



Rapport Projet 2023

Sujet : Développement D'un Jeu GO GAME

Ahmed bajadda
Ouabed ali
Mazouz ussama
Yassine dbaichi

Encadrement : Mme Mariam CHERRABI

SOMMAIRE

Rapport Projet 2023.....	
1. Introduction	
2. A propos du jeu	
1- Présentation du jeu :.....	
2- Règles du jeu	
➤ Chaînes et libertés	
➤ Capture	
➤ Coups interdits.....	
➤ Fin de partie	
➤ Komi	
➤ Territoire	
➤ Décompte final	
3. A propos du projet	
1- Travail demandé	
2- Les gros travaux réalisés dans le projet :.....	
3- Contraintes et problèmes générales rencontrés dans le projet :.....	
4- Outils et supports utilisés	
4. Navigation dans le jeu.	
➤ Page d'accueil	
➤ Les modes du jeu :	
➤ Joueur vs Joueur:	
➤ Les composantes du goban :	
➤ Joueur vs machine	
➤ Les coups interdits :	
➤ Capture	
➤ les messages d'orientation	
5. Conception du jeu :	
1- Représentation et stockage des informations collectés :	

- 2- Méthode de représentation et stockage des données collectés :
- 3- Représenter le goban de jeu :
- 6. Programmation des règles du jeu**
 - 1- regroupement des chaines avec leurs degres de liberte :
 - 2- capture.
 - 3- Suicide :
 - 4- le coup KO :
 - 5- traitement des territoires :
 - 6- calcul du score :
 - 7- Le joueur qui a le tour de rôle :
- 7. Architecture du jeu**
- 8. Intelligence artificiel**
 - Clôture :**

1-Introduction

Ce projet a été réalisé dans le cadre d'un module de programmation technique, au cours de travail nous avons été encadrés par notre professeur Mme M. Cherrabi, que nous remercions pour nous avoir donné l'opportunité de travailler sur un projet concret portant sur le jeu de Go.

Selon les exigences du cahier des charges, notre tâche consistait à programmer le jeu de Go en langage C et à produire deux versions du jeu : une version console et une version graphique implémentée à l'aide de la bibliothèque graphique SDL.

Au cours du travail sur ce projet, nous avons rencontré des difficultés que nous avons réussi à surmonter en faisant des choix stratégiques pour aboutir à notre produit final.

Dans ce rapport, nous présenterons les difficultés que nous avons rencontrées et expliquerons comment nous les avons surmontées. Nous justifierons également les choix de stratégies que nous avons adoptés en détaillant les limitations et les avantages de chaque proposition que nous avons envisagés, ainsi que les raisons qui nous ont poussés à faire ces choix.

2-A propos du jeu

1-Présentation du jeu :

Le jeu de Go est un jeu de stratégie abstrait, originaire de Chine, qui se joue sur un plateau de jeu quadrillé appelé Goban, avec des pierres noires et blanches. Le but du jeu est de contrôler le plus grand territoire possible en plaçant les pierres sur le Goban et en capturant celles de l'adversaire. Le Go est souvent considéré comme l'un des jeux de société les plus anciens et les plus complexes au monde. Il existe des preuves archéologiques suggérant que des jeux similaires ont été joués en Chine il y a plus de 2 500 ans. Le jeu de Go est très populaire en Asie, en particulier en Chine, au Japon, en Corée et à Taïwan, mais il est également pratiqué dans d'autres régions du monde. Le jeu a également une forte présence en ligne, avec de nombreux sites Web et applications proposant des parties en ligne. Le Go est souvent considéré comme un jeu qui développe la réflexion stratégique et la créativité, ainsi que la patience et la concentration. Il est également connu pour être un jeu de gentlemen, où le fair-play et le respect de l'adversaire sont valorisés.

2-Règles du jeu

❖ Chaînes et libertés

Les pierres qui ont la même couleur et qui se trouvent côte à côte en suivant les lignes de la grille (une pierre ayant donc jusqu'à quatre voisines) sont considérées comme connectées et forment une chaîne. Les intersections vides immédiatement adjacentes à une chaîne de pierres sont appelées libertés.

❖ Capture

La capture en jeu de Go se produit lorsqu'une ou plusieurs pierres d'un joueur sont entourées de pierres ennemies et ne sont pas connectées à d'autres pierres de leur propre couleur. Ces pierres capturées sont retirées du plateau et placées dans la réserve du joueur qui les a capturées.

❖ Coups interdits

- **Suicide**

Le suicide en jeu de Go se produit lorsqu'un joueur place une pierre dans une position où elle est immédiatement entourée de pierres ennemies et n'est pas connectée à d'autres pierres de sa propre couleur. Dans les règles officielles du jeu, le suicide est souvent interdit, sauf dans certaines situations où le suicide est un moyen de capturer des pierres ennemies ou d'éviter une capture imminente.

- **Ko**

Le ko est une situation dans le jeu de Go où une répétition de coups est possible, créant une boucle infinie. Cette situation se produit lorsqu'un joueur capture une pierre ennemie, et que cette capture est suivie immédiatement par une contre-attaque de l'adversaire qui recapture la pierre précédemment capturée, créant ainsi une boucle répétitive.

Pour éviter cette situation, les règles du jeu de Go prévoient une règle de ko qui interdit aux joueurs de répéter une position identique en jouant un coup. Si un joueur joue dans une telle position, l'adversaire a le droit de répondre et de supprimer la pierre qui a créé la répétition. Cette règle est importante car elle empêche les joueurs de se protéger de manière répétitive ou de prolonger inutilement le jeu.

❖ Fin de partie

La fin de partie en jeu de Go se produit lorsque les deux joueurs conviennent que le jeu est terminé ou lorsque les deux joueurs passent consécutivement. On comptabilise alors les points de chacun. Celui qui possède le plus de points

gagne. À ce stade, et les territoires sont comptabilisés pour déterminer le score final. Le score final est calculé en comptant les territoires contrôlés par chaque joueur, ainsi que les pierres capturées. Le joueur qui contrôle le plus de territoires et qui a capturé le plus de pierres gagne la partie.

❖ Komi

Le komi est généralement exprimé sous forme de points, qui sont ajoutés au score du joueur qui joue en second à la fin de la partie. Ces points sont généralement compris entre 5 et 7,5 points dans les parties de niveau professionnel, mais peuvent varier en fonction des règles utilisées.

Le komi est un élément important du jeu de Go, car il permet d'équilibrer les chances des joueurs et de rendre la partie plus compétitive. Sans komi, le joueur qui commence la partie aurait un avantage injuste, ce qui réduirait l'intérêt et l'équité du jeu.

❖ Territoire

Dans le jeu de Go, le territoire désigne les intersections vides du plateau de jeu qui sont entourées par les pierres d'une même couleur.

❖ Décompte final

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 6 points et demi (le demi-point sert à éviter les parties nulles). Le gagnant est celui qui a le plus de points.

3-A propos du projet

1-Travail demandé

L'objectif du projet est de programmer le jeu de Go en utilisant le langage C. Le jeu développé doit permettre à deux joueurs humains de jouer ensemble et également offrir la possibilité de jouer contre l'ordinateur. Le projet doit être réalisé en deux versions, une version console et une version SDL.




le travail final doit respecter les règles conventionnelles du jeu (capture, territoires, coups interdits...).

2-Les gros travaux réalisés dans le projet :

Dans notre travail nous avons réalisé :

- ✓ La version console du jeu
- ✓ Le fonctionnement de la capture des pierres n'ayant point de degrés de libertés.
- ✓ Le fonctionnement du calcul du degré de libertés des pierres et des groupes de pierres.
- ✓ Le fonctionnement du calcul des territoires occupés par les deux joueurs.
- ✓ Le fonctionnement nécessaire pour interdire tous les coups interdits comme le suicide et le KO.
- ✓ Faire en sorte que le jeu produit soit intègre et respecte le fonctionnement naturel du jeu GO et ses règles conventionnelles.
- ✓ Définir un exemple de AI avec trois fonctions (capture, Save, sur round

3-Contraintes et problèmes générales rencontrés dans le projet :

-  **Contrainte d'optimisation du jeu** : cette contrainte nous a pris beaucoup de temps, parce que dans un jeu go il y'a en général beaucoup plus de traitement à faire après chaque coup nous allons en citer (éviter les coups interdits, faire disparaître du goban tous les pierres capturés, calcul des territoires...) ce qui nous a poussé à déployer un effort supplémentaire pour optimiser le produit et final et le rendre un peu plus léger.
-  **Contrainte de temps** : malgré que nous ayons été conscient que le temps est très pressé dès la réception du cahier de charge, et malgré que nous ayons commencé de manœuvrer dès la réception du cahier de charges, cette problématique nous fut inévitable, cependant nous avons veuilles à réaliser le maximum des travaux possible.
-  **Comprendre le jeu et ces règles** : ce qui rend le jeu go très complexe, c'est le fait que les règles du jeu varient d'un pays à l'autre, par exemples il y'a des règles Chinois, des règles Japonais et des règles françaises. Dans

un certain moment lorsqu'on est amené à comprendre le fonctionnement du calcul des territoires, nous avons subi une grande confusion due aux plusieurs variantes des règles.

4-Outils et supports utilisés :

✓ Changer les couleurs du texte :

```
1 void color(int text_color, int bg_color)
2 {
3     HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
4     int color = text_color + (bg_color * 16);
5     SetConsoleTextAttribute(hConsole, color);
6 }
```

✓ Branchement inconditionnel

goto : est un mot-clé en C qui permet de transférer le contrôle du programme à une étiquette spécifique dans le code.

✓ Système Pause :

```
system("pause");
```

✓ Nettoyer la console :

Avec un simple appel à la fonction au-dessous on pourra nettoyer l'écran pour la remplir avec le contenu généré pour le coup de rôle suivant.

```
system("cls");
```

4-Navigation dans le jeu

Nous allons maintenant naviguer à travers les différents menus de notre jeu pour découvrir son ergonomie et son fonctionnement.

❖ Page d'accueil

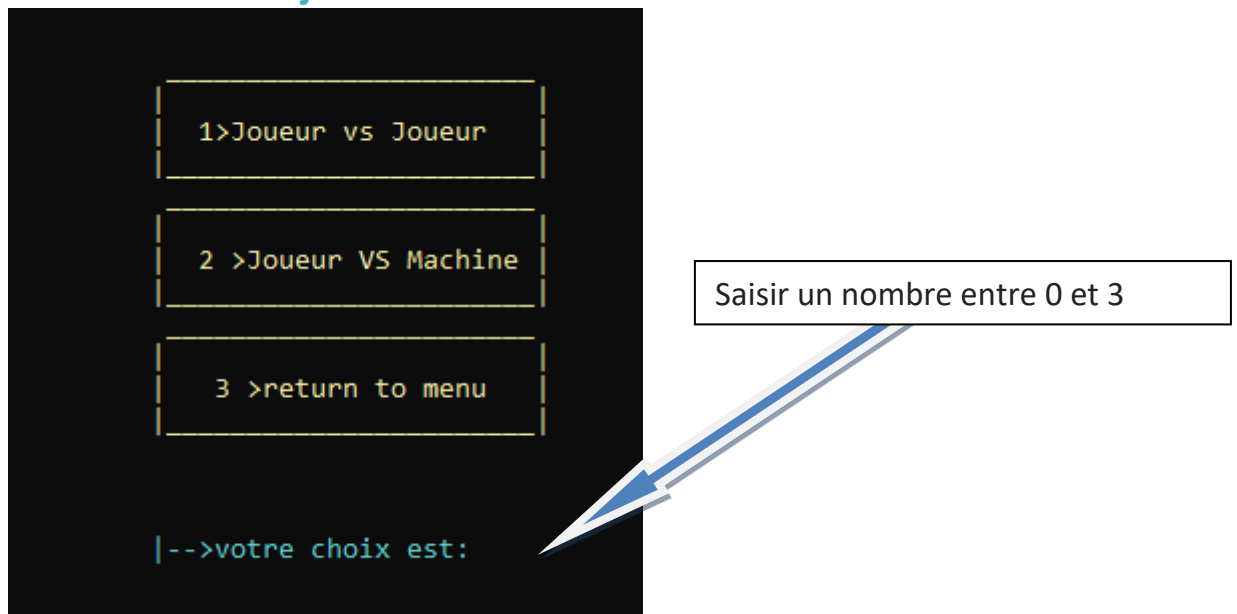


Entrez un nombre entre 1 et 3

La page d'accueil offre 3 choix :

- ✓ **Jouer** : permet d'accéder aux modes du jeu
- ✓ **Rules** : permet une consultation des règles principales du jeu
- ✓ **Exit** : pour fermer le jeu définitivement.

❖ Les modes du jeu :



Lorsqu'on saisit 1 pour jouer dans le menu d'accueil on accède directement aux modes du jeu, on a réalisé deux modes comme exigé par le cahier de charges.

- ✓ **Mode joueur vs joueur** : ici pour donner la main à deux joueurs humains pour jouer.
- ✓ **Mode joueur vs machine** : pour donner la main à un seul joueur pour qu'il puisse jouer contre l'ordinateur.
- ✓ **Return to menu** : pour retourner au menu principal.

❖ Joueur vs Joueur :

Lorsqu'on écrit 2 de menu des modes du jeu on accède à une autre page dans laquelle on écrit les noms des deux joueurs avec un tirage de sort qui détermine le joueur qui va commencer premièrement (toujours qui a le noir commence premièrement).

```

=>Veuillez saisir le nom du premier joueur |->:A

=>Veuillez saisir le nom du deuxieme joueur |->:B

Joueur1>>B:start first with black pion!
Appuyez sur une touche pour continuer...

```

Apres :

```

| 404 to exit | | 505 to pass |
|-----|
| player_2:1 | | white_stones: 0 | capture 6.5 | white_territory:0
|-----|
| player_1:1 | | black_stones: 0 | capture 0 | black_territory:0
|-----|
#TSUME_GO#
      A      B      C      D      E      F      G      H      I
0  0  --- 1  --- 2  --- 3  --- 4  --- 5  --- 6  --- 7  --- 8
1  9  ---10 ---11 ---12 ---13 ---14 ---15 ---16 ---17
2 18  ---19 ---20 ---21 ---22 ---23 ---24 ---25 ---26
3 27  ---28 ---29 ---30 ---31 ---32 ---33 ---34 ---35
4 36  ---37 ---38 ---39 ---40 ---41 ---42 ---43 ---44
5 45  ---46 ---47 ---48 ---49 ---50 ---51 ---52 ---53
6 54  ---55 ---56 ---57 ---58 ---59 ---60 ---61 ---62
7 63  ---64 ---65 ---66 ---67 ---68 ---69 ---70 ---71
8 72  ---73 ---74 ---75 ---76 ---77 ---78 ---79 ---80

|-->Joueur1 '1':your turn!
|->votre choix d'indice i>>

```

C'est parti ! Nous avons le goban, et nous avons décidé de mettre en place une conception différente du jeu original. Au lieu de l'utilisation des abscisses et des ordonnées des intersections nous avons ainsi un goban contenant des intersections numérotées de 0 et 81. Pour poser une pierre sur une intersection, l'utilisateur n'a besoin que de saisir un seul nombre, plutôt que deux nombres. Cela rend l'expérience utilisateur plus fluide et agréable. Cette conception différente nous a également aidé du côté de la programmation, car grâce à elle, nous avons travaillé avec des tableaux à une seule dimension. Cela permet de parcourir les tableaux avec un seul indice, plutôt que deux indices, ce qui rend la manipulation des tableaux plus facile au niveau de codage de toutes les fonctions.

❖ Les composantes du goban :

- ✓ **Si vous insérez 505** : cela indique le joueur qui a le tour de rôle, et qui permet aussi aux joueurs d'abandonner son tour de rôle et de passer la main à l'autre joueur pour jouer.
- ✓ **Si vous insérez 404** : pour quitter le jeu.
- ✓ **Les informations sur chaque joueur** : on a un tableau qui donne les informations de chaque joueur : le nom ; le nombre des pierres dans le goban ; le territoire ; le nombre de pierres capturées.

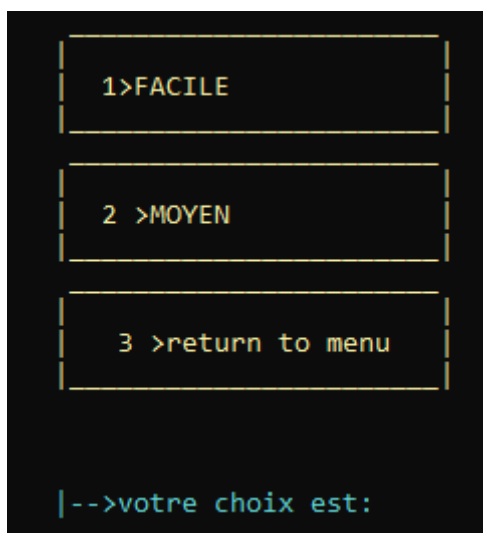
```

| player_2:ALI | | white_stones: 5 | capture 9.5 | white_territory:2
| player_1:YASSINE | | black_stones: 4 | capture 3 | black_territory:2
|-----#TSUME_GO#

```

- **Remarque** : Les trois types de points indiqués au-dessus sont importants pour le calcul du score de chaque joueur et détermination du gagnant.

❖ Joueur vs machine :

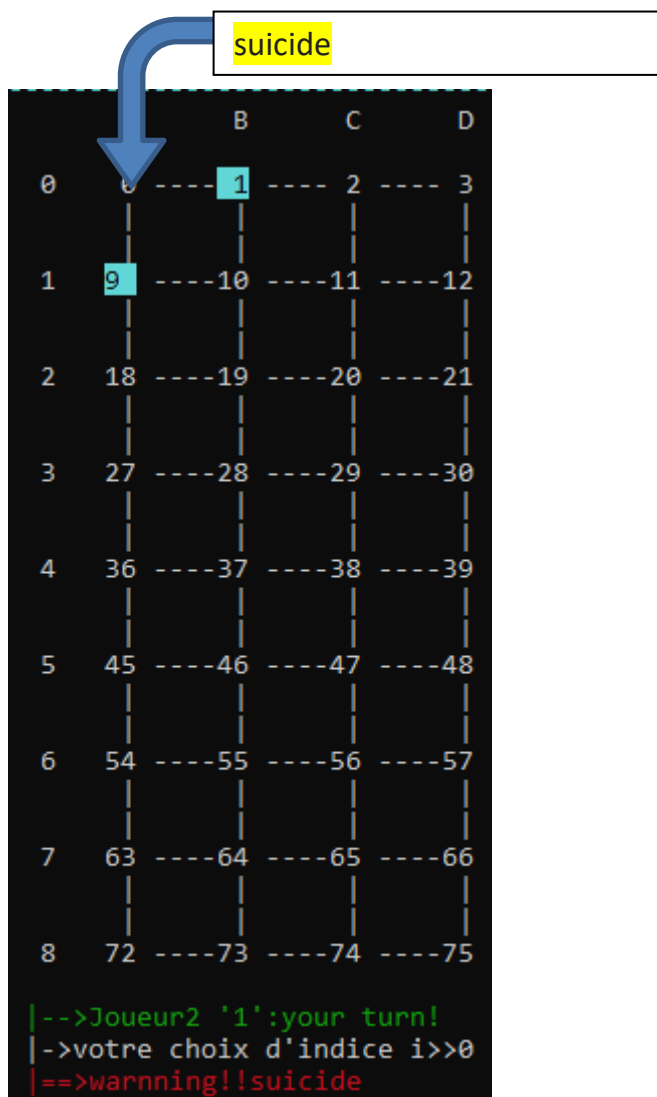


Quand on insère 2 dans les modes de jeu on obtient directement 2 différents niveaux de difficultés pour le jouer contre la machine, facile et moyen.

❖ Les coups interdits :

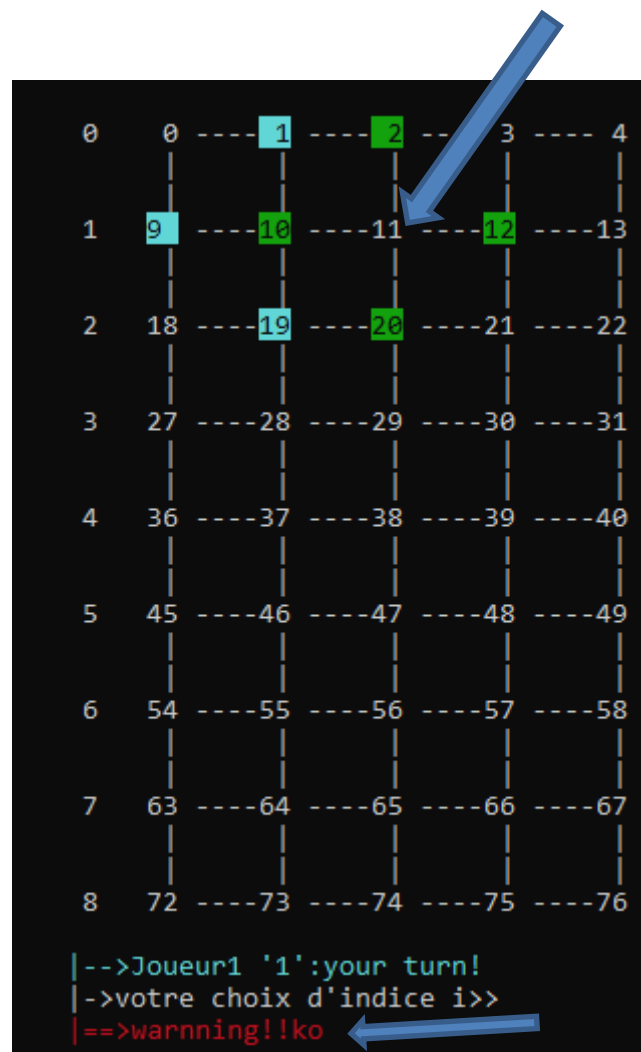
- **Suicide :**

Dans le jeu de go, le suicide se produit lorsqu'un joueur place une pierre dans une position qui rendrait son propre groupe de pierres entièrement capturé. En d'autres termes, le joueur met en danger son propre groupe sans mettre en danger le groupe de l'adversaire. Dans les règles du jeu, le suicide est interdit, et toute tentative de suicide est considérée comme un coup illégal et est donc immédiatement retirée du plateau de jeu.



- **Le coup KO :**

Nous avons déjà expliqué le coup KO dans la partie règles du jeu, ce dernier est considéré un coup interdit dans le jeu, nous avons respecté cette règle, voici une figure de KO :



Dans notre programme nous avons implémenté une fonction qui évite que ses cas se produisent, et tous les autres cas si y'en a.

Capture :

AVANT :

```

| 404 to exit | | 505 to pass |
|-----|
| player_2:1 | | white_stones: 5 | capture 6.5 | white_territory:0
|-----|
| player_1:1 | | black_stones: 8 | capture 0 | black_territory:0
|-----|
#TSUME_GO#
  A   B   C   D   E   F   G   H   I
0  0   1   2   3   4   5   6   7   8
1  9  10  11  12  13  14  15  16  17
2 18 19 20 21 22 23 24 25 26
3 27 28 29 30 31 32 33 34 35
4 36 37 38 39 40 41 42 43 44
5 45 46 47 48 49 50 51 52 53
6 54 55 56 57 58 59 60 61 62
7 63 64 65 66 67 68 69 70 71
8 72 73 74 75 76 77 78 79 80

|-->Joueur2 '1':your turn!
|->votre choix d'indice i>>

```

après :

```

| player_2:1 | | white_stones: 0 | capture 6.5 | white_territory:0
|-----|
| player_1:1 | | black_stones: 9 | capture 5 | black_territory:72
|-----|
#TSUME_GO#
  A   B   C   D   E   F   G   H   I
0  0   1   2   3   4   5   6   7   8
1  9  10  11  12  13  14  15  16  17
2 18 19 20 21 22 23 24 25 26
3 27 28 29 30 31 32 33 34 35
4 36 37 38 39 40 41 42 43 44
5 45 46 47 48 49 50 51 52 53
6 54 55 56 57 58 59 60 61 62
7 63 64 65 66 67 68 69 70 71
8 72 73 74 75 76 77 78 79 80

|-->Joueur2 '1':your turn!
|->votre choix d'indice i>>

```

Un autre cas de capture :

```

-----
| player_2:1 | | white_stones: 8 | capture 6.5 | white_territory:68
-----
| player_1:1 | | black_stones: 4 | capture 0 | black_territory:1
-----
#TSUME_GO#

```

	A	B	C	D	E	F
0	0	1	2	3	4	5
1	9	10	11	12	13	14
2	18	19	20	21	22	23
3	27	28	29	30	31	32
4	36	37	38	39	40	41

Après :

```

-----
| player_2:1 | | white_stones: 9 | capture 10.5 | white_territory:72
-----
| player_1:1 | | black_stones: 0 | capture 0 | black_territory:0
-----
#TSUME_GO#

```

	A	B	C	D	E	F
0	0	1	2	3	4	5
1	9	10	11	12	13	14
2	18	19	20	21	22	23
3	27	28	29	30	31	32
4	36	37	38	39	40	41

❖ Les messages d'orientations

Des messages ont été intégrés sur le goban pour guider l'utilisateur en cas d'erreurs, afin de lui montrer ce qui ne va pas. Voici quelques exemples de messages d'orientation pour l'utilisateur : Par défaut, l'utilisateur est invité à saisir un nombre entier compris entre 0 et 80.

```

|-->Joueur1 '1':your turn!
|->votre choix d'indice i>>

```

Si l'utilisateur a donné une valeur qui n'est pas comprise entre 0 et 80 on lui affiche un message d'erreur qui indique la source de l'erreur.

```
|-->Joueur1 '1':your turn!  
|->votre choix d'indice i>>400  
|==>waranning!!donne une indice entre 0 et 80!
```

Si l'utilisateur a saisi la valeur d'une intersection qui est déjà remplie, on lui affiche le message suivant.

```
|-->Joueur1 '11':your turn!  
|->votre choix d'indice i>>4  
|==>waranning!!occupied position
```

Parmi les coups interdits il y a KO, si l'utilisateur a donné une valeur qui amènera à la situation du KO on lui affiche le message suivant.

```
|-->Joueur1 '1':your turn!  
|->votre choix d'indice i>>  
|==>warning!!ko
```

Un autre coup interdit, quand l'utilisateur entre la valeur d'une intersection dont le degré de liberté est nul ou bien une valeur qui va amener à la situation d'un suicide on lui affiche le message suivant.

```
|-->Joueur2 '1':your turn!  
|->votre choix d'indice i>>0  
|==>warning!!suicide
```

Sur le goban on a implémenté une possibilité de quitter le jeu en saisissant le nombre 404, (et 505 pour un pass) si l'utilisateur demande de quitter une certaine partie on lui demande d'abord de confirmer qu'il veut quitter.

```
|-->Joueur2 '1':your turn!  
|->votre choix d'indice i>>404  
|==>voulez vous vraiment quitter?[ 1 = YES ] [ anything else = NO ]  
1) OUI 2) NON -->
```

5-Conception du jeu :

Dans cette phase très importante nous allons voir comment on a répondu à aux questions qui tournent autour de la conception et codage du jeu :

1-Représentation et stockage des informations collectés:

D'abord la question était auparavant, est-ce qu'on doit stocker les informations qui comportent les coups déjà jouées ou bien on n'a besoin que de les afficher. Effectivement la question était légitime car parfois on n'a pas besoin de stocker certaines informations et on se contente juste de les afficher dès le tour en question, mais dans le jeu GO on aura besoin de calculer les degrés de liberté qui peuvent être changés après chaque tour de rôle, et parfois on doit même éliminer les pierres dont les degrés de libertés sont nuls.

- ✓ **Conclusion** : on doit stocker les informations qui se collectent durant les tours de rôles

Maintenant qu'on a conclu que le stockage des informations collectés et légitime, une autre question se pose, c'est la suivante

2-Méthode de représentation et stockage des données collectés :

On va maintenant parler de quelques propositions qu'on s'est proposé durant la phase de conception et voir justifier le choix qu'on a effectué.

❖ matrice de deux dimensions (stratégie éliminé) :

On a décidé d'éviter l'utilisation d'une matrice a deux dimensions car elle représente plusieurs contraintes :

- 1- Taille de la structure de données : une matrice peut être très grande et nécessiter beaucoup de mémoire pour stocker toutes les informations nécessaires pour représenter un plateau de jeu de Go de taille standard
- 2- Mauvaise lisibilité et faible manipulation du code

- ✓ **Conclusion** : un tableau de deux dimensions n'est pas le choix le plus optimale pour représenter les informations collectées

❖ Notre solution : Tableau à une dimension :

La décision que nous avons déjà prise d'utiliser des entiers de 0 à 80 pour identifier les intersections du goban pour chaque joueur, comme nous l'avons

également discuté précédemment dans la section sur la navigation dans le jeu, nous offre l'avantage de pouvoir représenter toutes les données avec des tableaux à une seule dimension.

- ✓ **Conclusion** : on va prendre toutes les informations qu'on collecte durant une partie et on va les stocker dans des tableaux d'une dimension.

3-Représenter le goban sur de jeu :

on a représenté le goban par une table tel que les intersections ont un indice unique entre 0 et 80 séparées par des étoiles et des « | ».

6-Programmation des règles du jeu

1-regroupement des chaines avec leurs degrés de liberté:

```

2- /*check-group_lib:The check_group function is called recursively to
3- traverse the group and mark all stones as checked and calculate liberte degree of every position(0 si position vide 1 sinon)*/
4-
5- void check_group_lib(int i,int M[81],int checked[81],int liberte[81])
6- {
7-     if (M[i]!=0)
8-     {
9-         checked[i]=1;
10-        int h,b,g,d;
11-        h=position_haut(i);
12-        b=position_bas(i);
13-        g=position_gauche(i);
14-        d=position_droit(i);
15-
16-        if(h!=-1 && M[h]==0)//tester l'existence de la position et est ce qu'elle vide
17-            liberte[h]=1;//si h vide :la position h dans liberte prend la valeur 1
18-        if(h!=-1 && M[h]==M[i]&&checked[h]==0)
19-        {
20-            /*la position haut existe dans le goban et remplit par une
21-            pion(soit noir soit blanche et cette position n'est pas parcouru auparavant.*/
22-
23-            checked[h]=1;//mark this position as checked
24-            check_group_lib(h,M,checked,liberte);
25-        }//de mm pour cette position
26-        if(b!=-1 && M[b]==0)
27-            liberte[b]=1;
28-        if(b!=-1 && M[b]==M[i]&&checked[b]==0)
29-        {

```

```

30     checked[b]=1;
31     check_group_lib(b,M,checked,liberte);
32 }
33 if(d!=-1 && M[d]==0)
34     liberte[d]=1;
35 if(d!=-1 && M[d]==M[i]&&checked[d]==0)
36 {
37     checked[d]=1;
38     check_group_lib(d,M,checked,liberte);
39 }
40 if(g!=-1 && M[g]==0)
41     liberte[g]=1;
42 if(g!=-1 && M[g]==M[i]&&checked[g]==0)
43 {
44     checked[g]=1;
45     check_group_lib(g,M,checked,liberte);
46 }
47 }
48 }
49 //calculer degre de liberte d un grp
50 int count_liberte(int liberte[81])
51 {
52     int sum_degre=0;
53     for(int j=0; j<81; j++)
54     {
55         sum_degre+=liberte[j];
56     }
57     return sum_degre;
58 }

```

La première fonction, "check_group_lib", prend en entrée quatre paramètres:

"i", qui est un entier représentant l'indice de la pierre à vérifier dans le tableau "M".

"M", qui est un tableau d'entiers représentant le goban (plateau de jeu).

"checked", qui est un tableau d'entiers de même taille que "M" et qui indique si une pierre a déjà été vérifiée ou non (groupe des amis de l'intersection i) .

"liberte", qui est un tableau d'entiers de même taille que "M" et qui stocke les degrés de liberté de chaque groupe de pierres.

La fonction vérifie si la pierre à l'indice "i" est non vide (différente de zéro). Si c'est le cas, elle marque la pierre "i" comme vérifiée dans le tableau "checked".

Ensuite, elle calcule les indices des pierres voisines de la pierre "i" dans les directions haut, bas, gauche et droite en appelant des fonctions auxiliaires "position_haut", "position_bas", "position_gauche" et "position_droit".

Si une pierre voisine est vide, la fonction met à jour son degré de liberté dans le tableau "liberte".

Si la pierre voisine n'est pas vide et qu'elle appartient au même groupe de pierres que la pierre "i" (c'est-à-dire qu'elle est de la même couleur), la fonction marque la pierre voisine comme vérifiée et rappelle

récurivement la fonction "check_group_lib" avec la pierre voisine comme argument. Ainsi, la fonction parcourt tous les groupes de pierres connectées à la

pierre "i" et met à jour leurs degrés de liberté dans le tableau "liberte".

La seconde fonction, "count_liberte", prend en entrée un tableau "liberte" contenant les degrés de liberté de chaque groupe de pierres, et retourne la somme de tous les degrés de liberté. Elle est utilisée pour calculer le degré de liberté total d'un groupe de pierres, ce qui permet de déterminer si ce groupe de pierres est capturé ou non.

2 - capture :

En utilisant les deux dernières fonctions et en parcourant le goban après chaque coup, si un groupe d'adversaires à zéro degré de liberté, il sera enlevé du goban et chaque pierre capturée ajoutera un point au score du joueur. (Voir code console).

3-suicide :

[illegible]

[illegible]

Nous avons vu la définition du « suicide »

Pour vérifier si un mouvement est un suicide dans le Go, nous avons traité les cas suivants :

- 1/Vérifiez les libertés des pierres voisines : si une pierre voisine a une liberté ouverte, le mouvement n'est pas un suicide.
- 2/Vérifiez si le mouvement crée un nouveau groupe : si le mouvement crée un nouveau groupe avec des libertés, alors il n'est pas un suicide.
- 3/Vérifiez si le mouvement capture les pierres de l'adversaire : si le mouvement capture une ou plusieurs pierres de l'adversaire, alors ce n'est pas un suicide.

On l'implémente en spécifiant trois cas où le coup n'est pas un suicide ; on a utilisé une nouvelle matrice (temporaire pour vérifier les cas ci-dessus) à remplir par les éléments de M après chaque coup.

4- le coup KO :

On a utilisé deux fonctions « ko move » et « ko prevent » .

La première fonction, "ko_move", prend deux tableaux d'entiers "pre_M" et "after_M" de 81 éléments chacun, qui représentent respectivement l'état du plateau de jeu avant et après un coup joué par un joueur. La fonction compare les deux tableaux élément par élément et compte le nombre d'éléments qui sont identiques. Si tous les éléments sont identiques, cela signifie que le coup

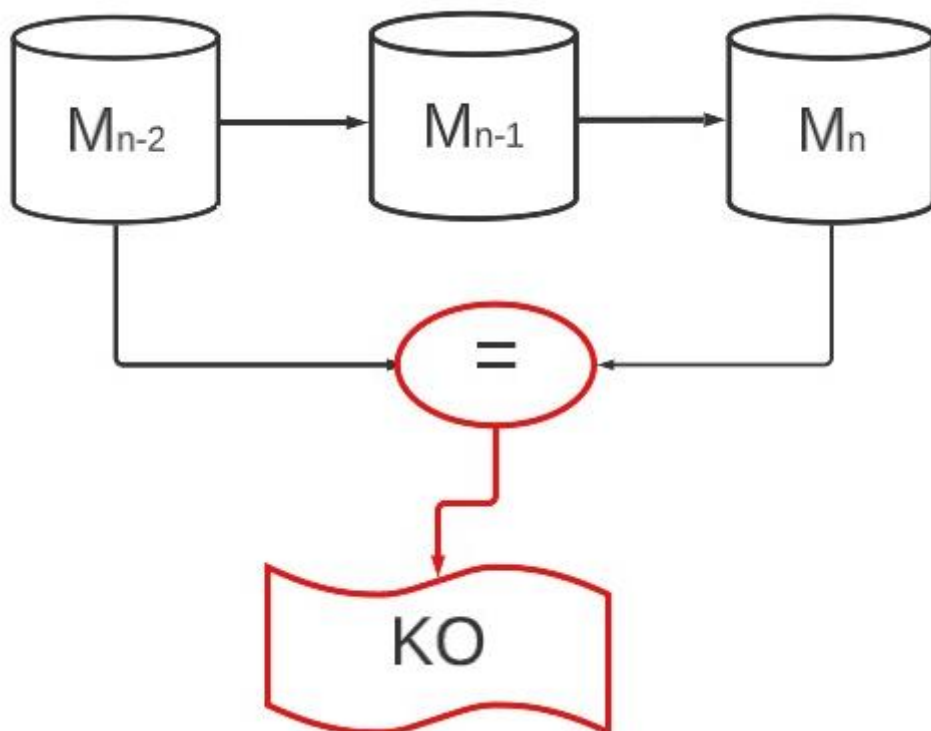
joué par le joueur est un ko, et la fonction renvoie 1. Sinon, elle renvoie 0 pour indiquer que le coup n'est pas un ko.

La deuxième fonction, "ko_prevent", utilise la première fonction pour détecter si le coup joué par un joueur est un ko. Si c'est le cas, la fonction effectue une série d'actions pour empêcher le joueur de jouer ce coup.

Tout d'abord, si la variable "ko" est égale à 0, la fonction enregistre l'état actuel du plateau de jeu dans les tableaux "before_move1_M", "after_move1M" et "after_move2M". Ces tableaux servent à sauvegarder l'état précédent du plateau de jeu afin de pouvoir le restaurer si le coup est un ko.

Ensuite, la fonction utilise la fonction "ko_move" pour vérifier si le coup joué par le joueur est un ko. Si c'est le cas, la fonction change la variable "player" pour donner une autre chance de jouer au joueur. Elle restaure également l'état précédent du plateau de jeu en copiant les tableaux "after_move1M" et "before_move1_M" dans le tableau "M". Enfin, si le joueur qui a joué le coup est le joueur blanc, la fonction décrémente la variable "count_captured_white" de 1, sinon elle décrémente la variable "count_captured_black" de 1.

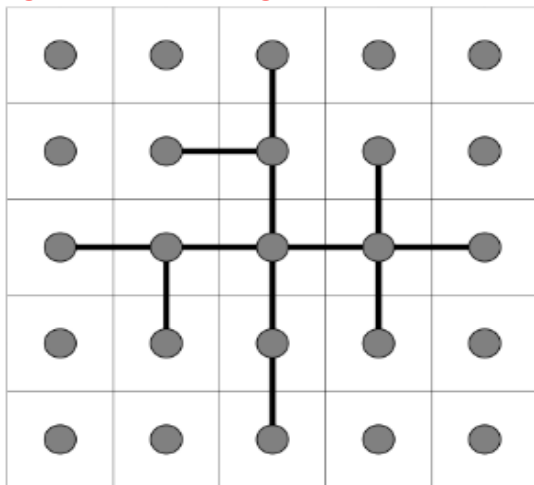
En résumé, ces deux fonctions permettent de détecter et de prévenir les coups ko dans le jeu de Go, conformément à la règle du jeu.



5-Traitement des territoires :

La fonction qui nous a pris une grande partie de temps est effectuée de la même manière que la fonction "checke_groupe" (flood fill algorithm : L'algorithme de remplissage par diffusion, également appelé algorithme de remplissage par inondation, est une méthode récursive utilisée pour remplir une zone fermée à l'intérieur d'une image avec une couleur ou une texture spécifique (voir figure).) et est réalisée de manière récursive. La difficulté réside dans le fait que le traitement est effectué sur les intersections qui sont vides, comme le montre le code.

figure : flood fill algorithm



```

1 //accumuler les intersections vide qui appartient au meme groupe
2 void territory(int i,int visited[81],int color)//color to definir le territoire de qui?
3 {
4     adversaire=(color==1)?2:1;//Le joueur adversaire
5     if (M[i]==0)
6     {
7         visited[i]=1;//mark it as checked
8         int h,b,g,d;
9         h=position_haut(i);
10        b=position_bas(i);
11        g=position_gauche(i);
12        d=position_droit(i);
13
14        if(h!=-1 && M[h]==0&&visited[h]==0)//position existe et non encore traite
15
16        {
17            visited[h]=1;//mark this position as checked
18            territory(h,visited,color);
19        }
20        if(h!=-1&&M[h]==adversaire)//si on rencontre un pion d'adversaire lors de parcours
21            territory_found=0;//in this case no territory is founded
22        if(b!=-1 && M[b]==0&&visited[b]==0)
23
24        {
25            visited[b]=1;//mark this position as checked
26            territory(b,visited,color);
27        }
28        if(b!=-1&&M[b]==adversaire)
29            territory_found=0;

```

```

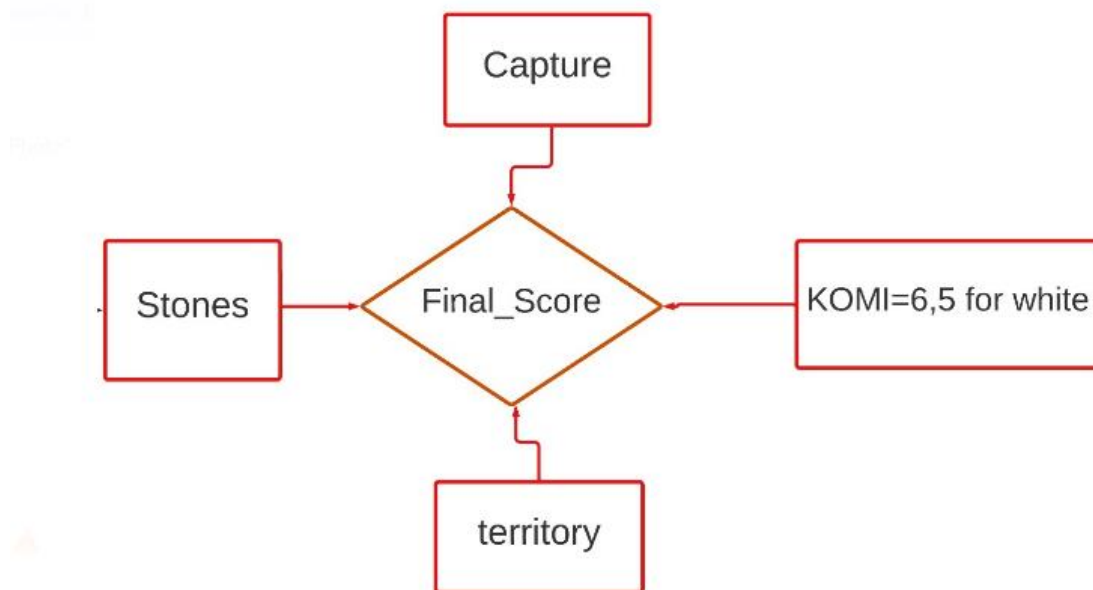
29        territory_found=0;
30        if(d!=-1 && M[d]==0&&visited[d]==0)
31
32        {
33            visited[d]=1;//mark this position as checked
34            territory(d,visited,color);
35        }
36        if(d!=-1&&M[d]==adversaire)
37            territory_found=0;
38        if(g!=-1 && M[g]==0&&visited[g]==0)
39
40        {
41            visited[g]=1;//mark this position as checked
42            territory(g,visited,color);
43        }
44        if(g!=-1&&M[g]==adversaire)
45            territory_found=0;
46    }
47    else
48        territory_found=0;
49 }

```

6-calcul du score :

D'abord le gagnant sera déterminé par la méthode des règles française, le calcul du score total sera compté par la somme des captures, des territoires et des

pierres que chaque joueur a sur le goban.



```

1 void Final_Score()
2 {
3     float sum_black=0;
4     float sum_white=0;
5     sum_black=count_captured_white+count_stones(1)+black_territory;
6     sum_white=count_captured_black+count_stones(2)+white_territory;
7     if(sum_black>sum_white)
8     {
9         color(2,0);
10        printf("\n\n\n\n\n\t\t black won with %.2f",sum_black-sum_white);
11        color(7,0);
12    }
13    else
14    {
15        color(11,0);
16        printf("\n\n\n\n\n\t\t white won with %.2f",sum_white-sum_black);
17        color(7,0);
18    }
19 }

```

7-Le joueur qui a le tour de rôle

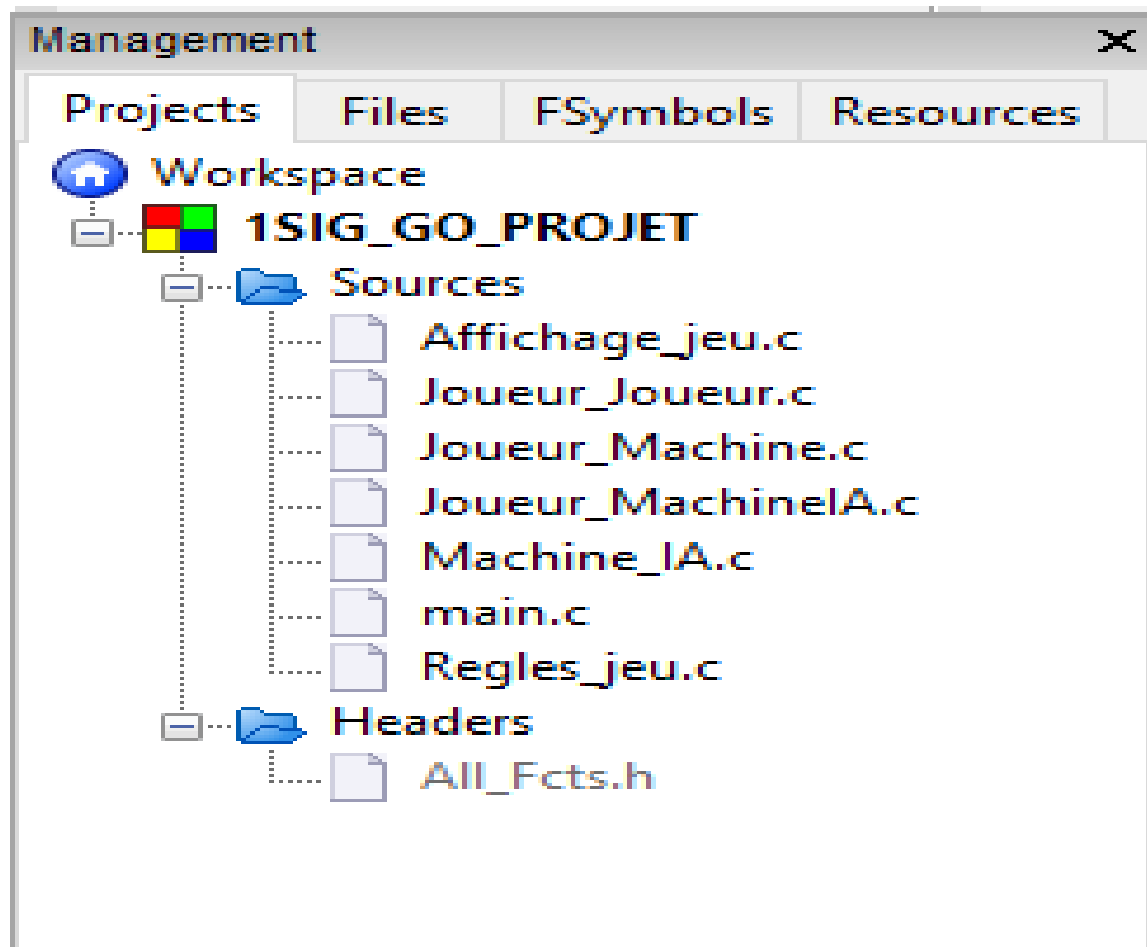
```

//organiser les tours de chaque joueur
void pass(void)
{
    player=(player==1)? 2:1;
}

```

7-Architecture du jeu

Nous avons fragmenté le code source du jeu et l'avons réparti sur des fichiers .c et .h afin de le rendre plus lisible et plus accessible. De plus, nous avons allégé la fonction principale main qui doit représenter l'interface de notre code, afin qu'elle ne soit pas trop encombrée.



8-Intelligence artificiel :

On a défini 3 fonctions par ordre de priorité suivant :
machine_capture ,machine_save,machine_surround .

S

1.Machine_capture()

```

1 int machine_capture()
2 {
3     for (int i=0; i<81; i++)
4     {
5         int checked[81]= {0};
6         int liberte[81]= {0};
7
8         if(M[i]==joueur.id)//Les pions d'adversaire
9         {
10             check_group_lib(i,M,checked,liberte);//calculer la liberte of chaque groupe du joueur dans le goban
11             if(count_liberte(liberte)==1) //si on a un groupe d'adversaire a juste un seul deg de liberte donc essaye de capturer
12             {
13                 for(int k=0; k<81; k++)
14                 {
15                     if((liberte[k]==1)&&(Suicide(k)==0))//vient a cette intersection et capturer le grp
16                     {
17                         return k;
18                     }
19                 }
20             }
21         }
22     }
23     return -1;//si on a pas de grp d'adversaire avec un seul deg de liberte
24 }

```

2.Machine_save()

```

1 int save_group()//essaye d'augmenter le degre de liberte de votre group si le group n'est pas dans les frontieres du goban
2 {
3     for (int i=0; i<81; i++)
4     {
5         int checked[81]= {0};
6         int liberte[81]= {0};
7
8         if(M[i]==machine.id)//pion de machine
9         {
10             check_group_lib(i,M,checked,liberte);
11             if(count_liberte(liberte)==2)//si le degre de liberte de chaque grp egale a 2 (eviter un deg de liberte car l'adversaire peut capturer votre grp a la fin)
12             {
13                 for(int k=0; k<81; k++)
14                 {
15                     int h,b,g,d;
16                     h=position_haut(k);
17                     b=position_bas(k);
18                     g=position_gauche(k);
19                     d=position_droit(k);
20                     if((liberte[k]==1)&&(h!=-1&&g!=-1&&b!=-1&&d!=-1)&&(minimise_deg_lib(k)==0)&&(Suicide(k)==0))//pour assurer que le grp n'est pas dans les frontieres
21                     {
22                         return k;
23                     }
24                 }
25             }
26         }
27     }
28     return -1;
29 }

```

3.Machine_surround()

```

1 int surround()//try to surround the adversaire
2 {
3     int min_liberte=82;//any value superieur a 80
4     for (int i=0; i<81; i++)//parcours le goban
5     {
6         int checked[81]= {0};
7         int liberte[81]= {0};
8         if(M[i]==joueur.id)//adversaire pion
9         {
10             check_group_lib(i,M,checked,liberte);//calcule le degre de liberte de chaque grp
11             if(min_liberte>count_liberte(liberte))//help us to knowc the grp that has the minimum of deg liberte
12             {
13                 min_liberte=count_liberte(liberte);
14             }
15         }
16     }
17     for (int i=0; i<81; i++)
18     {
19         int checked[81]= {0};
20         int liberte[81]= {0};
21
22         if(M[i]==joueur.id)
23         {
24             check_group_lib(i,M,checked,liberte);
25             if(count_liberte(liberte)==min_liberte)//choose the grp that has the minimum of degre liberte
26             {
27                 for(int k=0; k<81; k++)
28                 {
29                     if((liberte[k]==1)&&(Suicide(k)==0))//empty position(in grp that has the min deg de liberte) and not suicide
30                     {
31                         return k;
32                     }
33                 }
34             }
35         }
36     }
37     return -1;
38 }

```

Clôture :

Ce projet a été une expérience très enrichissante pour nous. Nous avons pris beaucoup de plaisir à travailler sur ce jeu de Go, surtout parce que c'est le premier vrai projet sur lequel nous avons travaillé dans notre vie. Nous avons pris cela comme un défi personnel et nous avons voulu produire quelque chose dont nous serions fiers. Nous avons fait de notre mieux pour fournir le meilleur travail possible, et nous sommes satisfaits de ce que nous avons accompli, même si nous aurions aimé faire encore mieux .

Ce projet nous a permis d'apprendre beaucoup de choses en surmontant les difficultés que nous avons rencontrées. C'est également le premier projet de programmation où nous avons utilisé une feuille de papier et un stylo pour élaborer une vraie réflexion et concevoir un algorithme. C'était un grand pas pour nous, car cela nous a permis de produire notre premier projet en dehors du cadre des exercices d'algorithmique minimaux.