurliste;
```

2.2 Code du jeu

– possiblecase.c :

```c
void hori_possible(matrice M,int l, int k,char car1,char car2) {…}
void verti_possible(matrice M,int l, int k,char car1,char car2) {…}
void diag_possible(matrice M, int l, int k,char car1,char car2) {…}
void anti_diag_possible(matrice M, int l, int k,char car1,char car2) {…}
int possible_case(matrice M) {…}
```

Input : M matrice qui décrit le plateau du jeu, i et j cordonnées de la case traitée, car1 et car2 les couleurs N et B selon le tour.
Output : ne retourne aucune valeur.

– ColorChange.c :

```
void hori_color_change(matrice M, int k, int l, char car1,char car2) {⋯}
void verti_color_change(matrice M, int k, int l, char car1,char car2) {⋯}
void diag_color_change(matrice M, int k, int l, char car1,char car2) {⋯}
void anti_diag_color_change(matrice M, int k, int l, char car1,char car2) {⋯}
```

Input : M matrice qui décrit le plateau du jeu, k et l cordonnées de la case choisie par le joueur, car1 et car2 les couleurs N et B selon le tour.
Output : ne retourne aucune valeur.

- Others.c :
  - Afficher_plat :

```
void afficher_plat(matrice M) {⋯}
```

Input : M matrice qui décrit le plateau du jeu.
Output : void.

  - pawn_calculator :

```
couple pawn_calculator(matrice M) {⋯}
```

Input : la matrice M.
Output : structure de type couple

  - Plein :

```
int plein(matrice M) {⋯}
```

Input : la matrice M.
Output : Si M est non plein elle retourne 0, et 1 sinon.

- Play_humain :

```
void play_human(playerID player1, playerID player2, int load,char partID[30],int up) {⋯}
```

Input : player1 et player2 de type la structure player ID, partID le nom de la partie, load variable qui vérifie si le joueur veut charger une partie sauvegardée auparavant, up variable qui traduit la possibilité de charger une partie.
Output : void

2.3 Livrable 1

```
void stocker(playerID player) {…}
void print_moves(int tab[64][2], int iter){…}
void upload(int tab[64][2],int iter) {…}
cordonnee reload(cordonnee cord, char partID[30]){…}
joueurliste inorder_insert(joueurliste liste,joueur *newplayer){…}
void topten(){…}
```

- Stocker :
  Input : structure de type playerID.
  Output : void.
- Printmoves :
  Input : tab : tableau de taille nombres de cases * 2 , iter compteur du nombre des tours passés.
  Output : void.
- Upload :
  Input : tab : tableau de taille nombres de cases * 2 , iter compteur du nombre des tours passés.
  Output : void.
- Reload :
  Input : coord : une structure de type coordonnee , partID le nom caractérisant la partie à charger.
  Output : retourne le structure de type coordonnee.
- topten :
  Input : void.
  Output : void.

2.4 Livrable 2

```
void play_AI(playerID player1,int level){…}
```

play_IA :
Input : player1 une structure de type playerID, level : variable entière qui égale à 0 s'il s'agit d'un niveau facile et 1 sinon.
Output : void.

2.5 Livrable 3

```
void login(playerID player) {…}
void signup(playerID player1) {…}
```

- Login :
  Input : player de type playerID.

Output : void.

- Signup :
  Input : player de type playerID.
  Output : void.

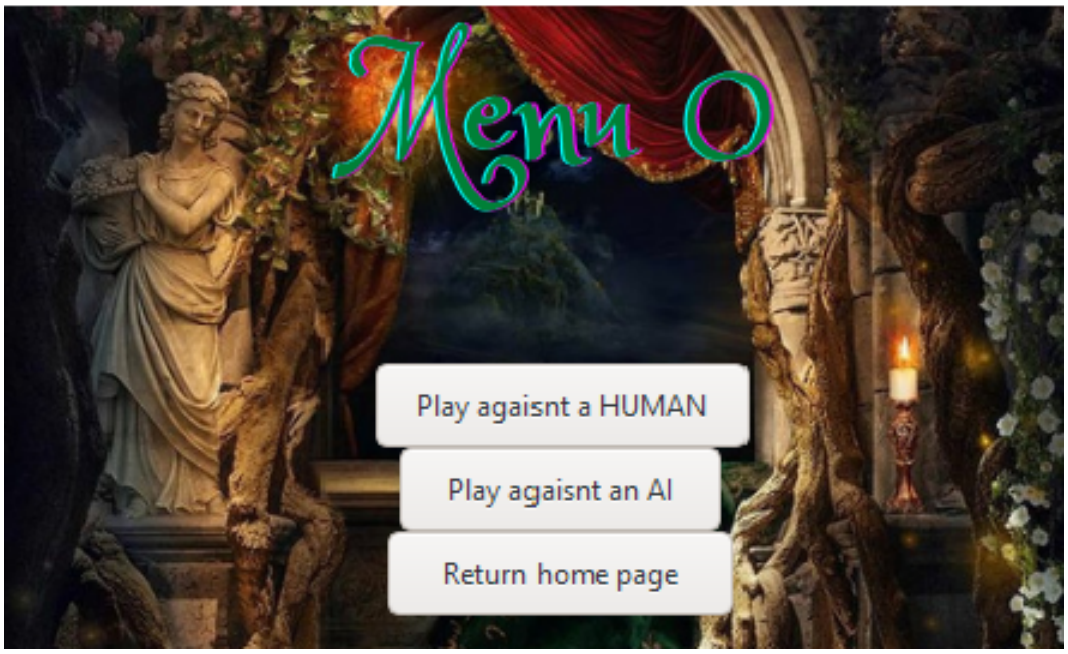
3 Interface Graphique

L'interface graphique a permis de faciliter la gestion du jeu à travers les menus,la bonne visualisation du table du jeu et facilite surtout la communication avec l'utilisateur :

- main :

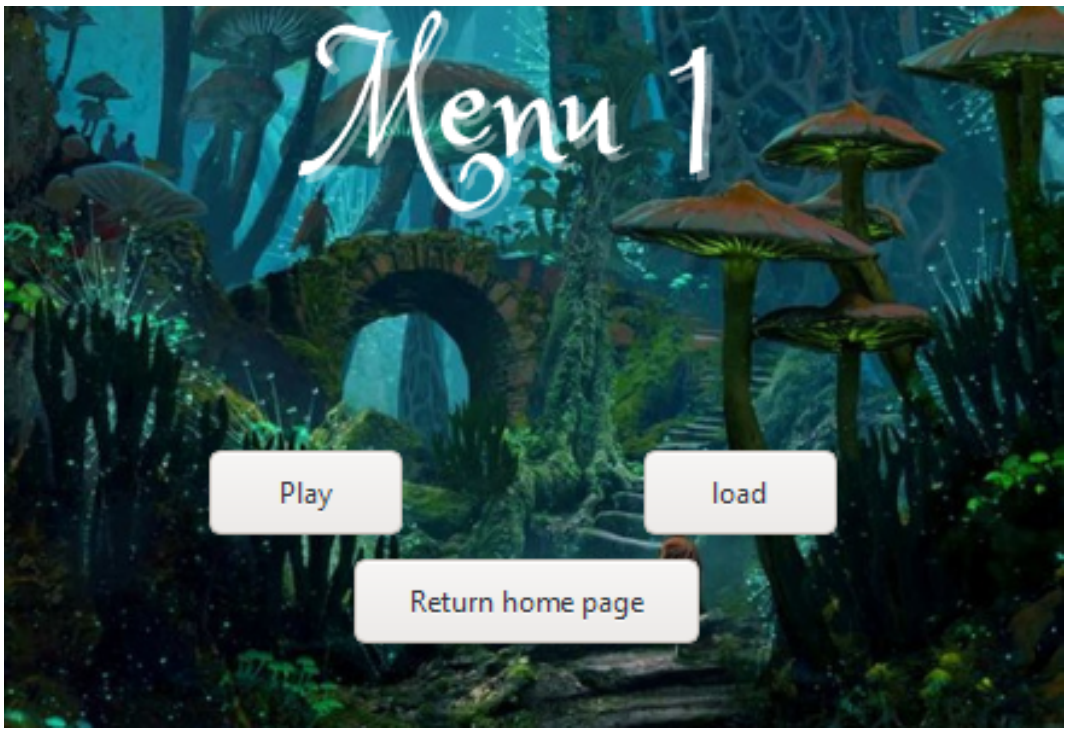
Le bouton login permet l'authentification du joueur.
Le bouton sign up permet l'inscription d'un nouveau joueur.
Le Bouton exit pour quitter.

- menu0 :



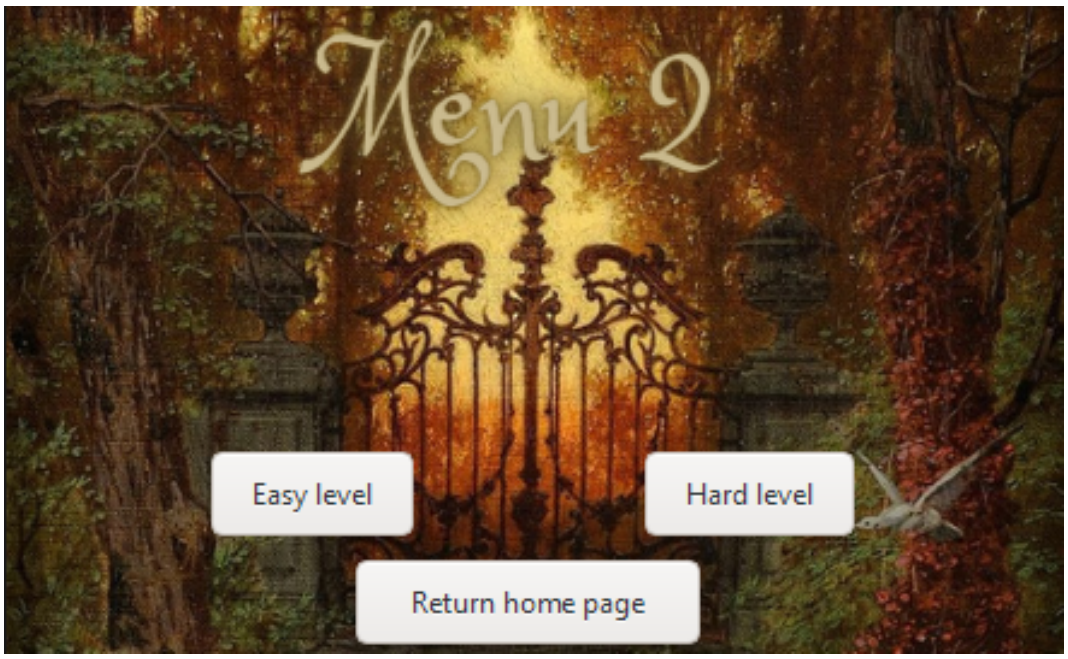
Le bouton play against human pour jouer contre un humain invité, il renvoie vers menu1.
Le bouton play against AI pour jouer contre un non humain, il renvoie vers menu2.
Le bouton return home pour retourner home page.

— menu1 :



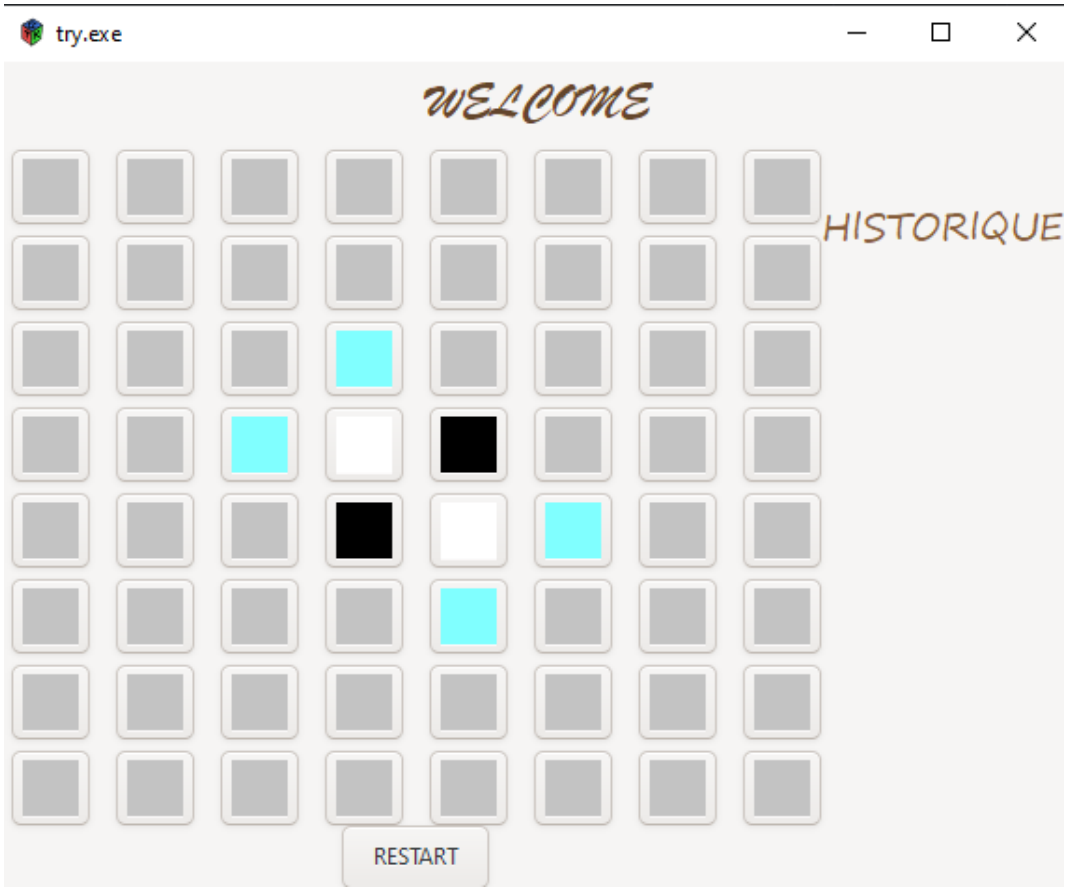
le bouton play pour jouer.
Le bouton load pour charger une partie sauvegardée auparavant.
Le bouton return pour retourner vers menu0.

— menu2 :

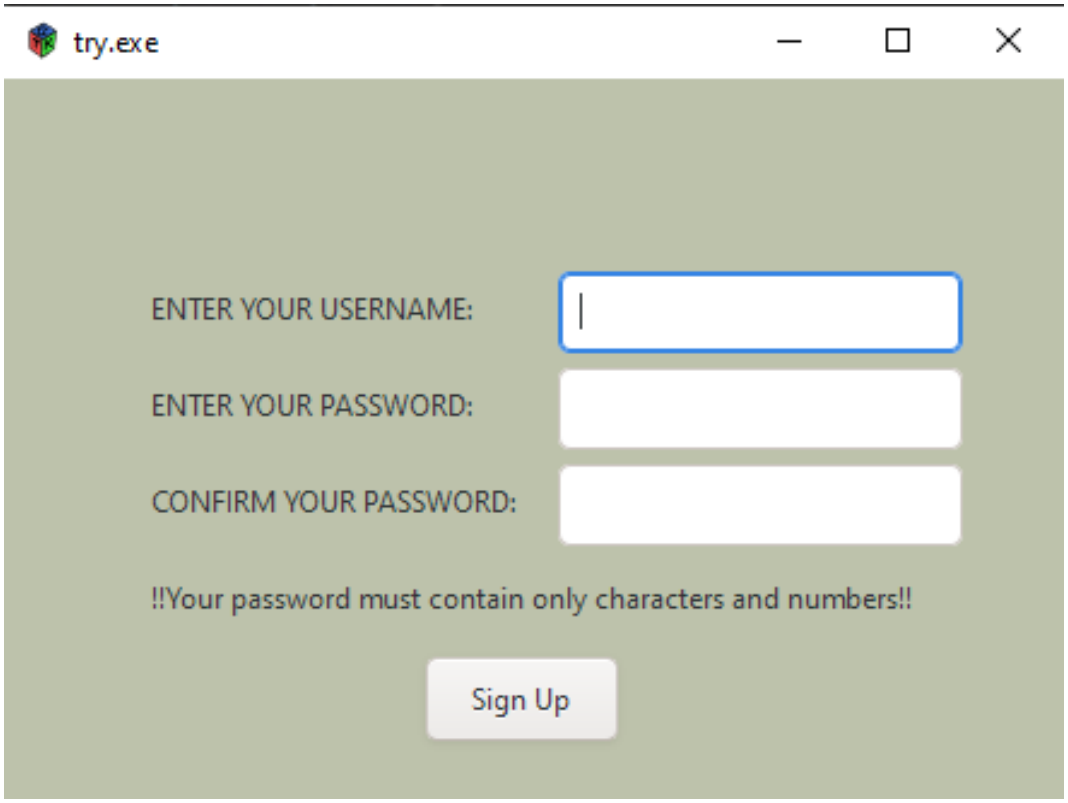


Le bouton easy pour un niveau facile lors du jeu contre l'IA.
Le bouton hard pour un niveau plus difficile.
Le bouton return pour retourner vers menu0.

– Table du jeu :



– La fonction Sign$_U p$



4 Difficultés rencontrées

La découverte de diverses technologies durant ce projet telles que latex, git. . ., l'Installation et la lecture de la documentation de GTK, la familiarisation avec Glade, étaient des taches difficiles à être réalisées dans un temps si serré. En outre, le chevauchement du projet c avec les examens a été un obstacle devant le perfectionnement du projet C, Surtout que la compréhension du cahier de charges nous a consommé beaucoup de temps. Suite à ces contraintes, nous n'avons pas pu terminer l'interface graphique, ni arriver à exécuter correctement l'algorithme minimax.

Conclusion

Notre projet a consisté à la réalisation du jeu Othello. Ce qui nous a permis de cristalliser nos connaissances théoriques acquises durant ce semestre.Ce projet nous a introduit à la théorie des jeux. La majorité des fonctionnalités demandées dans le cahier de charges ont été réalisées , espérons que nous serons à la hauteur de vos estimations.

Bibliographie

https://developer.gnome.org/gtk3/stable
https://glade.gnome.org/
https://www.youtube.com/watch?v=6ehiWOSp_wk
https://www.overleaf.com/
https://www.ffothello.org/
https://github.com/Orphia/Othello