
GO
Rapport
Jeu de Go
Projet de programmation en C
Élaboré par: Abchir Ouijdane
Zugari Amal
Encadré par: Mme Cherrabi

TABLE DES MATIERES

I – INTRODUCTION

- **Présentation 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. Il est un

jeu à origines mystérieuses. Une possibilité est qu'il soit né au Tibet il y a plusieurs milliers d'années, ce qui en ferait le jeu le plus ancien encore pratiqué de nos jours.

En Chine, des légendes évoquent la paternité de l'empereur Yao ou de son successeur Shun qui auraient inventé ce jeu afin d'éduquer leurs fils.

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.

• Règles du jeu

- ✓ On tire au sort pour savoir qui jouera avec les pierres noires.
- ✓ Le premier jouera avec les pierres noires.
- ✓ Les mouvements se succéderont qu'avec un tour de rôle entre les deux joueurs. Ils ne consistent qu'à ajouter de nouvelles pierres sur le goban, une seule pierre par joueur et par tour.
- ✓ Les pierres déjà déposées sur le plateau ne peuvent pas être déplacées.
- ✓ A tour de rôle, les joueurs posent une pierre de leur couleur sur une intersection inoccupée du goban ou bien ils passent (cela entraîne la fin de la partie).
- ✓ Les libertés d'une pierre sont les intersections inoccupées voisines.
- ✓ Les libertés d'une chaîne (ensemble de pierres voisines de proche en proche) sont les intersections inoccupées voisines des pierres de cette chaine.
- ✓ Lorsqu'un joueur supprime la dernière liberté d'une pierre ou une chaîne adverse, il la capture en la retirant du goban.
- ✓ Un joueur en posant une pierre, ne doit pas redonner au goban un état identique à l'un de ceux qu'il lui avait déjà donné.
- ✓ Un joueur ne doit pas poser une pierre sur une intersection n'ayant pas une liberté et entourée par les pierres adverses..
- ✓ 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 le noir. Aussi, dans une partie à égalité, le blanc reçoit en échange des points de compensation, appelés Komi : 7 points et demi.
- ✓ Il existe toujours un vainqueur. Le match nul n'existe pas.

Remarque : Il est également possible de jouer contre un adversaire plus faible en accordant un certain nombre de pierres dites de handicap qui permettent d'équilibrer la partie.

• 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, mais aussi de pouvoir jouer contre la machine.

Dans ce projet, on travaillera sur la version de **taille 9x9**. Concernant les couleurs, le premier à jouer aura les pierres noirs, alors que le deuxième les pierres blanches.

• Travail réalisé

Version Console :

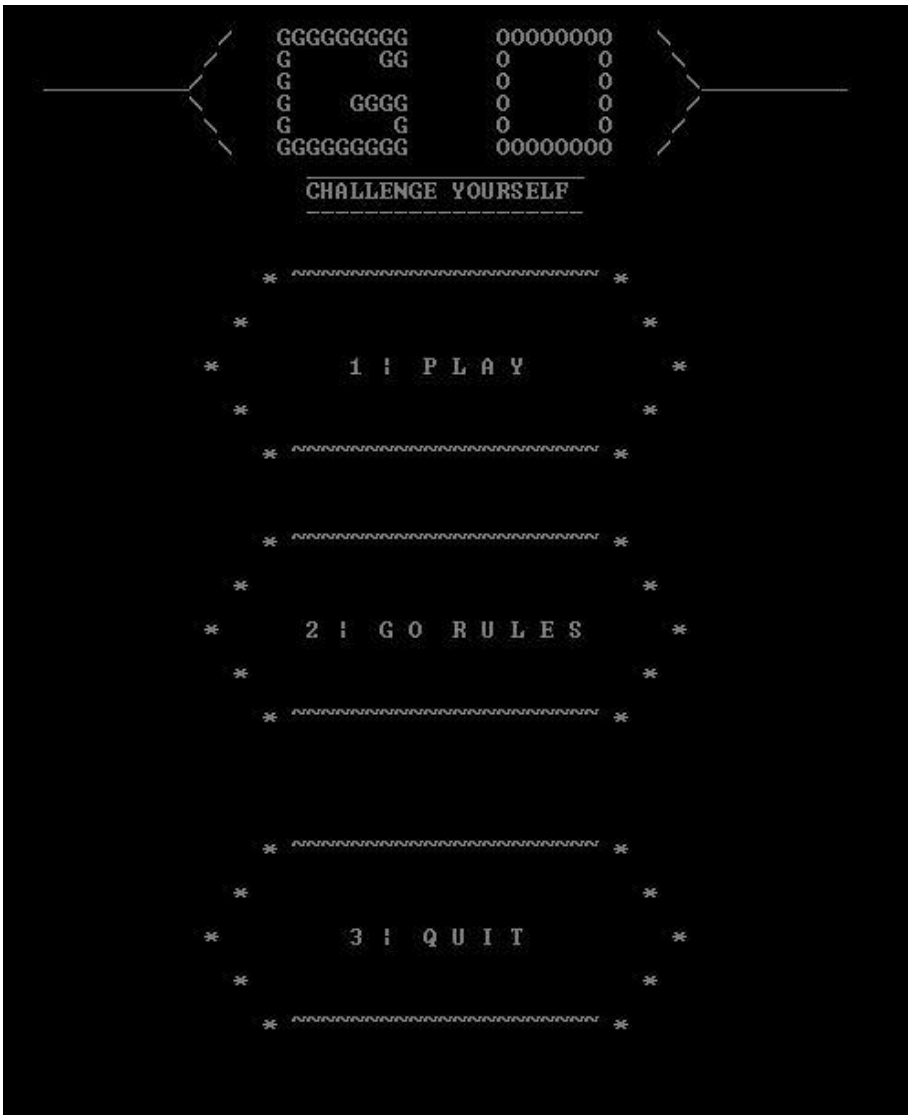
- Ecrire un programme en C automatisant les règles générales et quelques stratégies du jeu de GO par le biais de quelques implémentations simples de l'intelligence artificielles, en considérant 3 TSUME GO dans le cas d'un joueur humain pouvant jouer contre le CPU.

II – SIMULATION DU JEU

Dans cette partie, on va essayer de montrer le scénario du jeu et son fonctionnement.
On expliquera le rôle des différents menus du jeu.

• Page d'accueil

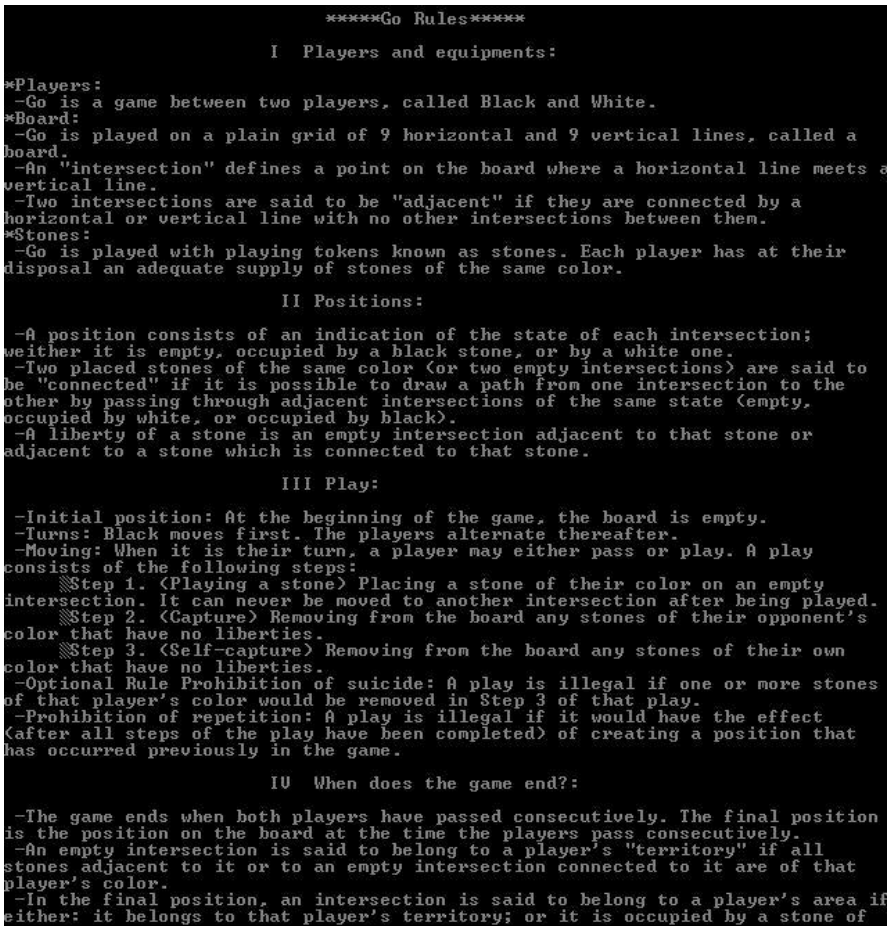
La première fois qu'on lance le jeu, voici la fenêtre qui s'affiche :



Le menu principal est constitué de 3 choix :

- **« PLAY »** permettant d'accéder au jeu principal
- **« GO RULES »** permettant d'afficher les règles de jeu (voir ci-dessous)

Simple fenêtre affichant les règles de jeu :

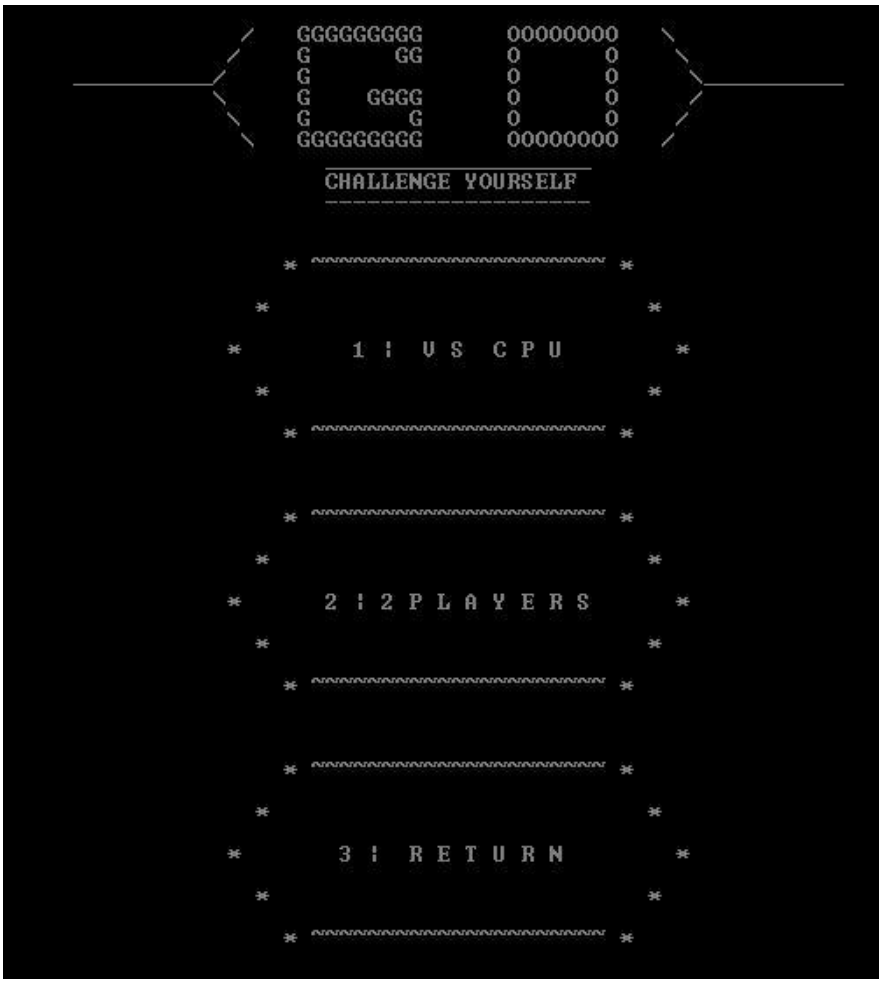


« QUIT » pour quitter le jeu.

- # MODE DE JEU

Si on clique sur « PLAY », on accède immédiatement à la page contenant les différents modes de jeu, 2 modes sont disponibles selon le cahier des charges :

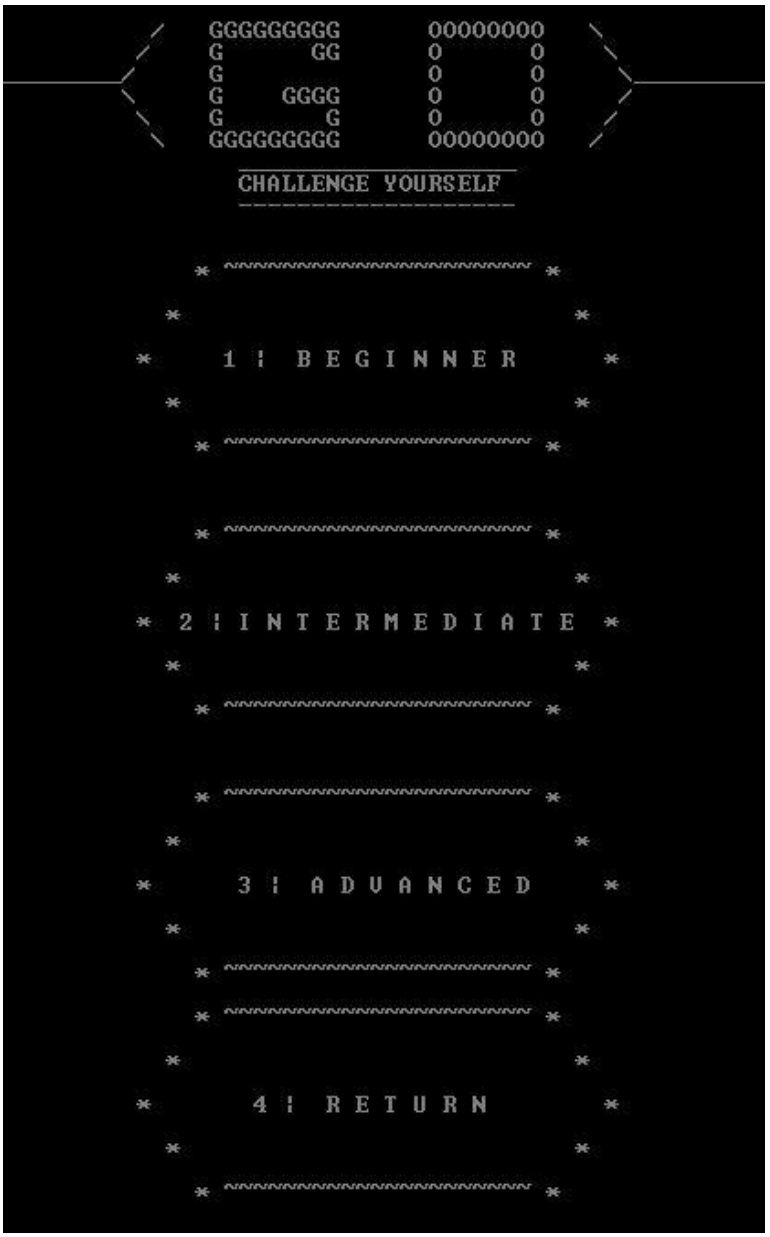
- « VS CPU » : c'est également le mode Homme vs. Machine. Le joueur Homme aura comme adversaire le CPU
- « 2 PLAYERS » : Représente le mode Homme vs. Homme. Elle permet d'avoir une partie entre 2 joueurs humains.
- « RETURN » : C'est le bouton de retour vers le menu principal / page d'accueil.



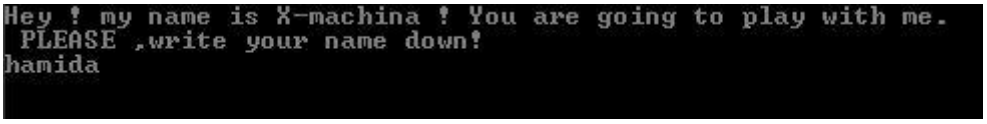
- # Niveaux de jeu

Si on a effectué le premier choix **« 1 PLAYER »** afin de jouer contre le CPU, on sera immédiatement dirigé vers les différents choix de la difficulté. 3 options sont disponibles :

- **« BEGINNER »** c'est le mode facile, la machine se contentera de jouer de manière aléatoire.
- **« INTERMEDIATE »** à ce stade-là, la machine essayera de vous empêcher de construire une ligne connectée, ainsi il n'est plus facile comme avant.
- **« ADVANCED »** Là, la difficulté augmente manifestement. C'est le mode difficile et qui implémente une intelligence simple avancée par rapport au cas précédents. La machine essayera de vous bloquer et de gagner en même temps.
- **« RETURN » C'est le bouton de retour vers le menu des modes de jeu.**



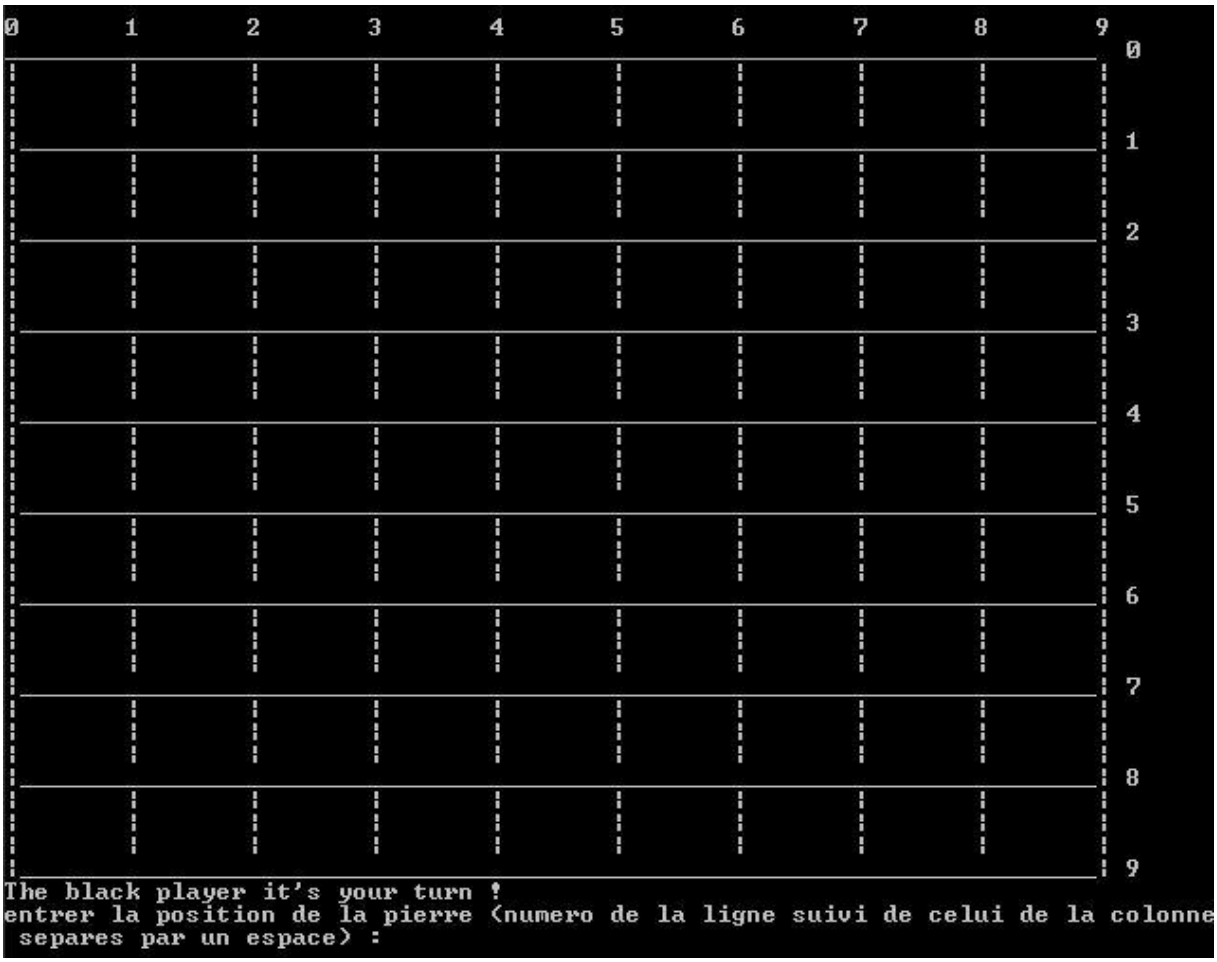
Après avoir choisi la difficulté désirée, la machine se présente et demande du joueur d'entrer son nom :



on sera dirigé vers une fonction random qui décidera le premier à jouer ; l'Homme ou la machine (Le cahier des charges insiste que le premier à jouer doit être décidé aléatoirement) .

- # LA PARTIE COMMENCE

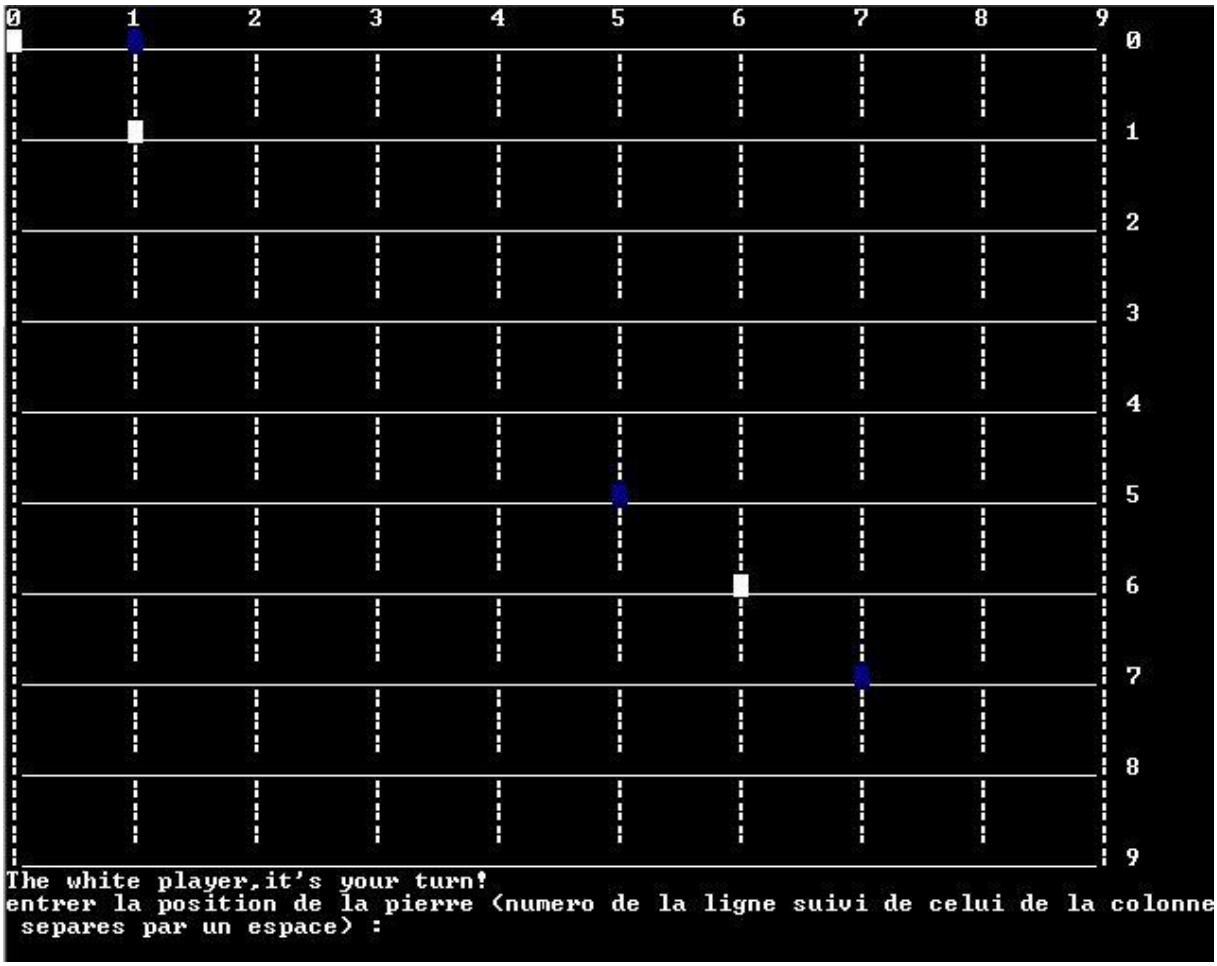
Après qu'on a choisi le mode de jeu, le niveau et celui qui aura le premier tour, on affichera finalement le goban du jeu :



C'est celui ayant les pierres noires qui a le droit de jouer le premier tour.

- En bas, on écrit toujours une phrase qui affiche à qui est le tour de jouer.

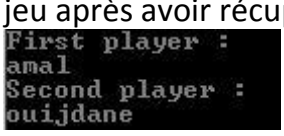
Voici un exemple d'un goban rempli avec les différentes pierres :

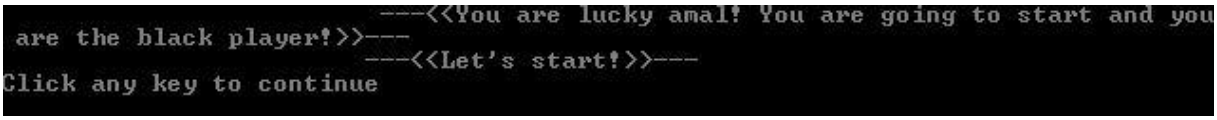


On a modélisé les pierres par des petites carrées blanches et bleues.

Enfin il ne reste qu'à féliciter le joueur gagnant (Il n'y a pas de match nul en GO).
Une fenêtre où il est écrit le nom de joueur gagnant s'affiche.

Remarque : Si on choisit le mode **« 2 PLAYERS »** on passe directement au tablier de jeu après avoir récupéré les noms des 2 joueurs et choisi le joueur noir.





III-PHASE D'ANALYSE ET DE CONCEPTION ;

Durant cette phase, on effectuera l'étude des données et l'étude des traitements à effectuer. Tout en travaillant sur leur modélisation.
On essayera de décrire les structures de données éventuelles à créer, la façon d'interaction du joueur avec le jeu, l'architecture et la cohésion des différentes entités du jeu, les programmes à écrire et la manière dont tout cela va être intégré.

- # ANALYSE DE Problématique

La formulation d'une bonne architecture pour notre jeu nécessite une analyse profonde de tous ses aspects. C'est pourquoi il est nécessaire de tester le jeu plusieurs fois, cela aide énormément à avoir une idée précise sur les multiples besoins nécessaires pour construire notre propre version de jeu.

Après avoir joué le GO sur mon smart phone nombreuses fois et en ligne sur le site recommandé par notre encadreur, sans oublier aussi la construction de son goban suivant à la demande de notre encadreur aussi. Il est devenu clair que la construction d'un tel jeu requiert la réponse à certaines questions principales :

- Comment représenter le goban du jeu et les pierres?
- Comment vérifier si l'un des deux joueurs a gagné la partie ?
- Comment pouvoir distribuer les tours entre les deux joueurs ?
- Comment construire une intelligence artificielle capable de confronter le joueur Homme ?

Il existe également d'autres questions dont la réponse vise à perfectionner la qualité logicielle du jeu :

- Quelle serait la meilleure interface qui permettra au joueur une fluidité dans l'utilisation ?
- Quelles fonctionnalités additionnelles peut-on intégrer pour améliorer l'expérience de l'utilisateur ?

• CONCEPTION&MODELISATION :

Dans cette étape, on répondra aux questions posées tout en modélisant les résultats.

✓ Représentation de plateau de jeu :

La goban de GO se compose de 81 case, et 100 intersections et chaque intersection peut représenter 3 états différents :

1. Intersection vide
2. Intersection remplie par le Joueur 1 ou CPU (pierre noire)
3. Intersection remplie par le Joueur 2 ou CPU (pierre bleue)

Nous avons représenté alors le goban par un tableau de caractères nommé tab à une seule dimension de taille 79 en colonne et 36 en ligne. Pour avoir l'état d'une case, il suffit de son numéro de ligne et de colonne comme indice et la tableau retournera l'information recherché. La modélisation a été fait comme ceci : Pour 0≤i≤80 et 0≤j≤37 :
si tab[i] [j]= ' ' ou '_' ou '|' alors la case (i, j) est vide
si tab[i] [j]= 'B' alors la case (i, j) est remplie par une pierre noire (Joueur 1)
si tab[i] [j]= 'W' 2 alors la case (i, j) est remplie par une pierre bleue (Joueur 2)

Remarque : Vu que le goban de GO ressemble à une matrice, c'est mieux de le représenter par un tableau deux dimensions.

On va regrouper alors toutes les structures

✓ La vérification si l'un des joueurs est gagnant :vérification si l'un des joueurs est gagnant :

Le camp ayant acquis le plus de territoire gagne ma partie.(les pierres capturées sont incluses dans le décompte.)
Un Brainstorming a été effectué dans cet égard, ce qui a amené à plusieurs idées, parmi les idées échouées on trouve :

1. Parcourir le tableau et compter les positions remplies par les pierres noires et blanches
   ➔ **Idéee non réussie :** les pierres capturées ne seront pas comptées et le territoire des intersections dominées ne seront pas comptées également. Alors ce n'est pas suffisant.

2. Incrémenter à chaque fois que les fonctions de capture noires et blanches capturent une pierre. Ainsi qu'une fonction qui incrémente les 2 variables s'elle trouve des intersections vides dominées par le noir ou le blanc.

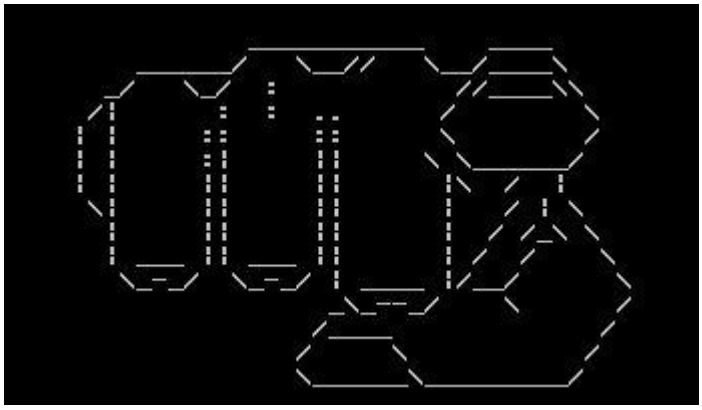
➔ **Idée réussie :**

➔ Points forts : *Ceci permet de parcourir le minimal des cases possible et donc une complexité minimale lors de vérification.*

➔ Toutes les fonctionnalités nécessaires pour la vérification du gagnant seront regroupées afin d'être utilisées dans une fonction sous le nom de gagnant.

Cela peut être assuré à travers 2 variables qu'on va nommer **cw : comptage white, cb : comptage black** qui seront de type entier. Et une fonction **gagnant** permettant de déterminer le gagnant.

On affiche cette forme accompagnée de félicitations du gagant : Congratulations,You are thewinner !



➔ Les différentes intelligences implémentées dans le jeu de 2 joueurs et avec le CPU seront intégrées dans différentes entités

✓ La meilleure interface pour naviguer le jeu facilement :

Lors de cette étape, l'importance a été accordé au respect du cahier des charges et la cohésion de jeu afin d'améliorer l'aisance d'utilisation de ses différents aspects, d'où les sections suivantes ont été décidées :

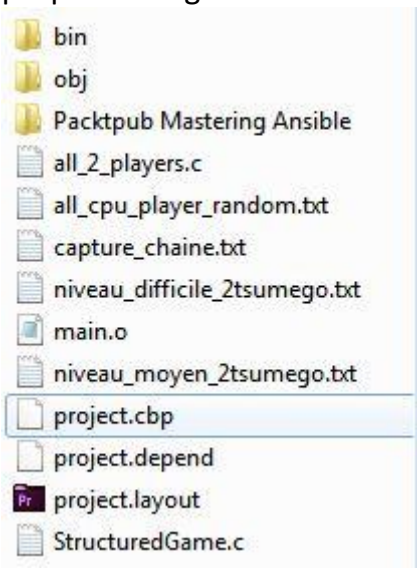
- **PLAY** : c'est ce qui permettra de jouer le jeu effectivement tout en permettant de choisir les modes/niveaux désirées par le joueur
- **GO RULES** : Si on va jamais distribuer un jeu, il faut assurer que ses règles y figurent quelque part.
- **QUIT** : l'option de quitter le jeu est un droit légitime pour les joueurs.

➔ Les différentes fenêtres présentant l'interface de navigation de jeu vont constituer l'entité IHM.

✓ Perfectionner l'expérience des joueurs :

Pour rendre le jeu harmonieux, rassurant et pour donner l'impression que le jeu est VIVANT, on exploitera les interfaces et les dialogues, pour rendre le jeu plus interactif.

En conclusion, La clarté de la problématique et son traitement nous a menés de proposer l'organisation suivante afin de répondre au besoin posé :



IV – DÉVELOPPEMENT DU JEU

Le développement et programmation de jeu s'est appuie sur l'architecture définie lors de la phase de conception, on va alors traiter chaque bloc/entité et expliquer le fonctionnement.

• ARCHITECTURE DE JEU

Voici les entités principales de cette version :

- **Comptage_territoire_blanc :**

  Prend comme argument : tableau actuel.
  Retourne : un entier qui représente le nombre total de pierres blanches présentes sur le goban.
  Intérêt : Permettra de déterminer le score du deuxième joueur, et servira par la suite pour définir le gagnant de la partie.

- **Comptage_territoire_noir :**

  Prend comme argument : tableau actuel.

Retourne : un entier qui représente le nombre total de pierres noires présentes sur le goban.
Intérêt : Permettra de déterminer le score du premier joueur, et servira par la suite pour définir le gagnant de la partie.

- **Specific_degree** :
  Prend comme argument : tableau actuel + tableau vide + coordonnées de la position (i et j).
  Retourne : 1 si la position admet au moins un degré de liberté, 0 sinon
  Intérêt : Fonction élémentaire pour la capture.

- **Capture_assure** :
  Prend comme argument : tableau actuel + tableau vide + coordonnées de la position.
  Retourne : 1 si l'on peut capturer la pierre se trouvant dans cette intersection, 0 sinon.
  Intérêt : Permettra de donner le feu vert pour la capture et évite la répétition (assurer la règle de répétition).

- **Capture_suicide**
  Prend comme argument : tableau actuel + coordonnées de la position.
  Retourne : 0 si toutes les intersections adjacentes contiennent des pierres adverses, 1 sinon, et les affecte dans des tableaux (listeb et listew).
  Intérêt : Permettra de déterminer si le joueur peut placer une pierre dans cette position sans se suicider.

- **Capture :**
  Prend comme argument : tableau actuel + coordonnées de la position.
  Retourne : Modification du tableau en supprimant une pierre, avec un compteur de pierres capturées.
  Intérêt : Permettra d'effectuer la capture d'une pierre, en vérifiant toutes les conditions nécessaires.

- **Capture_else :**
  Prend comme argument : tableau actuel + coordonnées de la position.
  Retourne : Elimine l'intersection d'entré et applique la règle de capture sur les pierres restantes du goban.

Intérêt : Permettra d'effectuer la capture d'une pierre avec exception.

- **Limitation_white :**

  Prend comme argument : tableau actuel + tableau vide.
  Retourne : La position qui permettra d'annuler le degré de liberté d'une pierre blanche pour laquelle il ne reste qu'un seul degré de liberté.
  Intérêt : Permettra au CPU (étant le joueur aux pierres noires) de positionner sa pierre dans cette position afin d'annuler les degrés de liberté de la pierre adverse et donc avoir plus de chance pour gagner.

- **Limitation_black:**

  Prend comme argument : tableau actuel + tableau vide.
  Retourne : La position qui permettra d'annuler le degré de liberté d'une pierre noire pour laquelle il ne reste qu'un seul degré de liberté.
  Intérêt : Permettra au CPU (étant le joueur aux pierres blanches) de positionner sa pierre dans cette position afin d'annuler les degrés de liberté de la pierre adverse et donc avoir plus de chance pour gagner.

- **Forbidden_black:**

  Prend comme argument : tableau actuel + tableau vide + coordonnées de la position.
  Retourne : 1 si l'affectation d'une pierre noire dans cette position entrainera une capture de celle-ci, 0 sinon.
  Intérêt : Permettra au CPU (étant le joueur aux pierres noires) d'éviter la capture de ses pierres dans certains cas (si la position n'offre qu'un seul degré de liberté, et que les pierres adjacentes sont toutes adverses).

- **Forbidden_white :**

  Prend comme argument : tableau actuel + tableau vide + coordonnées de la position.
  Retourne : 1 si l'affectation d'une pierre blanche dans cette position entrainera une capture de celle-ci, 0 sinon.
  Intérêt : Permettra au CPU (étant le joueur aux pierres blanches) d'éviter la capture de ses pierres dans certains cas (si la position n'offre qu'un seul degré de liberté, et que les pierres adjacentes sont toutes adverses).

- **Capture_avoid_black:**

Prend comme argument : tableau actuel + tableau vide.
Retourne : la position qui permettra d'éviter la capture d'une pierre noire n'ayant plus qu'un degré de liberté (si on y dépose une pierre blanche, la noire sera capturée).
Intérêt : Permettra au CPU (étant le joueur aux pierres noires) d'éviter la capture de ses pierres en y déposant une pierre noire par la suite.

- **Capture_avoid_white:**

  Prend comme argument : tableau actuel + tableau vide.
  Retourne : la position qui permettra d'éviter la capture d'une pierre blanche n'ayant plus qu'un degré de liberté (si on y dépose une pierre noire, la blanche sera capturée).
  Intérêt : Permettra au CPU (étant le joueur aux pierres blanches) d'éviter la capture de ses pierres en y déposant une pierre blanche par la suite.

- **Create_suicide_black:**

  Prend comme argument : tableau actuel.
  Retourne : Une position vide qui n'admet qu'un degré de liberté et qui est entourée de pierres noires uniquement.
  Intérêt : Permettra au CPU (étant le joueur aux pierres noires) d'annuler les degrés de liberté de la position en l'entourant de pierres noires, et donc créer une position de suicide pour l'adversaire.

- **Create_suicide_white:**

  Prend comme argument : tableau actuel.
  Retourne : Une position vide qui n'admet qu'un degré de liberté et qui est entourée de pierres blanches uniquement.
  Intérêt : Permettra au CPU (étant le joueur aux pierres blanches) d'annuler les degrés de liberté de la position en l'entourant de pierres blanches, et donc créer une position de suicide pour l'adversaire.

- **Dominance territoire:**

  Prend comme argument : tableau actuel.
  Retourne : Modifie les compteurs (score).
  Intérêt : Permettra de considérer les espaces de suicide pour l'un en incrémentant le score de l'autre (une position de suicide pour les noires ->un point de plus pour les blanches, et vice versa).

- **Capture_chaine:**

  Prend comme argument : tableau actuel + tableau vide + position.
  Retourne : 0 si présence d'une capture chaine, 1 sinon.
  Intérêt : Permettra de donner le feu vert à la capture d'une chaine.

- **Kill_B:**

  Prend comme argument : tableau actuel + tableau vide + position.
  Retourne : Suppression des pierres capturées (cas de capture de pierres noires), avec compteur (incrémentant le score du joueur aux pierres blanches).
  Intérêt : Effectuer la capture et modifier les scores.

- **Kill_W:**

  Prend comme argument : tableau actuel + tableau vide + position.
  Retourne : Suppression des pierres capturées (cas de capture de pierres blanches), avec compteur (incrémentant le score du joueur aux pierres noires).
  Intérêt : Effectuer la capture et modifier les scores.

- **Capture_complete_chaine:**

  Prend comme argument : tableau actuel + tableau vide + position.
  Retourne : Effectue la capture en vérifiant les conditions nécessaires, avec compteur (incrémentant le score du joueur adverse).
  Intérêt : Effectuer la capture et modifier les scores en vérifiant toutes les conditions nécessaires, et en modifiant les scores.

- **Gagnant:**

  Prend comme argument : Rien
  Retourne : Affiche le gagnant.
  Intérêt : Permettra d'énoncer le gagnant en fin de partie.

- # OUTILS EMPLOYES :

C'est l'équivalent de la SDL sur la console. Elle permet plusieurs fonctions intéressantes permettant de rendre le jeu plaisant à naviguer et voir.

➔ MANIPULATION DES COULEURS

```
void color(int t,int f)
{
HANDLE H=GetStdHandle(STD_OUTPUT_HANDLE);
SetConsoleTextAttribute(H,f*16+t);
}
```

Cette fonction permet d'écrire avec la **couleur t** sur un arrière-plan de **couleur de f,** voici la palette des couleurs Possibles (voir tableau à droite).

Colour codes available:

Code (Hex)	Color
0	black
1	blue
2	green
3	cyan
4	red
5	magenta
6	yellow/brown
7	white
8	gray
9	bright blue
A	bright green
B	bright cyan
C	bright red
D	bright magenta
E	bright yellow
F	white

➔ Nettoyage de l'ecran

L'appel de la fonction suivante permet de nettoyer l'écran :

```
void clear()
{
    system("cls");
}
```

✓ ASCII Art :

Ascii Art est l'ensemble des dessins qu'on peut réaliser avec ASCII

Exemple : Le logo du jeu de GO.

V– LES PROBLÈMES RENCONTRÉS

- **Au niveau du code / programmation:**

  - Nous n'étions pas familiarisés avec le langage C, étant donné que c'est notre première année de manipulation.
  - Difficultés à localiser des erreurs commises, surtout en ce qui concerne les fonctions imbriquées.
  - L'abondance d'idées, et difficulté de choisir ainsi que de poursuivre une même idée jusqu'au bout.
  - Notre appréhension du jeu demeure de niveau amateur, étant donné que nous ne sommes pas suffisamment expérimentés pour bien exploiter sa complexité; cela a rendu l'implémentation des tsume-go encore plus compliquée.
  - Le temps étant réduit, on a plus de pression, ce qui affecte notre rendu.
  - La variété des cas, et donc l'obligation de traiter chaque cas n'a pas été chose facile, et a pris plus de temps (Même si ton code pourrait paraitre bien implémenté il se peut qu'il ne marche pas pour un cas donné).
  - La vérification de certains cas pour s'assurer que le programme est bien implémenté nous prend aussi beaucoup de temps.
  - Grande difficulté à implémenter la capture chaine.

- **Au niveau du design / esthétique**

  - ✓ Création des graphiques (SDL):

Nous accordons trop d'importance au côté esthétique jusqu'à ce que le résultat devienne satisfaisant. Cela nous coûte énormément de temps.

Prenons l'exemple du goban du jeu :

C'est illisible, difficile à manipuler et douloureux pour les yeux.

VI– CONCLUSION

Ce travail a été très enrichissant que ce soit au niveau de la programmation et le codage, la gestion du temps et du stress, mais aussi au niveau du travail en groupe, et la coordination entre nous.

Il est à noter que malgré toutes les difficultés que nous avons pu rencontrer, nous n'avions pas baissé les bras, et avions défié ce que nous paraissait impossible et inatteignable au début.

Il est à ajouter que cela a joué aussi sur notre mental, nous sommes passé par des hauts et des bas ; quand tu passes énormément de temps sur une fonction sans résultats, ou quand après plusieurs tentatives ton programme marche !

Pour finir, le travail en groupe était agréable et nous a permis de nous soutenir l'une l'autre, nous sommes aussi fières d'avoir eu l'occasion de réaliser un tel projet, sans lequel, nous n'aurions pas pu assimiler un bon nombre de notions.

Remerciement : ***nous nous permettons d'adresser nos sincères remerciements à notre encadrante :*** **Mme Cherrabi**, ***pour ses directives et ses conseils précieux, et nous profitons d'exprimer notre reconnaissance pour la bienveillance au bon déroulement de la formation en C de Mme cherrabi qui nous aidé à réaliser ce projet.***