

JEU DE GO



PROJET DE PROGRAMMATION EN C

| REALISE PAR:

- * YASSINE AMCHAROD
- * SALAH-EDDINE ETANGI

| ENCADRE PAR:

- * Mme Cherrabi

TABLE DES MATIERES :

| | |
|---------------------------------------|-----------|
| 1. INTRODUCTION..... | 3 |
| 2. DECOVERTE DE NOTRE JEU..... | 6 |
| 3. ANALYSE ET CONCEPTION..... | 22 |
| 4. DEVELOPEMENT DU JEU..... | 28 |
| 5. AMELIORATION..... | 54 |
| 6. PROBLEMES RENCONTRES..... | 59 |
| 7. CONCLUSION..... | 62 |

I. INTRODUCTION :

● PRESENTATION DU JEU :

Le jeu de go est né en Chine il y a plusieurs milliers d'années. Il se joue au Japon depuis 1200 ans, mais il ne s'est répandu que récemment en occident. Le but du jeu est la constitution de territoires en utilisant un matériel des plus simples : un plateau, appelé goban, sur lequel est tracé un quadrillage et des pions, appelés pierres, que l'on pose sur les intersections de ce quadrillage à tour de rôle. Les règles s'apprennent en quelques minutes et permettent aux débutants de faire rapidement des parties passionnantes. Ensuite, ceux qui voudront explorer plus avant les subtilités du jeu pourront rejoindre un club et participer à des tournois. Ils pourront alors constater que sous son apparente simplicité qui le rend accessible même aux plus jeunes, le jeu de go est d'une richesse inépuisable.

● REGLES DU JEU :

a) MATERIEL :

Le matériel de jeu traditionnel se compose d'un **goban** sur lequel est tracé un quadrillage de 19x19 lignes, soit 361 intersections, et de pierres qui sont soit noires, soit blanches.

Mais rien n'empêche les joueurs d'utiliser un autre matériel, et en particulier des gobans de 13x13 ou 9x9 lignes pour les parties d'initiation.



b) DÉROULEMENT DU JEU :

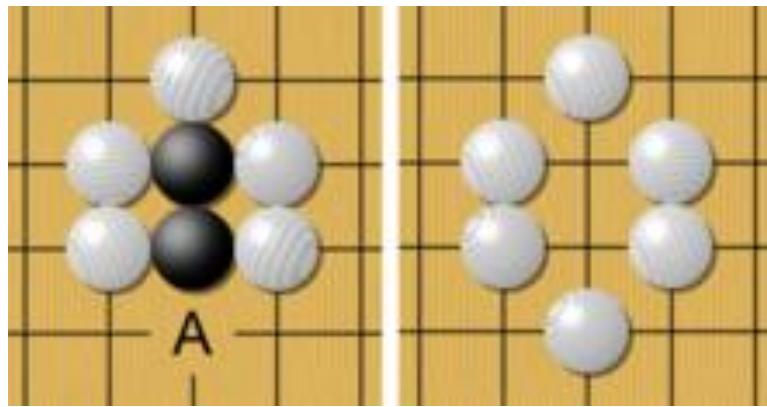
Le go se joue à deux. Celui qui commence joue avec les pierres noires et l'autre avec les blanches. A tour de rôle, les joueurs posent une pierre de leur couleur sur une intersection inoccupée du goban ou bien ils passent.

Passer sert essentiellement à indiquer à l'adversaire que l'on considère la partie terminée.

c) CAPTURE :

Lorsqu'un joueur supprime la dernière liberté d'une chaîne adverse, il la capture en retirant du goban les pierres de cette chaîne. De plus, en posant une pierre, un joueur ne doit pas construire une chaîne sans liberté, sauf si

par ce coup il capture une chaîne adverse. Lorsqu'une chaîne n'a plus qu'une liberté, on dit qu'elle est en **Atari**.



d) FIN DE PARTIE :

La partie s'arrête lorsque les deux joueurs passent consécutivement. On compte alors les points. Chaque intersection du territoire d'un joueur lui rapporte un point, ainsi que chacune de ses pierres encore présentes sur le goban.

Par ailleurs, commencer est un avantage pour Noir. Aussi, dans une partie à égalité, Blanc reçoit en échange des points de compensation, appelés komi. Le komi est habituellement de 7 points et demi (le demi-point sert à éviter les parties nulles). Le gagnant est celui qui a le plus de points.

● BUT DU PROJET :

Le projet consiste à développer le jeu de GO en utilisant le langage C, le jeu à créer doit implémenter la possibilité d'avoir 2 joueurs Hommes, et aussi de pouvoir jouer contre la machine, sans oublier les problèmes du TSUME GO. Dans ce projet, on travaillera sur la version qui est de **taille 9x9**.

● TRAVAIL REALISE :

On a réalisé une version du jeu de Go en console en utilisant le langage C .

II. DECOUVERTE DE NOTRE JEU :

Dans ce paragraphe, on va essayer de montrer le scénario de jeu et son fonctionnement. On expliquera le rôle des différents menus et l'intégralité des options offertes par le jeu.



PAGE D'ACCUEIL :

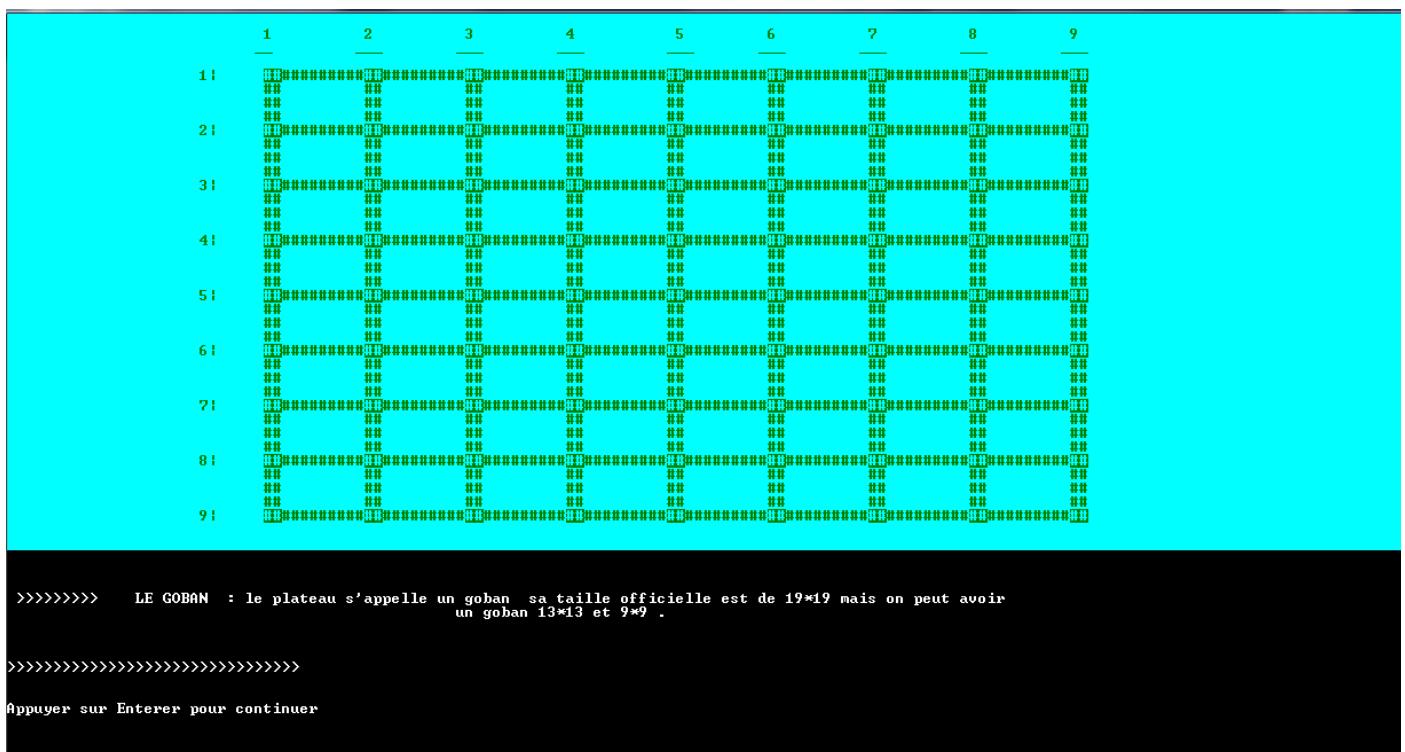
La première interface qui s'affiche lors du lancement de jeu contient trois options (JOUER, REGLES et QUITTER) :



 **JOUER** : permet d'accéder au jeu .

 **REGLES** : permet d'afficher les règles du jeu écran par écran (voir ci-dessous) :

Pour chaque règle il y a une simulation qui clarifier les notions et les outils traités dans le texte.



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|----|----|----|----|----|----|----|----|----|----|
| 11 | ## | ## | ## | ## | ## | ## | ## | ## | ## |
| 21 | ## | ## | ## | ## | ## | ## | ## | ## | ## |
| 31 | ## | ## | ## | ## | ## | ## | ## | ## | ## |
| 41 | ## | ## | ## | ## | ## | ## | ## | ## | ## |
| 51 | ## | ## | ## | ## | ## | ## | ## | ## | ## |
| 61 | ## | ## | ## | ## | ## | ## | ## | ## | ## |
| 71 | ## | ## | ## | ## | ## | ## | ## | ## | ## |
| 81 | ## | ## | ## | ## | ## | ## | ## | ## | ## |
| 91 | ## | ## | ## | ## | ## | ## | ## | ## | ## |

>>>>> LES JOUEURS : le GO se joue à deux et chaque joueur est associé à un couleur soit blanc ou noir .

>>>>> JOUER : les joueur peuvent poser un pion de leur couleur sur une intersection inoccupée en choisissant la ligne et la colonne .

>>>>>>>>>>>>>>>

Appuyer sur Entrer pour continuer

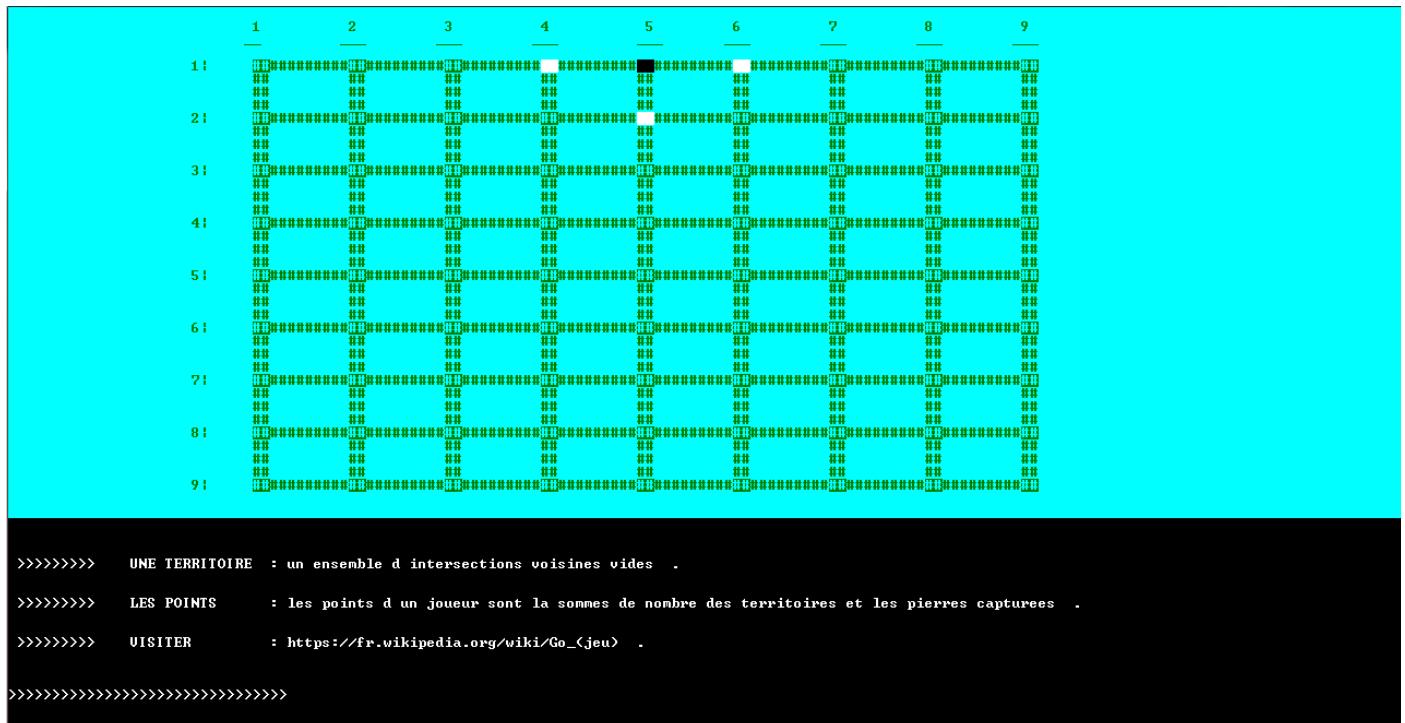
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|----|----|----|----|----|----|----|----|----|----|
| 11 | ## | ## | ## | ## | ## | ## | ## | ## | ## |
| 21 | ## | ## | ## | ## | ## | ## | ## | ## | ## |
| 31 | ## | ## | ## | ## | ## | ## | ## | ## | ## |
| 41 | ## | ## | ## | ## | ## | ## | ## | ## | ## |
| 51 | ## | ## | ## | ## | ## | ## | ## | ## | ## |
| 61 | ## | ## | ## | ## | ## | ## | ## | ## | ## |
| 71 | ## | ## | ## | ## | ## | ## | ## | ## | ## |
| 81 | ## | ## | ## | ## | ## | ## | ## | ## | ## |
| 91 | ## | ## | ## | ## | ## | ## | ## | ## | ## |

>>>>> UNE CHAINE : deux pierres voisines et une chaîne est un ensemble de pierres voisines de proches en proches .

>>>>> CAPTURE : si un joueur supprime la liberté d'une chaîne de l'autre joueur il la capture .

>>>>>>>>>>>>>

Appuyer sur Entrer pour continuer



 **QUITTER** : pour sortir du jeu .

➤ COMMENCER A JOUER :

Si on choisit JOUER , on passe immédiatement à la page contenant les différentes modes de jeu (JOUEUR vs JOUEUR et JOUEUR vs MACHINE).

- **JOUEUR vs JOUEUR** : permet d'avoir une partie entre deux joueurs humains.
- **JOUEUR vs MACHINE** : permet d'avoir une partie entre la machine et un joueur humain.
- **RETOUR** : permet de retourner vers la page d'accueil.



❖ JOUEUR vs JOUEUR :

La première interface qui s'affiche demande aux deux utilisateurs leurs noms :



Puis une autre interface qui demande leurs niveaux (qui doivent être entre 1 et 9) pour voir est ce qu'il existe un handicap entre les deux joueurs:

Puis une interface qui associe à chaque joueur un couleur (noir ou blanc).

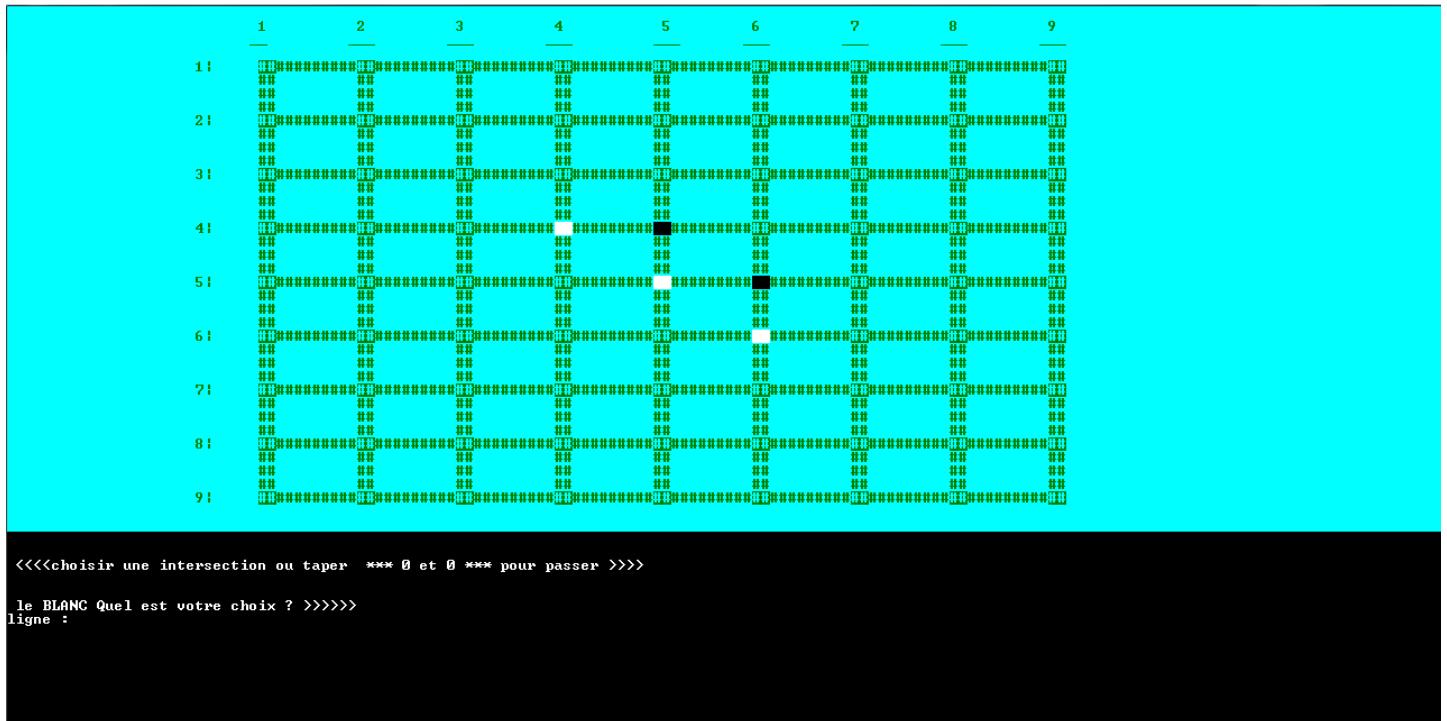
Remarque :

- Si le handicap est égal à 0 la machine attribue à chaque joueur son couleur d'une manière aléatoire.
 - Si le handicap est égal à n ($n < 9$) le joueur blanc a comme avantage de poser n pierres successivement avant que le noir puisse jouer.

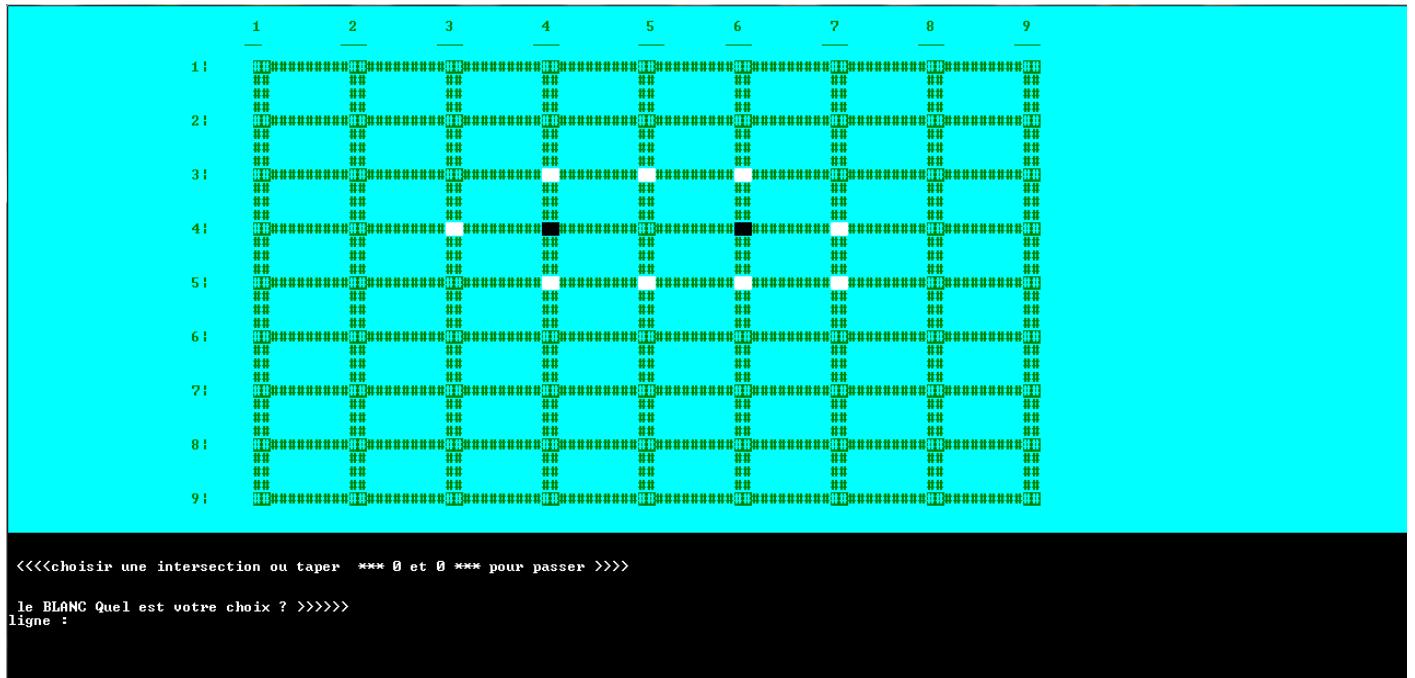
```
<<< choisir une intersection [ 2 pierre de handicap ] >>>  
  
le BLANC Quel est votre choix ? >>>>  
ligne :
```



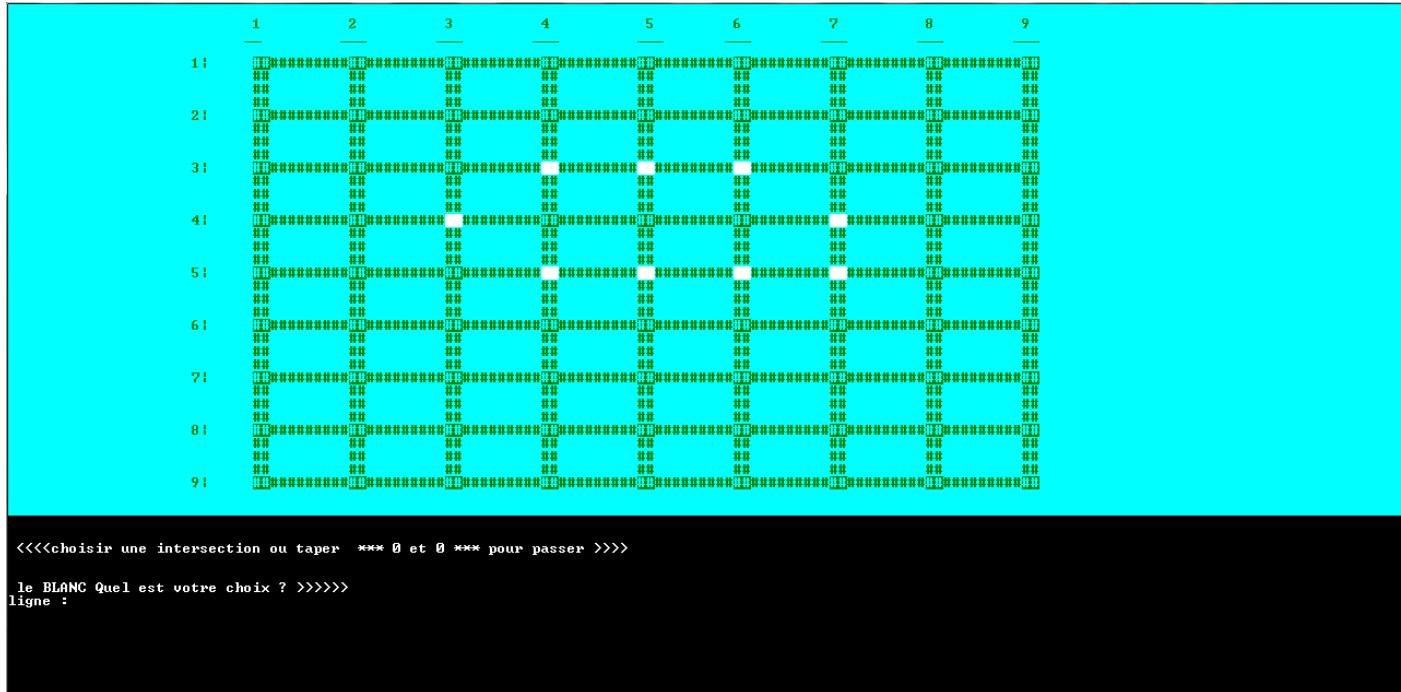
Apres avoir joué les pierres d'handicap le joueur noir a le droit de jouer son tour :



Si on suppose que le blanc entoure une chaîne de noir :



Le blanc capture les trois pierres noires ;



Si les deux joueur choisissent de **passer**, la partie se termine puis une interface se lance nommant le joueur gagnant et le score des deux joueurs :

Ou bien :

Dans le cas où le noir gagne .

❖ JOUEUR vs MACHINE :

Dans ce choix il existe deux options majeures

- + FACILE : permet de jouer avec la machine qui joue aléatoirement.
 - + TSUMEGO : qui contient des problèmes de GO et la machine connaît leurs solutions



A. < FACILE >

La première interface qui s'affiche demande de l'utilisateur son nom :

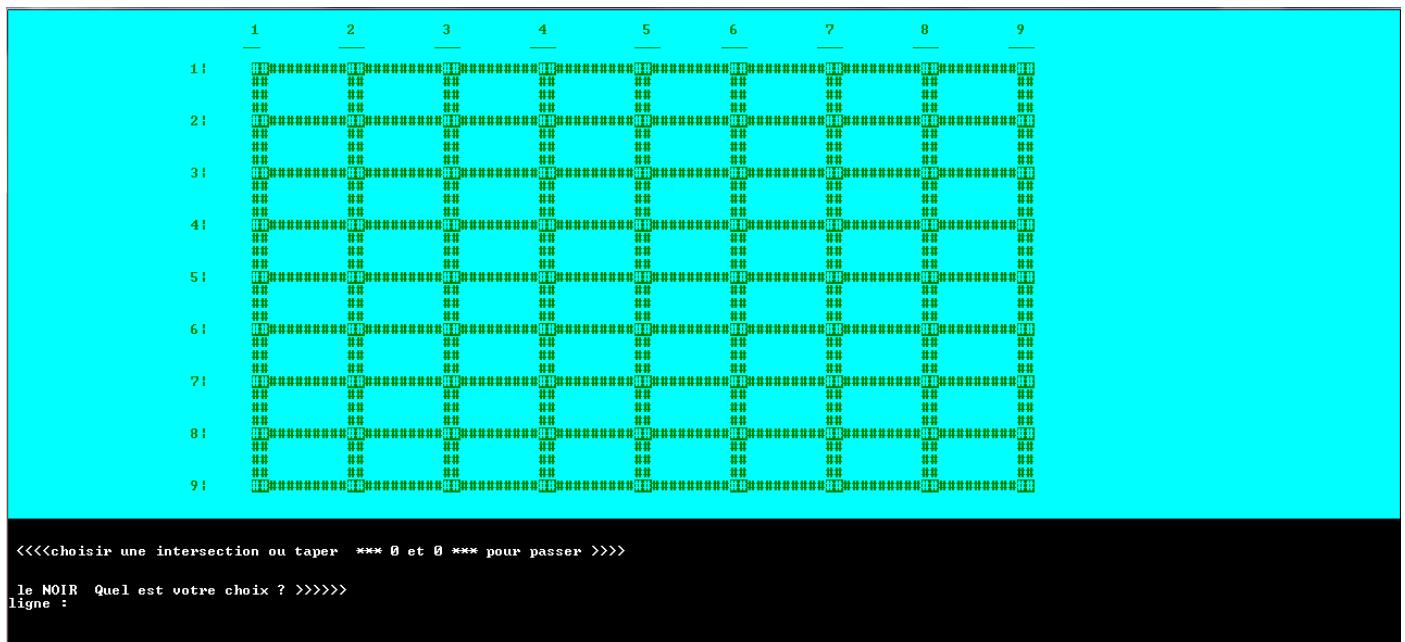


Puis la machine associe **aléatoirement** un couleur (noir ou blanc) au joueur et a elle-même.



Puis la partie commence :

La machine choisit d'une manière aléatoire comme si tu joue avec un bébé.



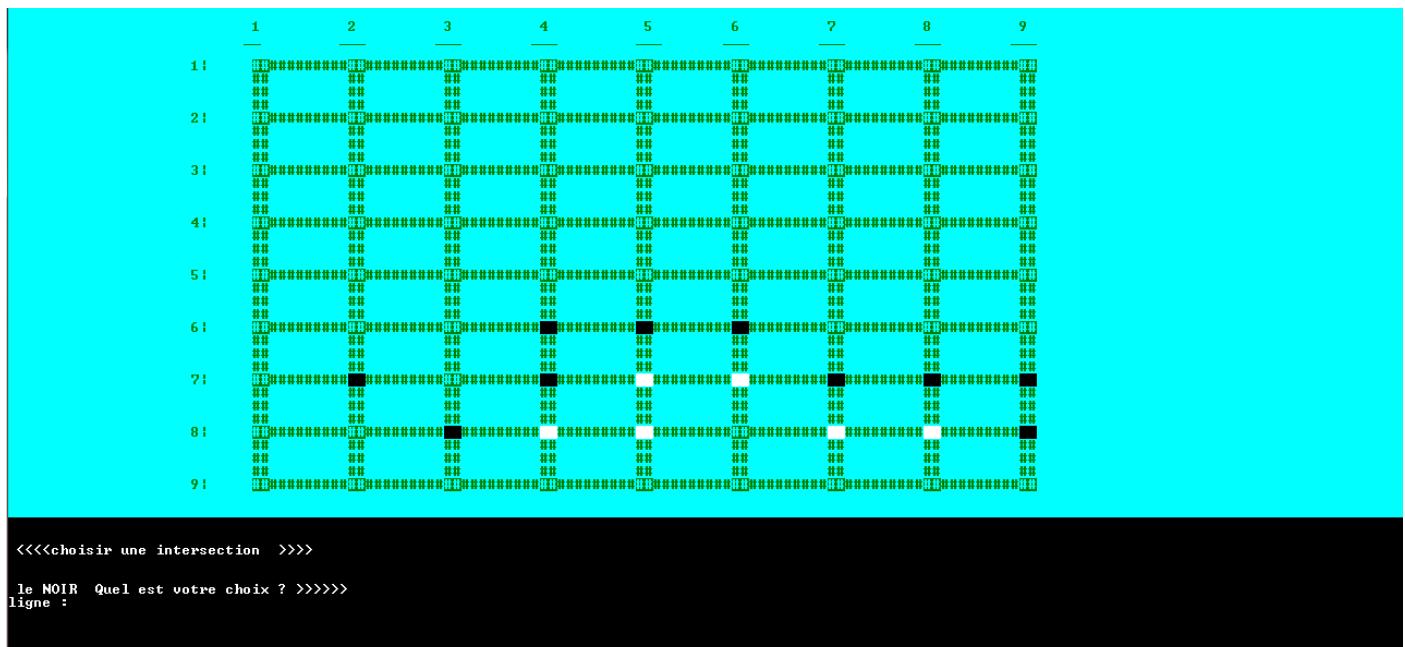
B. < TSUMEGO >

Après ce choix, une interface accueillante s'affiche pour clarifier les choses et expliquer l'objectif de cette partie:



En appuyant sur entrée le premier problème se commence (en fait , on a manipuler 6 problèmes de TSUMEGO) :

1 :



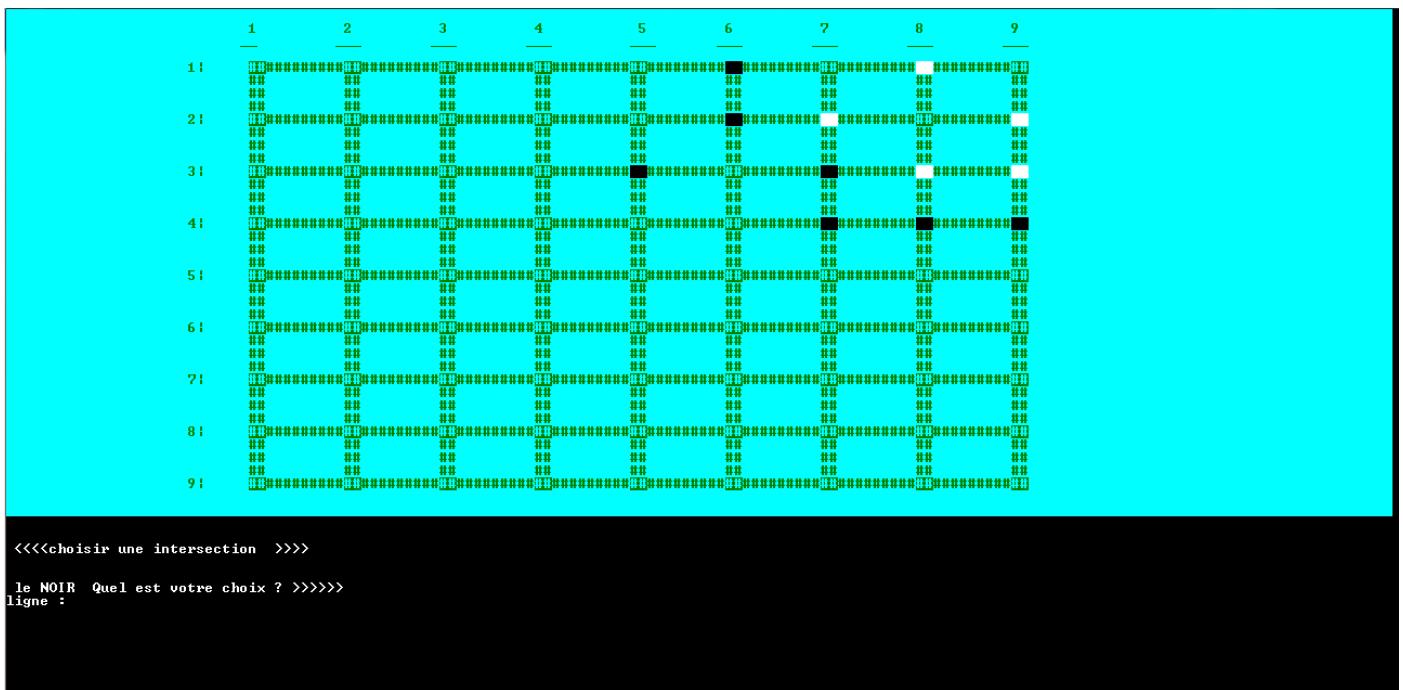
2 :

3 :

4 :

5:

6 :



Remarque :

- le joueur joue toujours le noir et la machine le blanc.
- Si La machine essaye de faire deux yeux dans le territoire donné, le joueur doit essayer de l'arrêter et vice-versa.
- Si le noir a réussi (c'est-à-dire la machine n'a pas deux yeux dans le territoire) le noir gagne le territoire :

```

>>>> le noir bravo tu a gagne ce territoire
***** entrer 1 pour ressayer et 2 pour passer au suivant *****

```

- Si la machine a deux yeux dans ce territoire le blanc gagne ce territoire :

```
>>>> le blanc a gagne ce territoire  
***** entrer 1 pour ressayer et 2 pour passer au suivant *****
```

III . ANALYSE ET CONCEPTION :

i. ANALYSE DE PROBLEMATIQUE :

Pour construire le jeu de go il faut d'abord tester le jeu et analyser tous ses aspects, et voir ses profondeurs. c'est pourquoi on a consacré beaucoup du temps en jouant le go sur notre smartphones et dans des sites spécialisé dans ce jeu qui est vraiment difficile.

Après avoir joué le GO de nombreuses fois. Il est devenu clair que la construction d'un tel jeu requiert la réponse à certaines questions principales :

- o Comment représenter le plateau de jeu et les pierres ?*
- o Comment vérifier si l'un des deux joueurs a gagné la partie ?*
- o Comment construire une intelligence artificielle capable de confronter le joueur humain ?*

Il existe également d'autres questions dont la réponse vise à augmenter la qualité de notre propre jeu :

o Quelle serait la meilleure interface qui permettra au joueur une fluidité dans l'utilisation ?

o Quelles fonctionnalités additionnelles peut-on intégrer pour améliorer l'expérience de l'utilisateur.

ii. CONCEPTION et MODELISATION :

Dans cette partie on essayera à répondre aux questions précédemment posées.



le plateau de jeu et les pierres :

Le GOBAN de GO se compose de 81 intersections, et chaque intersection peut prendre un état parmi les trois états suivants :

1. intersection vide
2. intersection remplie par le Joueur 1 (pierre noire)
3. intersection remplie par le Joueur 2 (pierre blanche)

On a représenté alors le plateau par un tableau à deux dimension (matrice) de taille 9×9 nommé « matrice ». Pour avoir l'état d'une intersection, il suffit de la donner ses coordonnées comme indice. La modélisation a été fait comme ceci : Pour $1 \leq i \leq 9$ et $1 \leq j \leq 9$:

- si $\text{matrice}[i][j] = 0$ alors l'intersection « i, j » est vide
- si $\text{matrice}[i][j] = 1$ alors l'intersection « i, j » est remplie par une pierre noire (Joueur noir).
- si $\text{matrice}[i][j] = 2$ alors l'intersection « i, j » est remplie par une pierre blanche (Joueur blanc).

➤ Vérifier si l'un des deux joueurs a gagné la partie :

On ne cherche le gagnant qu'après la fin de la partie (pour nous c'est quand les deux joueurs passent). Si c'est le cas on calcule le score ;

Avec *le score= nombre de prisonniers + le nombre des intersections dans son territoire.*

Pour cela on a défini pour chaque joueur des variables de type entiers qui sont (score_noir , terre_noir , score_blanç et terre_blanç)

Score_noir (resp. score_blanç) ont aussi aidé pour contenir le nombre des pierres prisonniers de l'adversaire.

Terre_noir (resp. terre_blanç) contient le nombre des intersections qui appartiennent aux joueurs.

Donc pour nous le score de noir= score_noir + terre_noir.

Et le **gagnant** pour nous c'est celui qui a **le score le plus élevé.**

➤ construire une intelligence artificielle :

Pour faire une intelligence on a manipulé des problèmes de TSUME GO et aussi leurs solutions ; dans ces problèmes le joueur qui a les pierres noires essaye d'arrêter la machine qui a les pierres blanches de prendre un territoire donné et vice-versa.

C'est pour cela on a cherché des problèmes de TSUME GO et de les résoudre puis de faire apprendre ces solutions à la machine ; et cela a donné à notre programme une base de 6 problèmes auxquels il connaît la solution auparavant. La fonction qui assure cette partie des TSUME GO s'appelle « joueur_ordinateur_tsumego » .

➤ la meilleure interface qui permettra au joueur une fluidité dans l'utilisation :

Pour simplifier l'utilisation de notre jeu on a intégré des options qui vont aider nos utilisateurs et donner un bon style à notre jeu.

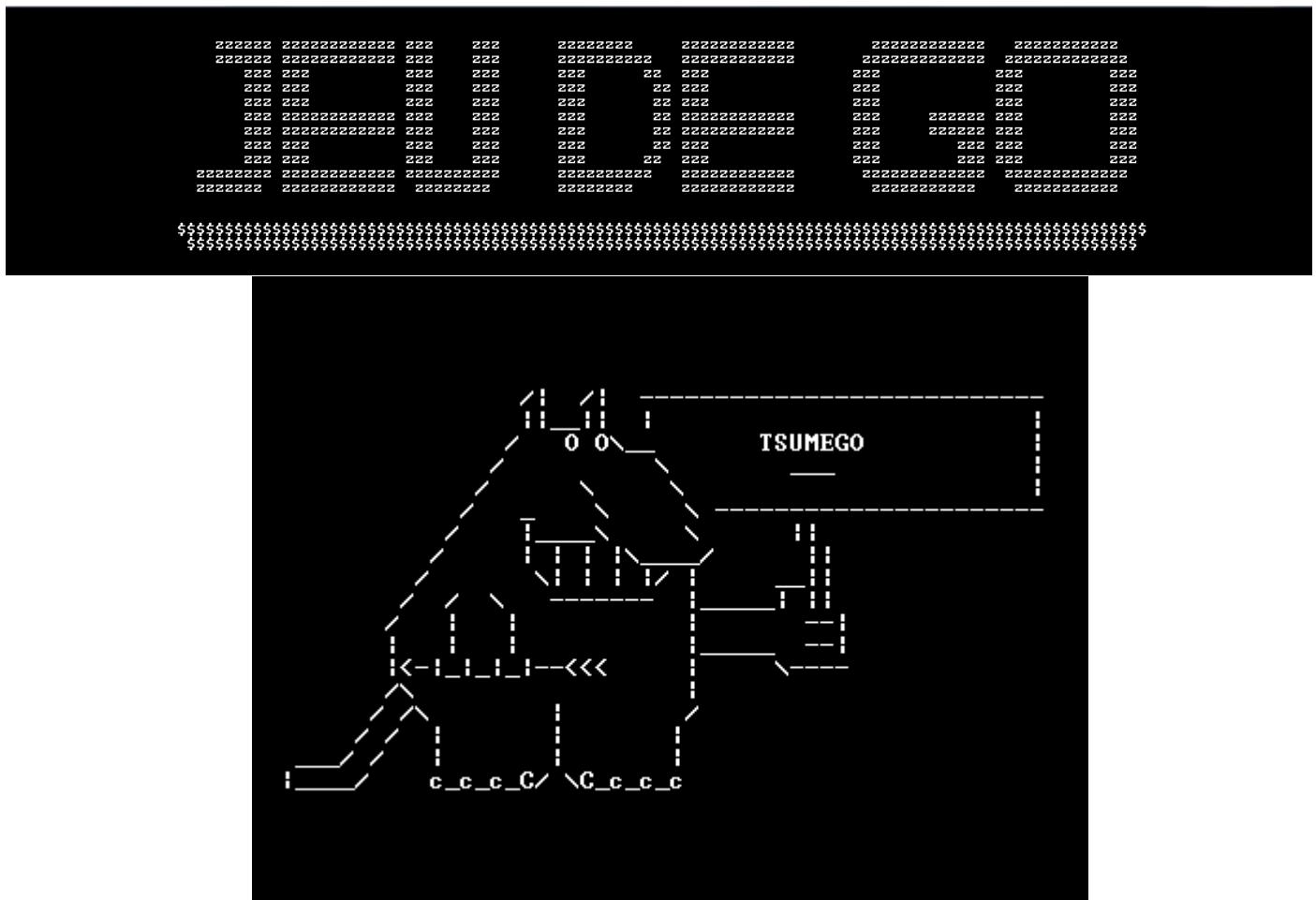
« RETOUR » : pour naviguer entre les différents écrans du jeu.
« QUITTER » : pour sortir du jeu .
« REGLES » : pour expliquer les règles de bases du jeu.
« suivant » et « ressayer » : dans les TSUME GO.

➤ fonctionnalités additionnelles qu'on puisse intégrer pour améliorer l'expérience de l'utilisateur .

■ ascii art :

Pour rendre la console plus agréable on a utilisé l'ascii art qui consiste à dessiner en utilisant les caractères du clavier.

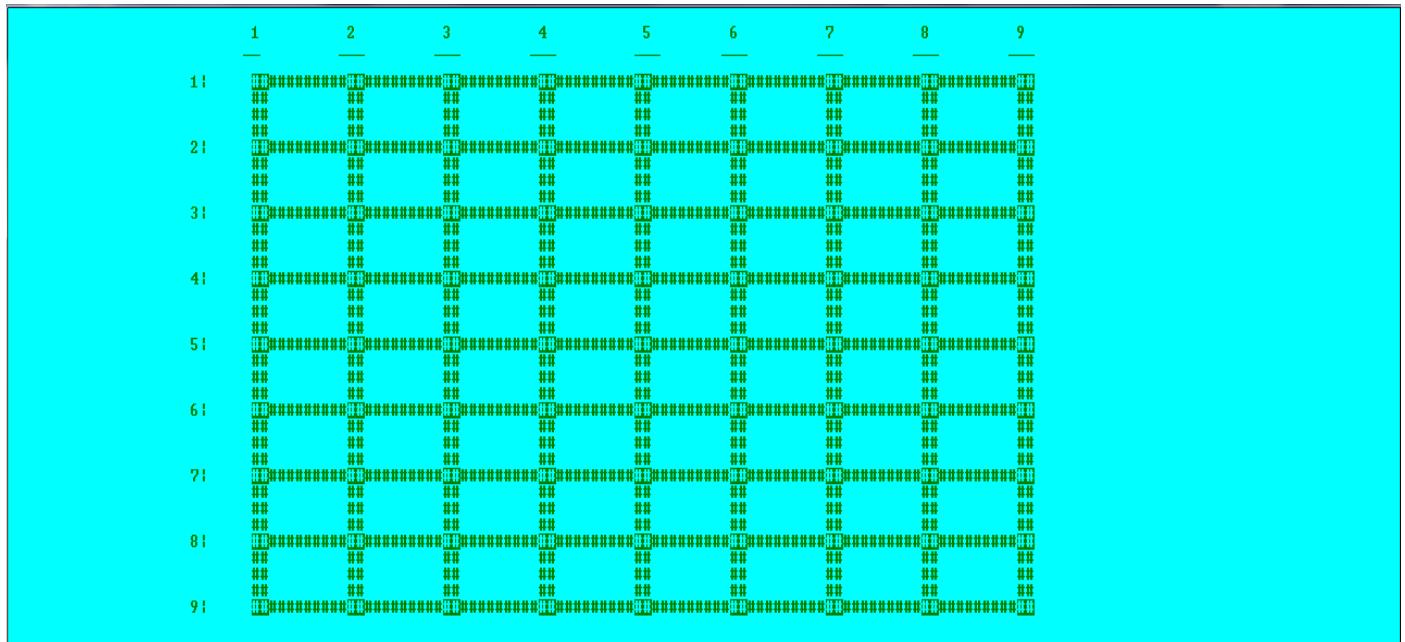
Et voici quelque texte et image en ascii art qu'on a utilisé :



■ Couleur :

On a utilisé une fonction qui permet de changer les couleurs « Color »

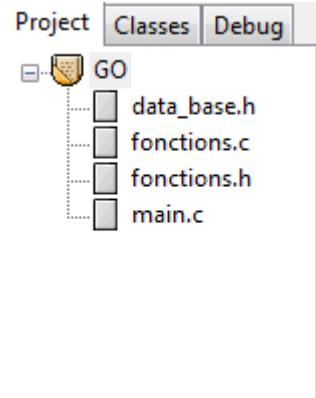
Qui prend en paramètre les nombres représentant le couleur du texte et celui de l’arrière-plan .



IV.DEVELOPEMENT DU JEU :

Dans cette partie on va voir le code source de notre jeu puis expliquer son fonctionnement .

Premièrement voici une vision globale sur notre code :



Le projet en général → → →

Les variables qu'on a utilisé tout au long de notre projet sont les suivantes :

```

main.c data_base.h fonctions.h fonctions.c
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<windows.h>
4 #include <time.h>
5
6 char J1[15];//joueur 1
7 char J2[15];// joueur 2
8 int rep_gagnant=0;// constante de fct gagnant
9 int rep_score=0;// constante de fct choix_score
10 int score_noir=0;//score du joueur noir
11 int score_blan=0;//score du joueur blanc
12 int chasse_user=0;// est ce que chasse est utilise ou non pour voir le coups autorise
13 int m=0; //qui va commencer l ordinateur ou le joueur (m=1 ou m=2) vient de rand
14 typedef struct {
15     int i;
16     int j;
17 }coordonn ;//structure coordonnees
18 coordonn noir[40][40];// les sous groupes de noir
19 coordonn blanc[40][40];//les sous groupes de blanc
20 int handicap=0;// pour traiter l handicap
21 int matrice[9][9]; //matrice principale
22 coordonn surfaces_goban[40][40]; // les surfaces occupees dans le goban (pour le territoire )
23 int domination=0; // constante qui donne qui domine dans un territoire (=1 noir =2 blanc =0 personne ou les deux )
24 int terre_noir=0; // les intersections du noir
25 int terre_blan=0;// les intersections du bland

```

Les fonctions qu'on a utilisées pour réussir notre jeu :

Dans la suite on va voir ces fonctions en détail et on va expliquer leurs fonctionnements et leur importance.

```
main.c data_base.h [*] fonctions.h fonctions.c
1 #include "data_base.h"
2 void Color(int couleurDuTexte,int couleurDeFond); // fonction d'affichage de couleurs
3 void interface1();
4 void interface2();
5 void interface3();
6 void blancnoir(int i , int j); // fct qui colorier les intersections
7 void interface04();
8 void JvsJ_initial(); // les noms et qui va commencer le premier
9 void JvsM_initial(); // le nom et qui va commencer est ce que la machine ou le joueur
10 int random(int min,int max); // un entier entre min et max
11 void interface4();
12 void rules();// les regles
13 void gagnant(); // qui a gagné la partie
14 void choix_score();// pour choisir quel score aimerez vous s'afficher
15 int degree_liberte(int i ,int j); // degree de liberte d'une boule
16 int nbr_voisin(int i,int j); // nombre de boules voisins de meme couleur
17 int degree_group( int k , coordonn p[40][40]); // calculer le degre de liberte d un groupe
18 int group_vide(int k , coordonn p[40][40]); // verifier est ce que le group est vide ou non
19 int quel_group(int i,int j ,coordonn p[40][40]); // chercher le groupe du point (i,j)
20 void ajouter_a_group(int i,int j,int k ,coordonn p[40][40]); // ajouter les coordonnee d un point donne a un group
21 void fusion_group(int i,int j,int k,coordonn p[40][40]); // fusionner les groupes pour avoir un seul group
22 void chasse(int k , coordonn p[40][40]); // fct de recuperation
23 void joueur_joueur(); // la fct de joueur vs joueur
24 void joueur_ordinateur(); // la fct de joueur vs machine mode facile
25 // pour les problemes du tsumego
26 void tsumego_initial(); // les regles our la partie des tsume go
27 void tsumego1(); // probleme 18
28 void tsumego2(); // probleme 17
29 void tsumego3(); // probleme 9
30 void tsumego4();
31 void tsumego5();
32 void tsumego6();
33 // pour calculer le territoire
34 void surface();// les surface divisé sur le goban
35 void dominer_par(int k); // donne qui est dominé dans une surface
36 void les_biens(); // conte les espace dominés par chaque joueur
37
```

Fichier principal:

```
main.c data_base.h [*] fonctions.h fonctions.c
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<windows.h>
4 #include <time.h>
5 #include "fonctions.c"
6 |
7 int main()
8 {
110
```

=>



Les interfaces :

1.Le GOBAN :

On a utilisé une matrice de taille 9*9 pour représenter la grille et voici de quoi elle ressemble dans notre code :

```

int main()
{
    int i,j;
    for(i=1;i<=9;i++)
    {
        for(j=1;j<=9;j++)
        {
            cout<<board[i][j];
        }
        cout<<endl;
    }
    return 0;
}

```

En zoomant sur une partie

```

int("          1   2   3   4   5   6   7   8   9   \t\t\t\t\t\t\n");
int("          —   —   —   —   —   —   —   —   —   \t\t\t\t\t\t\n");
int("          ");blancnoir(1 , 1);printf("#####");blancnoir(1 , 2);printf("#####");blancnoir(1 , 3);printf("#####");blancnoir(1 , 4);printf("#####");blancnoir(1 , 5);printf("#####");blancnoir(1 , 6);printf("#####");blancnoir(1 , 7);printf("#####");blancnoir(1 , 8);printf("#####");blancnoir(1 , 9);printf("#####");\n";
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ");blancnoir(2 , 1);printf("#####");blancnoir(2 , 2);printf("#####");blancnoir(2 , 3);printf("#####");blancnoir(2 , 4);printf("#####");blancnoir(2 , 5);printf("#####");blancnoir(2 , 6);printf("#####");blancnoir(2 , 7);printf("#####");blancnoir(2 , 8);printf("#####");blancnoir(2 , 9);printf("#####");\n";
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ");blancnoir(3 , 1);printf("#####");blancnoir(3 , 2);printf("#####");blancnoir(3 , 3);printf("#####");blancnoir(3 , 4);printf("#####");blancnoir(3 , 5);printf("#####");blancnoir(3 , 6);printf("#####");blancnoir(3 , 7);printf("#####");blancnoir(3 , 8);printf("#####");blancnoir(3 , 9);printf("#####");\n";
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ");blancnoir(4 , 1);printf("#####");blancnoir(4 , 2);printf("#####");blancnoir(4 , 3);printf("#####");blancnoir(4 , 4);printf("#####");blancnoir(4 , 5);printf("#####");blancnoir(4 , 6);printf("#####");blancnoir(4 , 7);printf("#####");blancnoir(4 , 8);printf("#####");blancnoir(4 , 9);printf("#####");\n";
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ");blancnoir(5 , 1);printf("#####");blancnoir(5 , 2);printf("#####");blancnoir(5 , 3);printf("#####");blancnoir(5 , 4);printf("#####");blancnoir(5 , 5);printf("#####");blancnoir(5 , 6);printf("#####");blancnoir(5 , 7);printf("#####");blancnoir(5 , 8);printf("#####");blancnoir(5 , 9);printf("#####");\n";
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ");blancnoir(6 , 1);printf("#####");blancnoir(6 , 2);printf("#####");blancnoir(6 , 3);printf("#####");blancnoir(6 , 4);printf("#####");blancnoir(6 , 5);printf("#####");blancnoir(6 , 6);printf("#####");blancnoir(6 , 7);printf("#####");blancnoir(6 , 8);printf("#####");blancnoir(6 , 9);printf("#####");\n";
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ");blancnoir(7 , 1);printf("#####");blancnoir(7 , 2);printf("#####");blancnoir(7 , 3);printf("#####");blancnoir(7 , 4);printf("#####");blancnoir(7 , 5);printf("#####");blancnoir(7 , 6);printf("#####");blancnoir(7 , 7);printf("#####");blancnoir(7 , 8);printf("#####");blancnoir(7 , 9);printf("#####");\n";
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ");blancnoir(8 , 1);printf("#####");blancnoir(8 , 2);printf("#####");blancnoir(8 , 3);printf("#####");blancnoir(8 , 4);printf("#####");blancnoir(8 , 5);printf("#####");blancnoir(8 , 6);printf("#####");blancnoir(8 , 7);printf("#####");blancnoir(8 , 8);printf("#####");blancnoir(8 , 9);printf("#####");\n";
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ");blancnoir(9 , 1);printf("#####");blancnoir(9 , 2);printf("#####");blancnoir(9 , 3);printf("#####");blancnoir(9 , 4);printf("#####");blancnoir(9 , 5);printf("#####");blancnoir(9 , 6);printf("#####");blancnoir(9 , 7);printf("#####");blancnoir(9 , 8);printf("#####");blancnoir(9 , 9);printf("#####");\n";
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
int("          ##  ##  ##  ##  ##  ##  ##  ##  ##  \t\t\t\t\t\t\n");
    lor(15,0);
}

```

La fonction << blancnoir >> permet de placer une couleur noir ou blanc sur la grille ou bien laisser une intersection vide (c'est à dire ‘ ‘##’ ’ en vert) ;


```

void rules()
{
    interfaced();
    Color(15,0);
    printf("\n\n\n >>>>>> LE GOBAN : le plateau s'appelle un goban sa taille officielle est de 19*19 mais on peut avoir \n t\|t\|t\|t\|t un goban 13*13 et 9*9 .");
    puts("\n\n\n\nAppuyer sur Enterer pour continuer ");
    getchar();
    while (getchar() != '\n');
    system("cls");
    matrice[1][5]=1;
    matrice[1][6]=2;
    interfaced();
    Color(15,0);
    printf("\n\n\n >>>>>> LES JOUEURS : le GO se joue a deux et chaque joueur est associe a un couleur soit blanc ou noir .");
    printf("\n\n\n >>>>>> JOUER : les joueur peuvent poser un pion de leur couleur sur une intersection inoccupee en choisissant la ligne et la colonne .");
    puts("\n\n\n\nAppuyer sur Enterer pour continuer ");
    while (getchar() != '\n');
    system("cls");
    matrice[2][5]=2;
    matrice[2][6]=1;
    interfaced();
    Color(15,0);
    printf("\n\n\n >>>>>> UNE CHAINE : deux pierres voisines et une chaîne est un ensemble de pierres voisines de proches en proches .");
    printf("\n\n\n >>>>>> CAPTURE : si un joueur supprime la liberte d une chaîne de l'autre joueur il la capture .");
    puts("\n\n\n\nAppuyer sur Enterer pour continuer ");
    while (getchar() != '\n');
    system("cls");
    matrice[2][5]=0;
    matrice[2][6]=0;
    matrice[1][4]=2;
    interfaced();
    Color(15,0);
    printf("\n\n\n >>>>>> UNE TERRITOIRE : un ensemble d intersections voisines vides .");
    printf("\n\n\n >>>>>> LES POINTS : les points d un joueur sont la sommes de nombre des territoires et les pierres capturees .");
    puts("\n\n\n\nAppuyer sur Enterer pour continuer ");
    while (getchar() != '\n');
    matrice[1][5]=0;
    matrice[1][6]=0;
    matrice[1][4]=0;
    matrice[1][0]=0;
    matrice[1][3]=0;
    system("cls");
}

```

❖ LA RECUPERATION :

En gros voici les sous-fonctions et les variables qui nous ont aidées à construire une fonction de récupération des pierres qui a pour nom <<< chasse >>> :

```

typedef struct {
    int i;
    int j;
}coordonn ;//structure coordonnees
coordonn noir[40][40];// les sous groupes de noir
coordonn blanc[40][40];//les sous groupes de blanc

```

```

// les fonction pour la recuperation

int degree_liberte(int i ,int j) // degree de liberte d'une boule
{
int nbr_voisin(int i,int j) // nombre de boules voisins de meme couleur
{
int degree_group( int k, coordonn p[40][40]) // calculer le degree de liberte d un groupe
{
int group_vide(int k ,coordonn p[40][40]) // verifier est ce que le group est vide ou non
{
int quel_group(int i,int j ,coordonn p[40][40]) // chercher le groupe du point (i,j)
{
void ajouter_a_group(int i,int j,int k ,coordonn p[40][40]) // ajouter les coordonnee d un point donne a un group
{
void fusion_group(int i,int j,int k,coordonn p[40][40]) // fusioner les groupes pour avoir un seul group
{
void chasse(int k , coordonn p[40][40]) // fct de recuperation
{

```

- On a défini une structure qui contient deux éléments de type entier qui vont représenter les coordonnées des intersections.
- Le groupe pour nous est un ensemble de pierres enchaînées de même couleur.
- Le degré de liberté d'une pierres pour nous est le nombre des cases voisines vide de cette pierres.
- Chacun des deux joueurs possède une matrice dont chaque ligne contient les coordonnées des pierres d'un groupe ;
- La fonction << degree_liberte >> donne le degré de liberté d'une pierre donnée.
- La fonction << nbr_voisin >> donne le nombre de pierres voisines de même couleur qui occupe un degré de liberté d'une pierre donnée.
- La fonction << quel_group >> donne le groupe auquel appartient une pierre, et la fonction <<degree_group >> donne le degré de liberté d'un group qui est égale a la somme des degré de liberté de ses membres.
- Si le degré d'un group est nul ce groupe va être capturé par l'adversaire à l'aide de la fonction <<chasse>>.

- La fonction <<chasse>> ajoute aussi au score du joueur le nombre de prisonniers.

```

void chasse(int k , coordonn p[40][40]) // fct de recuperation
{
    if( degree_group(k,p)==0)
    {
        for(int r=1;r<40;r++)
            if(p[k][r].i!=0 && p[k][r].j!=0)
            {
                if(matrice[(p[k][r]).i][(p[k][r]).j]==1)
                {
                    score_blan=score_blan+1;
                }
                else
                {
                    score_noir=score_noir+1;
                }
                matrice[(p[k][r]).i][(p[k][r]).j]=0;
                p[k][r].i=0;
                p[k][r].j=0;
                chasse_user=1;
            }
    }
}

```

Remarque :

<< Chasse_user >> est une variable utilisée pour savoir est ce que chasse a bien récupéré les pierres ; cela nous a aidé dans le traitement du KO.

❖ JOUER :

1. Joueur vs Joueur :

La fonction qui permet à deux êtres humains de jouer entre eux s'appelle comme il a été demandé dans le cahier de charge-
 <<< JOUEUR_JOUEUR >>> qui est représentée dans notre code comme ceci :

```

void joueur_joueur() // la fct de joueur vs joueur
{
    int ko=1;// pour savoir est ce qu'on est dans un KO
    int ko_noir=1;// pour connaître qui a commencer dans le ko
    int ko_blan=1;
    int pass1=1;
    int pass2=1;
    while (pass1!=0 & pass2!=0)
    {
        if(handicap==0)
        {
            interface4();
            int i=0;
            int j=0;
            Color(15,0);
            printf("\n\n <<<choisir une intersection ou taper *** 0 et 0 *** pour passer >>> \n");
            L:
            printf("\n\n le NOIR Quel est votre choix ? >>> \n");
            printf("ligne :");
            scanf("%d",&i);
            printf("colonne :");
            scanf("%d",&j);
            if(matrice[i][j]!=0)
            {
                printf(" deja remplit choisir une autre place \n");
                goto L;
            }
            if((j<0||j>9)|| (i<0||i>9))
            {
                printf("les parametres doivent etre entre 1 et 9\n");
                goto L;
            }
        }
    }
}
  
```

Dans cette fonction on a traité :

- ✚ L'handicape calculé auparavant par la fonction << JvsJ_initiale >> à partir des niveaux saisis par les deux joueurs.
- ✚ si l'intersection choisie par un joueur n'est pas valide le programme refuse son choix et il lui demande de ressayer.
- ✚ Si les coordonnées entrées ne sont pas valides le programme demande encore une fois de ressayer.

```

if (i==0 || j==0)
{
    pass1=0;
    if(ko_blancl==0)
        { ko=ko+1;
          ko_blancl=ko_blancl+1;
        }
    system("cls");
}
else
{
    matrice[i][j]=1;
    if(degree_liberte(i,j)==0 && nbr_voisin(i,j)==0)// pour voir est ce que le coups est autorise
    {
        if(ko!=0)
        {
            chasse_user=0;
            for(int k=1;k<40;k++)
                chasse(k,blanc);
            if(chasse_user==0) // chasse n a pas ete utiliser
            {
                printf("\n***** coups interdit *****");
                matrice[i][j]=0;
                goto L;
            }
            else
            {
                ko=ko-1;
                ko_noir=ko_noir-1;
            }
        }
    }
}

```

Cette fonction permet à un joueur de jouer son choix si et seulement si son choix respecte certains conditions :

- Le KO : la fonction sait est ce qu'il y a un ko à partir d'une variable qui s'appelle <<ko>>
- Dans le cas où il y a un ko, le joueur ne peut jouer dans cette intersection qu'après un coup intermédiaire.
- Si le degré de liberté d'une intersection est nul le joueur ne peut pas le jouer que si il a la possibilité de récupérer des pierres de l'adversaire.

```
        }
        else
        {
            printf("\n***** <KO> coups interdit il faut jouer un coup intermidiere *****");
            matrice[i][j]=0;
            goto L;
        }
    }
    if(ko_blanck==0)
    {
        ko=ko+1;
        ko_blanck=ko_blanck+1;
    }
    int k =quel_group(i,j,noir);
    ajouter_a_group(i,j,k,noir);
    fusion_group(i, j, k,noir);
    for(int k=1;k<40;k++)
        chasse(k,blanc);
    for(int k=1;k<40;k++)
        chasse(k,noir);
    system("cls");
}
interface4();
Color(15,0);
R:
printf("\n\n <<<choisir une intersection ou taper *** 0 et 0 *** pour passer >>>\n");
printf("\n\n le BLANC Quel est votre choix ? >>>\n");
printf("ligne :");
scanf("%d",&i);
printf("colonne :");
scanf("%d",&j);
if(matrice[i][j]!=0)
{
    printf(" deja remplit choisir un autre place \n");
    goto R;
}
if(( j<0||j>9 )||(i<0 || i>9))
{printf("les parametres doivent etre entre 1 et 9\n");
goto R;
}
if(i==0 || j==0)
{pass2=0;
if(ko_noir==0)
{ ko=ko+1;
ko_noir=ko_noir+1;
}
system("cls");}
else
{ matrice[i][j]=2;
if(degree_liberte(i,j)==0 && nbr_voisin(i,j)==0)
{
    if(ko!=0)
    {
        chasse_user=0;
        for(int k=1;k<40;k++)
            chasse(k,noir);
        if(chasse_user==0) // chasse n a pas ete utiliser
        {
            printf("\n***** coups interdit *****");
            matrice[i][j]=0;
            goto R;
        }
    }
}
```

```

        else
        {
            ko=ko-1;
            ko_blanco=ko_blanco-1;
        }
    else
    {
        printf("\n***** <KO> coups interdit il faut jouer un coup intermidiere *****");
        matrice[i][j]=0;
        goto R;
    }

}
if(ko_noir==0)
{
    ko=ko+1;
    ko_noir=ko_noir+1;
}
int k=quel_group(i,j,blanc);
ajouter_a_group(i,j,k,blanc);
fusion_group(i, j, k,blanc);
for(int k=1;k<40;k++)
    chasse(k,noir);
for(int k=1;k<40;k++)
    chasse(k,blanc);
system("cls");
}
if (pass1!=0 || pass2!=0)
{
    pass1=1; pass2=1;
}
}

```

Si l'handicap est non nul le joueur blanc a comme avantage de poser un nombre des pierres égales au handicap .

Cela est traité dans la partie qui suit ;

```

        else
    {
handicap=handicap-1;
int i=0;
int j=0;
interface4();
Color(15,0);
H:
printf("\n\n <<< choisir une intersection [ %d pierre de handicap ] >>> \n",handicap+1);
printf("\n\n le BLANC Quel est votre choix ? >>>\n");
printf("ligne :");
scanf("%d",&i);
printf("colonne :");
scanf("%d",&j);
if(matrice[i][j]!=0)
{
    printf(" deja remplit choisir un autre place \n");
    goto H;
}
if(( j<0||j>9 )||(i<0 || i>9))
    {printf("les parametres doivent etre entre 1 et 9\n");
    goto H;
}
if(i==0 || j==0)
    {printf(" il faut jouer !! \n");
    goto H;}
}

```

```

        if(i==0 || j==0)
            {printf(" il faut jouer !! \n");
             goto H;}
        else
        {  matrice[i][j]=2;
           int k=quel_group(i,j,blanc);
           ajouter_a_group(i,j,k,blanc);
           fusion_group(i, j, k,blanc);
           system("cls"); }

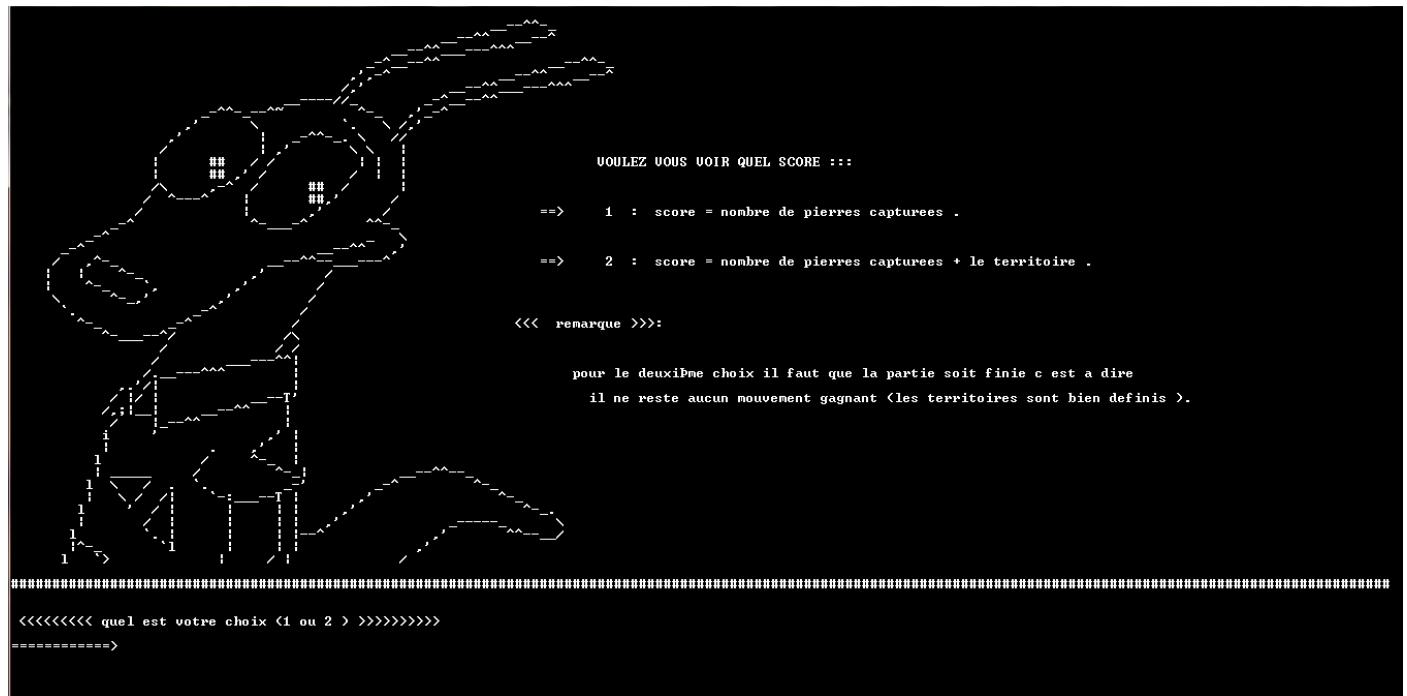
    }

choix_score();
if(rep_score ==2)
{
    surface();
    les_biens();
    score_noir=score_noir + terre_noir;
    score_blan=score_blan + terre_blan;
}
system("cls");
gagnant();

}

```

Si pass1 =0 et pass2=0 alors les deux joueurs passent ; dans ce cas la fonction <>choix_score>> se lance, c'est elle qui va demander au joueurs la nature de score qu'ils veulent voir ;



Puis la fonction <> gagnant>> va comparer les scores et annonce le joueur gagnant ;

2. Joueur vs machine mode aléatoire :

Dans cette partie au lieu d'avoir deux joueurs humains on a un joueur humain et la machine qui joue d'une manière aléatoire ;

- ✓ La fonction qui assure le choix de la machine s'appelle
 <<random>>

```
int random(int min,int max) // un entier entre min et max
{
    return (rand()% (max-min+1))+min;
}
```

Pour avoir des valeurs différentes on a utilisé **srand**

```
srand ( time(NULL) );// initialisation de temps
```

La fonction qui permet à un être humain de jouer contre la machine s'appelle-comme il a été demandé dans le cahier de charge -<<<
JOUEUR_ORDINNATEUR>>>.

Remarque :

La seule différence entre <<< JOUEUR_ORDINNATEUR >>> et <<< JOUEUR_JOUEUR >>> c'est que un des deux joueurs est remplacé par la machine qui choisit aléatoirement une intersection, donc les même cas précédents sont présents ici.

Voici le code :

```

void joueur_ordinateur() // la fct de joueur vs machine mode facile
{
    int ko=1; // pour savoir est ce qu'on est dans un KO
    int ko_noir=1; // pour connaitre qui a commencer dans le ko
    int ko_blanck=1;
    int pass1=1;
    int pass2=1;
    while (pass1!=0 & pass2!=0)
    {
        if(m==1)
        {
            interface4();
            int i=0;
            int j=0;
            Color(15,0);
            printf("\n\n <<<choisir une intersection ou taper *** 0 et 0 *** pour passer >>> \n");
            A:
            printf("\n\n le NOIR Quel est votre choix ? >>> \n");
            printf("ligne :");
            scanf("%d",&i);
            printf("colonne :");
            scanf("%d",&j);
            if(matrice[i][j]!=0)
            {
                printf(" deja remplit choisir une autre place \n");
                goto A;
            }
            if((j<0||j>9)|| (i<0||i>9))
            {
                printf("les parametres doivent etre entre 1 et 9\n");
                goto A;
            }
            if (i==0 || j==0)
            {
                pass1=0;
                if(ko_blanck==0)
                {
                    ko=ko+1;
                    ko_blanck=ko_blanck+1;
                }
                system("cls");
            }
            else
            {
                matrice[i][j]=1;
                if(degree_liberte(i,j)==0 && nbr_voisin(i,j)==0)// pour voir est ce que le coups est autorise
                {
                    if(ko!=0)
                    {
                        chasse_user=0;
                        for(int k=1;k<40;k++)
                            chasse(k,blanc);
                        if(chasse_user==0) // chasse n'a pas ete utiliser
                        {
                            printf("\n***** coups interdit *****");
                            matrice[i][j]=0;
                            goto A;
                        }
                        else
                        {
                            ko=ko-1;
                            ko_noir=ko_noir-1;
                        }
                    }
                }
            }
        }
    }
}

```

```

        else
        {
            ko=ko-1;
            ko_noir=ko_noir-1;
        }
    }
else
{
    printf("\n***** <KO> coups interdit il faut jouer un coup intermidiere *****");
    matrice[i][j]=0;
    goto A;
}

}
if(ko_blancl==0)
{
    ko=ko+1;
    ko_blancl=ko_blancl+1;
}
int k =quel_group(i,j,noir);
ajouter_a_group(i,j,k,noir);
fusion_group(i, j, k,noir);
for(int k=1;k<40;k++)
    chasse(k,blanc);
for(int k=1;k<40;k++)
    chasse(k,noir);
system("cls");
}

B:
i=random(0,9);
j=random(0,9);
if(matrice[i][j]!=0)
{
    goto B;
}

if(i==0 || j==0)
{
    pass2=0;
    if(ko_noir==0)
    {
        ko=ko+1;
        ko_noir=ko_noir+1;
    }
}
else
{
    matrice[i][j]=2;
    if(degree_liberte(i,j)==0 && nbr_voisin(i,j)==0)
    {
        if(ko!=0)
        {
            chasse_user=0;
            for(int k=1;k<40;k++)
                chasse(k,noir);
            if(chasse_user==0) // chasse n a pas ete utiliser
            {
                matrice[i][j]=0;
                goto B;
            }
            else
            {
                ko=ko-1;
                ko_blancl=ko_blancl-1;
            }
        }
        else
        {
            matrice[i][j]=0;
            goto B;
        }
    }
}
}

```

```

        if(ko_noir==0)
        { ko=ko+1;
          ko_noir=ko_noir+1;
        }
        int k=quel_group(i,j,blanc);
        ajouter_a_group(i,j,k,blanc);
        fusion_group(i, j, k,blanc);
        for(int k=1;k<40;k++)
            chasse(k,noir);
        for(int k=1;k<40;k++)
            chasse(k,blanc);
        system("cls");
    }
}
if (pass1!=0 || pass2!=0)
{
    pass1=1; pass2=1;
}

}

if(m==2)
{
int i=0;
int j=0;
C:
i=random(0,9);
j=random(0,9);
if(matrice[i][j]!=0)
{
    goto C;
}
if(i==0 || j==0)
{pass2=0;
if(ko_blanco==0)
{ ko=ko+1;
  ko_blanco=ko_blanco+1;
}
}
else
{
matrice[i][j]=1;
if(degree_liberte(i,j)==0 && nbr_voisin(i,j)==0)// pour voir est ce que le coups est autorise
{
    if(ko!=0)
    {
        chasse_user=0;
        for(int k=1;k<40;k++)
            chasse(k,blanc);
        if(chasse_user==0) // chasse n a pas ete utiliser
        {
            matrice[i][j]=0;
            goto C;
        }
        else
        {
            ko=ko-1;
            ko_noir=ko_noir-1;
        }
    }
    else
    {
        matrice[i][j]=0;
        goto C;
    }
}
}
}

```

```

        }
        if(ko_blancl==0)
        {
            ko=ko+1;
            ko_blancl=ko_blancl+1;
        }
        int k =quel_group(i,j,noir);
        ajouter_a_group(i,j,k,noir);
        fusion_group(i, j, k,noir);
        for(int k=1;k<40;k++)
            chasse(k,blanc);
        for(int k=1;k<40;k++)
            chasse(k,noir);
        system("cls");
    }

interface4();
Color(15,0);
printf("\n\n <<<choisir une intersection ou taper *** 0 et 0 *** pour passer >>> \n");
D:
printf("\n\n le BLANC Quel est votre choix ? >>> \n");
printf("ligne :");
scanf("%d",&i);
printf("colonne :");
scanf("%d",&j);
if(matrice[i][j]!=0)
{
    printf(" deja remplit choisir une autre place \n");
    goto D;
}
if((j<0||j>9)|| (i<0||i>9))
{
    printf("les parametres doivent etre entre 1 et 9\n");
    goto D;
}
if (i==0 || j==0)
{
    pass1=0;
    if(ko_noir==0)
    {
        ko=ko+1;
        ko_noir=ko_noir+1;
    }
    system("cls");
}
else
{
    matrice[i][j]=2;
    if(degree_liberte(i,j)==0 && nbr_voisin(i,j)==0)
    {
        if(ko!=0)
        {
            chasse_user=0;
            for(int k=1;k<40;k++)
                chasse(k,noir);
            if(chasse_user==0) // chasse n a pas ete utilise
            {
                printf("\n***** coups interdit *****");
                matrice[i][j]=0;
                goto D;
            }
            else
            {
                ko=ko-1;
                ko_blancl=ko_blancl-1;
            }
        }
    }
}

```

```

        }
    else
    {
        printf("\n***** <KO> coups interdit il faut jouer un coup intermidiere *****");
        matrice[i][j]=0;
        goto D;
    }

}

if(ko_noir==0)
{
    ko=ko+1;
    ko_noir=ko_noir+1;
}
int k=quel_group(i,j,blanc);
ajouter_a_group(i,j,k,blanc);
fusion_group(i, j, k,blanc);
for(int k=1;k<40;k++)
    chasse(k,noir);
for(int k=1;k<40;k++)
    chasse(k,blanc);
system("cls");
}
if (pass1!=0 || pass2!=0)
{
    pass1=1; pass2=1;
}
}

}

choix_score();
if(rep_score ==2)
{
    surface();
    les_biens();
    score_noir=score_noir + terre_noir;
    score_blan=score_blan + terre_blan;
}
system("cls");
gagnant();
}
}

```

3. JOUEUR vs MACHINE :TSUME GO :

Comme on a déclaré précédemment, les TSUME GO représentent une partie indispensable dans notre travail où on a essayé d'implémenter une simple intelligence en s'inspirant de la résolution de six problèmes de TSUME GO, qui sont codés comme ci-dessous :

```

void tsumego1() // probleme 18
{
    matrice[7][5]=2;
    matrice[7][6]=2;
    matrice[8][4]=2;
    matrice[8][5]=2;
    matrice[8][7]=2;
    matrice[8][8]=2;
    matrice[8][3]=1;
    matrice[8][9]=1;
    matrice[7][2]=1;
    matrice[7][4]=1;
    matrice[7][7]=1;
    matrice[7][8]=1;
    matrice[7][9]=1;
    matrice[6][4]=1;
    matrice[6][5]=1;
    matrice[6][6]=1;
}

```

```

void tsumego2() // probleme 17
{
    matrice[1][1]=1;
    matrice[2][1]=1;
    matrice[3][1]=1;
    matrice[4][1]=1;
    matrice[4][2]=1;
    matrice[5][3]=1;
    matrice[5][4]=1;
    matrice[5][5]=1;
    matrice[5][7]=1;
    matrice[1][5]=1;
    matrice[1][6]=1;
    matrice[1][7]=1;
    matrice[2][7]=1;
    matrice[3][5]=1;
    matrice[3][6]=1;
    matrice[1][2]=2;
    matrice[2][2]=2;
    matrice[3][2]=2;
    matrice[4][3]=2;
    matrice[1][4]=2;
    matrice[3][4]=2;
    matrice[4][4]=2;
    matrice[2][5]=2;
    matrice[2][6]=2;
    matrice[2][3]=2;
}

```

```

void tsumego3() // probleme 9
{
    matrice[5][2]=2;
    matrice[6][2]=2;
    matrice[7][2]=2;
    matrice[4][3]=2;
    matrice[8][3]=2;
    matrice[2][4]=2;
    matrice[3][4]=2;
    matrice[4][4]=2;
    matrice[8][4]=2;
    matrice[1][5]=2;
    matrice[7][5]=2;
    matrice[1][6]=2;
    matrice[5][6]=2;
    matrice[7][6]=2;
    matrice[2][7]=2;
    matrice[6][7]=2;
    matrice[2][8]=2;
    matrice[3][8]=2;
    matrice[4][8]=2;
    matrice[5][8]=2;
    matrice[7][8]=2;
    matrice[5][3]=1;
    matrice[6][3]=1;
    matrice[5][4]=1;
    matrice[7][4]=1;
    matrice[2][5]=1;
    matrice[3][5]=1;
    matrice[4][5]=1;
    matrice[5][5]=1;
    matrice[6][5]=1;
    matrice[2][6]=1;
    matrice[3][7]=1;
    matrice[4][7]=1;
    matrice[5][7]=1;
}

```

```

void tsumego4()
{
    matrice[1][9]=2;
    matrice[1][8]=2;
    matrice[1][7]=2;
    matrice[2][7]=2;
    matrice[3][7]=2;
    matrice[4][7]=2;
    matrice[5][7]=2;
    matrice[6][7]=2;
    matrice[7][7]=2;
    matrice[7][8]=2;
    matrice[7][9]=2;
    matrice[2][9]=1;
    matrice[2][8]=1;
    matrice[3][8]=1;
    matrice[4][8]=1;
    matrice[5][8]=1;
    matrice[6][8]=1;
    matrice[6][9]=1;
}

void tsumego5()
{
    matrice[1][7]=1;
    matrice[2][7]=1;
    matrice[2][8]=1;
    matrice[3][8]=1;
    matrice[3][9]=1;
    matrice[1][6]=2;
    matrice[2][6]=2;
    matrice[3][5]=2;
    matrice[3][7]=2;
    matrice[4][7]=2;
    matrice[4][8]=2;
    matrice[4][9]=2;
}

```

```

void tsumego6()
{
    matrice[2][7]=2;
    matrice[1][8]=2;
    matrice[2][9]=2;
    matrice[3][8]=2;
    matrice[3][9]=2;
    matrice[1][6]=1;
    matrice[2][6]=1;
    matrice[3][5]=1;
    matrice[3][7]=1;
    matrice[4][7]=1;
    matrice[4][8]=1;
    matrice[4][9]=1;
}

```

On a fait apprendre la solution à l'ordinateur sous forme des cas possibles dans chaque problème, à l'aide de la fonction

<<< Joueur_ordinateur_tsumego >>> .

Et voilà des captures d'écran qui illustrent le codage des solutions des différents problèmes :

```

void joueur_ordinnateur_tsumego() // la fct de joueur vs machine mode tsumego
{
    int t=1;// pour tsumego2
    int r=1; // pour tsumego3
    int z=3; // pour tsumego4
    int rep=0;// passer ou ressayer
    int nbr_tsume=6; // nombre des tsumegos
    while (nbr_tsume!=0)
    {
        system("cls");
        if(nbr_tsume==6)
        {
            tsumego1();
        }
        if(nbr_tsume==5)
        {
            tsumego2();
            if(matrice[2][4]==1)
            {
                matrice[2][5]=0;
                matrice[2][6]=0;
            }
            if(matrice[2][4]==1 && matrice[2][5]==0)
            {
                matrice[2][5]=2;
                matrice[2][4]=0;
            }
        }
    }
}

```

```

    if(nbr_tsume==4)
    {
        tsumego3();
    }
    if(nbr_tsume==3)
    {
        tsumego4();
    }
    if(nbr_tsume==2)
    {
        tsumego5();
    }
    if(nbr_tsume==1)
    {
        tsumego6();
    }
    interface4();
    int i=0;
    int j=0;
    Color(15,0);
    printf("\n\n <<<choisir une intersection >>> \n");
    A:
    printf("\n\n le NOIR Quel est votre choix ? >>>> \n");
    printf("ligne :");
    scanf("%d",&i);
    printf("colonne :");
    scanf("%d",&j);
    if(matrice[i][j]!=0)
    {
        printf(" déjà remplit choisir une autre place \n");
        goto A;
    }
    if((j<=0 || j>9) || (i<=0 || i>9))
    {
        printf("les parametres doivent etre entre 1 et 9\n");
        goto A;
    }
    if(i==2 && j==4 && nbr_tsume==5 && t==1)
    {
        matrice[2][4]=1;
    }
    else
    {
        matrice[i][j]=1;
        if(degree_liberte(i,j)==0 && nbr_voisin(i,j)==0)// pour voir est ce que le coups est autorise
        {
            printf("\n***** coups interdit *****");
            matrice[i][j]=0;
            goto A;
        }
        system("cls");
    }
    if(nbr_tsume==1)
    {
        if(nbr_tsume==2)
        {
            if(nbr_tsume==3)
            {
                if(nbr_tsume==4)
                {
                    if(nbr_tsume==5)
                    {
                        if(nbr_tsume==6)
                        {
                        }
                    }
                }
            }
        }
    }
    while(rep_gagnant!=1 & rep_gagnant!=2)
    {
        printf("\n\n\n <<<<< Page principale :: appuyer sur 1 >>>>>> \n\n");
        printf("<<<<< Quitter :: appuyer sur 2 >>>>>>\n\n");
        printf("===== ");
        scanf("%d",&rep_gagnant);
    }
}

```

Dans les (if) qui traitent les TSUME GO la machine connaît le meilleure coups pour gagner le territoire et est-ce-qu'elle a la possibilité de gagner ou non suivant le choix de joueur ;

Et voilà un exemple pour (le problème numéro 6) :

```

if(nbr_tsume==6)
{
    if(matrice[9][6]==2)
    {
        if(matrice[9][8]==0)
        {
            matrice[9][8]=2;
            if(matrice[9][7]==1)
                matrice[9][7]=0;
        }
        else
        {
            matrice[9][4]=2;
            if(matrice[9][5]==1)
                matrice[9][5]=0;
        }
        system("cls");
        interface4();
        printf("\n\n >>>> le blanc a gagne ce territoire \n");
        printf("\n ***** entrer 1 pour ressayer et 2 pour passer au suivant ***** ");
        scanf("%d",&rep);
        if (rep==2)
            nbr_tsume=nbr_tsume-1;
        for(int i=1;i<=9;i++)
            for(int j=1;j<=9;j++)
                matrice[i][j]=0;
    }
}

if(matrice[9][6]==0 && matrice[7][6]!=0)
{
    matrice[9][6]=2;
    if(matrice[8][6]==1)
    {
        matrice[8][6]=0;
    }
}
if(matrice[9][6]==1)
{
    system("cls");
    interface4();

    printf("\n\n >>>> bon choix le blanc a perdu ce territoire \n");
    printf("\n***** entrer 1 pour ressayer et 2 pour passer au suivant *****");
    scanf("%d",&rep);
    if (rep==2)
        nbr_tsume=nbr_tsume-1;
    for(int i=1;i<=9;i++)
        for(int j=1;j<=9;j++)
            matrice[i][j]=0;
}
}

```

Pour les cas qui reste c'est de la même manière, sauf que les conditions changent.

❖ Calcule du territoire :

Voici les variables et les fonctions qui nous ont aidées à calculer le territoire de chaque joueur :

```
coordonn surfaces_goban[40][40]; // les surfaces occupées dans le goban (pour le territoire )
int domination=0; // constante qui donne qui domine dans un territoire (=1 noir =2 blanc =0 personne ou les deux )
int terre_noir=0; // les intersections du noir
int terre_blan=0;// les intersections du blanc
```

```
// pour calculer le territoire

void surface()// les surfaces divisé sur le goban
{
    void dominer_par(int k) // donne qui est dominé dans une surface
}
void les_biens() // conte les espaces dominés par chaque joueur
{
```

La matrice de type **coordonn** sert à contenir les groupes des intersections et parmi ses groupes on peut choisir celle qui représente pour nous un territoire ; c'est-à-dire un groupe d'intersection vide délimiter par un joueur, ce dernier entoure ce groupe et domine l'espace des intersections de ce groupe sur le goban.

Pour savoir qui domine dans un groupe on a utilisé la fonction **<<< dominer_par >>>** qui reçoit comme paramètre le groupe puis elle donne quel joueur domine dans ce groupe ; puis on calcule le nombre des éléments de ce groupe et on les ajoute à la variable ‘ **terre_noir** ou **terre_blan**’ qui contient le nombre total des intersections dans lesquelles ce joueur domine.

Voici la fonction **<<<les_biens>>>** qui attribue à chaque joueur son territoire :

```

void les_biens() // conte les espace dominés par chaque joueur
{
    terre_noir=0;
    terre_blancl=0;
    for(int k=1;k<40;k++)
    {
        domination=0;
        dominer_par(k);
        if(domination==2)
        {
            for(int r=1;r<40;r++)
                if(surfaces_goban[k][r].i!=0 && surfaces_goban[k][r].j!=0)
                    terre_blancl=terre_blancl+1;
        }
        if(domination==1)
        {
            for(int r=1;r<40;r++)
                if(surfaces_goban[k][r].i!=0 && surfaces_goban[k][r].j!=0 )
                    terre_noir=terre_noir+1;
        }
    }
}

```

❖ MAIN :

Le fichier main doit contrôler et appeler les différentes fonctions complexes et assurer la liaison entre toutes les entités déjà traitées.

```

main.c
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<windows.h>
4 #include <time.h>
5 #include "fonctions.c"
6
7 int main()
8 {
9     N:
10    srand ( time(NULL) );// initialisation de temps
11    for(int i=1;i<40;i++)
12        for(int j=1;j<40;j++)
13        {
14            noir[i][j].i=0;
15            noir[i][j].j=0;
16            blanc[i][j].i=0;
17            blanc[i][j].j=0;
18        }
19    for(int i=1;i<=9;i++)
20        for(int j=1;j<=9;j++)
21            matrice[i][j]=0; // remplir la matrice principale par des 0
22    int rep=0;
23    g:
24    interface1();
25    scanf("%d",&rep);
26    system("cls");
27    if(rep==1)
28    {
29        l:
30        interface2();
31        scanf("%d",&rep);
32        system("cls");
33        if(rep==2)
34        {
35            interface3();
36            scanf("%d",&rep);
37            if(rep==1)
38            {
39                system("cls");

```

```
40         interface04();
41         JvsM_initial();
42         joueur_ordinnateur();
43         if(rep_gagnant==1)
44         {
45             rep_gagnant=0;
46             system("cls");
47             goto N ;
48         }
49
50         else
51             exit(1);
52         rep=0;
53     }
54     if(rep==2)
55     {
56         system("cls");
57         tsumego_initial();
58         joueur_ordinnateur_tsumego();
59         if(rep_gagnant==1)
60         {
61             rep_gagnant=0;
62             system("cls");
63             goto N ;
64         }
65
66         else
67             exit(1);
68     }
69     if(rep==3)
70     {
71         system("cls");
72         goto I;
73     }
74 }
75 if(rep==1)
76 {
77
78         system("cls");
79         interface04();
80         JvsJ_initial();
81         joueur_joueur();
82         if(rep_gagnant==1)
83         {
84             {
85                 score_noir=0;
86                 score_blan=0;
87                 rep_gagnant=0;
88                 system("cls");
89                 goto N ;
90             }
91
92             else
93                 exit(1);
94             Color(15,0);
95         }
96         if(rep==3)
97         {
98             system("cls");
99             goto g;
100         }
101     }
102     if (rep==2)
103     {
104         rules();
105         goto g;
106     }
107
108     return 0;
109 }
```

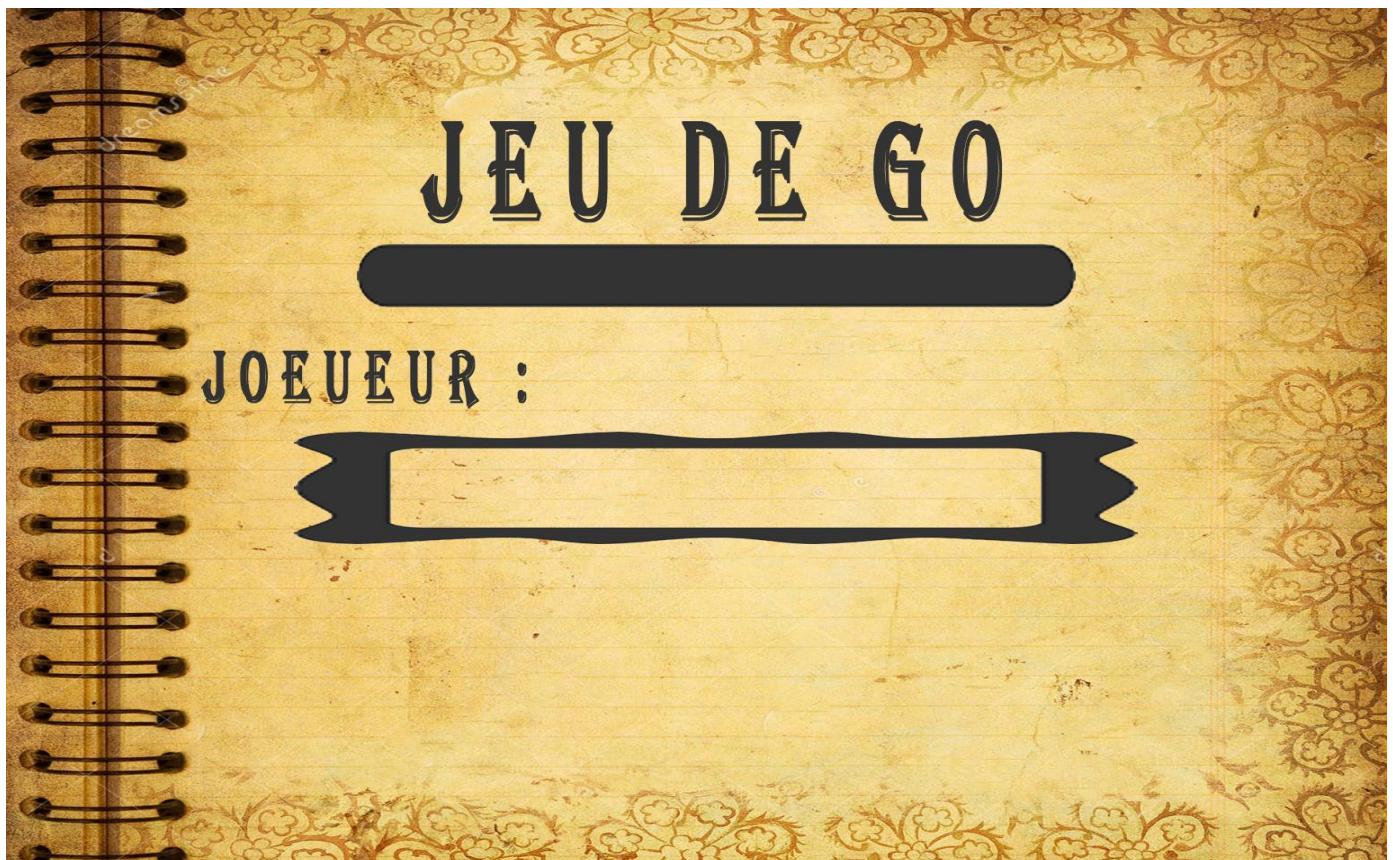
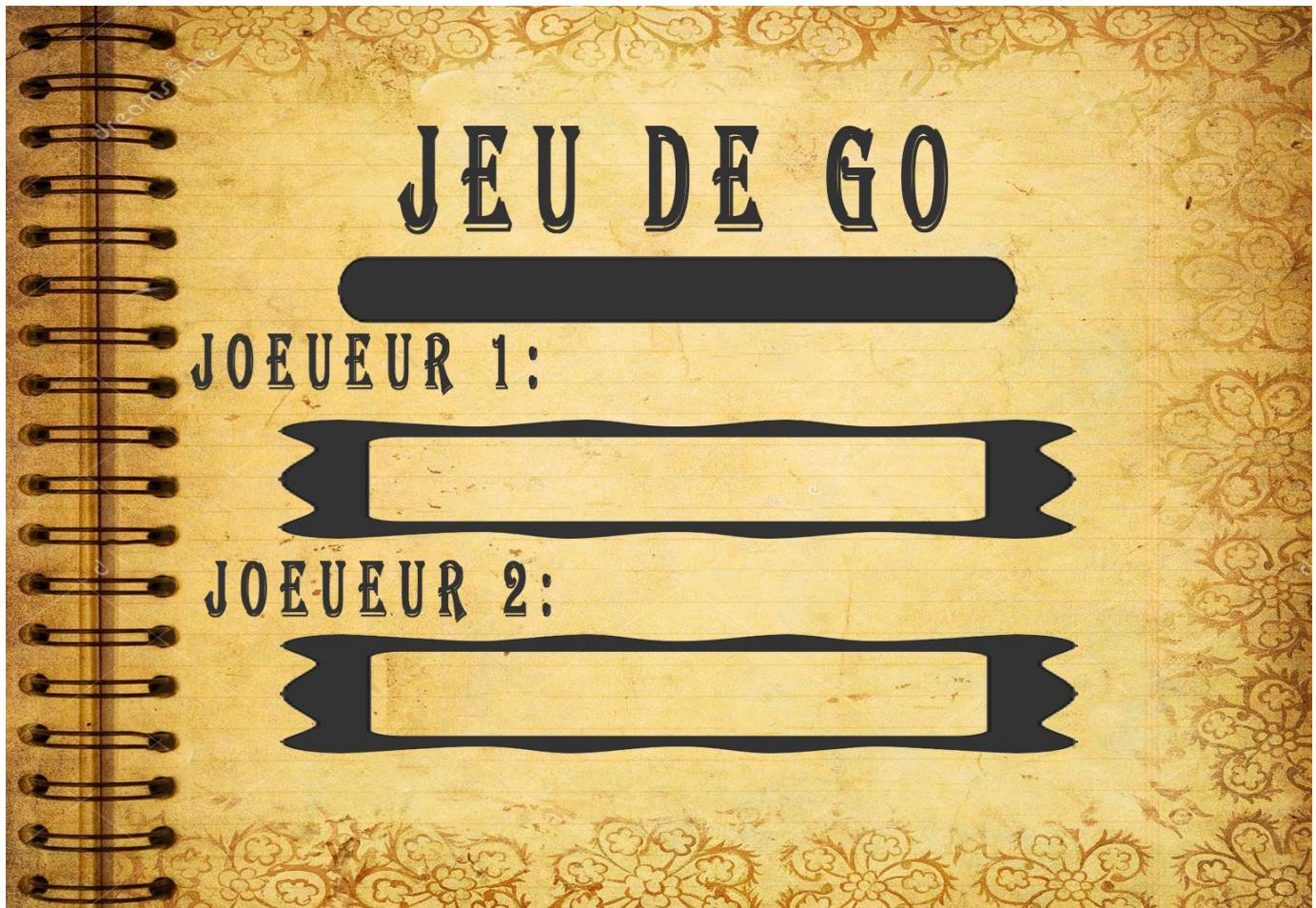
Les différents labels servent à rendre le jeu dynamique et souple, et les initialisations au début sert à assurer le bon fonctionnement du jeu.

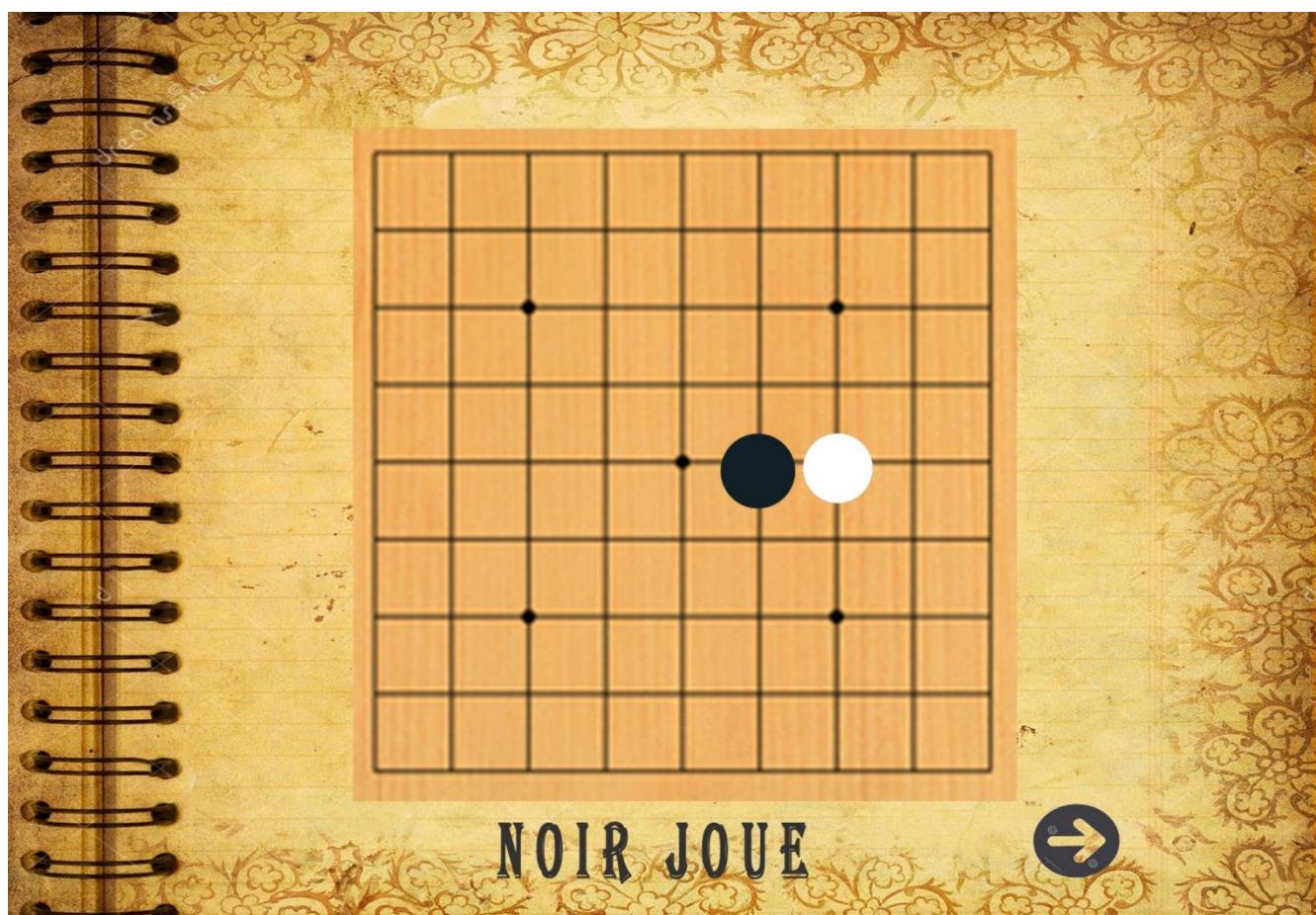
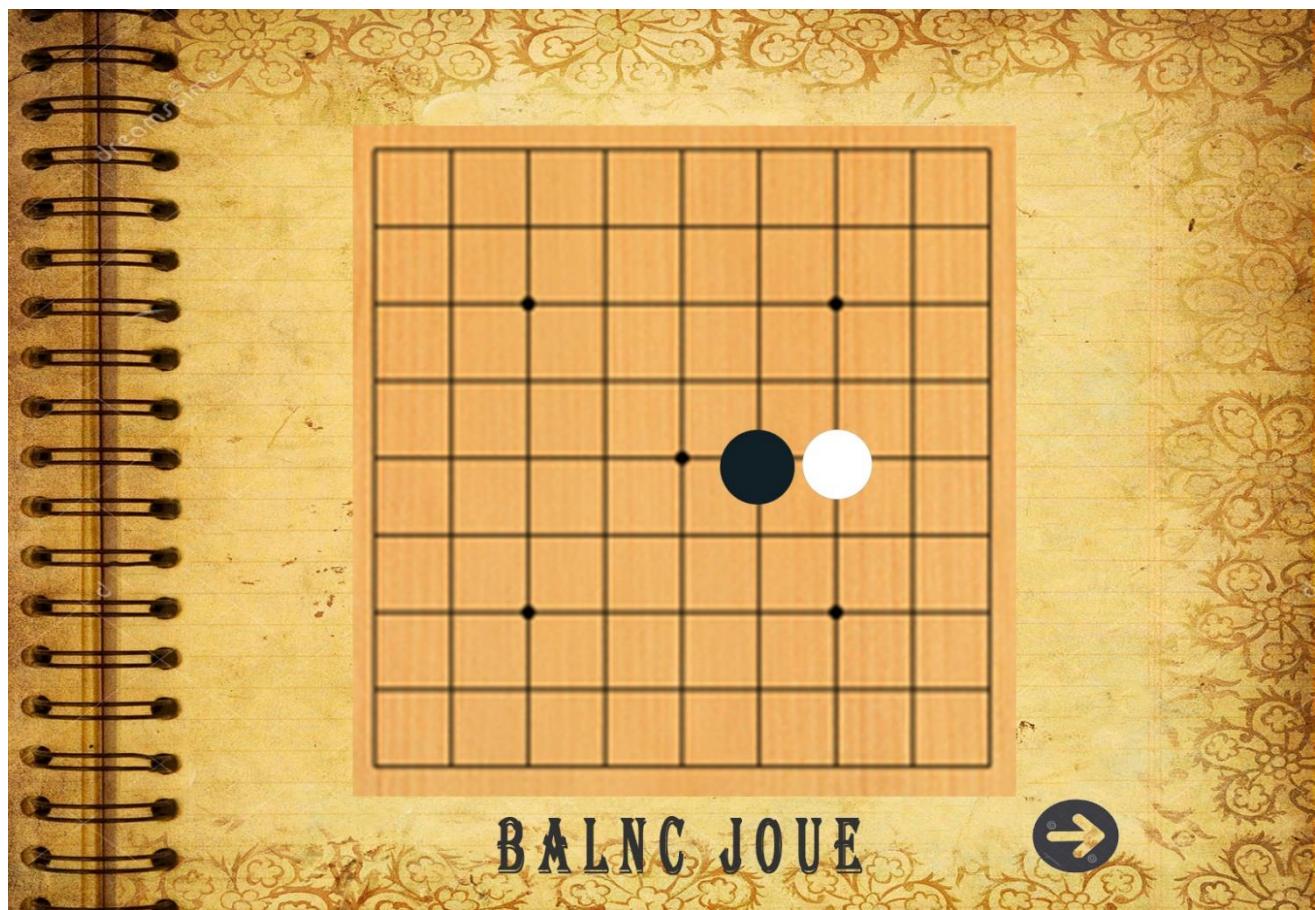
V. AMELIORATION :

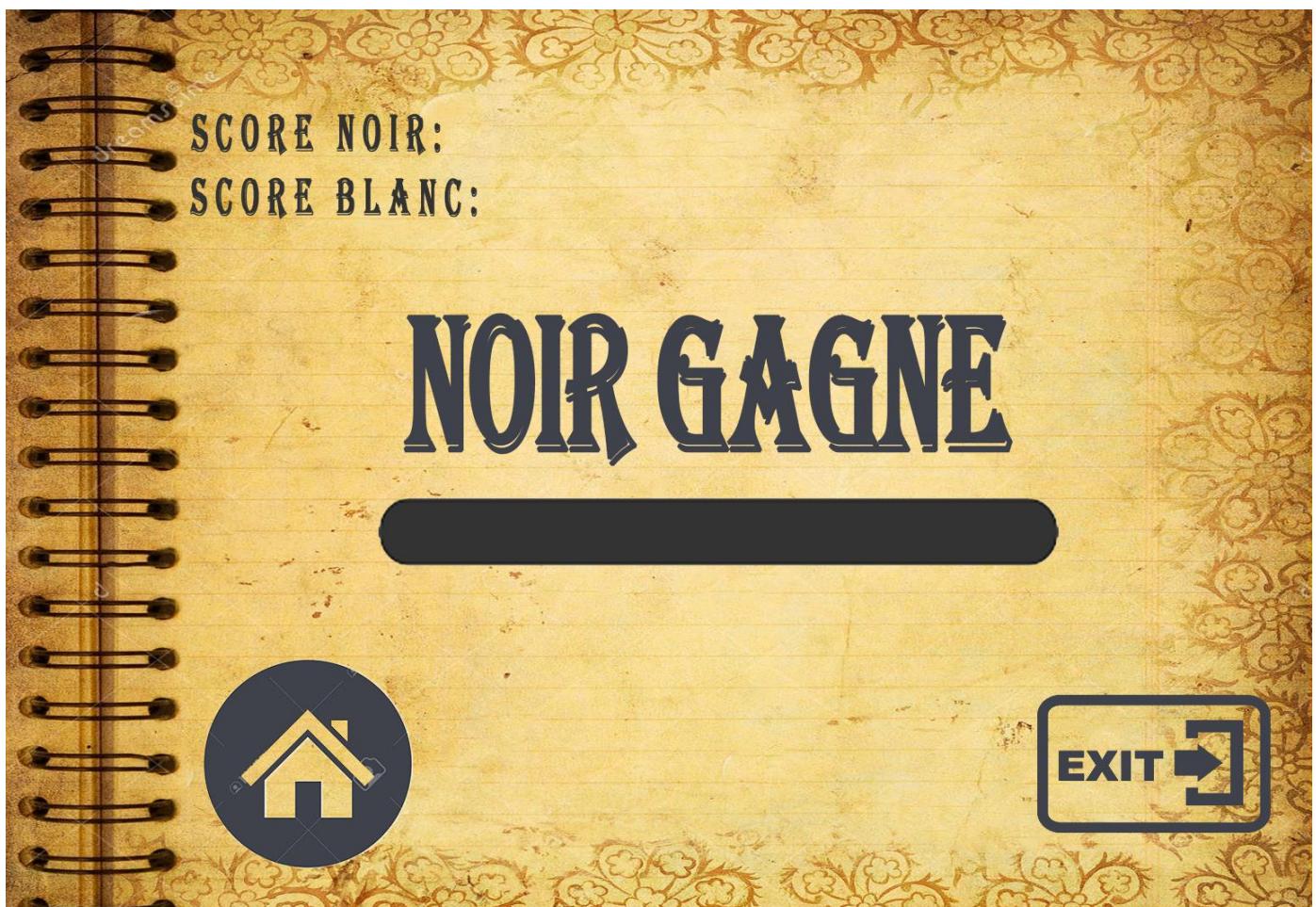
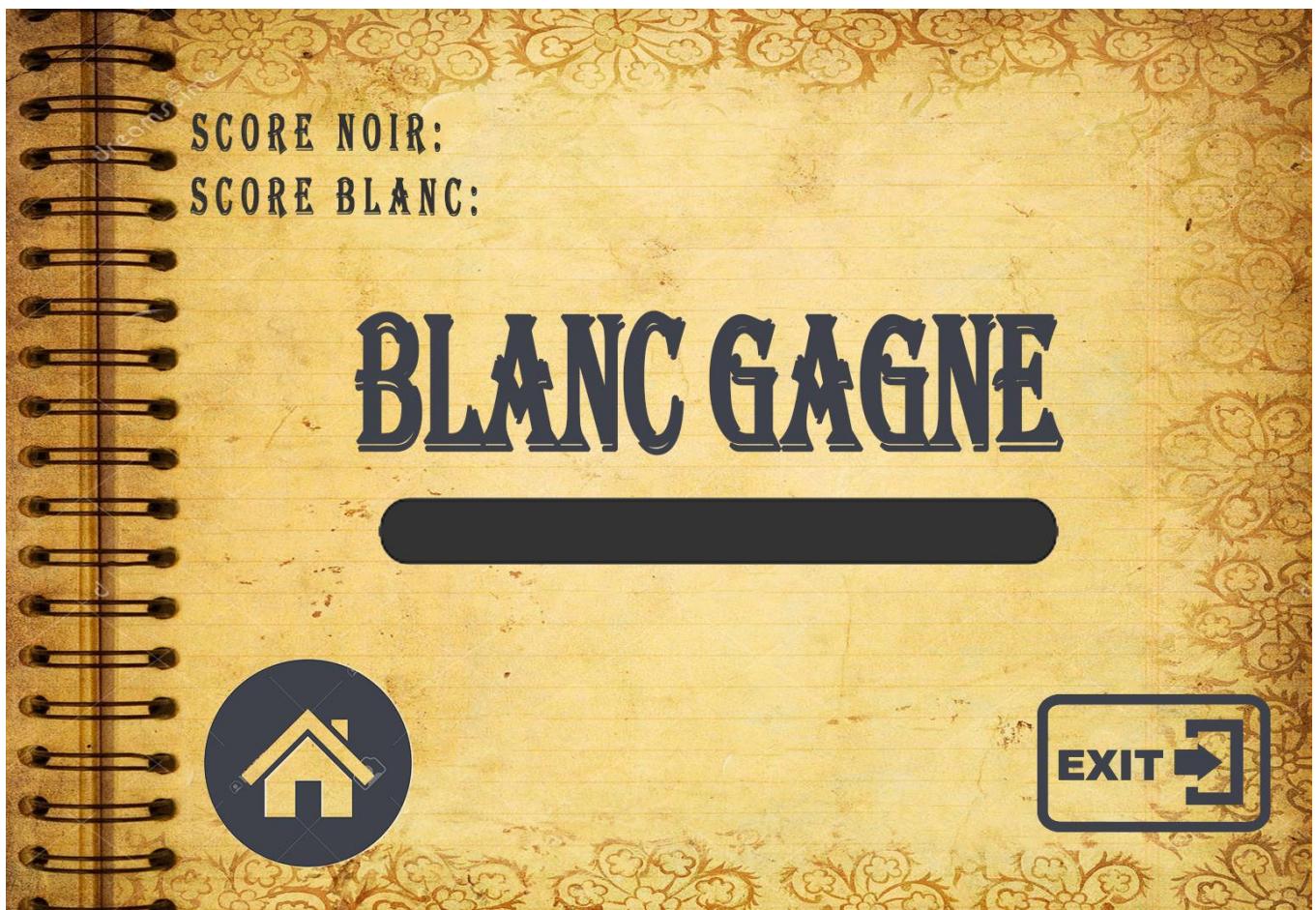
Pour améliorer notre jeu on a décidé de commencer une version SDL mais malheureusement on n'a pas encore terminé cette version.
Voilà quelques images de notre prochaine version :











VI. PROBLEMES RENCONTRES :



AU NIVEAU DU PROGRAMMATION :

Durant notre travail sur le jeu de GO on a changé et reformulé beaucoup de fonctions qui nous nécessite beaucoup du temps et d'efforts surtout celle de la récupération ; en effet , avant la méthode que nous avons expliqué au-dessus on a essayé de résoudre le problème de récupération avec la récursivité , et voilà le code qu'on a réalisé :

```

int degree_liberte(int i ,int j) // degree d'une boule
{
    int nbr_voisin(int i,int j) // nombre de boules voisins de même couleur
    {
        int dans_tab(int i, int j , int tab1[40],int tab2[40])
        {
            void ajouter(int i, int tab[40])
            {
                void chasse(int i,int j) // fct de recuperation
            }
        }
    }
}

```

Mais cette fois chasse est la suivante :

```

void chasse(int i,int j) // fct de recuperation
{
    if(degree_liberte(i,j)==0 && nbr_voisin(i,j)==0)
    {
        ajouter(i,les_i);
        ajouter(j,les_j);
        n=n+1;
    }
    if(degree_liberte(i,j)!=0)
    {
        fuite=fuite+1;
    }
    if(degree_liberte(i,j)==0 && nbr_voisin(i,j)!=0)
    {
        if(dans_tab(i,j,les_i,les_j)==0)
        {
            ajouter(i,les_i);
            ajouter(j,les_j);
            n=n+1;
        }
        if((i==1) && (j==1))
        {
            if((matrice[i][j]==matrice[i+1][j]) && (dans_tab(i+1,j,les_i,les_j)==0))
                chasse(i+1,j);
            if((matrice[i][j]==matrice[i][j+1]) && (dans_tab(i,j+1,les_i,les_j)==0))
                chasse(i,j+1);
        }
        if (i==9 && j==9)
        {
            if((matrice[i][j]==matrice[i-1][j]) && (dans_tab(i-1,j,les_i,les_j)==0))
                chasse(i-1,j);
            if((matrice[i][j]==matrice[i][j-1]) && (dans_tab(i,j-1,les_i,les_j)==0))
                chasse(i,j-1);
        }
    }
}

```

```

        }
        if (i==1 && j==9)
        {
            if((matrice[i][j]==matrice[i+1][j]) && (dans_tab(i+1,j,les_i,les_j)==0))
                chasse(i+1,j);
            if((matrice[i][j]==matrice[i][j-1]) && (dans_tab(i,j-1,les_i,les_j)==0))
                chasse(i,j-1);
        }
        if (i==9 && j==1)
        {
            if((matrice[i][j]==matrice[i-1][j]) && (dans_tab(i-1,j,les_i,les_j)==0))
                chasse(i-1,j);
            if((matrice[i][j]==matrice[i][j+1]) && (dans_tab(i,j+1,les_i,les_j)==0))
                chasse(i,j+1);
        }
        if(i==1 && j!=1 && j!=9)
        {
            if((matrice[i][j]==matrice[i+1][j]) && (dans_tab(i+1,j,les_i,les_j)==0))
                chasse(i+1,j);
            if((matrice[i][j]==matrice[i][j+1]) && (dans_tab(i,j+1,les_i,les_j)==0))
                chasse(i,j+1);
            if((matrice[i][j]==matrice[i][j-1]) && (dans_tab(i,j-1,les_i,les_j)==0))
                chasse(i,j-1);
        }
        if(i==9 &&j!=1 &&j!=9)
        {
            if((matrice[i][j]==matrice[i-1][j]) && (dans_tab(i-1,j,les_i,les_j)==0))
                chasse(i-1,j);
            if((matrice[i][j]==matrice[i][j+1]) && (dans_tab(i,j+1,les_i,les_j)==0))
                chasse(i,j+1);
            if((matrice[i][j]==matrice[i][j-1]) && (dans_tab(i,j-1,les_i,les_j)==0))
                chasse(i,j-1);
        }
        if(j==1 && i!=9 && i!=1)
        {
            if((matrice[i][j]==matrice[i+1][j]) && (dans_tab(i+1,j,les_i,les_j)==0))
                chasse(i+1,j);
            if((matrice[i][j]==matrice[i-1][j]) && (dans_tab(i-1,j,les_i,les_j)==0))
                chasse(i-1,j);
            if((matrice[i][j]==matrice[i][j+1]) && (dans_tab(i,j+1,les_i,les_j)==0))
                chasse(i,j+1);
        }
        if(j==9 && i!=9 && i!=1)
        {
            if((matrice[i][j]==matrice[i+1][j]) && (dans_tab(i+1,j,les_i,les_j)==0))
                chasse(i+1,j);
            if((matrice[i][j]==matrice[i-1][j]) && (dans_tab(i-1,j,les_i,les_j)==0))
                chasse(i-1,j);
            if((matrice[i][j]==matrice[i][j-1]) && (dans_tab(i,j-1,les_i,les_j)==0))
                chasse(i,j-1);
        }
        else
        {
            if((matrice[i][j]==matrice[i+1][j]) && (dans_tab(i+1,j,les_i,les_j)==0))
                chasse(i+1,j);
            if((matrice[i][j]==matrice[i-1][j]) && (dans_tab(i-1,j,les_i,les_j)==0))
                chasse(i-1,j);
            if((matrice[i][j]==matrice[i][j+1]) && (dans_tab(i,j+1,les_i,les_j)==0))
                chasse(i,j+1);
            if((matrice[i][j]==matrice[i][j-1]) && (dans_tab(i,j-1,les_i,les_j)==0))
                chasse(i,j-1);
        }
    }
}

```

```
        }
    }
    if(fuite==0)
    {
        for(int k=1;k<=n;k++)
        {
            matrice[les_i[k]][les_j[k]]=0;
        }
    }
    for(int k=1;k<=n;k++)
    {
        les_i[k]=0;
        les_j[k]=0;
    }
    fuite=0;
    n=0;
}
```

Cette fonction marche bien mais pour un certain niveau le programme se plante et on a consacré 3 jours pour chercher la source du problème mais on n'a pas pu réussir, et jusqu'à maintenant on ne sait pas la source du problème ; mais on a décidé de changer la méthode et c'est vraiment pénible de faire beaucoup d'effort alors que les résultats que nous attendons ne se réalisent pas.



AU NIVEAU DE DESIGN :

- **La Complexité d'affichage des dessins ASCII dans la Console :**
-

Et voilà l'exemple du GOBAN :

```
start();
int i,j;
int n=19;
int les_i[19];
int les_j[19];
int matrice[19][19];
int fuite=0;
for(i=0;i<n;i++)
{
    for(j=0;j<n;j++)
    {
        matrice[i][j]=0;
    }
}
for(i=0;i<n;i++)
{
    les_i[i]=0;
    les_j[i]=0;
}
fuite=0;
n=0;
}

void print()
{
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            cout<<matrice[i][j]<<" ";
        }
        cout<<\n;
    }
}

int main()
{
    start();
    print();
    return 0;
}
```

- **Création des graphiques (SDL) :**

On a essayé de faire une version en SDL mais on a trouvé des problèmes dans l'installation des outils de la SDL et comme on n'a pas beaucoup du temps on a laissé tomber cette version, mais on va la réaliser prochainement.

VII. CONCLUSION :

Ce projet a été une très bonne expérience, vu qu'il a testé nos connaissances en programmations acquises dans le 1er semestre. Mieux encore, ce projet est notre premier programme qui a dépassé les 2000 lignes de code est c'est vraiment magnifique puisque on a l'habitude de programmer des mini-projets qui ne dépassent même 100 lignes de code. En plus ce projet nous a appris à concevoir et modéliser avant de se lancer dans le développement, et résoudre les problèmes d'une manière professionnelle.

