Rapport de projet C - Go
L1X Informatique - Université Paris 8

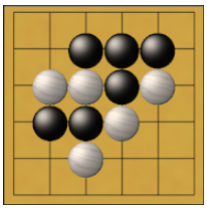
Axel LABARRE Vincent VILFEU

8 Avril 2021

Table des matières

1 Présentation du Projet

1.1 Enoncé du projet



- Sujet 24. (de * à *) Go. ( ) -**

**On peut trouver un certain nombre de problèmes de Go en ligne.
Il s'agit ici de mettre en place un système de résolution de problèmes.
La taille maximale du goban est fixée TMAX et, dans un premier temps, ne pourra pas dépasser 6 x 6.**

Voici les premières étapes de votre travail :

1. Très facile :
   - quel est le nombre de libertés de la pierre isolée (x,y) ?
   - quel est le nombre de libertés de la paire de pierres (x,y), (x', y') ?
   - quel est le nombre de libertés du triplet de pierres (x,y), (x', y'), (x", y") ?
   - la pierre (x,y) est-elle isolée ?
   - quel est le nombre de libertés de la pierre non isolée (x,y) ?
2. Quelle est la meilleure structure de données pour ce travail ?
3. Faites une fonction qui permet facilement de saisir un problème.
4. Écrivez un programme qui résout des problèmes simples de go.
5. Écrivez un programme qui résout des problèmes de go.

1.2 Règle courte du Jeu de Go

Déroulement du jeu :
Le go se joue à deux sur une grille de 19x19 lignes, avec des pierres noires et blanches. Parfois, des grilles plus petites sont utilisées, en particulier pour l'initiation.
Celui qui commence joue avec les pierres noires. À tour de rôle, les joueurs posent une pierre de leur couleur sur une intersection inoccupée de la grille ou bien ils passent (essentiellement pour indiquer qu'ils pensent la partie terminée).
Le but du jeu est d'occuper ou d'entourer avec ses pierres le plus grand nombre possible d'intersections de la grille, la partie s'arrêtant lorsque les deux joueurs pensent que ce nombre ne variera plus.
Une fois posées, les pierres ne se déplacent pas, mais elle peuvent être éventuellement capturées et retirées de la grille par l'adversaire.

2 Les choix faits

2.1 Quelle est la meilleure structure de données pour ce travail ?

Nous avons décidé de partir sur une structure composée de tableaux pour représenter le plateau à 2 dimensions (6x6).
Et des pointeurs utilisés par presque l'ensemble des fonctions de notre programme.
Ainsi que des énumérateurs pour représenter les pierres sous forme d'entier dans ce plateau(Goban).

3 Difficultés rencontrées

Notre première difficulté rencontrée fût la compréhension des règles du jeu de Go (qui possède plusieurs variantes et différentes selon les pays), méconnu à la sélection du projet par mon binôme et moi-même.
(Nous avons donc commencé par faire quelques parties de go en ligne, avant de se lancer dans ce projet)

La seconde difficulté était de trouver comment placer précisément une pierre sur le plateau, et par conséquent utiliser les coordonnées précises de l'intersection choisie.
Afin de répondre aux premières questions du projet, nous avions besoin de regarder les intersections adjacentes et avons donc définis les coordonnées : Nord, Est, Sud, Ouest et Actuel (représentant la case cible).

Concernant la représentation des pierres nous avions commencé par définir une structure, pour finalement nous tourner vers l'énumération beaucoup plus simple pour l'élaboration de notre Go.

Ensuite, on peut souligner une certaine difficulté à distinguer certaines coordonnées en particulier les coins et les bords du plateau qui comme vous le verrez nous oblige à les traiter au cas par cas alourdissant peut-être nos différentes fonctions.

La notion de groupe de pierre et de groupe adjacent n'a pas non plus été facile à gérer, nous avons utilisé la récursivité et le marquage de pierre d'un même groupe grâce à un algorithme de remplissage de pixel que nous avons adapté depuis une page wiki.

Enfin, pour la partie résolution de problèmes de go, nous avons essayé de faire une 'IA' qui scan le plateau et joue en fonction de certaine règle basique pour jouer son meilleur coup. La tâche étant un peu trop complexe nous manquions de temps pour la faire évoluer.

4 Mode d'emploi

A l'exécution du programme sur le terminal, vous aurez un 'Menu principal' et un visuel du plateau.
Ce menu se compose de différentes sections :

Questions du projet, Saisir un problème manuellement, Résoudre un problème simple de go, Jouer une partie libre contre l'IA, Vider le Goban, Quitter.

Pour Naviguer dans le menu, vous devrez entrer l'entier correspondant au menu.
Certains menus vous emmènent dans d'autres menus, mais le principe reste le même, il est assez intuitif.

1. **Questions du projet :**
   Réponses aux différentes questions du projet, par la sélection des différentes coordonnées, avec un menu de saisie manuelle de pierre sur le plateau

2. **Saisir un problème manuellement**
   Saisie manuelle de pierre sur le plateau par indications des coordonnées.

3. **Résoudre un problème simple de go**
   Sélection de problème simple de go, et leur résolution par l'IA.

4. **Jouer une partie libre contre l'IA**
   Prototype d'une partie contre l'IA

5. **Vider le Goban**
   Vider le plateau de pierre afin de le remettre à zéro.

6. **Quitter**
   Terminer le programme.

5 Impression de trace exécution

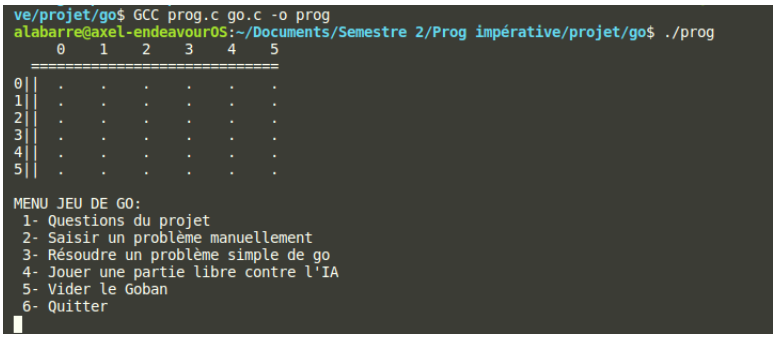


FIGURE 1 – *Execution : Menu principal*

FIGURE 1 – Execution : Menu principal

FIGURE 2 – *Menu resoudre*

FIGURE 2 – Menu resoudre

6 Listing du programme

6.1 Fichier go.h

```
1
2       #ifndef GO_H
3       #define GO_H
4
5       /* Definition de 5 constantes dont "actuel", represente la case sur laquelle
            on se trouve sur le plateau */
6       /* Et les points cardinaux pour les cases autour de celle-ci */
7       #define actuel tab->t[x * tab->col + y]
8       #define nord tab->t[(x - 1) * tab->col + y]
9       #define sud tab->t[(x + 1) * tab->col + y]
10      #define ouest tab->t[x * tab->col + (y - 1)]
11      #define est tab->t[x * tab->col + (y + 1)]
12
13      /* Definition d'un enumerateur pour differentier les pierres et une case vide
            */
14      enum content
15      {
16          EMPTY,
17          BLACK,
18          WHITE
19      };
20      typedef enum content pierre;
21
22      /* Definition d'un enumerateur pour le menu du projet */
23      enum section
24      {
25          S1 = 1,
26          S2,
27          S3,
28          S4,
29          S5,
30          S6
31      };
32      typedef enum section menu;
33
34      /* Definition d'un enumerateur pour les differentes questions du projet */
35      enum quest
36      {
37          Q0,
38          Q1,
39          Q2,
40          Q3,
41          Q4,
42          Q5,
43          Q6
44      };
45      typedef enum quest question;
46
47      /* Definition d'un enumerateur pour connaitre les differents coins du plateau
            et les cases qui ne sont pas des coins (PC) */
48      enum corner
49      {
50          PC,
51          NO,
52          SE,
53          NE,
```

```
54          SO
55      };
56      typedef enum corner coin;
57
58      /* Definition d'un pointeur de pierre */
59      typedef pierre *ptrpierre;
60
61      /* Definition d'une structure "tableau" avec notre pointeur de pierre, la
            colonne et la ligne */
62      struct tableau
63      {
64          ptrpierre t;
65          int col;
66          int lig;
67      };
68      typedef struct tableau plateau;
69
70      /* Definition d'un enumerateur pour les differents problemes */
71      typedef enum
72      {
73          PROB1 = 1,
74          PROB2,
75          PROB3,
76      } prob;
77
78
79      /* Creation d'un plateau */
80      plateau creer(int x, int y);
81
82      /* Fonction placer une pierre sur le plateau */
83      void placer_pierre(plateau *tab, int x, int y, pierre p);
84
85      /* Fonction affiche tableau */
86      void voirtab(plateau *tab);
87
88      /* Savoir si une pierre est dans un coin */
89      int est_coin(plateau *tab, int x, int y);
90
91      /* Savoir si une pierre est isolee */
92      int est_isole(plateau *tab, int x, int y);
93
94      /* Savoir le nombre de liberte de 1 seule pierre */
95      int nb_liberte(plateau *tab, int x, int y);
96
97      /* Si la pierre est a cote d'une seule autre pierre de la meme couleur
            retourne 1 sinon 0 */
98      int est_paire(plateau *tab, int x, int y, int x2, int y2);
99
100     /* Le nombre de liberte de la paire de pierre */
101     int nb_liberte_paire(plateau *tab, int x, int y, int x2, int y2);
102
103     /* Le nombre de liberte du triplet de pierre */
104     /* Fonctionne seulement dans l'ordre croissant ou decroissant ! */
105     int est_triplet(plateau *tab, int x, int y, int x2, int y2, int x3, int y3);
106
107     /* Le nombre de liberte du triplet de pierre */
108     /* Fonctionne seulement dans l'ordre croissant ou decroissant!*/
109     int nb_liberte_triplet(plateau *tab, int x, int y, int x2, int y2, int x3, int
            y3);
110
111     /* Marquage groupe pierre place en ajoutant +10 */
112     void ft_marquage(plateau *tab, int x, int y, pierre j, int m);
```

```
113
114     /* Appel de ft_marquage */
115     void marquage(plateau *tab, int x, int y, pierre j);
116
117     /* Marquage du groupe de pierre opposee adjacente    la pierre placee */
118     void opp_visiter(plateau *tab, int x, int y);
119
120     /* Marquage libertes groupe de la pierre j */
121     void marq_case_vide(plateau *tab, pierre j);
122
123     /* Somme libertes du groupe de la pierre j */
124     int somme_liberte(plateau *tab, pierre j);
125
126     /* Supprimmer un groupe entoure par l'adervsaire */
127     void eliminer(plateau *tab, pierre j);
128
129     /* Reset du tableau pour fin du tour */
130     void update_plateau(plateau *tab);
131
132     /* Saisir un probleme manuellement */
133     void saisir_probleme(plateau *tab);
134
135     /* Selectionner un probleme de Go */
136     void choix_probleme();
137
138     /* Selectionner une des questions du projet pour avoir une reponse */
139     void choix_question(plateau *tab);
140
141     /* Partie Libre contre l'IA comptage manuel */
142     void jouer(plateau *tab);
143
144     /* Fonction principale d'execution du programme */
145     void demarrer(plateau *tab);
146
147     /*Joue dans une intersection adjacente a la pierre placee en parametre*/
148     void ia(plateau *tab, int x, int y, pierre p);
149
150     /* Balayage + choix de faire un atari sur 1 pierre malgres la presence d'
            autres pierre sur le plateau */
151     void capturer_atari(plateau *tab);
152
153     /* Capture un groupe adjacent s'il n'a plus de libertes */
154     void capture_groupe(plateau *tab);
155
156     /* S'echappe si une liberte restante */
157     void echapper(plateau *tab);
158
159     /* 3eme probleme atari sur groupe */
160     void probleme_atari3(plateau *tab);
161
162     /* 2eme probleme atari sur groupe */
163     void probleme_atari2(plateau *tab);
164
165     /* Probleme atari simple */
166     void probleme_atari1(plateau *tab);
167
168     #endif
```

6.2 Fichier go.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include "go.h"
4
5   /*Creation d'un plateau*/
6   plateau creer(int x, int y)
7   {
8       plateau tab;
9       tab.col = y;
10      tab.lig = x;
11      tab.t = (ptrpierre)malloc((size_t)(y * x * sizeof(pierre)));
12      return tab;
13  }
14
15  /*fonction placer une pierre*/
16  void placer_pierre(plateau *tab, int x, int y, pierre p)
17  {
18      if (x < tab->lig && y < tab->col)
19      {
20          actuel = p;
21      }
22  }
23
24  /*fonction affiche tableau */
25  void voirtab(plateau *tab)
26  {
27      int x, y;
28      printf("     0    1    2    3    4    5\n");
29      printf("  ============================\n");
30      for (x = 0; x < tab->lig; x++)
31      {
32          printf("%d||", x);
33          for (y = 0; y < tab->col; y++)
34          {
35
36              if (actuel > 0)
37                  printf("  %d  ", actuel);
38              else if (actuel == 0)
39                  printf("  .  ");
40          }
41          printf("\n");
42      }
43  }
44
45  /*Savoir si une pierre est dans un coin*/
46  int est_coin(plateau *tab, int x, int y)
47  {
48
49      if (x + y == 0)
50          return NO;
51      if (x == tab->lig - 1 && y == tab->col - 1)
52          return SE;
53      if (x == 0 && y == tab->col - 1)
54          return NE;
55      if (x == tab->lig - 1 && y == 0)
56          return SO;
57      return PC;
58  }
59
```

```
60      /*savoir si un pierre est isolee*/
61      int est_isole(plateau *tab, int x, int y)
62      {
63          int coin = est_coin(tab, x, y);
64          if (actuel > 0)
65          {
66              /*si la pierre est dans un coin unique cas d'etre isolee*/
67              switch (coin)
68              {
69              case NO:
70                  if (sud + est == 0)
71                      return 1;
72                  break;
73              case NE:
74                  if (sud + ouest == 0)
75                      return 1;
76                  break;
77              case SO:
78                  if (nord + est == 0)
79                      return 1;
80                  break;
81              case SE:
82                  if (nord + ouest == 0)
83                      return 1;
84                  break;
85              case PC:        //Pas dans le coin
86                  if (x == 0) //au bord Nord
87                  {
88                      if (sud + est + ouest == 0)
89                          return 1;
90                  }
91                  else if (x == tab->lig - 1) //Bord Sud
92                  {
93                      if (est + nord + ouest == 0)
94                          return 1;
95                  }
96                  else if (y == 0) //Bord Ouest
97                  {
98                      if (sud + est + nord == 0)
99                          return 1;
100                 }
101                 else if (y == tab->col - 1) //Bord Est
102                 {
103                     if (sud + nord + ouest == 0)
104                         return 1;
105                 }
106                 else if (nord + sud + est + ouest == 0) //Au centre
107                 {
108                     return 1;
109                     break;
110                 }
111             default:
112                 break;
113             }
114         }
115         return 0;
116     }
117
118     /* nombre de liberte de 1 seule pierre */
119     int nb_liberte(plateau *tab, int x, int y)
120     {
121         int nb = 0;
```

```
122         int c = est_coin(tab, x, y);
123         switch (c)
124         {
125         case NO:
126             if (sud == 0)
127                 nb++;
128             if (est == 0)
129                 nb++;
130             break;
131         case NE:
132             if (sud == 0)
133                 nb++;
134             break;
135             if (ouest == 0)
136                 nb++;
137             break;
138         case SO:
139             if (nord == 0)
140                 nb++;
141             if (est == 0)
142                 nb++;
143             break;
144         case SE:
145             if (nord == 0)
146                 nb++;
147             if (ouest == 0)
148                 nb++;
149             break;
150         case PC:
151             if (x == 0)
152             {
153                 if (sud == 0)
154                     nb++;
155                 if (est == 0)
156                     nb++;
157                 if (ouest == 0)
158                     nb++;
159             }
160             else if (x == tab->lig - 1)
161             {
162                 if (est == 0)
163                     nb++;
164                 if (nord == 0)
165                     nb++;
166                 if (ouest == 0)
167                     nb++;
168             }
169             else if (y == 0)
170             {
171                 if (sud == 0)
172                     nb++;
173                 if (est == 0)
174                     nb++;
175                 if (nord == 0)
176                     nb++;
177             }
178             else if (y == tab->col - 1)
179             {
180                 if (sud == 0)
181                     nb++;
182                 if (nord == 0)
183                     nb++;
```

```
184                 if (ouest == 0)
185                     nb++;
186             }
187             else
188             {
189                 if (nord == 0)
190                 {
191                     nb++;
192                 }
193                 if (sud == 0)
194                 {
195                     nb++;
196                 }
197                 if (ouest == 0)
198                 {
199                     nb++;
200                 }
201                 if (est == 0)
202                 {
203                     nb++;
204                 }
205                 break;
206             }
207         default:
208             break;
209         }
210         return nb;
211     }
212
213
214     /*savoir si 2 pierres sont paires */
215     int est_paire(plateau *tab, int x, int y, int x2, int y2)
216     {
217         /*si les coordonnees de x,y et x2,y2 sont dans le tableau */
218         if (x >= 0 && x < tab->lig && y >= 0 && y < tab->col)
219         {
220             if (x2 >= 0 && x2 < tab->lig && y2 >= 0 && y2 < tab->col)
221             {
222                 /*si x,y est une pierre et que x2,y2 sont de la meme couleur*/
223                 if (actuel > 0 && tab->t[x2 * tab->col + y2] == actuel)
224                 {
225                     /*si x2,y2 est adjacent    x,y*/
226                     if (x2 >= x - 1 && x2 <= x + 1 && y2 >= y - 1 && y2 <= y + 1)
227                     {
228                         return 1;
229                     }
230                 }
231             }
232         }
233         return 0;
234     }
235
236     /* Le nombre de liberte de la paire de pierre */
237     int nb_liberte_paire(plateau *tab, int x, int y, int x2, int y2)
238     {
239         if (est_paire(tab, x, y, x2, x2))
240         {
241             return nb_liberte(tab, x, y) + nb_liberte(tab, x2, y2);
242         }
243         return -1;
244     }
245
```

```
246     /* Fonctionne seulement dans l'ordre croissant ou decroissant ! */
247     int est_triplet(plateau *tab, int x, int y, int x2, int y2, int x3, int y3)
248     {
249         if (est_paire(tab, x, y, x2, y2))
250         {
251             if (est_paire(tab, x2, y2, x3, y3))
252             {
253                 return 1;
254             }
255         }
256         return 0;
257     }
258
259     /* Fonctionne seulement dans l'ordre croissant ou decroissant!*/
260     int nb_liberte_triplet(plateau *tab, int x, int y, int x2, int y2, int x3, int
            y3)
261     {
262         if (est_triplet(tab, x, y, x2, y2, x3, y3))
263         {
264             return nb_liberte(tab, x, y) + nb_liberte(tab, x2, y2) + nb_liberte(
                    tab, x3, y3);
265         }
266         return -1;
267     }
268
269     /* Marquage groupe pierre place */
270     void ft_marquage(plateau *tab, int x, int y, pierre j, int m)
271     {
272         if (j == 0)
273         {
274             printf(" \n Il n'y a pas de pierre placer ici ! \n");
275             return;
276         }
277         m = 10;
278         if (j == actuel)
279         {
280             ft_marquage(tab, x - 1, y, j, actuel += m);
281             m = 0;
282             ft_marquage(tab, x + 1, y, j, actuel += m);
283             m = 0;
284             ft_marquage(tab, x, y + 1, j, actuel += m);
285             m = 0;
286             ft_marquage(tab, x, y - 1, j, actuel += m);
287             m = 0;
288         }
289     }
290
291     /* appel de ft_marquage */
292     void marquage(plateau *tab, int x, int y, pierre j)
293     {
294         ft_marquage(tab, x, y, j, 0);
295     }
296
297     /* marquage du groupe de pierre oppos e adjacente a la pierre placee */
298     void marquage_adjacent(plateau *tab, int x, int y)
299     {
300         if (actuel == 0)
301         {
302             printf(" \n Impossible sur une case vide (0) \n");
303             return;
304         }
305         if (nord != actuel && nord > 0)
```

```
306         {
307             marquage(tab, x - 1, y, nord);
308         }
309         if (est != actuel && est > 0)
310         {
311             marquage(tab, x, y + 1, est);
312         }
313         if (sud != actuel && sud > 0)
314         {
315             marquage(tab, x + 1, y, sud);
316         }
317         if (ouest != actuel && ouest > 0)
318         {
319             marquage(tab, x, y - 1, ouest);
320         }
321     }
322 
323     /*Marquage libertes groupe de la pierre j*/
324     void marq_case_vide(plateau *tab, pierre j)
325     {
326         int x, y = 0;
327         for (x = 0; x < tab->col; x++)
328         {
329             for (y = 0; y < tab->col; y++)
330             {
331                 if (actuel == j)
332                 {
333                     if (nord == 0 && x > 1)
334                     {
335                         nord = 10;
336                     }
337                     if (est == 0 && y < 5)
338                     {
339                         est = 10;
340                     }
341                     if (sud == 0 && x < 5)
342                     {
343                         sud = 10;
344                     }
345                     if (ouest == 0 && y > 1)
346                     {
347                         ouest = 10;
348                     }
349                 }
350             }
351         }
352     }
353 
354     /* somme liberte du groupe de la pierre j */
355     int somme_liberte(plateau *tab, pierre j)
356     {
357         int res, x, y;
358         res = 0;
359         marq_case_vide(tab, j);
360         for (x = 0; x < tab->col; x++)
361         {
362             for (y = 0; y < tab->col; y++)
363             {
364                 if (actuel == 10)
365                 {
366                     res++;
367                 }
```

```
368             }
369         }
370         return res;
371     }
372 
373     /* Supprimmer un groupe entoure */
374     void eliminer(plateau *tab, pierre j)
375     {
376         int x, y;
377 
378         if (somme_liberte(tab, j) == 0)
379         {
380             for (x = 0; x < tab->col; x++)
381             {
382                 for (y = 0; y < tab->col; y++)
383                 {
384                     if (actuel == j)
385                     {
386                         actuel = 0;
387                         printf("Pierre captur e en (%d,%d)\n", x, y);
388                     }
389                 }
390             }
391         }
392     }
393 
394     /*reset du tableau pour fin du tour*/
395     void update_plateau(plateau *tab)
396     {
397 
398         int x, y;
399 
400         for (x = 0; x < tab->col; x++)
401         {
402             for (y = 0; y < tab->col; y++)
403             {
404                 if (actuel == 11)
405                 {
406                     actuel = 1;
407                 }
408                 else if (actuel == 12)
409                 {
410                     actuel = 2;
411                 }
412                 else
413                 {
414                     if (actuel == 10)
415                         actuel = 0;
416                 }
417             }
418         }
419     }
420 
421     /*Saisir un probleme manuellement */
422     void saisir_probleme(plateau *tab)
423     {
424         char c;
425         int q;
426         int p = 0;
427         int x, y;
428         int fin = 1;
```

```
429         //printf("Veuillez saisir quelques pierres (\"BLACK\" = 1 ou \"WHITE\" =
                2) sur le plateau de jeu : \n");
430         while (fin != 0)
431         {
432             //voirtab(tab);
433         pierre_inconnu:
434             printf("Choisissez une pierre : \n 1- BLACK(1) \n 2- WHITE(2) \n");
435             scanf("%d", &p);
436             switch (p)
437             {
438             case BLACK:
439                 puts("Vous avez choisis une pierre: BLACK \n");
440                 break;
441             case WHITE:
442                 puts("Vous avez choisis une pierre: WHITE \n");
443                 break;
444             default:
445                 puts("Erreur, merci d'entrer l'entier coorespondant pour
                        s lectionner votre pierre \n");
446                 goto pierre_inconnu;
447             }
448         re_saisir_coord:
449             puts("Indiquer les coordonn es de la pierre sur le plateau");
450             puts("La valeur en x :");
451             scanf("%d", &x);
452             puts("La valeur en y :");
453             scanf("%d", &y);
454             if (x >= 0 && x <= tab->lig && y >= 0 && y <= tab->col)
455             {
456                 placer_pierre(tab, x, y, p);
457                 voirtab(tab);
458             }
459             else
460             {
461                 printf("ATTENTION, vous ne pouvez pas placer de pierre ici : [x =
                        %d, y = %d] \n", x, y);
462                 goto re_saisir_coord;
463             }
464             q = 1;
465             printf("Continuer ? (o/n)\n");
466             while (q == 1)
467             {
468                 scanf("%c", &c);
469                 if (c == 'o')
470                     q = 0;
471                 else if (c == 'n')
472                     return;
473             }
474         }
475     }
476
477     /* probleme atari simple */
478     void probleme_atari1(plateau *tab)
479     {
480         int pb = BLACK;
481         int pw = WHITE;
482         placer_pierre(tab, 1, 2, pb);
483         placer_pierre(tab, 2, 1, pb);
484         placer_pierre(tab, 2, 3, pb);
485         placer_pierre(tab, 2, 2, pw);
486         printf("\n");
487         voirtab(tab);
```

```
488         printf("JOUEUR = BLACK(1) -- IA = WHITE(2)\n");
489         printf("\n");
490     }
491 
492     /* 2 me  probleme atari sur groupe */
493     void probleme_atari2(plateau *tab)
494     {
495         int pb = BLACK;
496         int pw = WHITE;
497         placer_pierre(tab, 1, 1, pw);
498         placer_pierre(tab, 2, 0, pw);
499         placer_pierre(tab, 3, 0, pw);
500         placer_pierre(tab, 4, 1, pw);
501         placer_pierre(tab, 2, 1, pb);
502         placer_pierre(tab, 3, 1, pb);
503         placer_pierre(tab, 2, 2, pw);
504         printf("\n");
505         voirtab(tab);
506         printf("JOUEUR = BLACK(1) -- IA = WHITE(2)\n");
507         printf("\n");
508     }
509 
510     /* 3 me  probleme atari sur groupe */
511     void probleme_atari3(plateau *tab)
512     {
513         int pb = BLACK;
514         int pw = WHITE;
515         placer_pierre(tab, 2, 1, pb);
516         placer_pierre(tab, 2, 2, pb);
517         placer_pierre(tab, 1, 3, pb);
518         placer_pierre(tab, 2, 3, pw);
519         placer_pierre(tab, 3, 3, pb);
520         placer_pierre(tab, 4, 3, pw);
521         placer_pierre(tab, 3, 2, pw);
522         placer_pierre(tab, 4, 4, pb);
523         printf("\n");
524         voirtab(tab);
525         printf("JOUEUR = BLACK(1) -- IA = WHITE(2)\n");
526         printf("\n");
527     }
528 
529     /*S' chappe  si 1 liberte restante*/
530     void echapper(plateau *tab)
531     {
532         int x, y;
533         pierre pw = WHITE;
534         for (x = 0; x < tab->col; x++)
535         {
536             for (y = 0; y < tab->col; y++)
537             {
538                 if (actuel == pw)
539                 {
540                     marquage(tab, x, y, pw);
541                     marq_case_vide(tab, pw + 10);
542                 }
543             }
544         }
545         if (somme_liberte(tab, pw + 10) == 1)
546         {
547             for (x = 0; x < tab->col; x++)
548             {
549                 for (y = 0; y < tab->col; y++)
```

```
550                 {
551                     if (actuel == 10)
552                         actuel = pw;
553                 }
554             }
555         }
556         update_plateau(tab);
557     }
558 
559     /*Capture un groupe adjacent s'il n'a plus de libertes*/
560     void capture_groupe(plateau *tab)
561     {
562         int x, y, x_tmp, y_tmp;
563         int marq = 10;
564         pierre pw = WHITE;
565         pierre pb = BLACK;
566         for (x = 0; x < tab->col; x++)
567         {
568             for (y = 0; y < tab->col; y++)
569             {
570                 if (actuel == pw)
571                 {
572                     x_tmp = x;
573                     y_tmp = y;
574                 }
575             }
576         }
577         marquage_adjacent(tab, x_tmp, y_tmp);
578         marq_case_vide(tab, pb + marq);
579         if (somme_liberte(tab, pb + marq) == 1)
580         {
581             for (x = 0; x < tab->col; x++)
582             {
583                 for (y = 0; y < tab->col; y++)
584                 {
585                     if (actuel == 10)
586                         actuel = pw;
587                 }
588             }
589         }
590         voirtab(tab);
591         eliminer(tab, pb + marq);
592         update_plateau(tab);
593     }
594 
595     /* Balayage + choix de faire un atari sur 1 pierre malgres la pr sence d'
        autres pierre sur le plateau */
596     void capturer_atari(plateau *tab)
597     {
598         int x, y;
599         int marq = 10;
600         pierre pw = WHITE;
601         pierre pb = BLACK;
602         for (x = 0; x < tab->col; x++)
603         {
604             for (y = 0; y < tab->col; y++)
605             {
606                 if (actuel == pb)
607                 {
608                     if (nb_liberte(tab,x,y) == 1){
609                         if (nord + est + sud + ouest == 6){
610                             marquage(tab,x,y,pb);
```

```
611                         }
612                     }
613                 }
614             }
615         }
616         marq_case_vide(tab, pb + marq);
617         if (somme_liberte(tab, pb + marq) == 1)
618         {
619             for (x = 0; x < tab->col; x++)
620             {
621                 for (y = 0; y < tab->col; y++)
622                 {
623                     if (actuel == 10)
624                         actuel = pw;
625                 }
626             }
627         }
628         //voirtab(tab);
629         eliminer(tab, pb + marq);
630         update_plateau(tab);
631     }
632 
633     /* Fonction principale d'execution du programme */
634     void demarrer(plateau *tab)
635     {
636         int menu_select = 0;
637         int end = 0;
638         menu m;
639         while (!end)
640         {
641             printf("\n");
642             voirtab(tab);
643             printf("\n");
644             printf("MENU JEU DE GO: \n 1- Questions du projet \n 2- Saisir un
                    probl me manuellement \n 3- R soudre un probl me simple de go \
                    n 4- Jouer une partie libre contre l'IA \n 5- Vider le Goban \n 6-
                     Quitter \n");
645             scanf("%d", &menu_select);
646             m = (menu)menu_select;
647 
648             switch (m)
649             {
650             case S1:
651                 choix_question(tab);
652                 break;
653             case S2:
654                 saisir_probleme(tab);
655                 break;
656             case S3:
657                 choix_probleme();
658                 break;
659             case S4:
660                 jouer(tab);
661                 break;
662             case S5:
663                 for (int x = 0; x < tab->lig; x++)
664                     for (int y = 0; y < tab->col; y++)
665                         actuel = 0;
666                 break;
667             case S6:
668                 end = 1;
669                 break;
```

```
670             default:
671                 printf("Terminer \n");
672                 break;
673             }
674         }
675     }
676
677     /* Selectionner la question pour avoir une reponse */
678     void choix_question(plateau *tab)
679     {
680         question q;
681         int question_choisit = 0;
682         int x, y, x2, y2, x3, y3;
683         printf("\n");
684         voirtab(tab);
685         printf("\n");
686         printf("0- Placer des pierres sur le plateau \n");
687         printf("1- Quel est le nombre de libert s de la pierre isol e (x,y) ? \n
                ");
688         printf("2- Quel est le nombre de libert s de la paire de pierres (x,y), (
                x', y') ? \n");
689         printf("3- Quel est le nombre de libert s du triplet de pierres (x,y), (x
                ', y'), (x', y') ? \n");
690         printf("4- La pierre (x,y) est-elle isol e ?\n");
691         printf("5- Quel est le nombre de libert s de la pierre non isol e (x,y)
                ? \n");
692         printf("6- Menu principal\n");
693
694         scanf("%d", &question_choisit);
695         q = (question)question_choisit;
696
697         switch (q)
698         {
699         case Q0:
700             saisir_probleme(tab);
701             break;
702
703         case Q1:
704             voirtab(tab);
705             printf("1- Quel est le nombre de libert s de la pierre isol e (x,y)
                    ? \n");
706             puts("Indiquer les coordonn es de la pierre sur le plateau");
707             puts("La valeur en x :");
708             scanf("%d", &x);
709             puts("La valeur en y :");
710             scanf("%d", &y);
711             if (est_isole(tab, x, y))
712             {
713                 printf("R ponse : La pierre isol e en (%d,%d) a %d libert (s)\n
                        ", x, y, nb_liberte(tab, x, y));
714             }
715             else
716             {
717                 printf("Aucune pierre isol e    cet emplacement !\n");
718             }
719             break;
720
721         case Q2:
722             voirtab(tab);
723             printf("2- Quel est le nombre de libert s de la paire de pierres (x,y
                    ), (x', y') ? \n");
```

```
724             puts("Indiquer les coordonn es de la premi re pierre sur le plateau"
                    );
725             puts("La valeur en x :");
726             scanf("%d", &x);
727             puts("La valeur en y :");
728             scanf("%d", &y);
729             puts("Indiquer les coordonn es de la seconde pierre sur le plateau");
730             puts("La valeur en x :");
731             scanf("%d", &x2);
732             puts("La valeur en y :");
733             scanf("%d", &y2);
734             if (est_paire(tab, x, y, x2, y2))
735             {
736                 printf("La paire s lectionn e [(%d,%d),(%d,%d)] a %d libert (s)
                        \n", x, y, x2, y2, nb_liberte_paire(tab, x, y, x2, y2));
737             }
738             else
739             {
740                 printf("...Aucune paire n'a   t   trouv e...\n");
741             }
742             break;
743 
744         case Q3:
745             voirtab(tab);
746             printf("3- Quel est le nombre de libert s du triplet de pierres (x,y)
                    , (x', y'), (x', y') ? \n");
747             puts("Indiquer les coordonn es de la premi re pierre sur le plateau"
                    );
748             puts("La valeur en x :");
749             scanf("%d", &x);
750             puts("La valeur en y :");
751             scanf("%d", &y);
752             puts("Indiquer les coordonn es de la seconde pierre sur le plateau");
753             puts("La valeur en x :");
754             scanf("%d", &x2);
755             puts("La valeur en y :");
756             scanf("%d", &y2);
757             puts("Indiquer les coordonn es de la troisi me pierre sur le plateau
                    ");
758             puts("La valeur en x :");
759             scanf("%d", &x3);
760             puts("La valeur en y :");
761             scanf("%d", &y3);
762             if (est_triplet(tab, x, y, x2, y2, x3, y3))
763             {
764                 printf("Le triplet s lectionn  [(%d,%d),(%d,%d),(%d,%d)] a %d
                        libert (s) \n", x, y, x2, y2, x3, y3, nb_liberte_triplet(tab,
                         x, y, x2, y2, x3, y3));
765             }
766             else
767             {
768                 printf("...Aucun triplet n'a   t   trouv ...\n");
769             }
770             break;
771 
772         case Q4:
773             voirtab(tab);
774             printf("4- La pierre (x,y) est-elle isol e ?\n");
775             puts("Indiquer les coordonn es de la pierre sur le plateau");
776             puts("La valeur en x :");
777             scanf("%d", &x);
778             puts("La valeur en y :");
```

```
779             scanf("%d", &y);
780             if (est_isole(tab, x, y))
781             {
782                 printf("R ponse : Oui, la pierre s lectionn e en (%d,%d) est
                        isol e\n", x, y);
783             }
784             else if (!est_isole(tab, x, y))
785             {
786                 printf("R ponse : Non, la pierre s lectionn e en (%d,%d) n'est
                        pas is ol e\n", x, y);
787             }
788             else
789             {
790                 printf("...Aucune pierre s lectionn e...\n");
791             }
792             break;
793
794         case Q5:
795             voirtab(tab);
796             printf("5- Quel est le nombre de libert s de la pierre non isol e (x
                    ,y) ? \n");
797             puts("Indiquer les coordonn es de la pierre sur le plateau");
798             puts("La valeur en x :");
799             scanf("%d", &x);
800             puts("La valeur en y :");
801             scanf("%d", &y);
802             if (actuel > 0 && !est_isole(tab, x, y))
803             {
804                 printf("R ponse : La pierre non isol e en (%d,%d) a %d libert e
                        (s) \n", x, y, nb_liberte(tab, x, y));
805             }
806             else
807             {
808                 printf("...La pierre s lectionn e est isol e ou n'existe pas...
                         \n");
809             }
810             break;
811
812         case Q6:
813             return;
814
815         default:
816             printf("Terminer\n");
817             break;
818         }
819     }
820
821     /* S lectionner un probleme de Go */
822     void choix_probleme(void)
823     {
824
825         plateau goban;
826         goban = creer(6, 6);
827         plateau *atari;
828         atari = &goban;
829         int choix_tmp = 1;
830         prob choix;
831         char c;
832         int q;
833
834         printf("Veuillez choisir un probl me : \n");
835         printf("1- Echapper a un Atari Simple\n");
```

```
836         printf("2- Capturer en Atari\n");
837         printf("3- Capturer une pierre en Atari\n");
838 
839         scanf("%d", &choix_tmp);
840         choix = (prob)choix_tmp;
841         switch (choix)
842         {
843         case PROB1:
844             probleme_atari1(atari);
845             printf("WHITE(2) est entour e par BLACK(1) en atari et peut etre
                    supprimm e au prochain tour de BLACK(1)\n");
846             q = 1;
847             printf("R soudre ? (o/n)\n");
848             while (q == 1)
849             {
850                 scanf("%c", &c);
851                 if (c == 'o')
852                     q = 0;
853                 else if (c == 'n')
854                     break;
855             }
856             echapper(atari);
857             printf("\n");
858             voirtab(atari);
859             printf("JOUEUR = BLACK(1) -- IA = WHITE(2)\n");
860             printf("\n");
861             printf("WHITE(2) joue et s' chappe \n");
862             printf("Probl me r solu !\n");
863 
864             break;
865 
866         case PROB2:
867             probleme_atari2(atari);
868             printf("BLACK(1) est entour e par WHITE(2) en atari et peut etre
                    supprimm e au prochain tour de WHITE(2)\n");
869             q = 1;
870             printf("R soudre ? (o/n)\n");
871             while (q == 1)
872             {
873                 scanf("%c", &c);
874                 if (c == 'o')
875                     q = 0;
876                 else if (c == 'n')
877                     break;
878             }
879             capture_groupe(atari);
880             printf("\n");
881             voirtab(atari);
882             printf("JOUEUR = BLACK(1) -- IA = WHITE(2)\n");
883             printf("\n");
884             printf("WHITE(2) joue et prend le groupe BLACK(1) en un coup!\n");
885             printf("Probl me r solu !\n");
886 
887             break;
888         case PROB3:
889             probleme_atari3(atari);
890             printf(" C'est    WHITE(2) de jouer ! Bien qu'il y ait d'autres
                    pierres dans les environs, WHITE(2) devrait quand m me prendre
                    une pierre BLACK(1)\n");
891             q = 1;
892             printf("R soudre ? (o/n)\n");
893             while (q == 1)
```

```
894             {
895                 scanf("%c", &c);
896                 if (c == 'o')
897                     q = 0;
898                 else if (c == 'n')
899                     break;
900             }
901             capturer_atari(atari);
902             printf("\n");
903             voirtab(atari);
904             printf("JOUEUR = BLACK(1) -- IA = WHITE(2)\n");
905             printf("\n");
906             printf("WHITE(2) joue et prend la pierre BLACK(1) en atari!\n");
907             printf("Probl me  r solu !\n");
908             break;
909 
910         default:
911             break;
912         }
913     }
914 
915     /*Joue dans une intersection adjacente    la pierre placee en parametre*/
916     void ia(plateau *tab, int x, int y, pierre p)
917     {
918         //int i, j;
919         //int test = 0;
920         if (actuel == p)
921         {
922             marquage(tab, x, y, p);
923             marq_case_vide(tab, 11);
924             if (somme_liberte(tab, 11) > 0)
925             {
926                 if (nord == 10)
927                     nord = 2;
928                 else if (est == 10)
929                     est = 2;
930                 else if (sud == 10)
931                     sud = 2;
932                 else if (ouest == 10)
933                     ouest = 2;
934             }
935             eliminer(tab, 11);
936         }
937 
938         update_plateau(tab);
939     }
940 
941     /*Partie Libre contre l'IA comptage manuel*/
942     void jouer(plateau *tab)
943     {
944         int end = 1;
945         int pierre = 1;
946         int x, y;
947         char c;
948         int q = 1;
949         while (end)
950         {
951 
952             printf("\n");
953             voirtab(tab);
954             printf("JOUEUR = BLACK(1) -- IA = WHITE(2)\n");
955             printf("\n");
```

```
956             printf("Continuer ? (o/n)\n");
957             while (q == 1)
958             {
959                 scanf("%c", &c);
960                 if (c == 'o')
961                     q = 0;
962                 else if (c == 'n'){
963                     return;
964                 }
965             }
966             do
967             {
968                 puts("Indiquer les coordonn es de la pierre sur le plateau");
969                 puts("La valeur en x :");
970                 scanf("%d", &x);
971                 puts("La valeur en y :");
972                 scanf("%d", &y);
973                 if (x >= 0 && x <= tab->lig - 1 && y >= 0 && y <= tab->col - 1)
974                 {
975                     if (actuel == 0)
976                     {
977                         placer_pierre(tab, x, y, pierre);
978                         marquage_adjacent(tab, x, y);
979                         marq_case_vide(tab, 12);
980                         //voirtab(tab);
981                         eliminer(tab, 12);
982                         update_plateau(tab);
983                     }
984                     else
985                     {
986                         printf("\nIl y a dej  une pierre a cet emplacement. Vous
                                passez votre tour... \n");
987                     }
988                 }
989                 else
990                 {
991                     printf("ATTENTION, vous ne pouvez pas placer de pierre ici : [
                            x = %d, y = %d] \n", x, y);
992                 }
993             } while (actuel < 1);
994             puts("...L'IA Joue... ");
995             ia(tab, x, y, pierre);
996             q = 1;
997         }
998     }
```

6.3 Fichier prog.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include "go.h"
4
5   int main()
6   {
7       /*declaration plateau de jeu*/
8       int sx, sy;
9       plateau tab;
10      sx = 6;
11      sy = 6;
12      tab = creer(sx, sy);
13
14      /*creation de pierre*/
15      //pierre pj1 = BLACK;
16      //pierre pj2 = WHITE;
17
18      /*Pointeur vers tab*/
19      plateau *ptr;
20      ptr = &tab;
21
22      demarrer(ptr);
23      return 0;
24  }
```