Techniques De Programmation

Rapport de projet de jeu : GO

AJALE Saâd

03/09/2023

Ce rapport résume les différentes étapes de la réalisation de notre projet, présente les contraintes rencontrées, et explique notre conception ainsi que les différentes stratégies adoptées.

SOMMAIRE

I. Introduction

Ce projet a été réalisé dans le cadre du module technique de programmation, sous la supervision du professeur Mme M. Cherrabi, à qui nous exprimons notre gratitude pour nous avoir offert cette opportunité de travailler sur un projet concret qui portait cette année sur le jeu de GO.

Selon les spécifications du cahier des charges, notre tâche consistait à programmer le jeu de Go en utilisant le langage C et à créer par la suite une version console.

Au cours des jours de travail consacrés à ce projet, nous avons rencontré des difficultés que nous avons réussi à vaincre en prenant des décisions stratégiques, ce qui nous a finalement conduit à ce résultat.

Dans ce rapport, nous présentons toutes les difficultés rencontrées et expliquons comment nous les avons dépassées. Nous justifions les choix stratégiques que nous avons adoptés, en détaillant les limitations et les avantages de chaque proposition retenue. Nous expliquons également les raisons qui nous ont poussés à faire ces choix et les critères qui ont activé notre décision.

II. A propos du jeu

1-Présentation du jeu :

Le jeu de Go est un jeu de plateau d'origine chinoise qui oppose deux joueurs. Ils placent alternativement des pierres noires et blanches sur les intersections d'un plateau quadrillé appelé goban. L'objectif du jeu est de contrôler le terrain en construisant des "terrains" à travers les pierres posées. Les pierres qui se retrouvent encerclées deviennent des "prisonniers". Le joueur qui parvient à obtenir le plus de territoires et de prisonniers remporte la partie.

2-Règles du jeu

- Chaînes et libertés

Les pierres de même couleur qui se trouvent directement adjacentes en suivant les lignes de la grille sont considérées comme connectées et forment une chaîne. Chaque intersection vide voisine à une chaîne de pierres est appelée une "liberté".

- Capture

Une pierre isolée ou, plus généralement, une chaîne de pierres qui ne possède plus qu'une seule liberté est en situation d'Atari. Si cette chaîne perd sa dernière liberté, elle est capturée. La chaîne complète est retirée du goban et placée dans le tas de pierres capturées du joueur adverse.

- Coups interdits
  - Ko

    Pour éviter qu'une situation ne se répète à l'infini, la règle du ko (un mot japonais signifiant éternité) interdit de jouer un coup qui ramènerait le jeu dans une position déjà vue dans le courant de la partie.

- Fin de partie

Si aucun des joueurs n'a abandonné, la partie se termine après que les deux joueurs ont passé consécutivement, c'est-à-dire qu'ils décident de ne pas jouer de pierre à leur tour. À ce moment-là, on procède au décompte des points de chaque joueur. Celui qui possède le plus de points remporte la partie.

- Territoire

Un territoire est défini comme un ensemble d'une ou plusieurs intersections vides contiguës, délimitées par des pierres de la même couleur. Ces intersections inoccupées sont adjacentes les unes aux autres de manière continue

- Décompte final

La partie peut se terminer de deux manières :
- si l'un des joueurs abandonne, auquel cas il est considéré comme perdant.
- si les deux joueurs passent consécutivement.

Dans ce dernier cas, on procède au décompte des points. Chaque intersection du territoire d'un joueur lui rapporte un point, ainsi que chacune de ses pierres encore présentes sur le goban.

De plus, il est important de noter que commencer la partie en tant que joueur Noir offre un avantage.

Le gagnant de la partie est celui qui accumule le plus de points à l'émission du décompte.

III. A propos du projet

1- Travail demandé (cahier de charge dans l'annexe)

Le projet consiste à développer un jeu de Go en utilisant le langage C. Le jeu doit permettre à deux joueurs humains de s'affronter, ainsi que de jouer contre l'ordinateur.
La partie contre l'ordinateur comportera deux sous-parties. L'une d'entre elles sera dédiée à une intelligence artificielle capable de jouer contre le joueur humain, tandis que l'autre générera des mouvements aléatoires implémentés en langage C.
Le travail final devra respecter les règles conventionnelles du jeu de Go, notamment en ce qui concerne la capture des pierres, la gestion des territoires et les coups interdits.

2- Les gros travaux réalisés dans le projet :

Dans notre travail nous avons réalisé :

- ✓ la version console du jeu.
- ✓ Traitement d'une base de donnée sous forme de fichier.
- ✓ le fonctionnement de la capture des pierres n'ayant aucune 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 des joueurs.
- ✓ le fonctionnement nécessaire pour interdire tous les coups interdits comme le suicide et le KO.
- ✓ Interface interface homme-machine (IHM).
- ✓ Faire en sorte que le jeu produit soit intègre et respect le fonctionnement naturelle du jeu GO et ses règles conventionnels.
- ✓ L'intelligence artificielle

3- Contraintes et problèmes générales rencontrés dans le projet :

Par ordre de complexité on va citer les grosses contraintes rencontrées.

- **contrainte de temps :** Malgré la conscience du temps limité dès la réception du cahier des charges, et malgré que nous avons immédiatement commencé à travailler. Nous ayons été confrontés à cette contrainte, cependant nous avons fait de notre mieux pour accomplir le plus de travail possible.

- **contrainte d'optimisation du jeu :** Cette contrainte a demandé beaucoup de temps, en particulier lors du traitement de la partie base de données. En effet, dans un jeu de Go, il y a généralement de nombreux traitements à effectuer après chaque coup, tels que l'élimination des coups interdits, la suppression des pierres sollicitées et le calcul des territoires. Cela nous a incités à atteindre des efforts supplémentaires pour optimiser le produit final et le rendre plus léger.

4- Outils et supports utilisés

• La bibliothèque <windows.h> :

Cette bibliothèque contient des fonctions très intéressantes qui permet d'améliorer le graphique de la console et avoir un jeu console qui est mieux décoré et plaisant à

l'oie. Voici quelques fonctionnalités que permet les fonctions que contient cette bibliothèque :

✓ Changer les couleurs du text et d'arrière plan :

```
void color(int t)
{
    WORD wColor;
    HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    if (GetConsoleScreenBufferInfo(hStdOut, &csbi))
    {
        wColor = (csbi.wAttributes & 0xF0) + (ForgC & 0x0F);
        SetConsoleTextAttribute(hStdOut, wColor);
    }}
```

Cette fonction qu'on a construit à l'aide de quelques fonctions que contient la cette bibliothèque permet de changer la couleur du text. Les couleurs sont représentés par des entiers la figure suivante montre l'entier équivalent à chaque couleur.

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

✓ Nettoyer la console :

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

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

IV. Conception du jeu :

On met en place un système de comptes, ce qui signifie que les utilisateurs sont invités à se connecter à leur compte s'ils en possèdent un. Dans le cas contraire, un compte est créé automatiquement pour eux. Les informations d'identification sont enregistrées dans une base de données sous forme de fichiers. Chaque joueur possède ses propres données liées au jeu, notamment le nombre de parties gagnées, perdues et jouées, ainsi que son rang. Ce rang est évalué en fonction du ratio de parties gagnées par rapport aux parties jouées, et il se divise en quatre catégories : A, B, C et D. Ce processus vise à repérer les joueurs en situation d'handicap et à leur fournir un soutien adapté. Si l'un des deux joueurs impliqués dans une partie à un rang inférieur d'au moins deux niveaux par rapport à l'autre joueur, il est considéré comme ayant un handicap.

Nous avons travaillé sur trois parties majeures :

1- Extraire les informations stockées sur notre base de données concernant le niveau des joueurs.
2- Déterminer le type de la partie (partie handicap).
3- Modifier les informations conservées sur la base de données après avoir annoncé le vainqueur et le perdant.

Les trois parties seront plus détailler sur la partie Stockage de données.

V. Navigation dans le jeu.

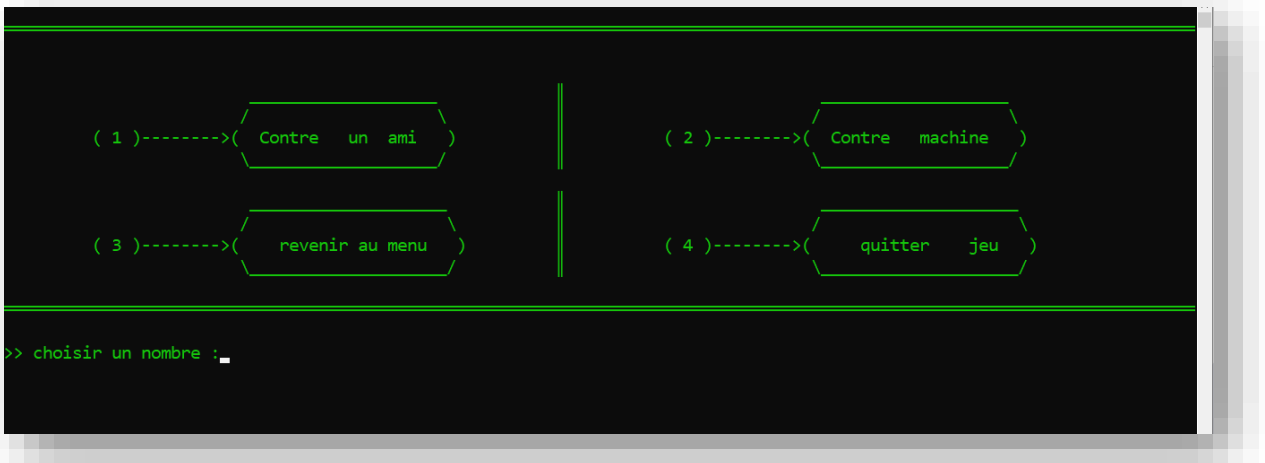
On va naviguer maintenant dans les différents menus du jeu pour découvrir son ergonomie et son fonctionnement.

- Page d'accueil

La page d'accueil offre 4 choix :

- ✓ **Jouer une partie** : permet d'accéder aux modes du jeu
- ✓ **Savoir sur « GO »** : permet une consultation des règles principales du jeu
- ✓ **Consulter vos donnes**: Facilite l'accès à des informations personnelles de chaque joueur, notamment le nombre total de parties jouées, gagnées, perdues, ainsi que leur rang dans le jeu.
- ✓ **Quitter le jeu** : pour fermer le jeu définitivement.

- Les modes du jeu :

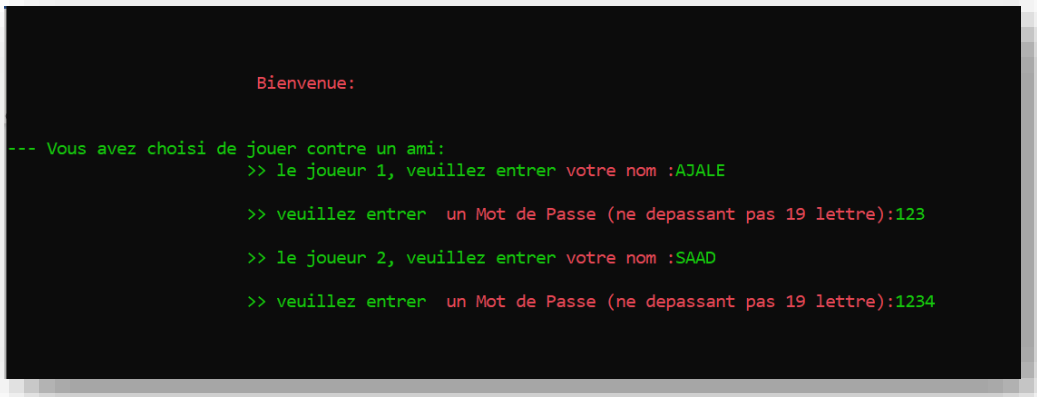


Lorsqu'on click sur **Jouer une partie** dans le menu d'accueil on accède directement aux modes du jeu, on a réalisé deux modes comme exigé par le cahier de charges.

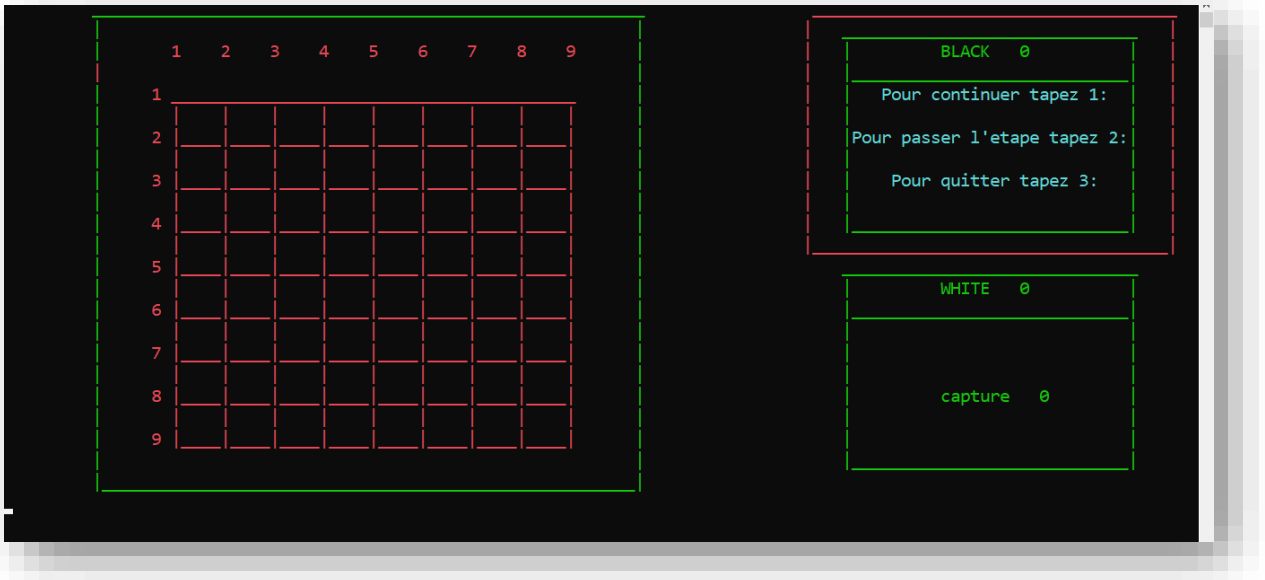
- ✓ **Mode 2players :** ici pour donner la main à deux joueurs humains pour jouer.
- ✓ **Mode 1player :** pour donner la main à un seul joueur pou qu'il puisse jouer contre le CPU.
- ✓ **Revenir au menu :** pour retourner au menu principal .
- ✓ **Quitter le jeu** : pour fermer le jeu définitivement.

- Joueurs humain vs humain (2 players) :

Lorsqu'on appui sur le bouton 2players de menu modes du jeu les joueurs sont invités à se connecter à leur compte .



Ensuite on accède directement au goban et la partie commence directement.



- Les composantes du goban :

- ✓ **Passer le role :** qui indique dans le bas 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.

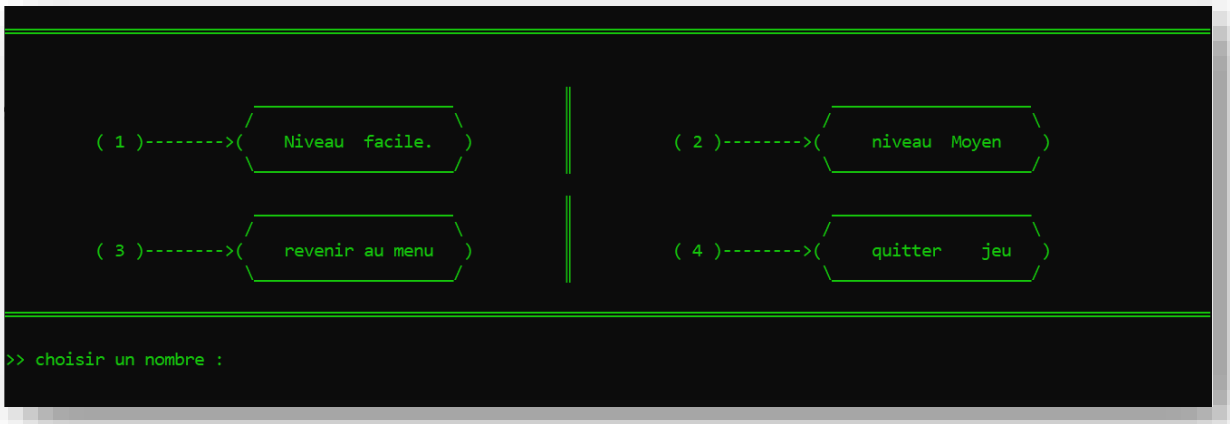
- ✓ **continuer :** qui donne la possibilité de positionner un jeton.

- ✓ **Quitter** : pour abandonner la partie.

- ✓ **Les informations sur chaque joueur :** on a les informations concernant le joueur noir du coté nord droit du goban, les informations sur le joueur blanc du coté sud droit du goban.

- ✓ **Capture :** cette rubrique indique le nombre de pierres capturés par chaque joueur

- Joueur humain vs machine



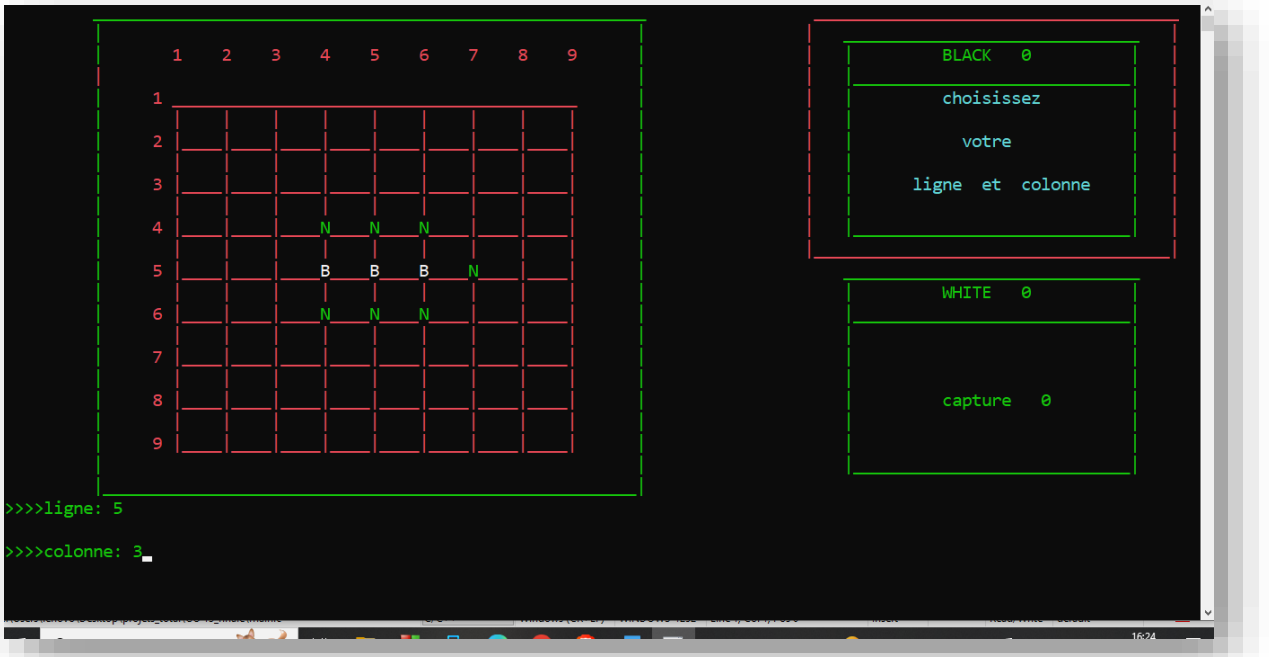
Quand on choisit sur **jouer contre la machine** on obtient directement les différents niveaux de difficultés pour le jouer contre la machine.

- Sur le GOBAN :

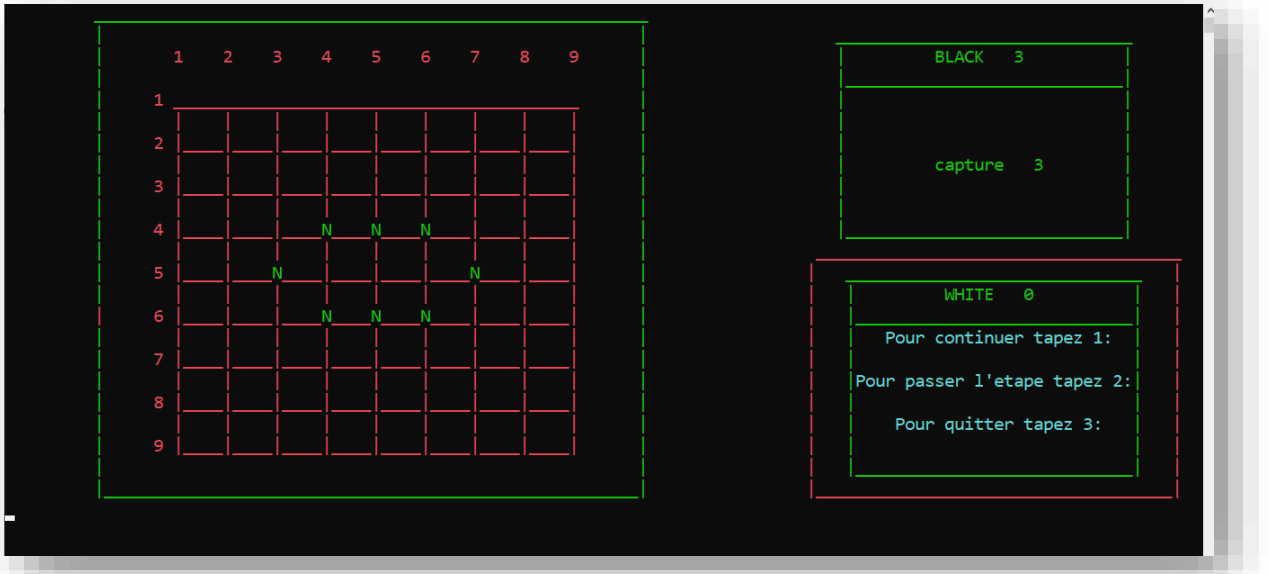
Une fois l'utilisateur sur le goban, entrain de jouer, le programme suit les coups produits par les utilisateurs (ou l'utilisateur et le cpu éventuellement) et puis produit les actions nécessaires tel que la capture est les coups interdits ainsi que les territoires occupées.
Maintenant on va explorer des exemples de situations dans lesquels le programmes produit les réactions nécessaires suivent les coup de l'utilisateur.

- Capture

Prenons par exemple cette situation



Lorsqu'on place une pierre noire sur l'intersection (5,3), le groupe des pierres blanches doivent être capturés et par la suite ils doivent disparaitre du goban.



C'est effectivement le cas, et c'est ce qui ce passé effectivement dans le programme, la figure sur dessus illustre le cas.

✓ **Remarque :** la capturation des pieres individuelles va de la même façon que la capturation des groupes de pierres. Dans le code qu'on a implémenté une pierre individuelle est considéré un cas particulier du groupe de pierres, en tt cas une pierre ou groupe de pierres n'ayant pas de degrés de libertés serons capturés.

- Les coups interdits :

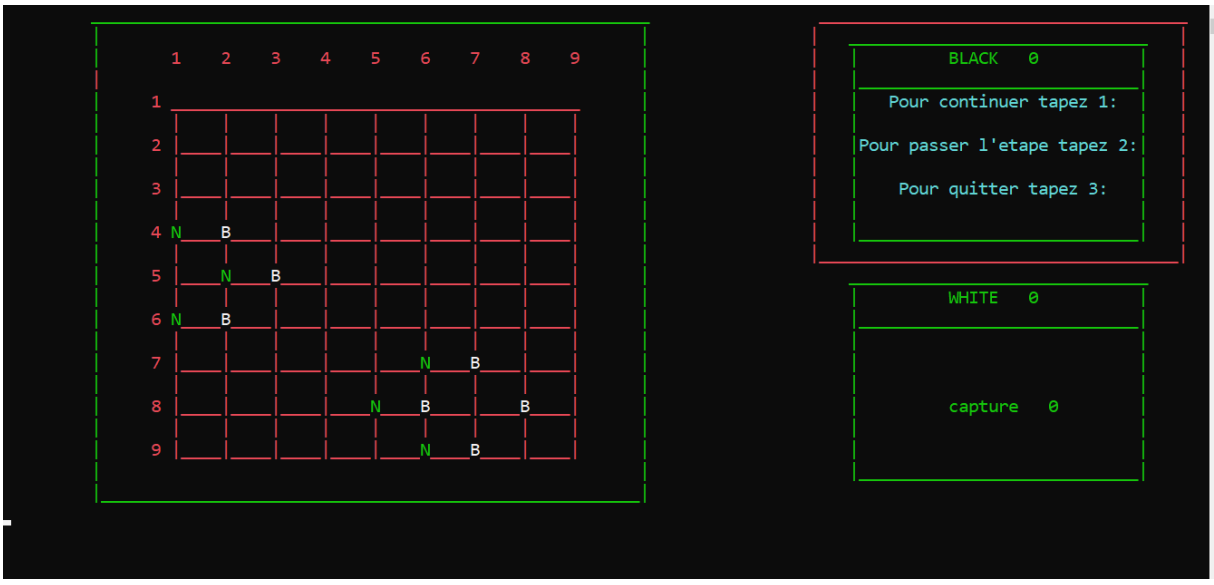
  - suicide :

Une pierre ne peut pas être posé sur une intersection si il aura une degré de liberté nulle. Nous avons respecté cette règle dans notre jeu, effectivement lorsqu'on essaie de poser une pierre sur une intersection sur lequel elle aura une degré de liberté nulle il nous empêche.

Un coup suicidaire peut facilement être identifié, il se produit lorsqu'on pose une pierre avec un groupe de pierres de telle façon à ce que ce groupe de pierres ait une degré de liberté nulle. Notre programme empêche ce cas de se produire (pace que sinon ce sera une limitation pour notre 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 quelques cas de figures des coups 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.

VI. Programmation des règles du jeu

1. Les règles générales :

- Test de position de jeton :

  Avant de poser un jeton dans un position on doit vérifier que cette position est convenable selon deux critères :
  ➢ Elle ne doit pas dépasser le nombre de lignes et de colonnes qu'on a sur le tableau.
  ➢ La position choisit ne doit pas être déjà réservé par une autre pierre.
  Ceci est présenté par la fonction : « test_de_position_de_jeton »

- Test de la validité des entrées:

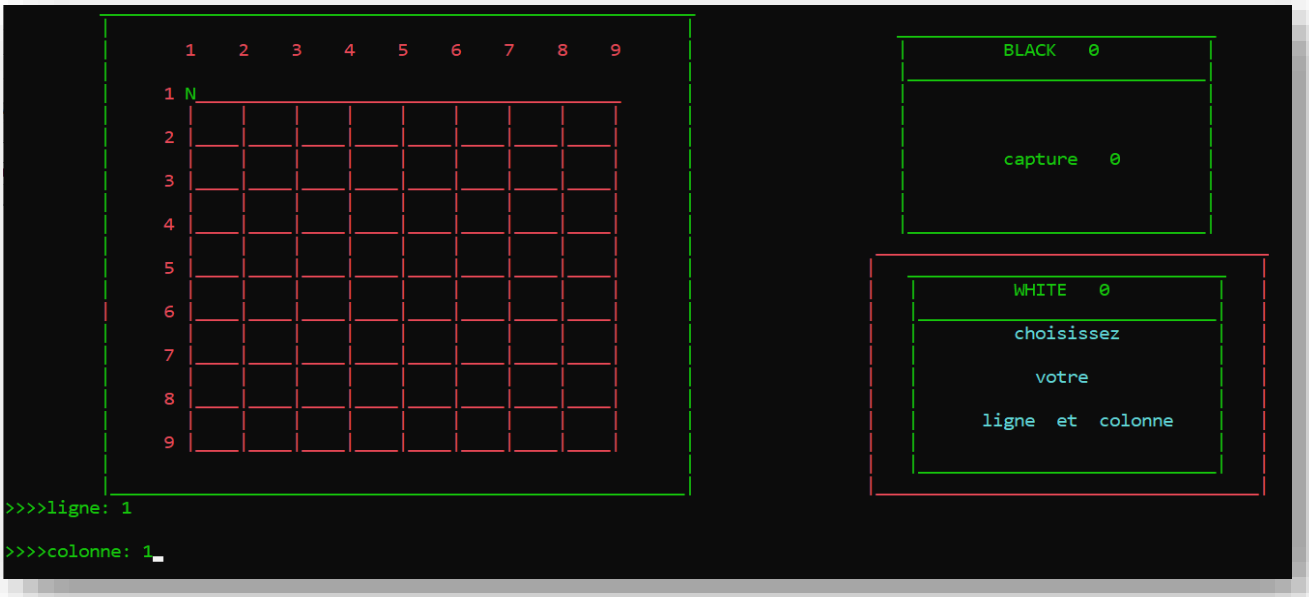
  Les entrées sont enregistrées sous forme de chaîne de caractères et sont ensuite testées pour déterminer si elles consistent en un chiffre décimal ou non.

- Test de fin:

  La fonction Testfin(char tab[][]) si dessus permet de tester il y a une intersection vacante dans la table.

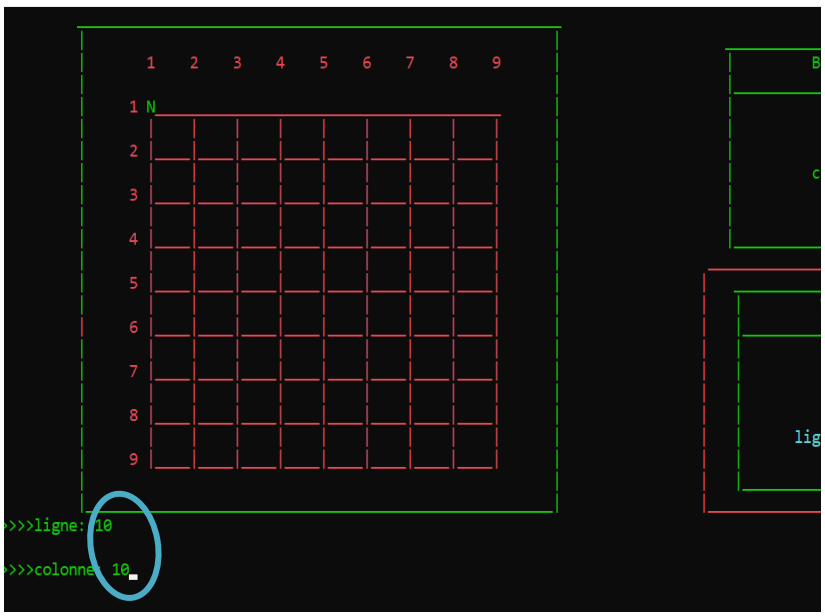
On peut voir ceci aussi lors de l'exécution de notre code :

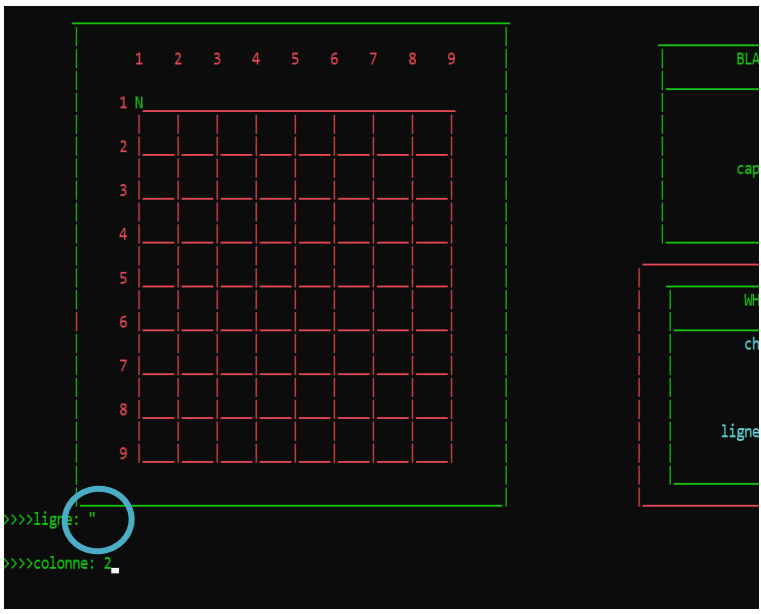
Si l'utilisateur a saisi la valeur d'une intersection qui est déjà remplie.

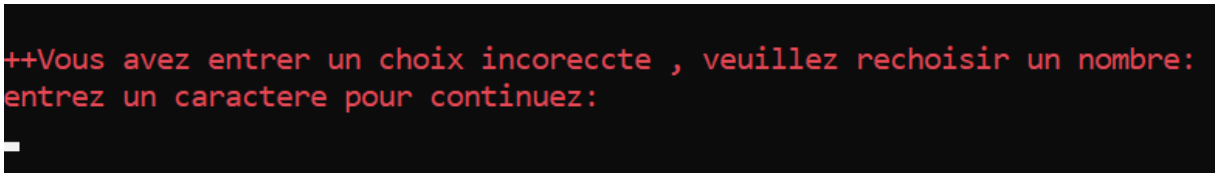


Un message d'erreur s'est affiché :

De même s'il a choisi une position hors goban ou il a entré un caractère non décimal







- Programmation de la capture des pierres :

La capture est dans le jeu go subie des règles très particulière on a déjà expliqué comment capturer une pierre dans la partie règles du jeu, reste à voir comment on a implémenté une fonction qui s'occupera de cette tâche.

Choisir une stratégie pour programmer la capture des pierres

Dans le début nous étions amenés à choisir la stratégie suivante :

- programmer une fonction qui calcul les degrés de liberté de chaque pierre et chaque groupe de pierres, sur la base de cette fonction sera basé une autre fonction qui va servir à capturer les pierres dont les degrés de libertés sont nulles.

Nous avons jugé judicieux de choisir cette stratégie pour la raison suivantes :

- On aura besoins des degrés de libertés des pierres et des groupes de pierres lorsqu'on a abordé la parie concernant l'intelligence artificiel, cette AI qu'on a eu l'intention de programmer se basera certainement sur les degrés de libertés pour choisir où positionner.

- **Conclusion :** pour aborder la capture des pierres on va programmer une fonction qui va calculer les degrés de libertés des pierres et des groupes de pierres, sur la base de cette fonction sera élaborée une autre fonction qui fera la capture.

Maintenant qu'on a opté pour une stratégie globale, venons découvrir les détails, dans la suite on va répondre à la question suivante, comment on va programmer la fonction qui s'occupera du calcul des degrés de libertés.

Opter pour une stratégie pour calculer les degrés de libertés

Étant donné que le principe de cette fonction sera réutilisé plusieurs fois, nous avons considéré qu'il était préférable de créer une fonction générale **'int parcoursfinal'** qui englobe toutes les situations et qui nous servira dans la règle de capture, calcul de territoire voire même la partie intelligence artificielle.

- ## Fonction récursive

La récursivité apporte souvent des solutions très optimales à des problématiques très compliqués, c'est pour cette raison qu'on a pensé d'utiliser une fonction récursive pour résoudre notre situation.

La fonction récursive **'int parcoursfinal'** prend en compte deux modes, 0 et 1, et effectue une traversée récursive d'un objet à parcourir (soit une intersection en mode 0, soit une pierre noire ou blanche en mode 1). Les positions de l'objet parcouru sont enregistrées dans un tableau, faisant partie de cette chaîne d'objets, tandis que les positions des objets bloquants (intersections en mode 0 ou pierres en mode 1) sont enregistrées dans un autre tableau distinct. À chaque itération de la traversée de 'l'objet à parcourir', celui-ci est ajouté à une liste noire (blacklist) pour éviter une deuxième traversée, prévenant ainsi les boucles infinies. Cette fonction revient 1 si l'objet à parcourir a un degré de liberté nul et 0 dans le cas contraire.

Après avoir calculé tous les degrés de liberté à l'aide de la fonction 'parcoursfinal', la prochaine étape consiste à choisir l'objet à parcourir ainsi que le mode approprié en utilisant la fonction 'parcoursB'. Ensuite, il s'agit de réaliser les captures nécessaires en utilisant la fonction ATARI_GO(). Cette fonction parcourt le goban, élimine toutes les pierres qui n'ont plus de degrés de liberté et ajoute les pierres capturées au score du joueur gagnant. Elle commence par prendre en compte la valeur retournée par la fonction précédente 'parcoursB' et fait également appel à la fonction 'supprimer_jeton'. (Voir script à l'annexe)

- Calcul de territoire :

Le calcul de territoire est basé principalement sur la fonction générale **'int parcoursfinal',** Il suffit de choisir le mode 1 comme argument pour la fonction. Cette fonction est conçue pour modifier l'objet à parcourir ainsi que la condition d'arrêt en fonction du mode correspondant. Un paramètre appelé "jeton" est ajouté pour identifier le joueur dont le territoire doit être calculé. Il renvoie 1 si le territoire est effectivement le sien et 0 s'il s'agit d'un territoire neutre.

Dans un autre goban temporaire, le territoire de chaque joueur est rempli avec ses propres pierres. Ensuite, la somme totale de toutes les pierres présentes sur le goban est calculée pour chaque joueur. Ces sommes sont ensuite ajoutées aux pierres capturées pour obtenir son score final. Cette procédure est répétée pour le deuxième joueur. (Voir script à l'annexe)

2. Traiter un cas particulier du jeu GO – le coup KO :

On a déjà expliqué le coup interdit KO dans la partie navigation dans le projet, maintenant on va juste expliqué comment on a traité ce cas.

On construit d'abord une matrice de même taille que notre goban 9 lignes et 9 colonnes et on l'initialise avec des zéros et qui va stocker des 1 sur les positions interdites (ko-suicide-partie AI):

```
int interditN[9][9];
int interditB[9][9];
for(int i=0;i<9;i++){
   for(int j=0;j<9;j++){
       interditN[i][j]=0;
       interditB[i][j]=0;
```

Quand l'utilisateur choisit une position, on teste son appartenance à notre matrice avant de placer la pierre sur le goban. Puis on construit un scénario qui diffère selon les cas de KO. (Voir le script à l'annexe)

VII. Stockage des données

1. Extraire les informations stockées sur notre base de données concernant le niveau des joueurs.

On récupère d'après un fichier texte les données des joueurs, le nom et le mot de passe dans chaque ligne en utilisant une boucle while qui parcourt le fichier ligne par ligne :

```
GoData1= fopen("GoData1.txt", "r");
while (1){

  fscanf(GoData1,"%s %s %d %d %d %c \n ", &NomJoueur,&MotPasse,&NombreJouee,&NombreGagnee,&NombrePerdues,&categorie);
  int count1=0; int count2=0;
  for(int i=0;i<1000;i++){

     if(NomJoueur[i]=='\0'){
       break;}

    count1++;
    }
   for(int i=0;i<1000;i++){

     if(MotPasse[i]=='\0'){
       break;}
    count2++;
   }

  char NomJoueurExactRecupere[count1];
  char MotPasseExactRecupere[count2];

  for(int i=0;i<count1;i++){

     NomJoueurExactRecupere[i]= NomJoueur[i];
  }
  for(int i=0;i<count2;i++){

    MotPasseExactRecupere[i]= MotPasse[i];
  }
```

Puis on récupère le nom et le mot de passe du joueur 1 :

```
int  count3=0; int count4=0;
 for(int i=0;i<1000;i++){

    if(nom_joueur1[i]=='\0'){
      break;}

   count3++;}
  for(int i=0;i<1000;i++){

    if(MotPasseJ1[i]=='\0'){
      break;}
   count4++;}

 char NomJoueurExactDonneJ1[count3];
 char MotPasseExactDonneJ1[count4];

 for(int i=0;i<count3;i++){

    NomJoueurExactDonneJ1[i]= nom_joueur1[i];
 }
 for(int i=0;i<count4;i++){

   MotPasseExactDonneJ1[i]= MotPasseJ1[i];
```

On répète le même traitement pour le joueur 2 :

```
int  count5=0; int count6=0;
  for(int i=0;i<1000;i++){

     if(nom_joueur2[i]=='\0'){
       break;}

    count5++;}
   for(int i=0;i<1000;i++){

     if(MotPasseJ2[i]=='\0'){
       break;}
    count6++;}

  char NomJoueurExactDonneJ2[count5];
  char MotPasseExactDonneJ2[count6];

  for(int i=0;i<count5;i++){

     NomJoueurExactDonneJ2[i]= nom_joueur2[i];
  }
  for(int i=0;i<count6;i++){

    MotPasseExactDonneJ2[i]= MotPasseJ2[i];
  }
```

2. Déterminer le type de la partie (partie handicap).

La partie handicap comme déjà expliquée consiste sur le fait de donner des coups supplémentaire pour le joueur qui a moins de chance pour gagner contre son adversaire , la fonction int DetermineHandicap(char nom_joueur1[100],char MotPasseJ1[20],char nom_joueur2[100], char MotPasseJ2[20]) prend comme input le nom du premier joueur et son mot de passe et le nom du deuxième joueur et son mot de passe puis il permet de retourner 3 valeurs :

- Elle retourne 1 si le joueur 1 qui est à handicap.
- Elle retourne 2 si le joueur 2 qui est à handicap.
- Elle retourne 0 si aucun d'autre eux n'est à handicap.

Si on trouve que les deux joueurs ont déjà joué avec notre jeu on compare leurs catégories, on a considéré qu'une partie n'est déclaré handicap que lorsque un joueur dépasse son adversaire par deux niveau (exemple : joueur 1= « A » & joueur 2= « C »)

```
if(feof(GoData1)){
            if(indexJ1==1 && indexJ2==1){

            if(  (categoriefinale1=='A' && (categoriefinale2=='C'| categoriefinale2=='D' |categoriefinale2=='E'))
                | (categoriefinale1=='B' && ( categoriefinale2=='D' |categoriefinale2=='E'))
                | (categoriefinale1=='C' && ( categoriefinale2=='E'))
                ){
                return 2;/// donc le J2 est en position de handicap

                }

            if(indexJ2==1)
            if(  (categoriefinale2=='A' && (categoriefinale1=='C'| categoriefinale1=='D' |categoriefinale1=='E'))
            | (categoriefinale2=='B' && ( categoriefinale1=='D' |categoriefinale1=='E'))
            | (categoriefinale2=='C' && ( categoriefinale1=='E'))
            ){
            return 1;/// donc le J2 est en position de handicap

            }
            }
```

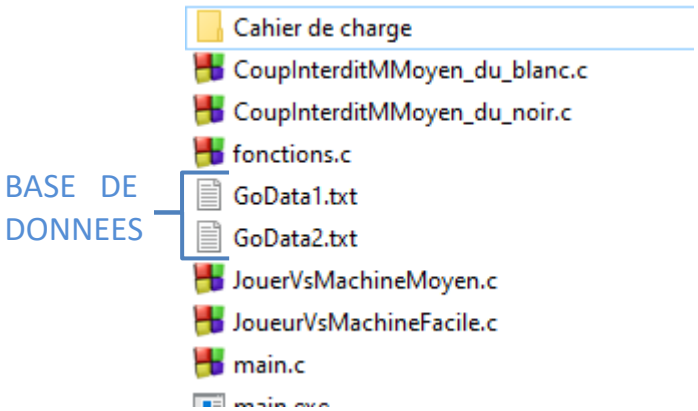
Si un des deux joueurs n'est pas reconnu dans notre base de donnée on l'ajoute dans le fichier, on ajoute son nom, son mot de passe, et on initialise les autres valeurs par des zéros, et enfin on retourne 0 pour dire qu'il s'agit pas d'une partie à handicap.

```
if(indexJ1==0)/// ceci dit que le joueur 1 n'est pas dans la base de donnees ; donc on, doit l'ajouter
{
    GoData1= fopen("GoData1.txt", "a");
    fprintf(GoData1,"%s %s %d %d %d %c \n",nom_joueur1,MotPasseJ1,0,0,0,'E');
    }
if(indexJ2==0)/// ceci dit que le joueur 2 n'est pas dans la base de donnees ; donc on, doit l'ajouter
    {
    GoData1= fopen("GoData1.txt", "a");
        fprintf(GoData1,"%s %s %d %d %d %c \n",nom_joueur2,MotPasseJ2,0,0,0,'E');
    }
if(indexJ1==0 && indexJ2==0){
    return 0;
```

Après Qu'on on détecte une partie à handicap on donne un coup supplémentaire au début au joueur le plus faible.

3. Modifier les informations conservées sur la base de données après avoir annoncé le vainqueur et le perdant.

En fin de chaque partie, on modifie les statistiques des joueurs dans notre base de données à l'aide de deux fichier texte, GOdata1.txt et Godata2.txt. On ajoute des partie gagnées et autres perdues ; on augmente le total jouées , et on affecte à nouveau la nouvelle catégorie correspondante à chaque joueur.



VIII. Intelligence artificielle

- Conception de la partie AI

L'intelligence artificielle est par excellence la partie qui a pris le plus de temps dans le jeu.
Comme expliqué précédemment, nous avons deux niveaux de difficulté : facile et moyen. Au niveau facile, la machine choisit une position aléatoire sur le plateau tout en respectant les règles générales du jeu. En revanche, au niveau moyen, l'ordinateur

effectue des choix plus pertinents. Il peut non seulement défendre et attaquer, mais également anticiper les développements futurs dans divers scénarios.

En effet, à chaque moment, l'ordinateur dispose de plusieurs options pour placer sa pierre, et ces options se multiplient à mesure que le jeu évolue. À un instant donné, il peut être judicieux pour la machine de privilégier la défense plutôt que l'attaque, tandis qu'à d'autres moments, une stratégie offensive peut être plus pertinente. Pour gérer cela, nous avons commencé par répertorier toutes les possibilités de coups d'attaque et de défense disponibles. Ensuite, nous avons réalisé une étude mathématique pour sélectionner le choix optimal. Cette analyse nous permet de prendre des décisions éclairées en pesant les avantages et les inconvénients de chaque option et en choisissant la meilleure stratégie en fonction de la situation actuelle du jeu.

- Stockage des coups

Dans cette partie on a utilisé une autre fois la fonction **'int parcoursfinal'**. Nous avons tiré parti de toutes ces caractéristiques. Il est important de rappeler que cette fonction traverse l'objet à parcourir et, pour chaque objet, renvoie les positions de toutes les pierres de la chaîne à explorer dans un tableau, ainsi que les positions de leurs degrés de liberté dans un autre tableau. De plus, la taille de la chaîne et son degré de liberté. Cependant, si cette fonction trouve une intersection vide, le processus de parcours n'a pas lieu. Les intersections non vides constituent un coup possible. Cette démarche est répétée 81 fois pour toutes les positions du plateau, et les résultats des coups possibles sont simultanément stockés dans un tableau de structures appelée chaineN[81] et chaineB[81] qui stocke les informations des pierres noires et celles blanches, ses informations sont :(nombre de vides, position des vides, nombre des pierres à capturer et position des pierres à capturer).

```
typedef struct{
    int nbr_de_vide;
    int  nbr_de_captures;
    int  position_de_captures[81];
    int position_de_vide[81];

} chaineN;


typedef struct{
    int nbr_de_vide;
    int  nbr_de_captures;
    int  position_de_captures[81];
    int position_de_vide[81];

} chaineB;
```

```
t=parcoursduvidesN(tab_p,i,j,tab_de_position_capture,capture,exist,tab_du_pos_de_vide,vides);
chainN[k].nbr_de_vide=t;
chainN[k].nbr_de_captures=*(capture);
```

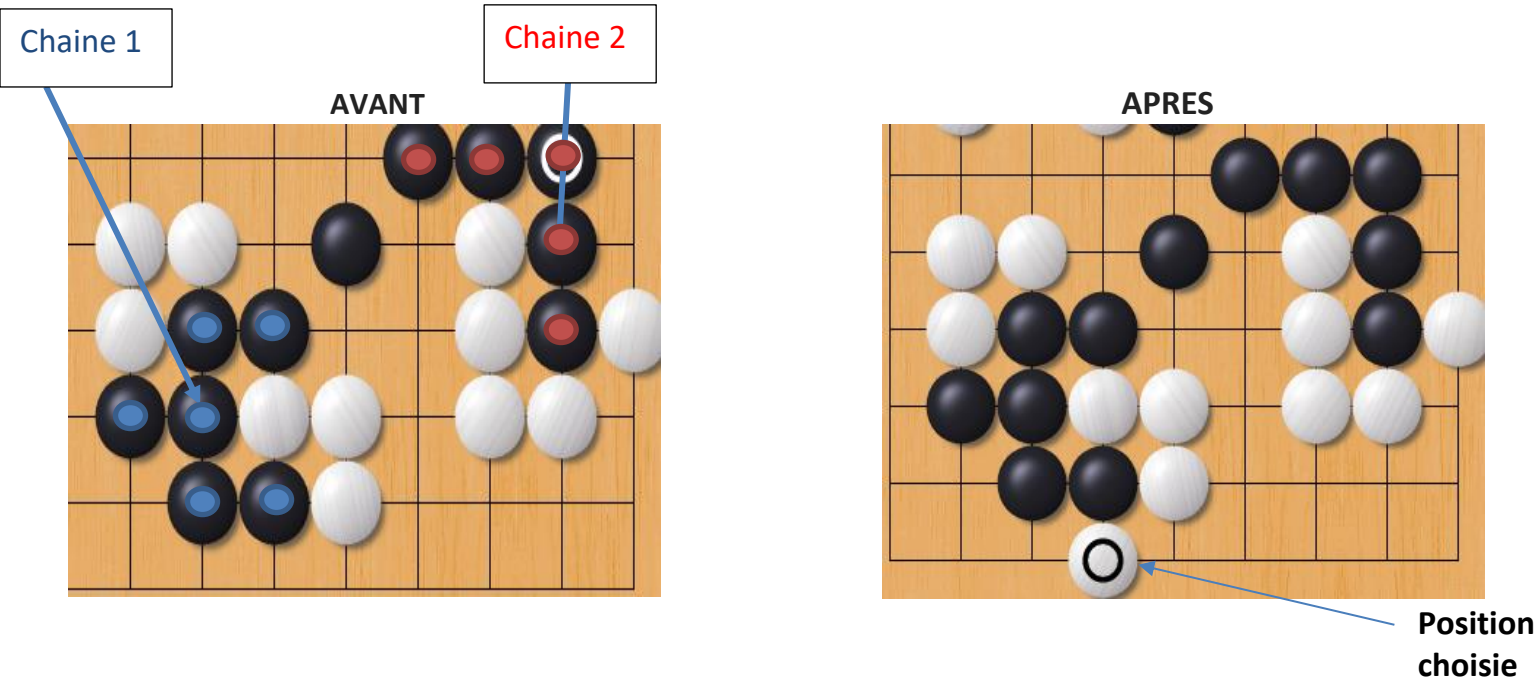
- Etude mathématique

Pendant notre étude, nous avons considéré la défense comme prioritaire lorsque le degré de liberté est de 1 pour une chaîne pouvant être capturée. En revanche, en ce qui concerne l'attaque, il se peut que plusieurs chaînes puissent être entourées en même temps. La question se pose alors de savoir comment choisir laquelle attaquer.
Pour résoudre ce dilemme, nous avons mis en place une approche d'évaluation multicritère. Nous avons développé un ensemble de critères et de règles pour évaluer les différentes options d'attaque. Ces critères pourraient inclure la taille des chaînes adverses, leur potentiel de capture, leur influence sur le plateau, et d'autres facteurs pertinents. En appliquant ces critères, nous sommes en mesure de classer les différentes possibilités d'attaque et de choisir celle qui offre le meilleur avantage stratégique à ce moment-là.
Cette approche implique de recueillir des données sur les décisions prises par des joueurs professionnels lorsqu'ils font face à une situation où plusieurs options d'attaque sont disponibles. Ensuite, ces données sont utilisées pour créer une équation mathématique qui modélise ces décisions en fonction de paramètres tels que la taille des chaînes adverses et leur potentiel de capture, tel que représenté par leur degré de liberté.

Ecart= nbr_de_captures-**h*** nbr_de_vide

Nous avons considéré qu'un coup judicieux a un écart plus important. Nous avons ainsi utilisé des données provenant de parties professionnelles jouées sur le site https://online-go.com/ pour prendre cette décision .

On prend de cette ce cas comme exemple :



Dans cette situation, le joueur est confronté à deux options d'attaque. Il a opté pour attaquer la première chaîne. Cela indique que le joueur considère la première chaîne comme plus favorable que la deuxième. En conséquence, on peut déduire que l'écart entre les deux options, noté comme "ecart2", est inférieur à l'écart initial "ecart1". À partir de cette information, nous pouvons maintenant résoudre la valeur de "h".

La chaine 1 de taille 6 a 6 degrés de liberté tandis que la chaine 2 de taille 5 a 7 degrés de liberté.
Soient nbr_de_captures2, nbr_de_vide2 la taille de chaine 2 et son degré de liberté ET nbr_de_captures1 et nbr_de_vide1 la taille de chaine 1 et son degré de liberté.

On a donc : Ecart2<ecart1

nbr_de_captures2-**h*** nbr_de_vide2< nbr_de_captures2-**h*** nbr_de_vide2

5-7*h<6-6*h

h>1

Initialement, nous avons pu obtenir une estimation approximative de la valeur de "h", mais elle n'était pas précise. Cependant, après avoir analysé d'autres coups de la même manière, nous avons pu resserrer la fourchette de valeurs possibles pour "h" entre 1 et 2. Face à ce résultat, nous avons choisi de fixer la valeur de "h" à **1,5**, même si elle n'est pas précise, car nous avions besoin davantage de plus de données pour obtenir une évaluation plus précise.

> Remarque :
> Exact, la chaîne cible est effectivement préalablement sélectionnée, mais la position précise où placer la pierre sera aléatoirement choisie parmi les emplacements possibles pour entourer cette chaîne. Cette approche ajoute une dynamique au jeu et évite le problème de répétition de choix du même emplacement d'attaque.

- Défense

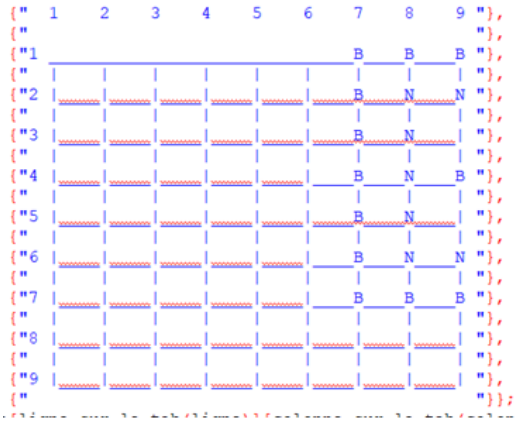
Le jeu progresse et l'ordinateur poursuit ses attaques jusqu'à ce qu'il repère une situation d'attaque adverse presque achevée. Cela signifie qu'une des chaînes n'a qu'un seul degré de liberté restant. À ce stade, l'ordinateur abandonne son attaque en cours et opte pour la défense de sa propre chaîne. Ce processus repose sur la fonction **'int parcoursfinal'**, qui explore toutes les chaînes de l'ordinateur. Si cette fonction détecte qu'une chaîne possède un unique degré de liberté, l'ordinateur abandonne alors son attaque actuelle et change de stratégie en passant à une action défensive.

```
 }
}for(int k=0;k<81;k++){
        if((chainB[k].nbr_de_vide==1)&&(*mB==0))
{
    {*mB=1;}
coupB=k;
            colonne=get_colo((chainB[coupB].position_de_vide[0]));
       ligne=get_li((chainB[coupB].position_de_vide[0]));
}
            }
            k=0;
```

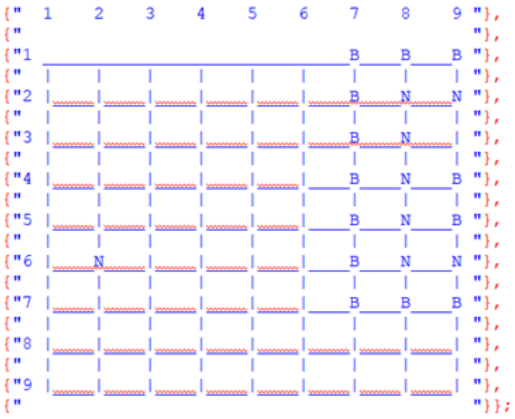
- TSUME-GO

Le tsume-go est une stratégie mise en œuvre pour anticiper l'avenir en ajoutant plusieurs scénarios à suivre dès que la machine se retrouve dans une situation donnée. Les tsume-go ont une priorité supérieure à la fois par rapport à la défense et à l'attaque. Nous avons tenté de mettre en œuvre trois tsume-go différents afin d'aborder différentes situations et de prendre des décisions éclairées basées sur ces scénarios prédéfinis.

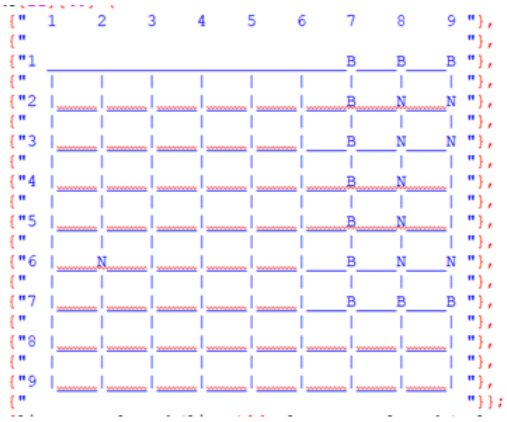
- 1er TSUME-GO :



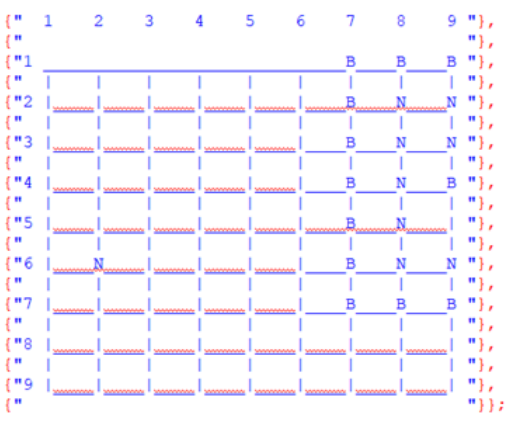
1er étape : En se trouvant dans cette situation, c'est l'utilisateur qui a le tour de jouer.



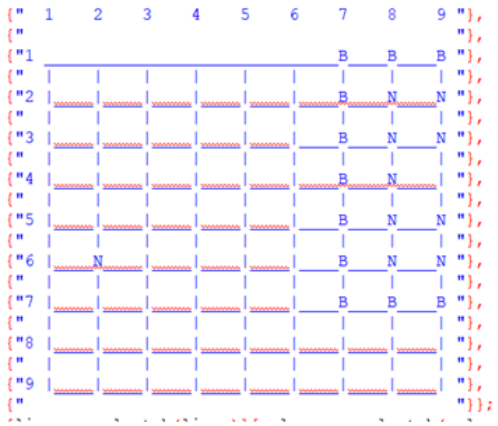
2ème étape : dans le cas où l'utilisateur joue ailleurs des 3 intersections :(3,9) ;(4,9) ;(5,9), la machine doit jouer dans l'intersection :(5,9).



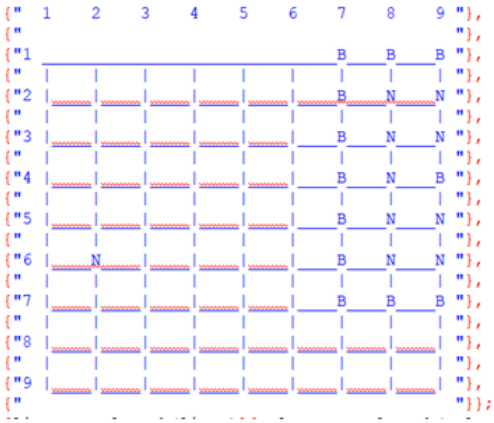
3ème étape : si l'utilisateur joue dans l'intersection (3,9), il mange les deux pierres situés dans les deux intersections :(4,9) et (5,9).



4ème étape : la machine alors doit poser sa pierre dans la position :(4,9)



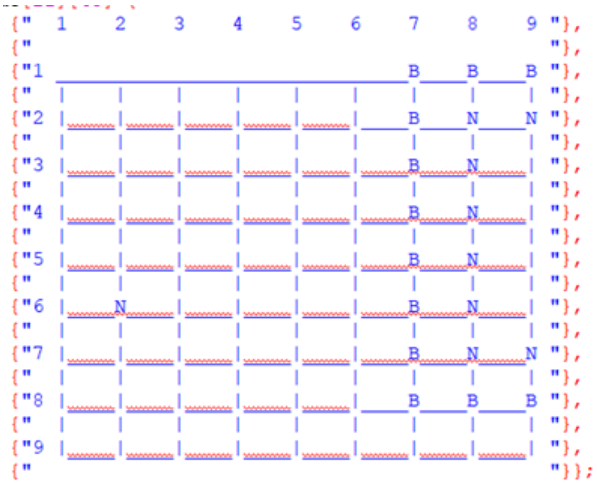
5ème étape : si l'utilisateur dépose sa pierre dans l'intersection : (5,9) il va manger la pierre (4,9).



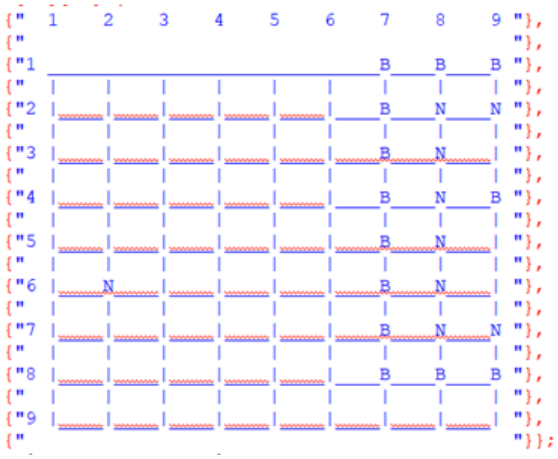
6ème étape : la machine alors doit déposer sa pierre dans l'intersection :(4,9)

Alors la machine va gagner ! Et elle va manger les 9 pierres situées dans les 9 intersections suivantes:
(2,9),(2,8),(3,8),(3,9),(4,8),(5,8),(5,9),(6,8),(6,9).

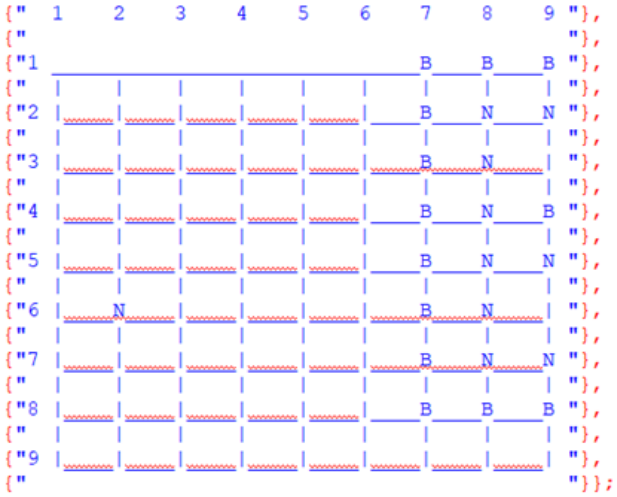
- 2eme TSUME-GO :



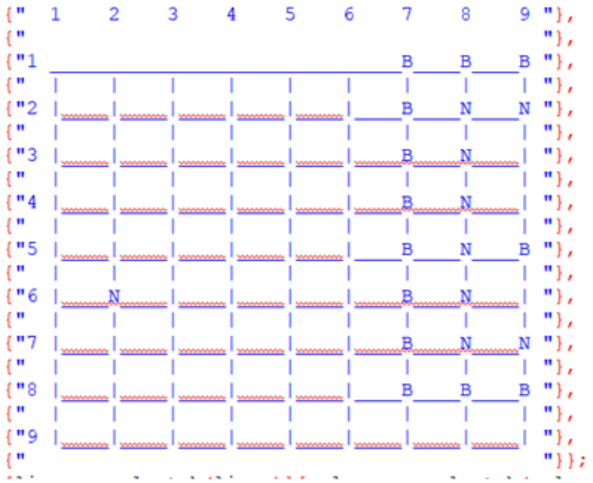
1er étape :On suppose ce scénario et c'est le tour de la machine.



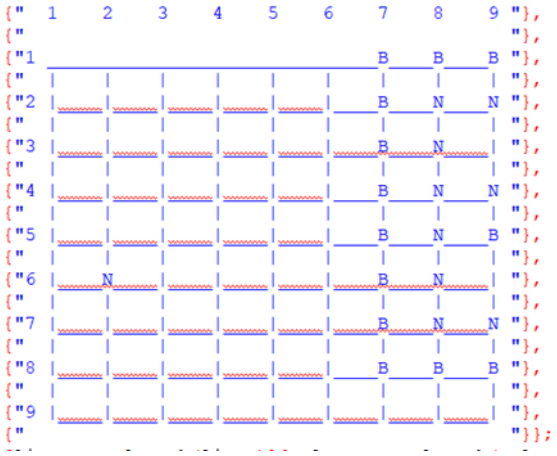
2ème étape :On suppose que la machine dépose sa pierre dans l'intersection (4,9) et on remarquera les résultats obtenus



3ème étape : surement si l'utilisateur connait les règles du jeu il va déposer sa pierre dans l'intersection(5,9), dans son cas la machine va perdre sa pierre placé «edans l'intersection (4,9)



4ème étape : supposons cette fois ci quela machine va déposer sa pierre dans l'intersection (5,9)

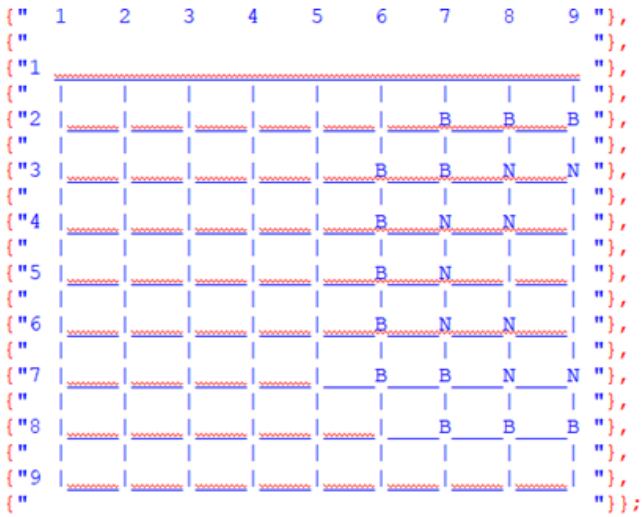


5ème étape : l'utilisateur peut alors déposer sa pierre dans l'intersection (4,9) et la machine perdra sa pierre.

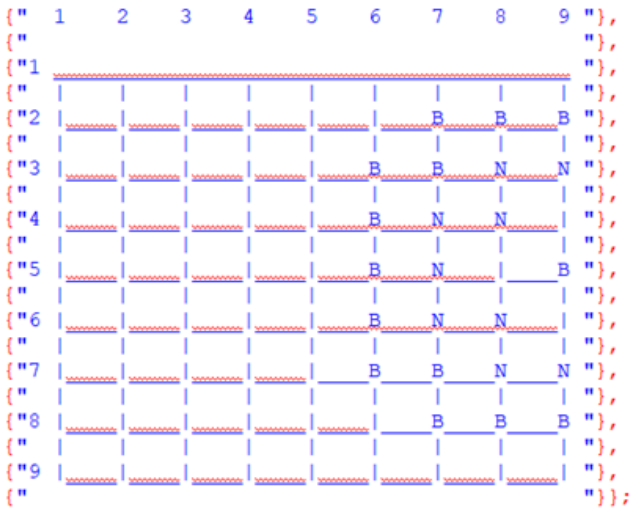
Conclusion :

la machine ne doit pas déposer sa pierre dans les quatre intersections suivantes : (3,9) ;(4,9) ;(5,9) ;(6 ,9).

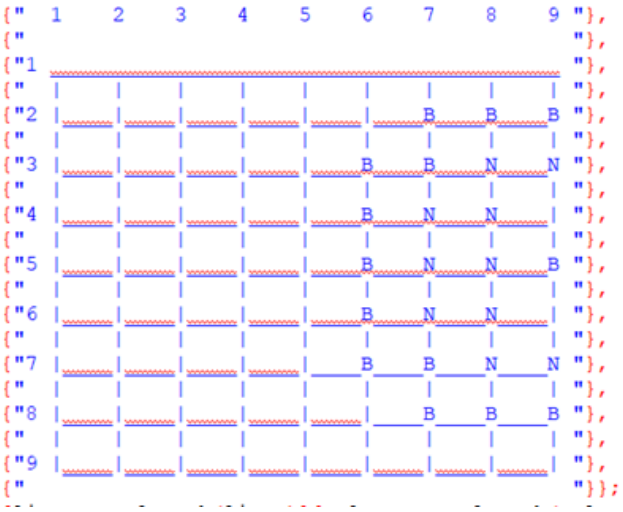
- 3eme TSUME-GO :



1er étape : On suppose le scénario suivant avec la machine qui a le droit de jouer



2ème étape : alors la machine doit déposer sa pierre dans l'intersection (5,9)



3ème étape : on suppose que l'utilisateur pose sa pierre dans l'intersection (5,8)

A ce stade, ce scénario ressemble au premier donc les coups suivants seront les mêmes

- Coups interdits

Les coups interdits ont la priorité supérieure, car il serait futile de se concentrer sur d'autres aspects pour ensuite finir par se suicider en poursuivant une stratégie d'attaque. Cependant, cette approche a des conséquences plus préjudiciables, car cela peut entraîner un suicide infiniment répété, si la machine pense que c'est son meilleur coup.

En outre, nous avons également dû trouver une solution à la boucle infinie générée par le choix répété de la même position, qui serait rejetée par le processus des coups interdits, bien que cette position soit considérée comme la plus pertinente. Pour éviter cette situation, nous avons jugé opportun de changer de coup une fois que la position choisie a été rejetée. Cela garantit une progression continue et empêche d'entrer dans des situations de répétition.

Annexe (Script)

- ATARI_GO

La fonction ATARI_GO_du_blanc(char tableaudynamique[21][45], int *scoreB,int *tab_de_position_noires,int *je,int *existe,char *tab_p) permet à la fois de supprimer les jeton captés et aussi ajouter les pierres gagnées dans le score du gagnant , elle prend d'abord la valeur retournée d'après la fonction précédente « parcoursB » et elle utilise aussi la fonction supprimer_jeton(char tableaudynamique[21][45],int k,int l).

```
int ATARI_GO_du_blanc(char tableaudynamique[21][45], int *scoreB,int *tab_de_position_noires,int *je,int *existe,char *tab_p){
    int tab_de_position_a_supprimer[21][45];
    (*scoreB)=0;
    for(int i=1;i<10;i++)
    {
     for( int j=1;j<10;j++)
      { tab_de_position_a_supprimer[ligne_sur_le_tab(i)][colonne_sur_le_tab(j)]=0;
      }
    }

    for(int i=1;i<10;i++)
    {
     for( int j=1;j<10;j++)
      {
        *(je)=0;
        for(int r=0;r<586;r++){
                    {
                         *(tab_de_position_noires+r)=0;
                    }
                }

        if( parcoursN(tab_p,i,j,tab_de_position_noires,je,existe)==1){
              tab_de_position_a_supprimer[ligne_sur_le_tab(i)][colonne_sur_le_tab(j)]=1;}
            }
        }

        for (int k=1;k<10;k++)
              {
                 for(int l=1;l<10;l++)
                 {


                if (tab_de_position_a_supprimer[ligne_sur_le_tab(k)][colonne_sur_le_tab(l)]==1){

                       supprimer_jeton(tableaudynamique,ligne_sur_le_tab(k),colonne_sur_le_tab(l));

                           (*scoreB)++;}}}
return(*scoreB);
```

- Calcul de territoire

```
    for (int i=1;i<=9;i++){
         for (int j=1;j<=9;j++){
         tab_p=malloc((45*13));
         tab_p=tableaudynamique;
         je=malloc(4);vides=malloc(4);
         tab_de_position_noires=malloc(4*45*13);
         tab_du_pos_de_vide=malloc(4*45*13);
         *(je)=0;*(vides)=0;
         for(int r=0;r<586;r++){*(tab_de_position_noires+r)=0;}
         for(int r=0;r<586;r++){*(tab_du_pos_de_vide+r)=0;}
         existe=malloc(4);
         *existe=0;
             test_du_finB=parcoursvidesB(tab_p,i,j,tab_de_position_noires,je,existe,tab_du_pos_de_vide,vides);

         if(test_du_finB==1)
             {
             tableaudynamique[ligne_sur_le_tab(i)][colonne_sur_le_tab(j)]='B';
             }
         }
     }
```

e

- KO

```
//ww milieu ww toic ww wwww
for(int li=1;li<10;li++)
    {
        for(int co=1;co<10;co++)
            {
                if((li-1>0)&&(co-1>0)&&(li+1<=9)&&(co+1<=9))
                    {
                            if ((tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co-1)]=='N')&&
                                (tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co)]=='B')&&
                                (tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co+1)]=='N')&&
                                (tableaudynamique[ligne_sur_le_tab(li+1)][colonne_sur_le_tab(co)]=='N')&&
                                (tableaudynamique[ligne_sur_le_tab(li-1)][colonne_sur_le_tab(co)]=='N')&&
                                (tableaudynamique[ligne_sur_le_tab(li+1)][colonne_sur_le_tab(co+1)]=='B')&&
                                (tableaudynamique[ligne_sur_le_tab(li-1)][colonne_sur_le_tab(co+1)]=='B')&&
                                (tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co+2)]=='B'))
                                    {
                                        supprimer_jeton(tableaudynamique,ligne_sur_le_tab(li),colonne_sur_le_tab(co));
                                        scoreNo++;
                                        interditB[li][co]=1;

                                    }
                            if ((tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co-1)]=='B')&&
                                (tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co)]=='N')&&
                                (tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co+1)]=='B')&&
                                (tableaudynamique[ligne_sur_le_tab(li+1)][colonne_sur_le_tab(co)]=='B')&&
                                (tableaudynamique[ligne_sur_le_tab(li-1)][colonne_sur_le_tab(co)]=='B')&&
                                (tableaudynamique[ligne_sur_le_tab(li+1)][colonne_sur_le_tab(co+1)]=='N')&&
                                (tableaudynamique[ligne_sur_le_tab(li-1)][colonne_sur_le_tab(co+1)]=='N')&&
                                (tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co+2)]=='N'))
                                    {
                                        supprimer_jeton(tableaudynamique,ligne_sur_le_tab(li),colonne_sur_le_tab(co+1));
                                        scoreNo++;
                                        interditB[li][co+1]=1;

                                    }
                    }
```

```
                                    }
                            }
                            if(li>=2 && li<=7)
{
    if ((tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co)]=='B') &&
            (tableaudynamique[ligne_sur_le_tab(li+1)][colonne_sur_le_tab(co+1)]=='B') &&
            (tableaudynamique[ligne_sur_le_tab(li+1)][colonne_sur_le_tab(co-1)]=='B') &&
            (tableaudynamique[ligne_sur_le_tab(li+2)][colonne_sur_le_tab(co)]=='B') &&
            (tableaudynamique[ligne_sur_le_tab(li-1)][colonne_sur_le_tab(co)]=='N') &&
            (tableaudynamique[ligne_sur_le_tab(li+1)][colonne_sur_le_tab(co)]=='N') &&
            (tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co+1)]=='N') &&
            tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co-1)]=='N')
            {
                 supprimer_jeton(tableaudynamique,ligne_sur_le_tab(li),colonne_sur_le_tab(co));
                                    scoreNo++;
                                    interditB[li][co]=1;
            }
             if ((tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co)]=='N') &&
            (tableaudynamique[ligne_sur_le_tab(li+1)][colonne_sur_le_tab(co+1)]=='N') &&
            (tableaudynamique[ligne_sur_le_tab(li+1)][colonne_sur_le_tab(co-1)]=='N') &&
            (tableaudynamique[ligne_sur_le_tab(li+2)][colonne_sur_le_tab(co)]=='N') &&
            (tableaudynamique[ligne_sur_le_tab(li-1)][colonne_sur_le_tab(co)]=='B') &&
            (tableaudynamique[ligne_sur_le_tab(li+1)][colonne_sur_le_tab(co)]=='B') &&
            (tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co+1)]=='B') &&
            tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co-1)]=='B')
            {
                 supprimer_jeton(tableaudynamique,ligne_sur_le_tab(li+1),colonne_sur_le_tab(co));
                                    scoreNo++;
                                    interditB[li][co]=1;
            }



     if(co==1){
                if ((tableaudynamique[ligne_sur_le_tab(li-1)][colonne_sur_le_tab(co)]=='N')&&
                    (tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co)]=='B')&&
                    (tableaudynamique[ligne_sur_le_tab(li+1)][colonne_sur_le_tab(co)]=='N')&&
                    (tableaudynamique[ligne_sur_le_tab(li-1)][colonne_sur_le_tab(co+1)]=='B')&&
                    (tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co+1)]=='N')&&
                    (tableaudynamique[ligne_sur_le_tab(li+1)][colonne_sur_le_tab(co+1)]=='B')&&
                    (tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co+2)]=='B'))
                        {
                                supprimer_jeton(tableaudynamique,ligne_sur_le_tab(li),colonne_sur_le_tab(co));
                                scoreNo++;
                                interditB[li][co]=1;

                        }

                if ((tableaudynamique[ligne_sur_le_tab(li-1)][colonne_sur_le_tab(co)]=='B')&&
                    (tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co)]=='N')&&
                    (tableaudynamique[ligne_sur_le_tab(li+1)][colonne_sur_le_tab(co)]=='B')&&
                    (tableaudynamique[ligne_sur_le_tab(li-1)][colonne_sur_le_tab(co+1)]=='N')&&
                    (tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co+1)]=='B')&&
                    (tableaudynamique[ligne_sur_le_tab(li+1)][colonne_sur_le_tab(co+1)]=='N')&&
                    (tableaudynamique[ligne_sur_le_tab(li)][colonne_sur_le_tab(co+2)]=='N'))
                        {
                                supprimer_jeton(tableaudynamique,ligne_sur_le_tab(li),colonne_sur_le_tab(co+1));
                                scoreNo++;
                                interditB[li][co+1]=1;

                        }

                }
```

Annexe (fiche des fonctions)

LA FONCTION : int parcoursB(char *tab,int ligne,int colonne,int *paru,int *j,int *existe)

LE ROLE : parcourir les jetons blanc et vérifier si on peut les capturer ou pas.
UITLISE DANS : ATARI_GO_du_blanc
FONCTIONS UTILISE DEDANS : NULL
INTPUT :

- char *tab : notre tableaudynamique sous forme d'un pointeur
- ,int ligne :ligne
- ,int colonne :colonne
- ,int *paru : un pointeur sur un tableau qui nous servis a tester si un jeton est déjà parcouru ou pas dans notre fonction recursive / l existence d un jeton dans « paru » signifie qu il est déjà parcouru
- ,int *j :un pointeur sur une variable afin de parcourir « paru »
- ,int *existe : pointeur sur une variable afin de tester l existence d un jeton dans « paru » il prend la valeur 1 si le jeton existe dans paru et 0 si non

OPERATION SUR LES INTPUT : (0 signifie pas de changement)

char *tab : 0
,int ligne :0
int colonne :0
,int *paru : a la fin on aura un tableau remplis par des positions des jetons blancs à capturés
,int *j : a la fin on aura le nombre des jetons blancs à capturés
,int *existe : il ne sert a rien a la fin

OUTPUT :

- 0 :si les jeton B dans la peremiere position est de coordonnés(ligne,colonne) ne sont pas entourés
- 1 :si les jeton B dans la peremiere position est de coordonnés(ligne,colonne) sont entouré

LA FONCTION : int parcoursN(char *tab,int ligne,int colonne,int *paru,int *j,int *existe)

Meme principe que parcoursB

FICHIER : #include "RegleKo_du_blanc.c"

LE ROLE verifier si un senario de KO existe si oui il interdit le noir à jouer dans la position interdite

UITLISE DANS : menujouerunepartie()
FONCTIONS UTILISEES DEDANS : NULL
INTPUT : interditN[9][9]
OPERATION SUR LES INTPUT :(0 signifie pas de changement)
Soient (x,y) les coordonnées d'une intersection dont il sera interdit à cause du KOinterditN[x][y] prend 1
OUTPUT :

- interditN[x][y] =0 : cet intersection n'est pas interdit
- interditN[x][y] =1 : cet intersection est interdit

principe)

FICHIER : TsumegoFondamental_du_N

LE ROLE : permettre a la machine de choisir le bon coup pour attaquer
UITLISE DANS : MachineMoyen
FONCTIONS UTILISE DEDANS : parcoursduvidesB ,(on a utilisé la structure chaine)
INTPUT : tableaudynamique
PRINCIPE : on utilise la structure chaine qui a comme attribue -- nbr_de_vide--nbr_de_captures;--position_de_captures[81]---position_de_vide[81]

Dans la premiere partie de ce fichier on essaye de parcourir tour les positions du tableau et pour chaqu'une on construit les informations suivante nbr_de_vide--nbr_de_captures;--position_de_captures[81]---position_de_vide[81] grace à la fonction parcoursduvidesB cette derniere a besoin des inputs qui sont definis dans ce fichier (voir principe de parcoursduvidesB)

On a construit un tableau de la structure « chaine » ou chaque case de la structure chaine contient les informations suivantes sur unintersection du tableau(nbr_de_vide--nbr_de_captures;--position_de_captures---position_de_vide) à l'aide de la fonction parcoursduvidesB.

Pour la 2 partie du fichier et après qu'on a eu les informations sur chaque intersection on doit etudier le coup parfait qui va etre choisipar la machine en fct du nombre de jeton à capturé et son degré de liberté en introduisant une information «écart= chainB[k].nbr_de_captures-h*chainB[k].nbr_de_vide vide » pour chaque intersection et la stocker dans untableau « tab_du_comparaison_coupB »

h: sera choisi par la suite du tel sorte que le coup ayant l écart le plus grand est le coup parfait (voir étude h)

Par la suite après avoir stocker toutes les écarts on a essayé de recuperer l écart maximal a l aide de la fonction max (qui prend c comme intput un tab_du_comparaison_coup et nous return l'index ayant l'écart le plus grand) cet index va nous aider a récupéré l'intersection ayant cet écart maximal(il s'agit du coup parfait) et donc on va recuperer la ligne et la colonne du coup parfait qui eststocké dans la structure « chaine » chainB[coup_parfaitB].position_de_vide[0]

Remarque :

Après qu on a ajouté la parties du coupinterdit on a eu quelque proble c est qu on tombe sur une boucle infinie donc on ajouer une autre parties pour resoudre ce probleme afin d ignorer le coup_parfait s il est interdit

LA FONCTION : parcoursduvidesB.

LE ROLE : parcourir les jetons blanc et nous donne les infos suivantes suivante nbr_de_vide--nbr_de_captures;--position_de_captures---position_de_vide.
UITLISE DANS : TsumegoFondamental
FONCTIONS UTILISE DEDANS : NULL
INTPUT : char *tab : notre tableau dynamique sous forme d'un pointeur

- ,int ligne :ligne
- ,int colonne :colonne
- ,int *paru : un pointeur sur un tableau qui nous servis a tester si un jeton est déjà parcouru ou pas dans notre fonction recursive / l existence d un jeton dans « paru » signifie qu il est déjà parcouru
- ,int *j :un pointeur sur une variable afin de parcourir « paru »
- ,int *existe : pointeur sur une variable afin de tester l existence d un jeton dans « paru » il prend la valeur 1 si le jeton existe dans paru et 0 si non
- ,int *paruvide : un pointeur sur un tableau qui nous servis a tester si un vide est déjà parcouru ou pas
- ,int *c : un pointeur sur une variable afin de parcourir « paruvide»

OPERATION SUR LES INTPUT :(0 signifie pas de changement)

- char *tab : 0
- ,int ligne :0
- ,int colonne :0
- ,int *paru : a la fin on aura un tableau remplis par des positions des jetons blancs à capturés
- ,int *paruvides : a la fin on aura un tableau remplis par des positions des vides qui entour un jetonB
- ,int *j : a la fin on aura le nombre des jetons blancs à capturés
- ,int *c : a la fin on aura le nombre des vides qui entoure un jetonB
- ,int *existe : il ne sert a rien a la fin

OUTPUT :
Le nombre des vides qui entour un une chaine de blanc ayant comme coordonnées de jeton étudié appartenant à la chaine

Annexe (cahier de charge)

Ecole Hassania des Travaux Publics
Casablanca ,le 10/10/2022 Département de Mathématiques ,Informatique et Géomatique
Filière SIG
Mlle Mariam CHERRABI

Programmation en C

-Projet à rendre-

Thème : Le Jeu GO (囲碁) :

On a dénommé le GO :

- GO ou iGO (japonais)
- Wéi qí (chinois)
- Baduk (coréen) GO (français) GO (anglais) .

On a décrit le GO : L'un des plus anciens jeux de stratégie au monde.

On a parlé du GO : « S'il y a une vie intelligente sur Mars, ses habitants doivent avoir découvert le jeu de go » L'ancien champion d'échecs Emanuel Lasker .

On a écrit à propos du GO : le recueil : 围棋十诀,« Wéi Qí Shí Jué » traduit en « Les 10 secrets du succès (Wéi Qí = Go, Shí = dix, Jué = "secret du succès ") , et renommé « Les 10 Commandements du GO » et aussi « Règles d'Or du ROI » (WANG traduit en ROI) . Selon ce recueil des 10 règles de succès inspirées par le GO :

- Règle 1 : (贪不得胜) Tān Bù Dé Shèng (La gourmandise n'apporte pas la victoire) .

- Règle 2: (入界宜缓)Rù Jiè Yí Huǎn (Restez posé lorsque vous entrez chez l'adversaire).
- Règle 3: (攻彼顾我)Gōng Bǐ Gù Wǒ (Quand vous attaquez votre adversaire, faites attention à vous même) .
- Règle 4: (弃子争先)Qì Zǐ Zhēng Xiān (Abandonnez des pierres pour gagner l'initiative) .
- Règle 5: (舍小救大)Shě Xiǎo Jiù Dà (Laissez tomber le petit pour vous intéresser au gros) .
- Règle 6: (逢危须弃)Féng Wēi Xū Qì (En cas de danger, abandonnez quelque chose) .
- Règle 7: (慎勿轻速)Shèn Wù Qīng Sù (Jouez des coups solides, évitez la précipitation) .
- Règle 8: (动须相应)Dòng Xū Xiāng Yìng (Gardez à l'esprit la situation globale ; vos différents coups devant agir de concert) .
- Règle 9: (彼强自保) Bǐ Qiáng Zì Bǎo (Si votre adversaire est fort, jouez la sécurité) .
- Règle 10: (势孤取和) Shì Gū Qǔ Hé (Au milieu d'une influence adverse, recherchez la paix).

On a automatisé le GO : MoGo, Crazy Stone ,AlphaGo ,AlphaGo zero.

Plan :

2. Annexe

Jeux combinatoires :

Un jeu combinatoire est un jeu à deux joueurs avec une information complète, des coups qui ne dépendent pas du hasard et se terminant par la victoire d'un des deux joueurs.

Un tel jeu est défini par un ensemble de positions, dont la position initiale et le joueur dont c'est le tour. Le jeu progresse d'une position à une autre, les joueurs exécutant leur coup alternativement, jusqu'à ce qu'une position terminale soit atteinte.

Une position terminale est une position à partir de laquelle aucun coup n'est possible.

L'un des joueurs est alors déclaré le vainqueur et l'autre le perdant.

Introduction au Jeu du GO :

Le jeu GO

 Origine et histoire :

Le jeu de go, l'un des plus vieux jeux de stratégie au monde, trouve son origine en Chine ou il aurait pris naissance il y a plus de 2000 ans avant J-C. Bien que la date précise de sa première apparition ainsi que la personne à l'origine de ce jeu reste floue, on attribue la création de ce jeu mythique à L'empereur Yao qui, afin d'enseigner la discipline, la concentration et l'équilibre à son fils turbulent, l'utilisait comme outil tutoriel.

On parle également d'anciens généraux qui utilisaient des pierres pour dresser leurs stratégies militaires ou encore de l'utilisation du matériel de go dans les domaines de l'astrologie et de la divination.

En Chine, le go était perçu comme un jeu réservé à l'aristocratie alors que les échecs chinois étaient le jeu du peuple. Le jeu est ainsi devenu l'un des 4 arts, avec la calligraphie, la peinture et la musique, auquel la jeunesse impériale ne pouvait se soustraire. Il se jouait sur une grille 17×17. Puis, sous la dynastie Tang (618-907) les grilles 19×19 sont devenues le standard que nous utilisons encore de nos jours
C'est aux alentours du Vème-VIIème siècle que le jeu de go s'est dispersé en Asie de l'Est, notamment en Corée et au Japon ou le jeu est respectivement appelé Baduk et igo. Comme en Chine, le jeu a d'abord touché l'aristocratie avant de devenir un jeu populaire. Aujourd'hui le go est un jeu extrêmement populaire en Chine, en Corée et au japon ou le nombre de joueurs amateurs et professionnels fait légion.

En Europe, c'est seulement à la fin du 19eme siècle que le go commence à se faire connaître, et c'est au début du 20eme siècle que le go s'installe en Allemagne et au sein de l'Empire Austro-hongroie avant de se répandre dans le reste du continent. En 2008, La Fédération Internationale de Go comptait 71 pays membres. Il semblerait selon leurs statistiques qu'au sein de la population mondiale 1 personne sur 222 joue au go. Il s'agit donc d'un jeu plus populaire qu'il n'y parait et qui semble voué à se développer fortement grâce à l'intérêt qu'il suscite, à la multiplication des clubs et à la possibilité de jouer en sur internet avec des personnes du monde entier.

 Description:

Le jeu GO se joue sur un tablier(碁盤 :goban) en japonais en bois de kaya est une grille de 19 lignes sur 19 rainurées formant 361 intersections. Les débutants jouent souvent sur des goban plus petits de 13×13 (169 intersections) ou 9×9 (81 intersections), sans autre aménagement des règles du jeu.

Les pierres (碁石 go-ishi en japonais) sont réparties en deux ensembles supposés infinis, mais en pratique limités à 181 pierres noires et 180 pierres blanches en forme de lentilles. Les pierres sont jouées sur les intersections libres du goban.

En général, les pierres sont rangées dans des bols (碁笥 go-ke en japonais) en bois, en plastique ou en osier, un par joueur. Le couvercle du bol est commode pour conserver les pierres capturées pendant la partie.

Ce jeu se caractérise paradoxalement par des règles extrêmement simples et naturelles, et une complexité croissante dans la mise en application des stratégies et tactiques en fonction du niveau des joueurs.

Principe : 2 joueurs, l'un disposant des pierres noires, l'autre des pierres blanches, posent à tour de rôle ces dernières sur le goban (plateau de jeu) dans le but d'obtenir le territoire le plus large possible au moyen de stratégies, de jeux d'influence, et de tactiques d'attaque, de défense et de capture.

Le jeu de go est donc accessible à tout un chacun mais une longue période de pratique vous attend si vous espérez atteindre un niveau vous permettant d'affronter les joueurs les plus expérimentés.

Il existe une échelle de niveau officielle permettant d'attribuer une position à chaque joueur professionnel et amateur.
Chez les amateurs, les niveaux s'échelonnent de 30 kiu à 1er kiu, puis de 1er dan à 9e dan.
Chez les professionnels les niveaux vont de 1er à 9e dan.

un 1er dan professionnel correspondant à peu près à un 7e dan amateur. Voici quelques exemples qui vous aideront à mieux comprendre l'organisation des niveaux:

*Un joueur 15K amateur est plus faible qu'un joueur 3K amateur.
*Un joueur 3K amateur est pus faible qu'un joueur 1D amateur.
*Un joueur 1D amateur est pus faible qu'un joueur 6D amateur.
*Un joueur 3D amateur est pus faible qu'un joueur 3D professionnel.
*Le niveau de 9e dan correspond en principe au plus haut niveau existant.

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. Entre amateurs, une pierre de handicap correspondra à un niveau d'écart. Entre joueurs professionnels, une pierre de handicap correspondra à environ trois niveaux d'écart

1.Début de jeu: "Noir" commence et place un pierre sur une intersection du plateau. Blanc à son tour pose sa pierre. Il n'est pas obligatoire de placer les pierres les unes à coté des autres. Les joueurs jouent à tour de rôle en disposant leurs pierres de façon à contrôler le plus de surface du plateau, en cherchant à encercler son adversaire et en le capturant. Les pierres capturées sont retirées du plateau libérant des intersections, le territoire adverse augmente.

2.La capture ou Atari-Go : Une pierre est encerclée puis capturée quand les pierres de son adversaire lui supprime toute liberté . Ce qui signifie que les intersections juste au dessus, au dessous (sur la ligne verticale), à gauche et à droite (sur la ligne horizontale) sont occupées par l'adversaire (les intersections en diagonales ne sont pas concernées) :

2.1 Exemple de capture de "blanc" :

Figure 1

Figure 2

Figure 3

2.2 Exemple de capture de "blanc" sur un bord du goban :

Figure 4

Figure 5

Figure 6

2.3 Exemple de capture de "blanc" en coin de goban :

Figure 7

Figure 8

Figure 9

2.4 Exemple de capture d'une chaine de "noir": Si une chaîne de pierre de même couleur, un groupe de pierre sont encerclées par l'adversaire, elles seront capturées si elles n'ont plus aucune liberté, c'est à dire plus aucune intersection libre à proximité d'une des pierres sur les lignes verticales ou horizontale.

Figure 10

Figure 11

Figure 12

Figure 13

Figure 14

Figure 15

3. La défense : Les pierres sont préservées en positionnant une pierre de la même couleur sur une intersection libre à proximité directe de la pierre à protéger.

Figure 16

Figure 17

4. La formation Ko: Il s'agit d'une combinaison symétrique entre les deux adversaires, le premier capture une pierre adverse, le deuxième joueur pourrai à son tour re-capturer la dernière pierre placer et ainsi de suite à l'infini. Le problème est résolu en interdisant la capture inverse immédiate, il est obligatoire de jouer un tour intermédiaire.

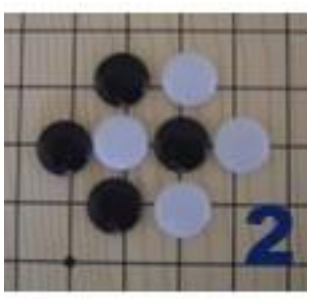


Figure 18 Figure 19 Figure 20 Figure 21

\

But du jeu:

Le camp ayant acquis le plus de territoire gagne la partie. (Les pierres capturées sont incluses dans le décompte.)
Figure 22. Noir a pris 20 points de territoire en haut à gauche et 22 points en bas à droite soit 42 points au total. Blanc a pris 22 points en haut à droite et 19 points en bas à gauche soient 41 points au total. Ainsi, le score est de 42 points pour Noir et 41 points pour Blanc : Le Noir gagne d'un point.

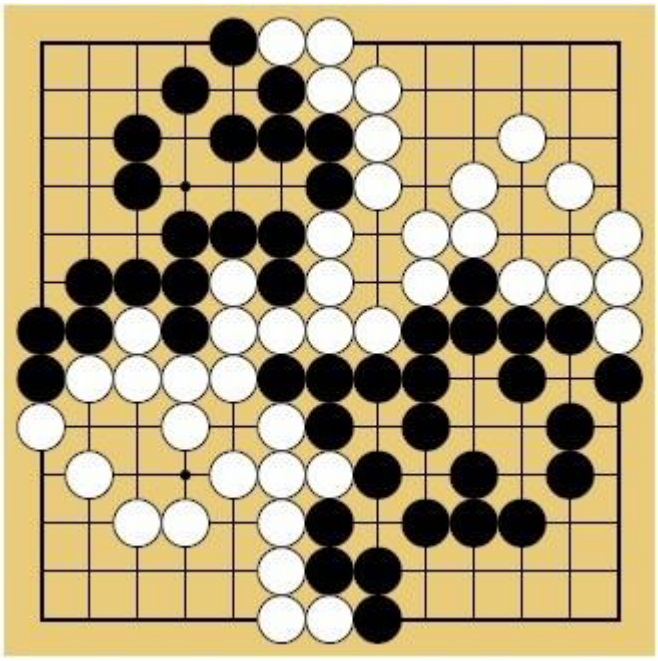


Figure 22

Et sachant que Blanc et Noir ont capturé 5 pierres chacun En fait, les pierres capturées sont replacées dans le territoire de l'adversaire, de sorte que le score final est de 37 points pour Noir et 36 points pour Blanc.

Règles du Jeu : (Voir la documentation en annexe)

Pour jouer au go, il suffit de connaître onze règles : six règles générales et cinq règles techniques.

1.1 Règles générales

1. Le go se joue à deux joueurs.
2. Un joueur joue avec les pierres noires, l'autre avec les pierres blanches. Les joueurs jouent leurs coups un par un par un à tour de rôle.

3. Un coup consiste à placer une pierre sur une intersection du quadrillage. Les pierres peuvent également être placées sur les bords de celui-ci.
4. Une fois qu'une pierre est placée sur une intersection, elle ne peut plus être déplacée.
5. Quand il y a une différence de force entre les deux adversaires, le joueur le plus faible place des pierres supplémentaires sur le plateau pour compenser cette différence de niveau.
6. Dans une partie à égalité, le joueur ayant les pierres noires joue toujours le premier, mais dans une partie à handicap, c'est le Blanc qui commence.

1.2 Règles techniques 1. Détermination du résultat.

2. Capture des pierres.
3. Coups interdits.
4. Ko.
5. Fin de partie.

TSUME-GO :

Un tsumego (詰碁) désigne un problème de GO, typiquement de « vie et mort ». On dit souvent que sans étudier le tsume-go il est impossible d'atteindre un bon niveau. C'est parce que la vie et la mort des pierres est un aspect fondamental du GO.

Comme une grande partie du lexique du go, tsumego (詰碁) est un terme japonais.

Littéralement, tsume (詰) signifie « groupe » ou encore « groupé », mais est utilisé plus techniquement au GO pour désigner des coups bloquant l'expansion d'un groupe, tandis que GO (碁) désigne le jeu de go lui-même (en kanji) ; on peut donc traduire par "GO dans des espaces limités".

Exemples :

Problème 1 : Noir joue et vit

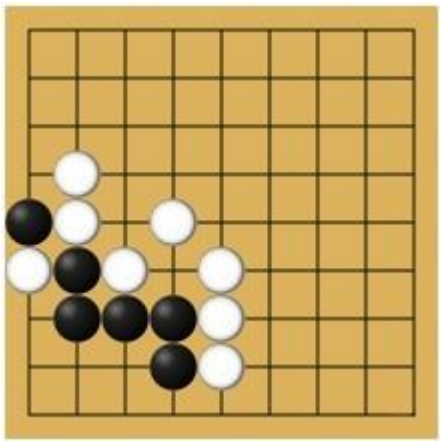


Figure 23

Solution :

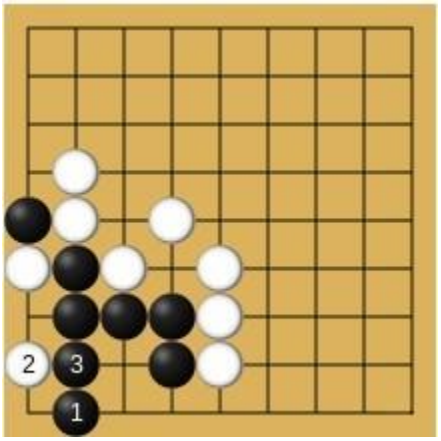


Figure 24

Problème 2 : Noir joue et vit

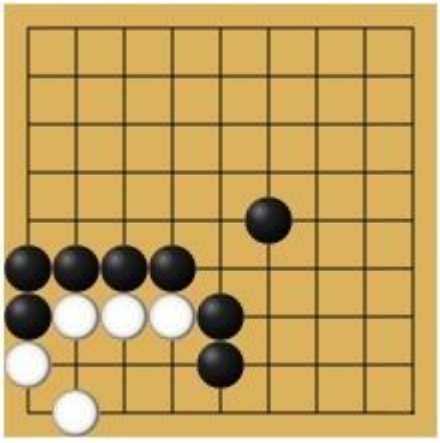


Figure 25

Solution :

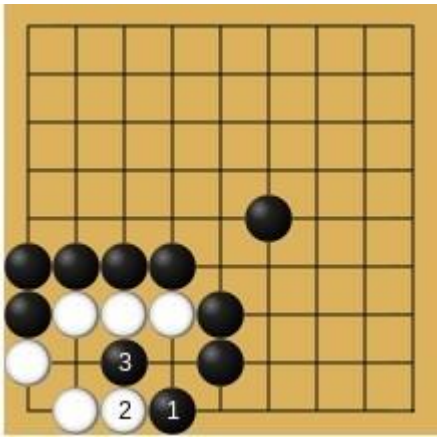


Figure 26

Travail demandé :

Le but de ce projet est d'écrire un programme en C automatisant les règles générales et quelques stratégies du jeu de GO.

On va travailler sur un plateau de 9*9.

Vous considérerez au moins 3 TSUME-GO élémentaires, choisis librement :

Pour établir la liste des choix : [Voir énoncé, annexe et pièces jointes]

1. https://artdugo.fr/glossaire
2. https://artdugo.fr/solution-tsumego-de-semaine-151017/
3. Initiation au go Kano Yoshinori (avec solutions, Annexe)
4. Cho Chikun's Encyclopedia of Life & Death Part 1—Elementary (Pièce Jointe version électronique, sans solutions)
5. Le Jeu de Go Principes Fondamentaux, Yilun YangTraduit de l'anglais par Finn Dickman (Pièce Jointe version électronique,sans solutions)

Chaque groupe choisira au moins 3 TSUME-GO élémentaires, qui vont constituer les scénarios possibles, et essayera de les implémenter pour définir l'intelligence souhaitée.

Tout effort supplémentaire sera comptabilisé et la difficulté liée à chaque choix sera prise en compte.

Dans un premier temps, on considérera le cas de deux humains jouant l'un contre l'autre.

Ensuite, on essayera d'améliorer le programme pour qu'un joueur humain puisse jouer contre l'ordinateur tout en envisageant deux niveaux : facile aléatoire et Moyen.

- Implémentez une version console du jeu GO permettant à deux joueurs humains de jouer l'un contre l'autre et ce en respectant le cadre de l'énoncé,. À vous de construire les structures nécessaires pour gérer l'affichage et la dynamique du jeu.
- Le programme doit être réalisé sous forme d' une application console C. Vous pouvez utiliser la bibliothèque SDL à condition de respecter les consignes et de présenter un travail complet .
- Tout effort supplémentaire sera comptabilisé.
- Une partie importante du projet consiste en un travail de réflexion sur comment créer et organiser les structures nécessaires au bon fonctionnement du jeu :

  Il est important de faire ce travail de conception avant même de chercher à programmer le jeu. Penser aussi à programmer la première partie de manière à pouvoir inclure, ensuite sans problème, ce qui vous sera demandé dans la deuxième partie (voir ci-dessous).

Partie 1 : Entre humains :

Dans un premier temps, on considérera le cas de deux humains jouant l'un contre l'autre.

À chaque tour de boucle, votre programme appellera la fonction joueur_joueur et récupérera le résultat.

- Il faudra ensuite afficher l'état du jeu, c'est-à dire la disposition.
- À la fin de la partie, le programme devra afficher le joueur gagnant.

On envisagera un tirage au sort pour décider du joueur qui commencera la partie.

Partie 2 : Avec l'ordinateur implémentant une intelligence simple :

On envisagera un tirage au sort pour décider du joueur (Homme Ou Machine) qui commencera la partie .

Vous définirez deux fonctions :

- type_retour_a_definir joueur_humain (Type_retour parametres)
- type_retour_a_definir joueur_ordinateur (Type_retour parametres)

Typiquement :

- la fonction joueur_humain affichera à l'écran la disposition à l'instant t et Demandera à l'utilisateur d'effectuer le mouvement suivant.
- La fonction joueur_ordinateur implémentera une intelligence.

L'intelligence implémentée n'a pas besoin d'être compliquée ; La fonction Pourra, par exemple, choisir un mouvement aléatoire.

À chaque tour de boucle, votre programme appellera les fonctions joueur_humain et joueur_ordinateur alternativement et récupérera le résultat..

- Il faudra ensuite afficher l'état du jeu, c'est-à dire la disposition.
- À la fin de la partie, le programme devra afficher le joueur gagnant.

Partie 3 : avec l'ordinateur implémentant une intelligence évoluée :

On dit qu'une position est gagnante s'il existe une stratégie pour le joueur dont c'est le tour permettant de gagner quoi que fasse l'autre joueur.

De façon duale, une position est dite perdante si, quoi qu'on fasse, l'adversaire a une position gagnante.

Les positions gagnantes et perdantes peuvent être définies récursivement de la façon suivante :

1. A partir d'une position perdante, tout coup mène à une position gagnante.
2. A partir d'une position gagnante, il existe au moins un coup menant à une position perdante.

Il est donc possible d'écrire une fonction qui détermine si une position est gagnante en parsant toutes les possibilités. Une telle fonction nécessitera l'utilisation d'autres mécanismes en introduisant la notion de l'intelligence artificielle.

A notre niveau, on va essayer d'approcher cette fonction pour que l'ordinateur choisisse le coup qu'il va jouer pour gagner. On explore le maximum des coups possibles.

Enfin, il est intéressant de définir un niveau de difficulté du jeu contre l'ordinateur. Pour cela, on pourra utiliser une fonction de choix aléatoire comme suit :

Random entre 2 float

```
float     random_entre_deux_floatfloat min, float max)
{
    returr (min + 1) + (((float) rand()) /(float) RAND_MAX) * (max - (min +    1));
}
```

Random entre 2 int

```
int     random_entre_deux_intfloat min, float max)
{
   returr rand() % (max - min) + min +  1;
```

Consignes :

- Le projet consiste en l'automatisation du jeu GO avec le langage C.

- L'énoncé du projet est accompagnée d'une annexe sur la documentation ,les règles du jeu ,des liens internet pour des vidéos explicatives de ces règles ainsi qu'un ensemble de documents en pièces jointes . Vous pouvez aussi jouer en ligne ,à partir du lien fourni ,pour se familiariser avec le jeu et tester les règles.
- L'énoncé (y compris) l'annexe s'étale sur 19 pages .
- Le nombre d'étudiants par groupe ne doit pas dépasser 3 .
- Vous devrez rendre un code source commenté. Les commentaires serviront de documentation. Vous devrez indiquer le rôle de chaque fonction, ainsi que le fonctionnement général du programme et son utilisation.
- L'interface de votre programme utilisera la ligne de commande (printf etc.).
- Tout effort supplémentaire pour réaliser une meilleure intelligence évoluée sera comptabilisé. Bien entendu, la note tiendra compte de la qualité de votre intelligence implémentée. Mais n'y passez pas trop de temps tout de même. Un programme qui s'exécute correctement malgré une intelligence simpliste dans le cadre demandé rapportera plus de points qu'un programme buggé et qui se concentre juste sur l'idée d'une intelligence plus élaborée.
- Le projet doit être réalisé sous forme d'un projet Code ::Blocks , vous devrez remettre TOUS les fichiers que vous avez créez et qui sont nécessaires au fonctionnement du jeu (par mail en respectant les conditions ci-dessous).
- Les projets doivent OBLIGATOIREMENT être envoyés dans des dossiers nommés ainsi :

2023_ProjetC_SIG_Etudiant1_Etudiant2_Etudiant3.zip

  - Mail d'envoi : mariam.cherrabi@gmail.com
  - Date Limite : A communiquer ultérieurement selon les dates limites de la DACE .

- Un rapport détaillé « version papier » (1 seul rapport par groupe) doit être déposé au secrétariat du département Mathématiques, Informatiques et SIG (MIG) au Nom de Mlle CHERRABI.

La note tiendra compte de la qualité du rapport :

(voir suite) Dépôt de rapport :

Département MIG

 Date Limite : A communiquer ultérieurement selon les dates limites de la DACE . Les dates et le déroulement des soutenances seront affichées ultérieurement

1. Liens Internet :

 Documentation en ligne du jeu :

- https://www.jeudego.com
- http://www.goproblems.com/
- https://artdugo.fr/
- https://tsumego.tasuki.org
- http://www.lr-studios.net/tsumego-books-pdf/
- https://smartgo.com

1 Documentation et règles du jeu en lign

L'association Word Pair Go -2022
- http://www.worldpairgo.org

L'association Americaine Use Go -2022
- https://www.usgo.org/

La fédération internationale du go
- http://intergofed.org/

La fédération européenne du Jeu de Go
- https://www.eurogofed.org

La fédération française du Jeu de Go
- http://ffg.jeudego.org/

Championnat de france 2018
- http://cdf.jeudego.org/

La fédération japonaise du Jeu de Go : the Nihon Ki-in にほん きーいん
- https://www.nihonkiin.or.jp/english/

L'association canadienne du jeu de GO
- https://canadiango.org/

British go association
- https://www.britgo.org/

- Chaine YOUTUBE : FulguroGo - Tutoriel du Jeu de Go

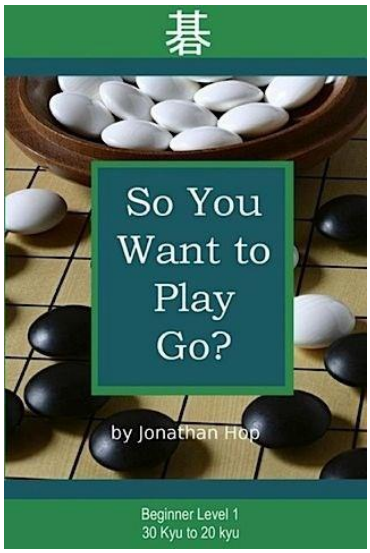
2. Pièces Jointes :

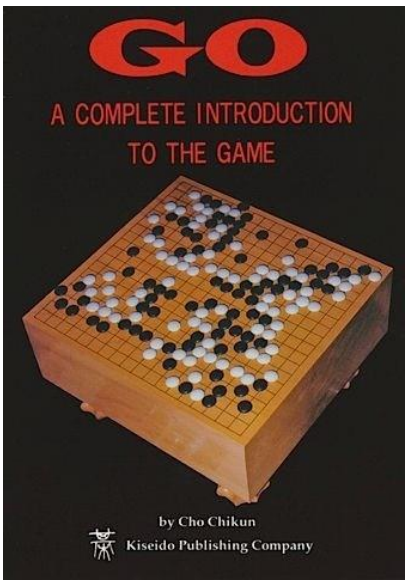
- Le Jeu de Go Principes Fondamentaux , Yilun Yang .
- Règle française du jeu de go.
- Le Jeu de GO -http://www.fqjr.qc.ca/go.html
- Initiation au go , Graded go problems for beginners - Volume1, K. Yoshinori, c1985

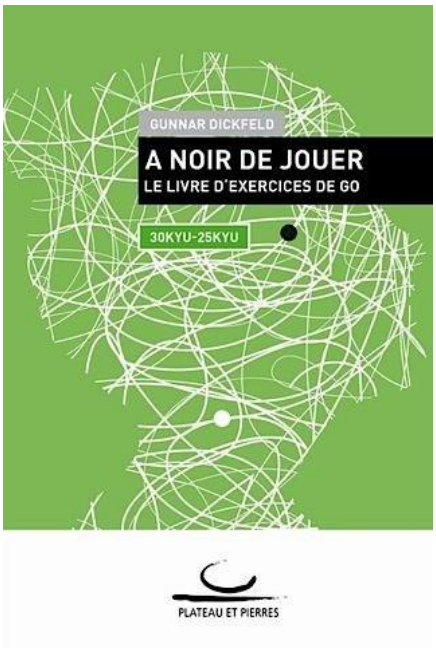
 Cho Chikun’s Encyclopedia of Life & Deat Part 1—

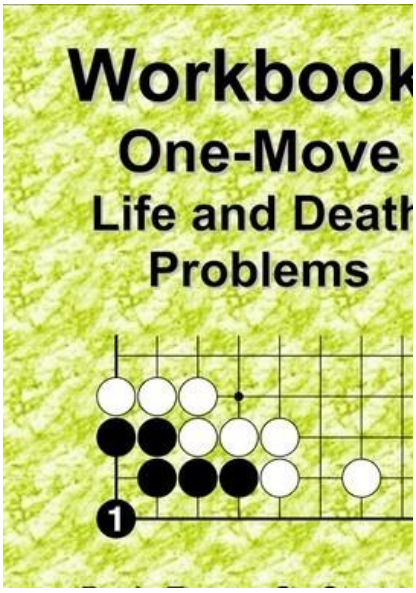
2 Livres autour du Jeu GO

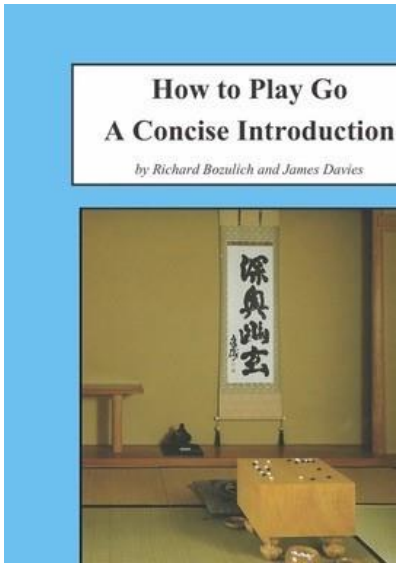
Elementary
 No.1000 Problems and answers 2000 Basic problems (Level 1)

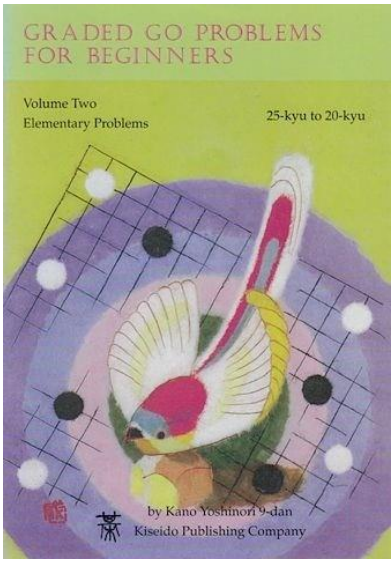


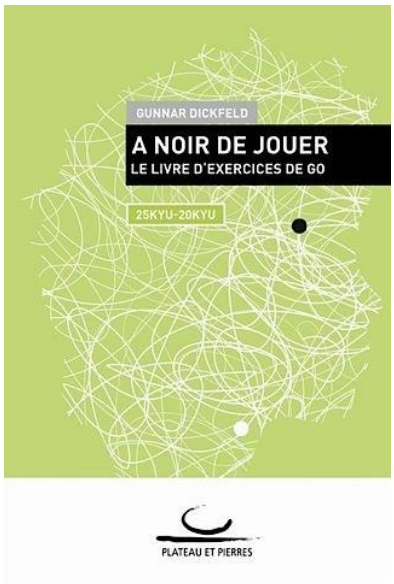


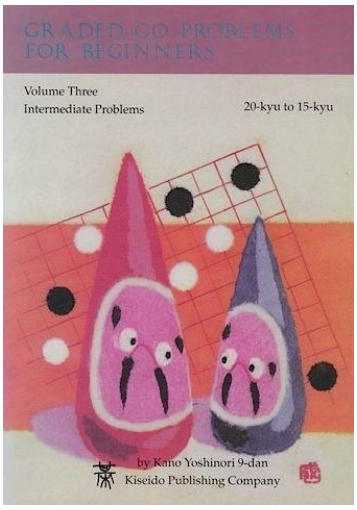

GUNNAR DICKFELD
A NOIR DE JOUER
LE LIVRE D'EXERCICES DE GO
30KYU-25KYU
PLATEAU ET PIERRES

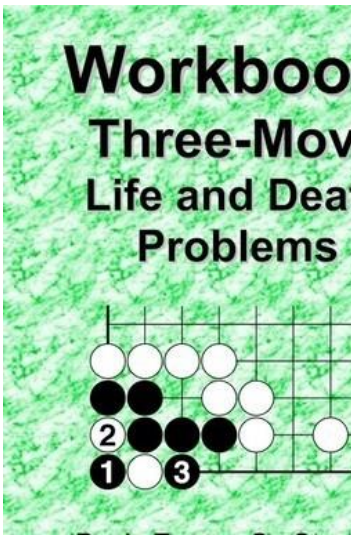

Workbook
One-Move
Life and Death
Problems
1

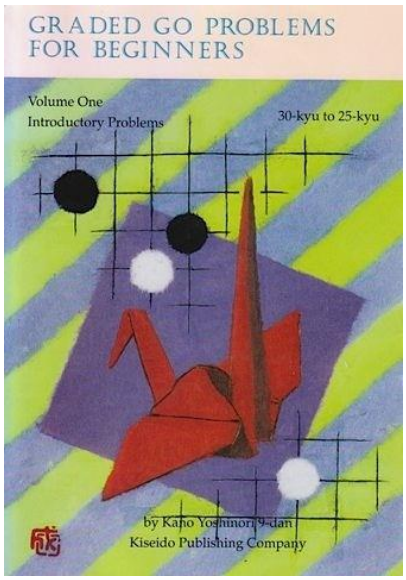

How to Play Go
A Concise Introduction
by Richard Bozulich and James Davies

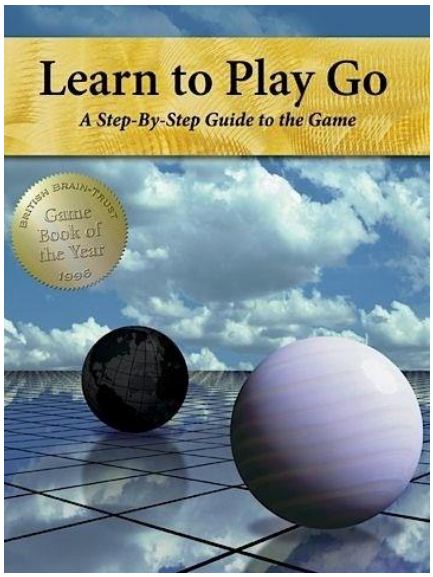

GRADED GO PROBLEMS
FOR BEGINNERS
Volume Two
Elementary Problems
25-kyu to 20-kyu
by Kano Yoshinori 9-dan
Kiseido Publishing Company

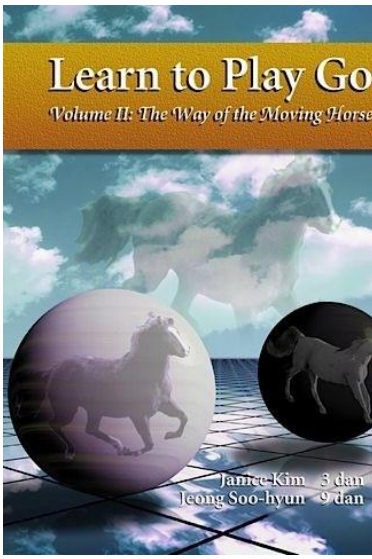

GUNNAR DICKFELD
A NOIR DE JOUER
LE LIVRE D'EXERCICES DE GO
25KYU-20KYU
PLATEAU ET PIERRES


GRADED GO PROBLEMS FOR BEGINNERS
Volume Three
Intermediate Problems
20-kyu to 15-kyu
by Kano Yoshinori 9-dan
Kiseido Publishing Company


Workboo
Three-Mov
Life and Dea
Problems


GRADED GO PROBLEMS FOR BEGINNERS
Volume One
Introductory Problems
30-kyu to 25-kyu
by Kano Yoshinori 9-dan
Kiseido Publishing Company


Learn to Play Go
A Step-By-Step Guide to the Game
BRITISH BRAIN-TRUST
Game Book of the Year
1996


Learn to Play Go
Volume II: The Way of the Moving Horse
Janice Kim 5 dan
Jeong Soo-hyun 9 dan

Fin

REMARQUE

VEILLEZ CONSULTER LE FICHIER « FONCTION.C » POUR CONSULTER TOUTES LES FONCTIONS UTILISEE