Étude du jeu de « GO »

Steunou Loic
&
Collet Clément

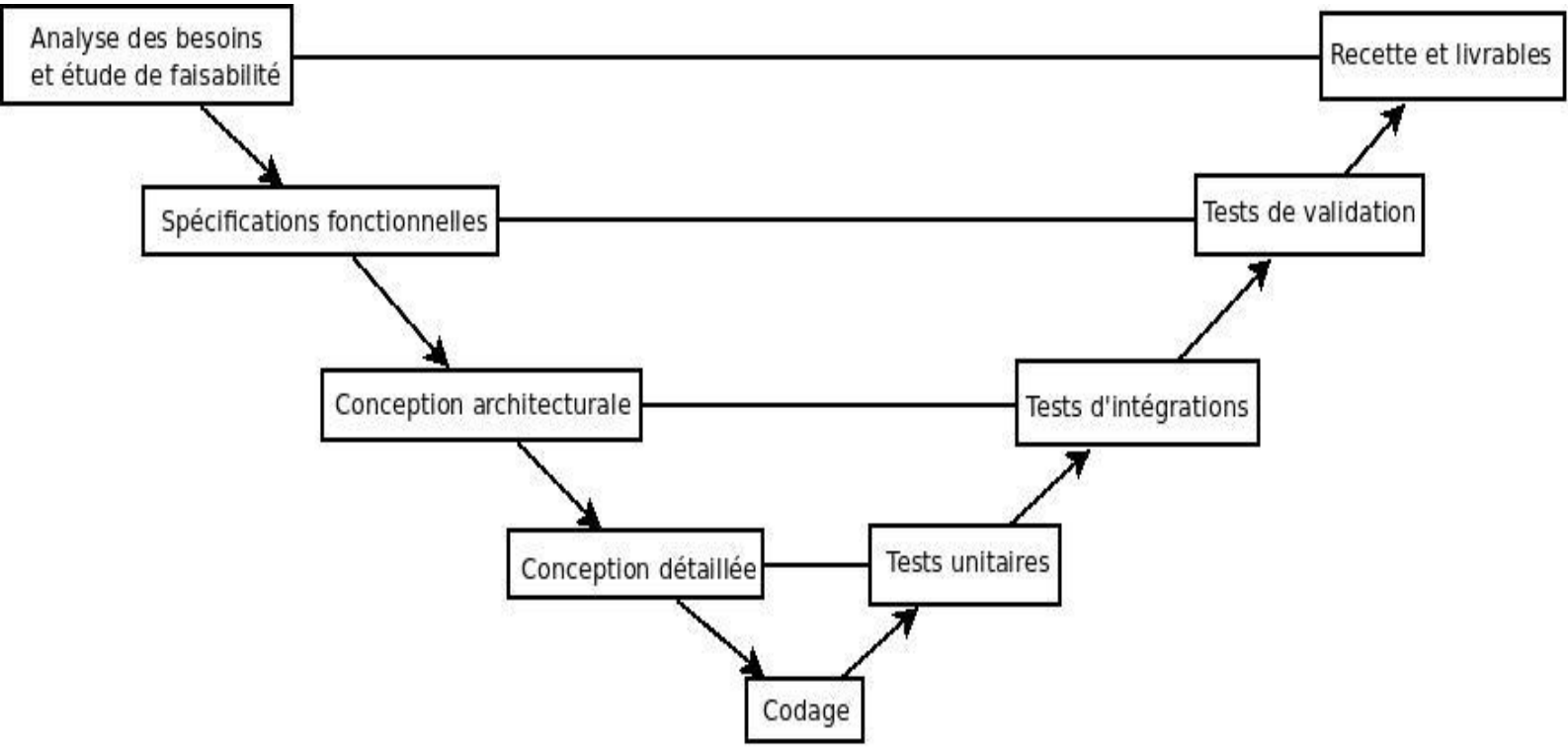
-

Introduction

Le but de ce projet informatique était de réaliser un logiciel informatique qui représente ce jeu et gère les tours de jeu, le score des joueurs et identifie un vainqueur.

Comme il faut le savoir, le codage ne représente pas tout le travail lors du développement d'un logiciel informatique. En effet, il y a un long travail à effectuer en amont et en aval comme l'analyse du sujets et de ses besoins, ses spécifications, la conception architecturale, la conception détaillée, les tests unitaire...

Pour rendre le développement d'un programme optimal, il est très conseillé de décomposer le travail selon le cycle en V :



Nous avons donc, dans notre rapport, rappelé les règles simples du jeu de go pour avoir une première idée sur comment développer un jeu de go.

Nous avons ensuite présenté le jeu comme un système avec entrées et sorties pour nous donner un objectif global.

Ensuite grâce à la notion d'un BPM pour donner un schéma du système à réaliser.

Nous sommes ensuite passé à la conception architecturale puis a la conception détaillée avant de nous lancé dans le codage.

Description du jeu et explication des règles :

L'histoire du jeu de go :

Le jeu de Go est un est ancien jeu de stratégie Chinois qui est populaire dans les pays asiatique comme la Chine, La Corée et le Japon.
Dans le reste du monde, sa cote de popularité est en pleine croissance notamment grâce à ses règles simples et ses nombreuses stratégies.

Les règles du jeu :

Le goban

Le jeu de go se joue sur un plateau, nommé goban, sur lequel est tracé un quadrillage de 19x19 lignes = 361 intersections (d'autres tailles de goban existent : 13x13 ou 9x9).

A tour de rôle, les joueurs vont placer des pierres (blanches et noires) sur les intersections du goban.

Voici quelques définitions importantes à connaître pour la suite des règles :

Chaîne, libertés et territoire

Deux intersections sont dites voisines quand elles sont sur la même ligne ou colonne et sans autre intersection entre elles.
Deux pierres sont voisines si elles occupent des intersections voisines.
Une chaîne est un ensemble de une ou plusieurs pierres de même couleur voisines de proche en proche. Les libertés d'une chaîne sont les intersections inoccupées voisines des pierres de cette chaîne.

Un territoire est un ensemble de une ou plusieurs intersections inoccupées voisines de proche en proche, délimitées par des pierres de même couleur.

Déroulement du jeu

A tour de rôle, les joueurs posent une pierre de leur couleur, le joueur qui commence joue avec les pierres noires, sur une intersection inoccupée du goban ou bien ils passent. Passer sert essentiellement à indiquer à l'adversaire que l'on considère la partie terminée.

Capture

Lorsqu'un joueur supprime la dernière liberté d'une chaîne adverse, une chaîne avec une seule liberté est dite atari, il la capture en retirant du goban les pierres de cette chaîne. En posant une pierre, un joueur ne doit pas construire une chaîne sans liberté, sauf si par ce coup il capture une chaîne adverse.

Répétition

Un joueur, en posant une pierre, ne doit pas redonner au goban un état identique à l'un de ceux qu'il lui avait déjà donné.

Fin de partie

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

De plus Blanc reçoit 7,5 points en plus car il a le désavantage de jouer en 2e. Ce bonus de point est appelé le komi. En cas d'handicap, le bonus est de 0,5 + nombre de tour d'handicap.

Présentation du jeu comme un système avec entrées et sorties

Pour présenter notre jeu comme un système avec entrées et sorties, nous avons choisis d'utiliser l'acronyme JEGO.

Pour que le jeu se déroule correctement nous avant pris l'entrée suivante, Tailleplateau (comme dit plus haut, au jeu de Go les tailles de plateau peuvent être différentes).

Une fois que le jeu se sera déroulé, alors JEGO nous donnera un vainqueur (joueur1 ou joueur2) ainsi que deux scores (score1 et score2 qui sont respectivement les scores du joueur 1 et du joueur 2).

Nous pouvons donc représenter JEGO grâce au schéma suivant :

Business Process Model (BPM)

Définition :

La BPM, Business Process Model est une notation graphique qui permet de représenter des procédures d'entreprise ou des processus de métier et d'en donner une représentation simple et compréhensible.

La BPM est un schéma comportant plusieurs entrées et sorties décrivant un projet : ses activités, ses participants, ses objectifs ainsi que différentes informations comme les ressources pour ce projet.

La BPM n'est donc pas nécessaire mais aide grandement au développement et suivit de projets.

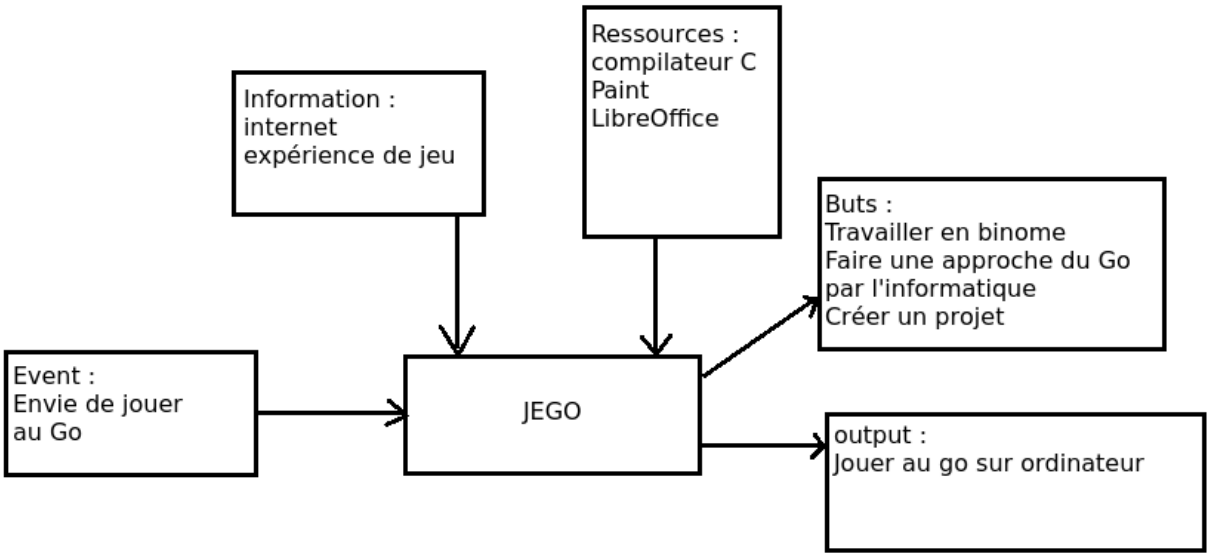
Notre BPM pour JEGO :

La BPM de JEGO comporte donc en entrées les informations suivantes :

- Une entré spécifique
- Des information
- Les ressources du projet

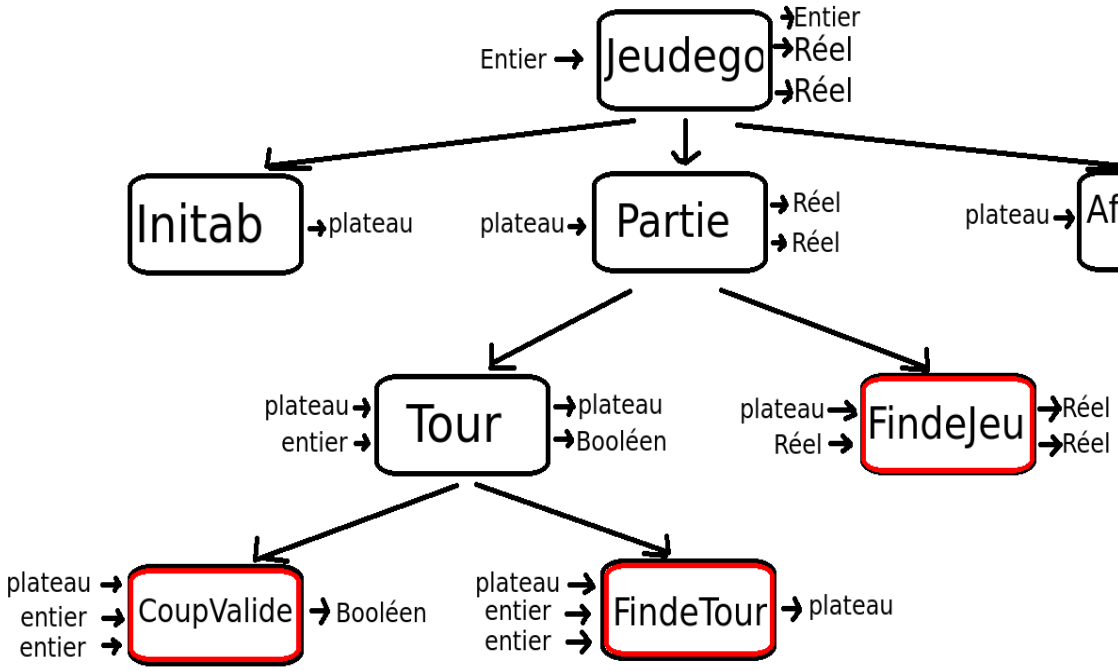
Et comporte en sortie :

- Le but du projet
- Une sortie spécifique



BPM appliqué au Jeu de Go

Analyse descendante et explications des fonctions



Signatures

- procédure Jeudego (E tailleplateau : entier ; S scoreN,scoreB : réel, indicevainqueur : entier)
- procédure Affichage (E plat : plateau)
- fonction Initab() : plateau
- procédure Partie (E plat : plateau ; S scoreN,scoreB : réel)
- procédure Tour (E / S plat1,plat2 plateau , E indicejoueur : entier ; S passer : booléen)
- module FindeJeu (E / S plat plateau , handicap : entier ; S ScoreN,ScoreB : réel)
- module FindeTour (E / S plat plateau ; E x,y : entiers)
- module CoupValide (E / S plat plateau ; E x,y entiers) : Booléen

Variables globales

Théoriquement, notre plateau est de taille variable puisqu'on demande en entrée la taille. Cependant, pour la suite, nous considérons que le plateau de jeu fait 19X19 cases. D'où la variable plateau un tableau 21X21 de caractère. On place des 'F' sur les bords qui indiquent les frontières afin d'éviter les tests sur des cases non possible. Les 'O' sont des cases vides, 'N' et 'B' des cases comprenant un pion et 'F' les limites donc.

On définit aussi verifplateau de la même taille. Ce tableau nous sert dans une de nos fonctions. Il est rempli de booléen. Dans notre code en C, il est rempli de 0 et de 1 qui nous servent de booléen.

Fonctions du Jeu de Go

En C, on ne peut pas tester ou affecter deux tableaux directement, nous avons donc créé deux fonctions permettant de manipuler les tableaux, nous ne les utiliserons pas dans le pseudo code.

- La fonction copietableau permet à un tableau de prendre la valeur d'un autre
- La fonction plateauidentique renvoie vrai si les plateau sont identiques et faux sinon.

La fonction Iniplateau :

La fonction iniplateau permet d'initialiser le goban, on utilise donc pour cela un tableau de dimension 21x21. Nous avons déclaré le type plateau qui sera utilisé dans tout le reste du programme.

Type plateau = tableau[21][21] de caractères

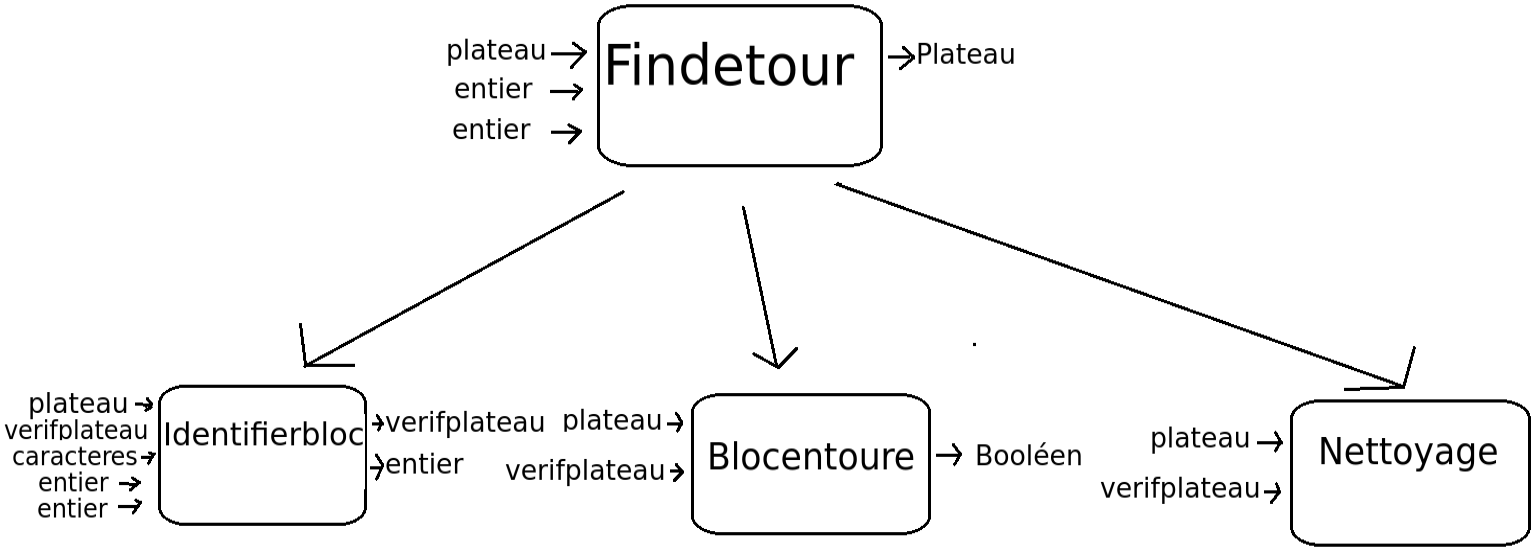
On a utilisé un tableau de dimension 21x21, car en bordure de ce tableau, on initialisera des 'F' pour frontières que l'on ne modifiera pas, et on initialise des 'O' dans le reste du tableau.

La procédure d'affichage :

Cette procédure affiche le plateau à l'écran pour que les utilisateurs de ce programme puissent le voir et donc réfléchir au coup qu'ils veulent effectuer.

Module Fin de Tour

Quand un jour a placé un pion à un endroit acceptable, il nous faut vérifier s'il a fermé un territoire en mettant ce pion, et donc « manger » des pièces qu'il conviendra d'effacer.



Les procédures/fonctions utilisées sont :
- Procédure Findetour (E/S plat : plateau ; E x,y : entiers)
On considère ici que x et y sont les coordonnées où le pion vient d'être posé.
- Fonction Remiseazeroverifplateau (plat : verifplateau) : verifplateau
- Fonction Identifierbloc (E/S verifplat : verifplateau ; E plat : plateau, lettre caractère, x : entier, y entier) réel
- Fonction Blocentoure (E plat : plateau ; verifplat : verifplateau) : booléen
- Procédure nettoyage (E/S plat : plateau ; E verifplat : verifplateau)

Voyons plus en détail le pseudo-code de identifierbloc, cette fonction part de la case (x,y) et parcourt le tableau tant qu'elle reste sur des cases comprenant le même caractère. Se faisant, elle remplie un verifplateau de vrai sur les cases du bloc en question, cela nous permet de ne pas compter deux fois la même case et d'identifier le bloc directement sur un verifplateau. A chaque fois,avant d'utiliser identifierbloc, il faut remplir le verifplateau de faux. C'est le rôle de la fonction remiseazeroverifplateau, nous ne nous pencherons pas dessus ici, son pseudocode étant très simple. De plus, on remarque que la fonction identifierbloc sort un réel alors que puisqu'il s'agit de cases d'un tableau, un entier aurait suffit. Nous avons choisi de sortir un réel car à la fin de la partie, il faudra compter les points et à cause du Komi (7,5), le score de blanc sera un réel. Donc nous avons préféré utiliser des réels dès le début pour éviter tout souci de changement de type.

```
Fonction Identifierbloc ( E/S verifplat : verifplateau ; E plat : plateau, lettre caractères, x : entier, y entier) : réel
Déclaration :
        a,b,c,d,e : réels
Debut
        Si verifplat[x][y] = vrai alors
                retourner 0
        Sinon
                Si plat[x][y] = lettre alors
                        verifplat[x][y] := vrai
                        a := 1
                        b := identifierbloc(verifplat,plat,lettre,x-1,y)
                        c := identifierbloc(verifplat,plat,lettre,x+1,y)
                        d := identifierbloc(verifplat,plat,lettre,x,y+1)
                        e := identifierbloc(verifplat,plat,lettre,x,y-1)
                        retourner a+b+c+d+e
                Sinon retourner 0
                Finsi
        Finsi
Fin
```

La fonction Blocentoure se sert d'un verifplateau où un bloc a été identifier, c'est à dire que le verifplateau contient des faux partout sauf sur les cases du bloc. Elle renvoie faux si au moins une case de ce bloc touche du vide, c'est à dire le caractère 'O'. On remarque qu'un « tant que » aurait été plus optimal, la fonction s'arrêtant au premier 'O' trouvé ; mais nous avons mis des « pour » par souci de clarté.

```
Fontion Blocentoure (plat : plateau ; verifplat : verifplateau) : booléen
Déclaration
        res,i,j : entiers
Début
    res := vrai
    Pour i:=1 à 19
        Pour j:=1 à 19
          Si verifplat[i][j]=vrai alors
              Si (plat[i-1][j]='O' ou plat[i+1][j]='O' ou plat[i][j+1]='O' ou plat[i][j-1]='O') alors
                  res := faux
              Finsi
          Finsi
        Finpour
    Finpour
Fin
```

La procédure « Nettoyage » prend en entrée un veriftableau où un bloc a été identifié, et remplace dans le plateau les cases du bloc identifié par des 'O', c'est à dire du vide.

La procédure Findetour englobe les fonctions et la procédure citées plus haut. On a joué en (x,y) et donc on se pose sur toutes les cases adjacentes à (x,y) pour verifier s'il n'y a pas eu de bloc encerclé (bloc entouré) , si c'est le cas on les retire (nettoyage).

Le Module coup valide

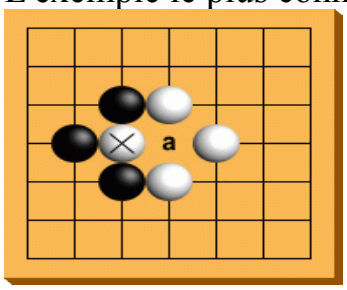
La fonction coup valide permet de savoir si l'intersection où un joueur veut placer sa pierre est autorisée par les règles du jeu de GO.

Au jeu de GO, il y a 3 façons de ne pas pouvoir placer sa pierre à l'endroit où on le souhaite :

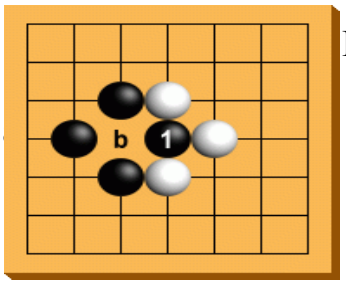
-L'intersection où l'on veut placer sa pierre contient déjà une pierre.

-L'endroit où l'on veut placer sa pierre redonne au plateau un état identique à celui qu'il avait un tour avant.

L'exemple le plus connu au go est le ko :



Ici, si Noir joue en a, alors il capturera la pierre blanche notée



Blanc ne peut donc pas rejouer en b et capturer la pierre noire car il redonnerait au plateau un état identique à celui un tour précédemment.

- Si l'endroit ou l'on veut placer notre pierre construirait une chaîne sans liberté, c'est à dire qu'elle se ferait manger instantanément. Sauf si en jouant à cette endroit, il mange d'autres pièces et donc ne serait pas manger.

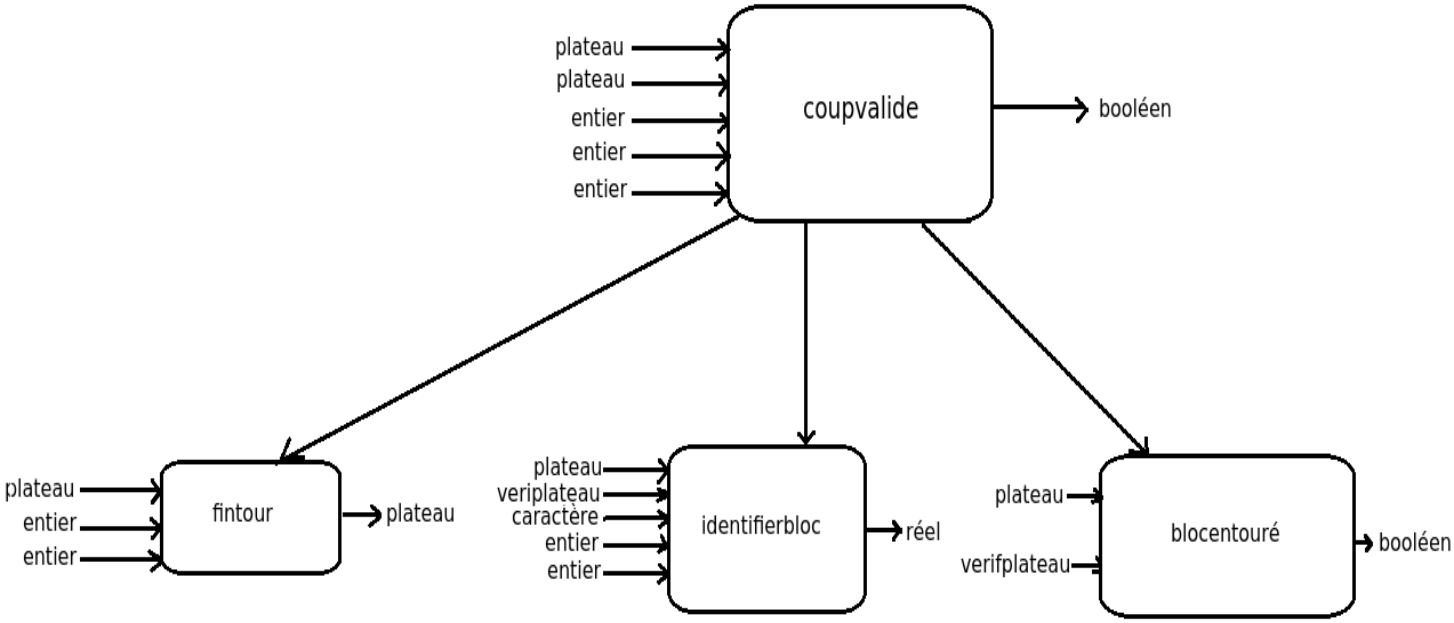
Dans ce bloc, il va donc falloir utiliser plusieurs test pour savoir si les conditions pour qu'un coup soit valide sont réunis, et si c'est le cas alors le bloc nous renverra Vrai si le coup est valide et Faux sinon.

Pour la première condition, on va juste vérifié si la case ou l'on veut jouer est bien vide dans le plateau actuel.

Ensuite, on va utiliser deux tableaux, le plateau actuel et un plateau nommé plateau ancien et nous allons voir si, après le coup joué, les deux tableaux ne sont pas identiques, si c'est la cas alors la condition n'est pas respecté.

Pour la dernière condition, cela est un peu plus compliqué car il faut vérifié que, lorsque l'on joue notre pierre, on ne créé pas une chaîne sans libertés, c'est à dire un bloc entouré. Dans ce cas le coup est non valide.

Il y a cependant une exception, créer un chaîne sans libertés pour capturer une chaîne adverse. Il faut donc vérifier si, l'un des blocs qui nous entourent, est lui même entouré une fois notre pierre jouée. Dans ce cas, le coup est valide.



Voici la signature, en pseudo code de la fonction coup valide :

Fonction Coupvalide(plateau plat1, plat2 ; entier indicejoueur, x, y) : Booléen

Cette fonction va faire plusieurs tests pour savoir si le coup est valide, elle renverra vrai si le coup est valide, et faux si le coup n'est pas valide.

Description des variables :

-platancien de type plateau : C'est le dernier plateau valide, par exemple, si c'est au tour de noir de jouer et que son coup est valide alors platancien sera le tableau avant ce coup, il permettra de savoir si le plateau n'est pas dans un état identique à précédemment (cf exemple du ko).
-platactuel de type plateau : C'est le plateau avant que l'on test si un coup est valide
-indicejoueur de type entier : indice joueur vaut 1 ou 2, 1 si c'est au noir de jouer et 2 si c'est au blanc
-x et y de type entier : les coordonnées du coup que l'on veut tester

Liste des fonctions utilisées :

-**Fonction** remiseazeroverifplateau(verifplateau plat) : verifplateau
-**Fonction** identifierbloc(plateau plat ; verifplateau verifplat ; caractère lettre ; entier x, y) : Entier
-**Fonction** blocentoure(plateau plat ; verifplateau veriplat) : Booléen
-**Fonction** findetour(plateau plat, entier x, y) : plateau
Ces trois fonctions que l'on a décrit plus haut nous permettent de savoir si un bloc est, ou non, entouré.

Fonction Coupvalide(plateau plat1, plat2 ; entier indicejoueur, x, y) : Booléen
Déclaration :
entouré : Booléen
plat3 : plateau
verifplat : verifplateau
a : caractère
Début
Si plat2[x][y] ≠ 'O' alors
retourner Faux
{Si l'intersection est déjà remplit on ne peut donc pas jouer ici : le coup est non valide}
Sinon
plat3 := plat2
Si indicejoueur = 1 alors
a='N'
Sinon
a='B'
FinSi
plat3[x][y]=a
findetour(plat3,x,y)
{la commande ci-dessus permet de simuler que ce coup a été jouer, pour éliminer les pierres, cela est nécessaire pour que platancien et plat3 puissent être identiques. Cf exemple du ko}
Si plat1=plat3 alors
retourner Faux
{si ils sont identiques alors : le coup est non valide}
Sinon
remiseazeroverifplateau(verifplat)
identifierbloc(plat3,verifplat,a,x,y)
coupval=blocentoure(plat3,verifplat)
Si coupval=0 alors
retourner Vrai
{Si le bloc n'est pas entouré une fois le coup joué alors le coup est valide}
Sinon
remiseazeroverifplateau(verifplat) ;
s=identifierbloc(plat3,verifplat,a,x+1,y) ;
coupval1=blocentouré(plat3,verifplat) ;

remiseazeroverifplateau(verifplat) ;
s=identifierbloc(plat3,verifplat,a,x-1,y) ;
coupval2=blocentouré(plat3,verifplat) ;

remiseazeroverifplateau(verifplat) ;
s=identifierbloc(plat3,verifplat,a,x,y+1) ;
coupval3=blocentouré(plat3,verifplat) ;

remiseazeroverifplateau(verifplat) ;
s=identifierbloc(plat3,verifplat,a,x,y-1) ;
coupval4=blocentouré(plat3,verifplat) ;

{grâce à ces commandes, on vérifie que, si le bloc créé par notre coup est entouré alors, ce n'est que car il y a une possibilité de capture. En effet si il y a une possibilité de capture, alors le bloc créé ne sera entouré que de blocs ennemis dont un ou plusieurs d'entre eux seront entourés et donc capturés}

Si (coupval1=0 et coupval2=0 et coupval3=0 et coupval=0) alors
retourner Faux
{en effet si aucun d'entre eux n'est entouré alors notre pierre, une fois jouée formera une chaîne sans libertés, le coup n'est pas valide}
Sinon
retourner Vrai
{au contraire si un ou plusieurs blocs qui entourent la chaîne créée sont entourés alors, le coup est valide et les blocs seront capturés}
FinSi
FinSi
FinSi
Fin

La procédure tour

La procédure tour, est la procédure qui appelle le module CoupValide ainsi que le module FinDeTour et qui permet de générer un tour. Cette fonction va donc prendre, en entrée, deux plateau, le plateau actuel et le plateau que l'on utilise pour la règle du ko (platancien) (cf fonction coupvalide).

Cette procédure va, ensuite ressortir deux nouveaux plateaux, platancien qui prendra la valeur de platactuel et un autre plateau, celui auquel on a ajouté le nouveau coup.

La procédure ressortira aussi un booléen, Vrai si le joueur a passé, dans ce cas les plateaux ne seront pas modifiés, et Faux si le joueur n'a pas passé et que les plateaux ont été modifié.

La signature de la procédure tour en pseudo-code :

Procédure Tour(E : indicejoueur : entier ; / E/S : platactuel , platancien : plateau : / S : passe : Booléen)

Liste des fonctions et procédures utilisées :
-**procédure** afficheplateau(E : plat : plateau)
-**Fonction** coupvalide(plateau plat1, plat2 ; entier indicejoueur, x, y) : Booléen
-**procédure** copietableau(E : plat1, plat2 : plateau)
-**Procédure** Findetour (E/S plat : plateau / E x,y : entiers)

Voici la procédure Tour en pseudo-code :

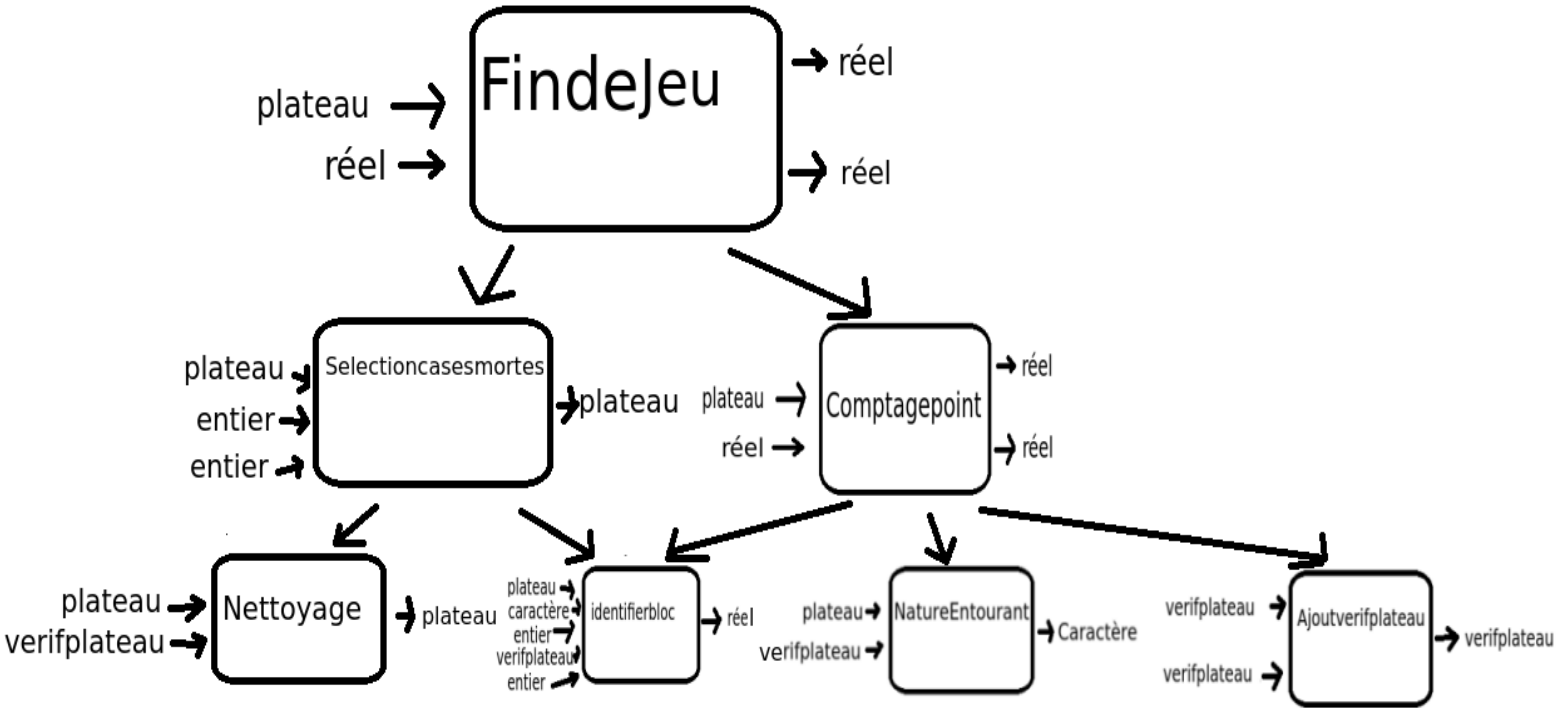
Procédure Tour(E : indicejoueur : entier ; platancien : plateau E/S : platactuel : plateau S : passe : Booléen)
Déclaration :
caractère : a
entier : action=[0;1], x, y
Booléen : coup ;
Début
    coup := FAUX
    afficheplateau(platactuel)
    écrire('que voulez vous faire ? Jouer : 1 / Passer : 0')
    Lire(action)
    Si (action=0) alors
        passe = VRAI
{si le joueur décide de passer, alors on renvoi vrai et le plateau n'est pas modifié}
    Sinon
        Tant que (coup=FAUX) faire
            écrire('où voulez vous jouer ?')
            Lire(x)
            Lire(y)

```
                    coup:= coupvalide(platancien,platactuel,indicejoueur,x,y)
```
{si le joueur décide de jouer alors on lui demande de rentré des coordonnées valides pour sont coup}

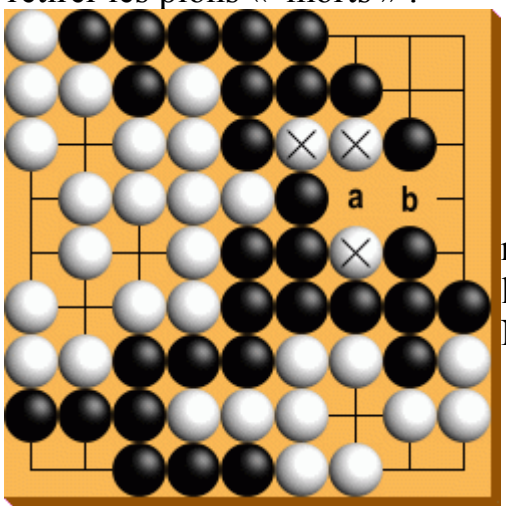
```
            FinTantque
            copieplateau(platactuel,platancien)
            Si (indicejoueur=1) alors
                a='N'
            Sinon
                a='B'
            FinSi
            platactuel[x][y]=a
            findetour(platactuel,x,y)
            passe=FAUX
```
{Si le coup est valide alors on modifie le plateau actuel et passe prend prend la valeur faux}

```
        FinSi
Fin
```

Module Findejeu



A la fin du jeu, il faut compter les points de chacun. Pour cela, il convient de retirer les pions « morts » .



Dans l'exemple à gauche, les 3 pions blancs marqués sont « morts » car si Noir joue en « a », il les mange. Si Blanc joue en « a » pour se sauver, Noir joue en « b » et mange les 4 pions blancs.

Puis de compter combien de pion chacun a sur le plateau ainsi que la taille des ensembles « controlés » (entouré). Et enfin d'ajouter le Komi.

Les procédures/fonctions utilisées sont :
- Procédure Findejeu (E plat : plateau, handicap : réel ; S scoreN, scoreN : réels)
- Fonction Remiseazeroverifplateau (plat : verifplateau) : verifplateau
- Fonction Identifierbloc (E/S verifplat : verifplateau ; E plat : plateau, lettre caractère, x : entier, y entier) réel
- Procédure Comptagepoint (E plat : plateau , handicap : réel ; S scoreN,scoreB :

réel)
- Procédure Nettoyage (E/S plat : plateau ; E verifplat : verifplateau)
- Fonction Natureentourant(plat : plateau ; verifplat : verifplateau) : caractère
- Fonction Ajoutverifplateau(verifplat1,verifplat2 : verifplateau) : verifplateau
- Procédure Selectionblocsmorts(E/S plat : plateau)

Remiseazeroverifplateau, Identifierbloc et Nettoyage ont déjà été expliqués dans le module Findetour.

La fonction Natureentourant prend en entrée le plateau et un verifplateau où un bloc de vide (un bloc de 'O') a été identifié. Elle sert donc à savoir qui « entoure » ce bloc. Elle parcourt le bloc et s'arrête lorsqu'elle a trouvé un 'N' ou un 'B'.

Fontion Natureentourant (plat : plateau ; verifplat : verifplateau) : caractère
Déclaration
i,j, : entier
fin : booléen
Début
```
i:=1 ;
j:=1 ;
fin:=faux
Tant que ( fin = faux ) et (i<20)
    Si verifplat[i][j]=1 alors
        Si (plat[i-1][j]='N' ou plat[i-1][j]='B') alors
            fin = vrai
            retourner plat[i-1][j]
        Sinon
            Si (plat[i+1][j]='N' ou plat[i+1][j]='B') alors
                fin = vrai
                retourner plat[i+1][j]
            Sinon
                Si (plat[i][j-1]='N' ou plat[i][j-1]='B') alors
                    fin = vrai
                    retourner plat[i][j-1]
                Sinon
                    Si (plat[i][j]='N' ou plat[i][j-1]='B')  alors
                        fin = vrai
                        retourner plat[i][j-1]
                    Finsi
                Finsi
            Finsi
        Finsi
    Finsi
    j:=j+1
    Si j=20 alors
```

```
            j:=1
            i :=i+1
        Finsi
FinTantque
Fin
```

La fonction Ajoutverifplateau permet de sortir un verifplateau contenant tous les vrais de verifplat1 à leur positions, plus ceux de verifplat2. Cette fonction va nous servir dans la procédure Comptagepoint. Son pseudo-code étant évident, nous ne nous pencherons pas davantage dessus.

La procédure Comptagepoint part d'un plateau à l'état totalement final, c'est à dire que les cases mortes ont été retirés et que les joueurs on joué au maximum qu'ils le pouvaient. Autrement dit, il est **obligatoire** que chaque territoire de vide ('O) soit entouré par un seul et même joueur. Cela afin de lui attribuer les points correspondants à la taille de bloc de vide. Le réel en entrée correspond à handicap éventuel. On ajoute 0.5 +nombre de coups d'avance de noir au score de blanc. Si l'handicap vaut 0, on attribue 7.5 points de Komi au score de blanc.

L'idée de cette procédure est d'utiliser 2 verifplateau. On parcourt le tableau, comptant les blocs de pions, et attribuant ceux de vide au bon joueur. On utilise pour cela le 2ème verifplateau qui est réinitialisé à chaque case. Pour éviter de compter deux fois les cases d'un même bloc, après chaque itérations, on ajoute le verifplateau2 à verifplateau1 et à chaque fois on teste si la case appartient à un bloc déjà testé.

Dans le cas du C, et donc de notre programme, nous avons défini 2 Comptagepoints, une pour les points de Noir et une pour les points de Blanc (les fonctions ne pouvant sortir qu'un argument à chaque fois). Cependant, le pseudo-code qui suit est bien plus général et pertinent .

```
Procédure Comptagepoint ( E  plat : plateau , handicap : réel ; S scoreN,scoreB : réel )
Déclaration
        scoreN, scoreB, score : réels
        i, j : entiers
        verifplat1, verifplat2 : verifplateau
        a : caractère
Début
        scoreN:=0
        Si handicap=0 alors
                scoreB=0.5+handicap
        Sinon
                scoreB=7.5
        remiseazeroverifplateau(verifplat1)
        Pour i de 1 à 19
                Pour j de 1 à 19
                        Si verifplat1[i][j]=0 alors
```

```
                        score :=0
                        remiseazeroverifplateau(verifplat2)
                        a :=plat[i][j]
                        score :=identifierbloc(plat,verifplat2,a,i,j)
                        Si a='N' alors
                                scoreN :=scoreN+score
                        Finsi
                        Si a='B' alors
                                scoreB :=scoreB+score
                        Finsi
                        Si a='O' alors
                                a :=natureentourant(plat,verifplat2)
                                Si a='N'
                                        scoreN :=scoreN+score
                                Finsi
                                Si a='B'
                                        scoreB :=scoreB+score
                                Finsi
                        Finsi
                        ajoutverifplateau(verifplat1,verifplat2)
                Finsi
        Finpour
Finpour
Fin
```

La procédure Selectionsblocmorts va demander aux joueurs de sélectionner les blocs de cases « mortes » (en réalité une case de chaque bloc mort suffira). On part du principe que les deux joueurs sont d'accord, s'ils ne le sont pas, il faudra qu'ils continuent à jouer. On va ici considérer qu'ils sont d'accord. De plus, on part ici du principe que l'utilisateur va répondre des valeurs acceptables. On ne traite pas les cas où les réponses sont des non-réponses (coordonnées hors du tableau, caractère à la place des chiffres demandés etc).

```
Procédure Selectionsblocmorts (E/S plat plateau)
Déclaration
        verifplat : verifplateau
        x,y,réponse : entier
        a : caractère
Début
        Répéter
                afficher('Voulez vous retirer des blocs de cases mortes ? 0 pour non et 1 pour vrai')
                lire(réponse)
                Si réponse=1 alors
                        afficher('x ? y?)
```

```
                lire(x,y)
                remiseazeroverifplateau(verifplat)
                a:=plat[x][y]
                identifierbloc(plat,verifplat,a,x,y)
                nettoyage(plat,verifplat)
            Finsi
    Jusqu'à réponse=0
Fin
```

La procédure Findejeu englobe les fonctions/procédures citées plus haut.

```
Procédure Findejeu ( E plat : plateau, handicap : réel ; S scoreN, scoreN : réels )
Déclaration
    scoreN,scoreB : réels
Début
    scoreN:=0
    scoreB:=0
    Selectionsblocmorts(plat)
    Comptagepoint(plat,handicap,scoreN,scoreB)
                                        Fin
```

La procédure partie

La procédure partie est la procédure qui va être appelée par le programme principal et qui va gérer la partie. Cette procédure va donc appelée la procédure tour et le module FinDeJeu.

Cette procédure va donc prendre en entré un tableau que l'on aura initialisé dans le programme principal.

Procédure partie(E : plat : plateau)

Elle va ensuite demandé aux utilisateurs si ils veulent jouer une partie avec handicap.

Si c'est le cas, elle va faire jouer le joueur 1 autant de tour que necessaire. Ensuite, la procédure va a tour de rôle faire jouer les deux joueurs jusqu'à ce que les deux passent. La procédure va ensuite calculer le score et l'afficher grâce à la procédure findepartie.

Les fonctions et procédures utilisées par cette procédure :
-**procédure** iniplateau(E : plat : plateau)
-**procédure** Tour(E : indicejoueur : entier ; platancien : plateau / E/S : platactuel : plateau / S : passe : Booléen)
-**procédure** findepartie(E : plateau : plat ; handicap : entier)

La fonction partie en pseudo-code :

Procédure partie(E : plat : plateau)
Déclaration :
plateau : platavant
entier : indicejoueur, passe1, passe2, handicap, i
Début
    iniplateau(platavant) ;
    indicejoueur=1
    passe1=0
    passe2=0
    écrire('handicap ?')
    Lire(handicap)
    Si (handicap ≠ 0) alors
        Pour i = 1 à handicap faire
            tour(platavant,plat,indicejoueur)
        Finpour
{si il s'agit d'une partie à handicap, noir dois jour un nombre de tour de plus que blanc au début de la partie. D'où la boucle pour qui permet cela}
    FinSi
    Tant que (passe1≠1 ou passe2≠1) faire

```
            Si (indicejoueur=1) alors
                écrire('noir c'est votre tour')
                passe1=tour(platavant,plat,indicejoueur)
                indicejoueur=2
{si c'est à noire de jouer alors l'indice du joueur est à 1 et on le passe à 2 pour que ce
soit à blanc de jouer et inversement}
            Sinon
                printf('blanc c'est votre tour')
                passe2=tour(platavant,plat,indicejoueur)
                indicejoueur=1
            FinSi
        FinTantque
        findepartie(plat,handicap)
Fin
```

Conclusion

Nous avons beaucoup appris durant ce projet et pris beaucoup de plaisir à tout d'abord décortiquer le jeu de go et le décomposer en fonctions et procédure et ensuite de le coder en C. La difficulté vient des nombreux cas particuliers à gérer. Pour chaque fonction / procédure, nous avons du tester tous ces cas.

De manière générale, la fonction la plus complexe et intéressante est « identifierbloc » car il s'agit d'une fonction récursive. Cette manière de se servir d'un tableau de booléen pour finir que nous n'avons pas déjà testé nous semble la plus facile pour parcourir le tableau facilement.

Le jeu de Go, par sa complexité, est passionnant à comprendre et réaliser l'analyse descendante est le véritable intérêt de ce projet, devant le simple codage.

Par rapport à ce dernier, nous pouvons l'améliorer en y incorporant une interface graphique plus attrayante. Nous utilisons actuellement le terminal ce qui est assez vite fatiguant pour les yeux et assez peu ludique. De plus, devoir à chaque fois rentrer les coordonnées des cases à sélectionner est assez pénible. Utiliser la SDL nous permettrait de simplement « cliquer » sur la cases pour la sélectionner, les cases étant en plus des véritables images de pions ou d'intersection.