Jeu de Go

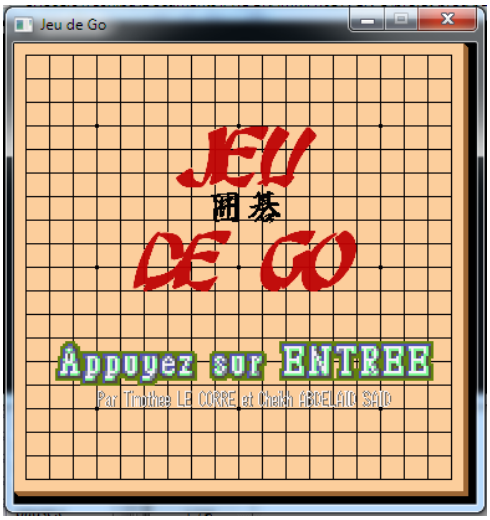
SOMMAIRE

I. Analyse fonctionnelle générale

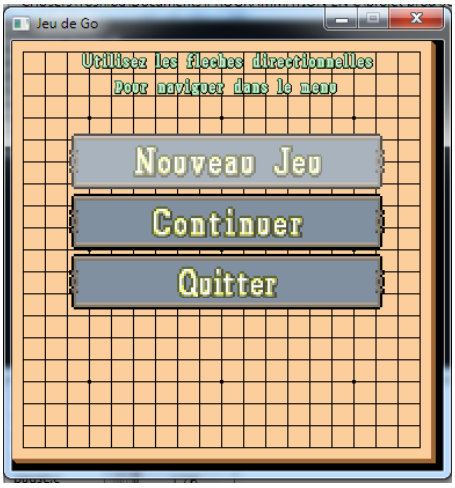
chargement de partie.txt	FAIT !	• pour charger le tableau aucune difficulté sauf qu'on n'avait pas utilisé la bonne fonction au départ • pour charger le tour du joueur aucune difficulté • pour charger si le joueur blanc est IA ou pas aucune difficulté	2 jours
sauvegarde de partie.txt	FAIT !	• pour charger le tableau aucune difficulté • pour charger le tour du joueur aucune difficulté • pour charger si le joueur blanc est IA ou pas aucune difficulté	2 jours
permettre au joueur de jouer contre un humain	FAIT !	• assigné le choix du joueur blanc a une variable .facile	1 jour(pour faire la fonction pour placer le pion)
permettre au joueur de jouer contre une machine	FAIT !	• pour reconnaitre que le joueur va jouer contre une machine pas de difficulté • pour faire l'IA quelques difficultés rencontré mais surmonté avec une IA de plus en plus performante	3 jours
Interdire au joueur de placer une pierre s'il en a pas le droit	FAIT !		
Compte les points	FAIT !	• difficile de distinguer les territoires	
Identifie la fin de la partie et donne le gagnant	FAIT !	• plutôt simple à gérer pour les deux joueurs • pour l'IA une petite complication qui nous a permis de constater une erreur dans la fonction placement possible	quelques heures

Le programme est organisé de cette manière-là :

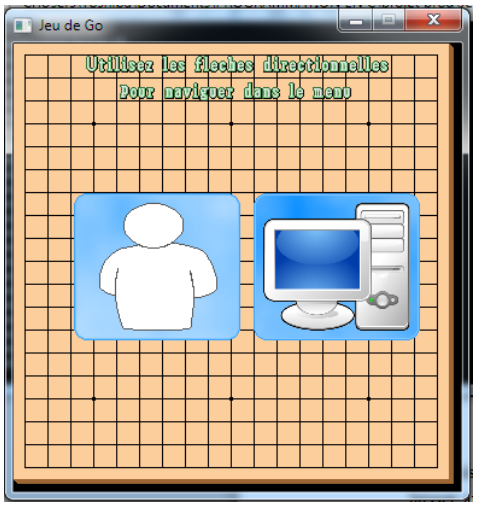
- Affichage de l'écran titre :
    - Appuie sur ENTREE pour commencer le jeu



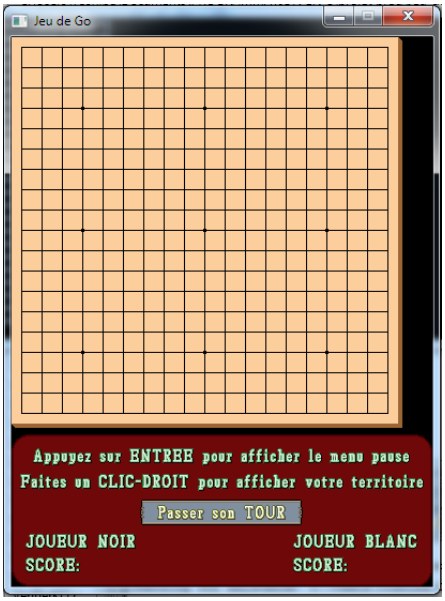
- Affichage du MENU :
  - Navigation avec les flèches directionnelles et Appuie sur ENTREE pour valider.
  - Avec un curseur symboliser par un éclairement de la case où pointe le choix du joueur
  - Choix entre :
    - Nouveau jeu : démarre une nouvelle partie
    - Continuer : charge une partie sauvegardée dans un fichier .Txt
    - Quitter : quitte le programme



- Si le joueur choisi Nouveau jeu :
  - Affichage de deux choix avec les flèches directionnelles et Appuie sur ENTREE pour valider :
    - icone personnage : pour jouer contre un joueur non IA
    - icone ordi : pour jouer contre l'IA
    - Eclairement de l'icône où pointe le choix du joueur
  - Eclairement de l'icône où pointe le choix du joueur

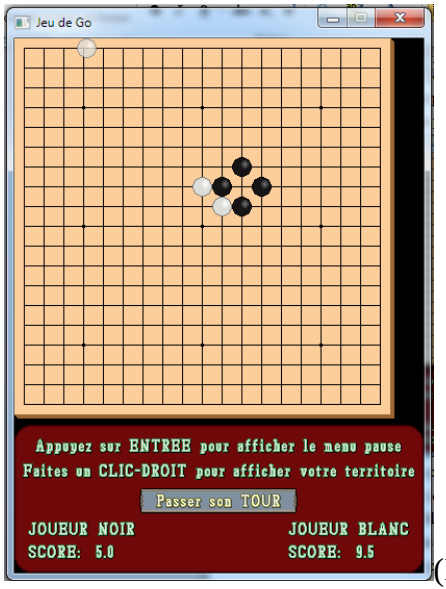


– Une fois tous les paramètres précédents réglés, la taille de l'écran est augmenté pour laisser apparaitre le plateau (19*19) ainsi qu'une « case » sous le plateau avec les scores, le bouton « passer son tour » et info sur des touches. La raison du changement d'écran est qu'on ne voyait pas quoi mettre sous le plateau pour l'écran titre, le menu et le choix de l'adversaire.



– Phase de jeu
  - Début de la boucle du jeu (tant qu'aucun joueur ne gagne ou tant que le joueur ne quitte pas le jeu)
  - Le joueur noir joue, son pion apparait sur le plateau en transparence mais n'est pas placé il se déplace avec la souris, il appui sur CLIC-GAUCHE pour placé son pion (si il en a le droit, il peut y avoir des cas où il peut pas)

- Le programme regarde si une capture est possible et l'effectue

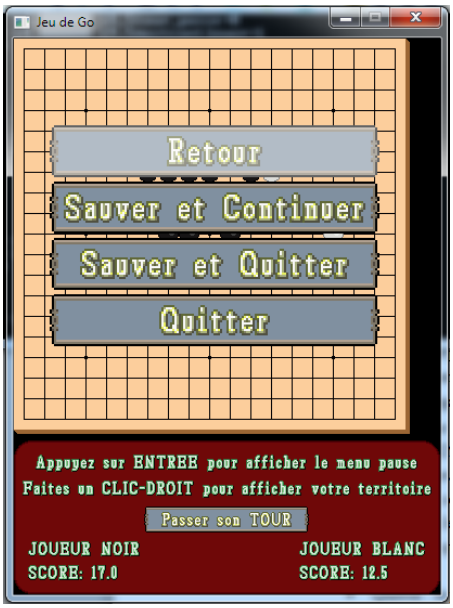


(Ici entre les 4 noirs il y avait un pion blanc qui a été capturé

(la capture marche aussi avec les chaines) :

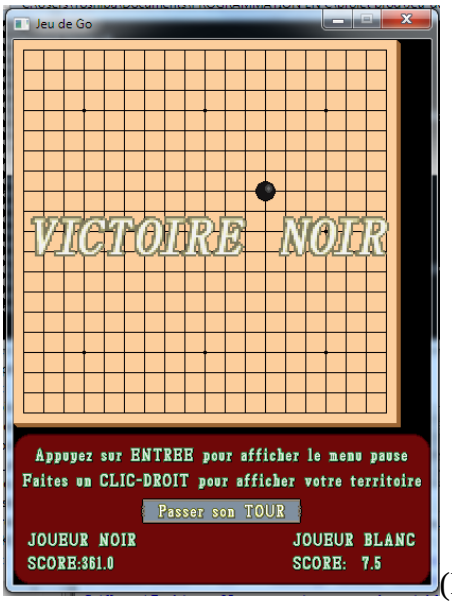
(Ici dans les intersections entre l'ensemble des pions noir il y avait des pions blancs qui ont été capturés)

- Le joueur blanc effectue la même action (si le blanc est contrôlé par un joueur le blanc placera le pion comme le joueur noir si il est contrôlé par l'IA il sera placé automatiquement par la fonction gérant l'IA)
- Le programme regarde si une capture est possible et l'effectue
- Au tour de chaque joueur le joueur a la possibilité d'appuyer sur ENTREE pour ouvrir le MENU PAUSE (organisé comme le MENU) :
  - Retour : reviens au jeu
  - Sauver et continuer : le programme effectue une sauvegarde mais ne quitte pas le jeu
  - Sauver et quitter : le programme effectue une sauvegarde et quitte le jeu
  - Quitter : le programme s'arrête



- Les Scores des joueurs sont affichés dans la case sous le plateau sous le joueur correspondant
- Le joueur peut passer son tour en effectuant un CLIC-GAUCHE sur la case « Passer son tour »
- En haut de l'écran est affiché le joueur qui a passé son tour pour informer le prochain joueur, si aucun ne passe son tour on n'affiche pas qu'un joueur a passé son tour

- On a voulu crée une fonction qui affiche le territoire du joueur quand on fait un CLIC-DROIT mais nous n'avons pas pu la réaliser par manque de temps
- Si les deux joueurs passent leur tour la partie est terminée et un « Victoire de BLANC » ou « Victoire de NOIR » est affiché selon le joueur qui a le score le plus élevé et le jeu se termine.



(Ici on voit la Victoire du joueur Noir ainsi que la non-intelligence du joueur blanc qui laisse tout le plateau comme territoire pour le pion noir mais c'est qu'un exemple.

II. Analyse fonctionnelle détaillée

1. Fonctions utilisant la SDL

Dans cette partie nous allons nous interessé de plus près aux fonctions dont on a parlé brièvement dans l'Analyse fonctionnelle générale pour ne pas devoir expliqué 2000 lignes de codes dont la plupart sont les bases de la SDL(SDL_Surface , SDL_Rect , SDL_Flip()…), on ne verra que les points importants en séparant le côté qui a attrait avec la SDL(affichage d'image et boucle événement) des manipulations sur le tableau représentant le plateau.

a) Print_image

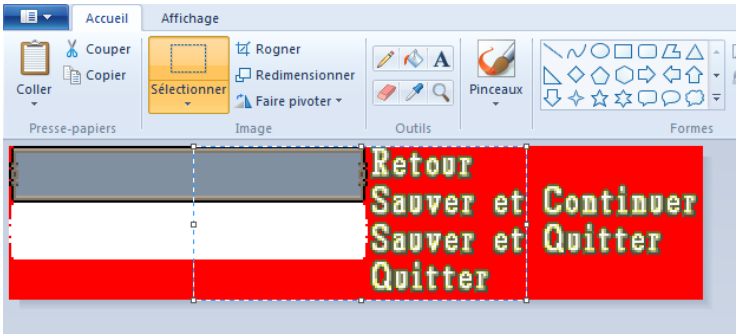
Avant de commencer à rentrer dans le vif du sujet, on doit voir une fonction qu'on utilisera tout le temps pour plus de rapidité :

```
fonction print_image(entree :SDL_Surface *ecran,SDL_Surface *surface,entier x,entier
y,SDL_Rect selection,entier opacite,entier R,entier V,entier B,entier *continuer
{
   SDL_Rect coordonnees;
   SDL_SetColorKey(surface, SDL_SRCCOLORKEY, SDL_MapRGB(surface->format,R,V,B));
   SDL_SetAlpha(surface, SDL_SRCALPHA,opacite);
   coordonnees.x←x;
   coordonnees.y←y;
   coordonnees.w←0;//les coordonnées n'ont pas besoin de stocker une taille
   coordonnees.h←0;
   if (SDL_BlitSurface(surface,&selection,ecran,&coordonnees) != FALSE )
      {
         fprintf(stderr,"Erreur de copie de la surface sur l'écran\n");
         *continuer←FALSE;
      }
}
```

Cette fonction print_image sert à affecter une image à une surface en utilisant tous les paramètres possible :

- mettre la couleur de fond en transparence : « SDL_SetColorKey(surface, SDL_SRCCOLORKEY, SDL_MapRGB(surface->format,R,V,B)); » ici R,V,B signifie Rouge,Vert,Bleu il s'agit du code de couleur RVB, la couleur désigné par ces trois variables sera mis en transparence avec cette fonction de la SDL.
- l'opacité : pour que l'image soit plus ou moins transparente. On l'utilise avec le « SDL_SetAlpha(surface, SDL_SRCALPHA,opacite); » avec ceci on modifie l'opacité de la surface

- coordonnées : la position de l'image stockée dans une structure avec 4 valeurs.
- sélection : la surface sélectionné dans l'image, cela correspond à la sélection qu'on pourrai faire sur un logiciel graphique comme Paint par exemple (voir ci-dessous)



Cette fonction a beau être plus compliqué à gérer vu le nombre de paramètres dans l'entêtes de la fonction mais c'est plus simple qu'on a à gérer l'opacité et la couleur d'arrière-plan tout au long du programme(surtout l'opacité vous le verrez plus tard).

Maintenant que cette fonction est présenté nous pouvons nous occupé des différentes fonctions du jeu de go, que je présenterai dans l'ordre d'appel dans le programme.

b) Title screen

fonction title_screen(entrée :SDL_Surface *screen, entier *mustcontinue)

Ici on met la Surface désignant l'écran en paramètre pour afficher une surface sur cet écran, on met *mustcontinue en paramètre pour pouvoir le modifier :

- *mustcontinue=FALSE (FALSE=0) si les images ne se sont pas affichés correctement, si le joueur appui sur la croix de la fenêtre ou si le joueur appui sur échap.

La fonction title_screen ne fait qu'afficher des images, jouer de la musique et démarrer le jeu, donc cette fonction ne dépend d'aucune variable du jeu de go, c'est un peu une fonction « passe partout ».

La fonction affiche le 1er titre, « jeu de go » écrit en chinois qui apparait en fondu, ici on joue sur l'opacité et le temps d'exécution.

Le principe est le suivant :

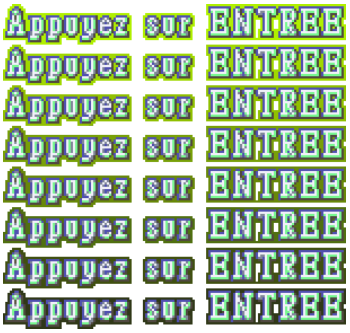
- On crée deux variables entières temps_actuel et temps_precedent initialisé tous les deux à 0
- On crée une variable entier i qui est un compteur qui sera plus tard l'opacité qu'on initialise à 0
- On crée une boucle événement tant que continuer=TRUE (TRUE=1)

- Après les switch des évènements (appui sur ENTREE et SDL_Quit()=appuie sur la croix de la fenêtre) on donne a temps_actuel= SDL_GetTicks() donc il va recevoir le temps en millisecondes depuis l'initialisation de la SDL.
- on teste si temps_actuel – temps_precedent>120 (sous-entendu 120 millisecondes) si le test est positif i est incrémenté de 1 temps_precedent prend la valeur de temps_actuel, sinon on effectue un SDL_Delay(120-(temps_actuel-temps_precedent)) cela va stopper le programme durant une durée déterminée par le calcul entre parenthèses
- Puis on affiche l'image avec print_image et SDL_Flip(screen) (pour mettre à jour l'écran) et on obtient une image qui apparait en fondu de plus en plus opaque

On affiche également un 2ème titre, « Jeu de Go » en français cette fois, on l'affiche de la même manière que pour le 1er titre sauf qu'il faut que ce dernier soit affiché avant lui.

On affiche aussi un texte « Appuyez sur ENTREE » pour indiquer au joueur le bouton sur lequel il doit appuyer, cette surface est afficher de la même manière que les deux autres sauf qu'en plus on effectue une animation entre plusieurs images pour donner un effet de clignotement de l'Appuyez sur ENTREE pour cela on va :

- créer 2 variables : entier image=1 et entier tour=0, image correspond à « l'indice de l'image » de l'Appuyez sur ENTREE et tour prendra 1 si image=8 autrement dit si toutes les images ont été affiché.



- en utilisant toujours les mêmes temps_actuel et temps_precedent on rajoute dans la boucle qui test si temps_actuel – temps_precedent>120 si tour=0 image est incrémenté de 1 sinon si tour=1 image est décrémenté de 1.
- si tour=0 et image<=8 on va modifié l'ordonnée y d'où va débuter la sélection de l'image(les 8 images sont dans le même fichier) qui font chacune la même taille en x (abscisse) et en y(ordonnée), pour cela on multiplie 36 qui est la taille en y par image

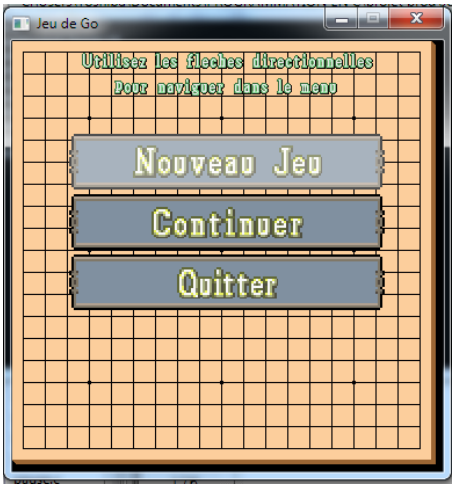
- si image>8 ou tour=1, peu importe sa valeur tour prendra la valeur 1 et si image=1 c'est-à-dire si toutes les images ont été affiché en partant de la 8ème jusqu'à la 1ère, tour=1.

c) Menu

menu (entrée :SDL_Surface *screen, int *mustcontinue,char p[][19]→sortie : entier) ;

Image source :

Dans cette fonction on affiche un menu, le joueur peut naviguer dans le menu grâce aux flèches directionnelles, le joueur voit où pointe son choix grâce à un « éclairement » de la case du choix :



L'éclairement de la case Nouveau Jeu est fait en affichant la case blanche dans l'image source en transparence par-dessus Nouveau Jeu, il suffit juste de donner la position en y de la case blanche la position en y de la case Nouveau Jeu si le joueur pointe sur cette case.

Le programme fait le pointage avec une variable choix qui est égal à 1 si nouveau Jeu, 2 si continuer et 3 si quitter, selon la valeur de choix on change la position en y de case blanche. Cette variable choix est modifié si le joueur appui sur la touche HAUT ou BAS, dans tous les cas on réaffiche toutes les images (Nouveau jeu, continuer, quitter et les cases) pour éviter que toute les cases finisse par être éclairé voir blanchie.

- Si le joueur appui sur ENTREE en étant :
  - Sur la case Nouveau Jeu : une nouvelle partie est lancé

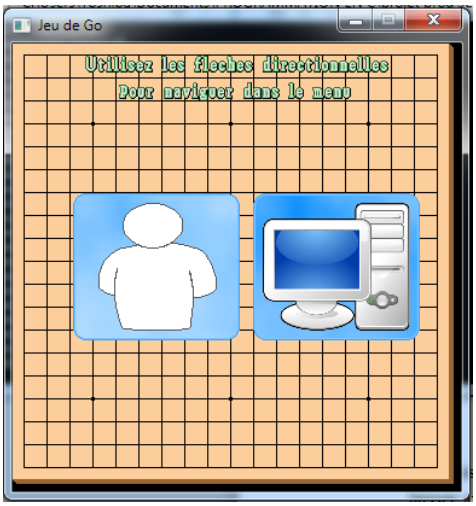
- Sur la case Continuer : une variable appelé charger prend la valeur TRUE(TRUE=1) cela indique au programme qu'une partie sauvegarder doit être chargé.
- Sur la case Quitter : le programme se ferme

d) Choix Adversaire

fonction choix_adversaire(entree :SDL_Surface* screen,entier*mustcontinue →sortie :entier) ;

Cette fonction sert à choisir l'adversaire du joueur Noir, si c'est un autre joueur ou un IA, elle est effectué si et seulement si le joueur a choisi de lancer un Nouveau Jeu (charger=FALSE). Ici c'est le même system que pour le menu une case blanche est afficher par-dessus les cases joueur et IA en fonction d'une variable choix qui sera retourné au programme après que le joueur ai appuyé sur ENTREE.

- Si choix=1 : le joueur Blanc est un joueur
- Si choix=2 : le joueur Blanc est un IA



e) refresh plateau

void refresh_plateau(SDL_Surface* ecran,SDL_Surface* pions,SDL_Rect select_pion,char p[][19],int *continuer)

Cette fonction comme on vient de le dire va mettre à jour le plateau, c'est-à-dire qu'elle va placée tous les pions qui étaient présents sur le plateau et stocké dans le tableau.

Pour cela on a besoin de récupéré en argument :

- SDL_Surface ecran : pour afficher la surface dessus
- SDL_Surface pions : pour afficher le pion
- SDL_Rect select_pion : pour connaitre la taille du pion
- char p[][19] : pour pouvoir parcourir le tableau
- int *continuer : si une image a été mal afficher *continuer=FALSE

J'ai préféré ne pas envoyer le plateau dans la fonction pour éviter d'avoir trop de paramètres, j'ai préféré afficher le plateau après la fonction refresh_plateau.

Pour mettre à jour le plateau on :

- parcourt le tableau 2D pour j allant de 0 à 19 pour i allant 0 à 19 avec j et i incrémenté de 1
- on calcul la position sur le plateau x=i*19 et y=j*19
- si p[i][j]='N' on sélectionne le pion noir avec SDL_Rect en changeant select_pion.x
- sinon si p[i][j]='B' on sélectionne le pion blanc avec SDL_Rect en changeant select_pion.x
- on affiche le pion avec comme coordonnées x et y calculés précédemment.

f) place_pion

void place_pion(SDL_Surface* ecran,SDL_Surface* plateau,char p[][19],char joueur,int *continuer, int * param, int * nbelCapt,int choix,int *passe_tour,int tour_passe)

Cette fonction effectue beaucoup d'actions, il s'agit de la fonction principale du programme celle qui va gérer tout le jeu (excepté la victoire d'un joueur).

On affiche le pion correspondant à la couleur du joueur dans cette fonction.

Pour faciliter la compréhension de la fonction je vais la détailler en partant de la création de la fonction aux dernières modifications.

A la création de la fonction elle remplaçait le curseur de la souris par un pion de la couleur du joueur affiché en transparence sur le plateau.

- pour que le curseur de la souris n'apparaisse plus on utilise la fonction SDL_ShowCursor(0)
- pour qu'à la place on affiche le pion en transparence on doit utiliser une boucle événement avec comme événement SDL_MOUSEMOTION.
- on récupère la position du curseur avec event.motion.x et event.motion.y qu'on stocke dans event_x et event_y
- on affiche le plateau car si on ne l'affiche pas il sera possible de « dessiner » sur le plateau avec l'image du pion qui se déplace
- Puis on affiche le pion avec ce dernier en transparence

Jusqu'ici on a fait que permettre au joueur déjà de voir qui va jouer à ce tour là et de voir son pion se balader sur le plateau.

Maintenant on voudrait pouvoir placé le pion c'est ce qu'on va faire tout de suite :

- on utilise un événement SDL_MOUSEBUTTONDOWN, en précisant sur quel bouton le joueur doit appuyer sur la souris soit le bouton gauche : SDL_BUTTON_LEFT
- on récupère la position du curseur avec event.button.x et event.button.y qu'on stocke dans event_x et event_y
- si on affiche le pion à partir de ces coordonnées le pion risque de se placer en dehors du plateau, pour éviter ça on a :
    - ✦ interdit event_x d'être inférieur à la longueur du pion(qui est égale aussi à la largeur d'une case du plateau) afin que le centre du pion se trouve bien sur la première colonne du plateau, la variable x prend la valeur de la largeur du pion.
    - ✦ interdit event_x d'être supérieur à 361 qui est la longueur maximal du plateau(ici on parle qu'en comptant les cases il y a 19 cases ayant chacun une taille de 19 pixels donc 19*19=361), si il est supérieur le pion ne sera pas sur le plateau et ne s'affichera peut être même pas à cause de ce qu'on fera par la suite, la variable x prend la valeur 361 ;
    - ✦ interdit les mêmes valeurs pour event_y pour les mêmes raisons et on affecte les valeurs de la mêmes façon
    - ✦ si event_x et event_y sont comprises entre ces valeurs ont les affectent à x et y

- Ici on a juste empêché le pion de se déplacer partout sur l'écran, mais le pion ne s'affiche pas exactement sur les intersections peu importe comme c'est demandé dans le cahier des charges pour cela on a :
    - ✦ donné à x la valeur de (x/19)*19 donc la position x du clic divisé par la taille d'une case multiplié par la taille d'une case, vu comme ceci cela est inutile mais pas en C car quand on fait (x/19) on obtient un réel si on calcule nous -

même ou avec une calculette, mais en C on obtient un entier donc un résultat tronquer, ici on obtient un indice d'une case d'un tableau 2D qu'on utilisera plus tard, en multipliant le résultat par 19 on obtient la position en x de la case.

- ✦ fait la même chose pour y
- ✦ on décrémente x et y de 19, la raison se trouve dans une nouvelle version du SDL_MOUSEMOTION, on a constaté que le curseur ne se mettait pas au milieu du pion mais en haut à gauche il a fallu trouvé un moyen de le centrer qui est le suivant :
  - event_x prend la valeur de event_motion.x moins la longueur du pion ce qui équivaut à afficher le pion une case avant le curseur puis à event_x on lui retire 10 qui est la moitié(environ) du pion comme ça il sera afficher sur l'intersection.
  - meme chose pour event_y.

  On retourne dans SDL_BUTTONDOWN, on a fait une décrémentation de 19 pour placé le pion sur l'intersection sa correspond au fait au calcul : x-position du curseur+la moitié du pion= x-29-10 =x-19

- On affiche le plateau puis le pion.

Voilà maintenant le pion est placé, mais on voudrait pouvoir le stocké dans un tableau pour pouvoir en placé d'autre en gardant le pion sur le plateau, pour cela on utilise le x/19 et y/19 dont on parlait tout à l'heure qui seront les indices du tableau 2D on les affecte aux variables case_x et case_y et on affecte p[case_x][case_y]=joueur.

Maintenant le problème est qu'au tour du joueur 'B' le pion noir va disparaitre, on a donc crée une fonction qui remet à jour le plateau, qui est la fonction **refresh_plateau** ***(voir 1.g)*** Cette fonction sera utilisé dans place pion à chaque fois qu'un pion est affiché sur le plateau, la fonction est toujours accompagné de l'affichage du plateau.

Maintenant qu'on peut réafficher tous les pions sur le plateau, il a fallu contrôler le placement du pion c'est-à-dire interdire au pion de se placer à tel ou tel endroit dans certains cas, avec la fonction **placement_possible(*voir 2.g)***,ainsi qu'effectuer (ou pas) la capture d'une chaine avec la fonction **capture*(voir 2.f)*** et réafficher le plateau après conversion du tableau 1D avec la fonction **tab_1D_to_2D*(voir 2.a)*** avec **refresh_plateau *(voir 1.g).***

A partir d'ici on a un jeu qui fonctionne bien avec un joueur comme adversaire mais le cahier des charges nous indique qu'un IA doit être fait, on a d'abord pensé à faire un IA qui place

des pions aléatoirement, par peur de ne pas pouvoir réussir à crée une IA convenable mais après plusieurs essais on a pu arriver à un bon résultat.

Avant de parler du fonctionnement de l'IA il faut définir le cas où l'IA joue pour cela on crée une boucle tant que :

- cont_deplace=TRUE : variable qui indique si le joueur place son pion
- joueur='B'
- choix=2 : variable égal FALSE si le joueur n'est pas un IA et TRUE si le joueur est IA

Si on rentre dans la boucle on va :

- affecter à la variable IA_active la valeur TRUE, donc le programme sait que le joueur blanc est un IA.
- on utilise la fonction **tab_2D_to_1D** ***(voir 2.a)*** pour convertir un tableau 2D en 1D (car la fonction IA ainsi que d'autres fonctions utilise un tableau 1D, mon binôme tenait à utiliser un tableau 1D dans son travail)
- puis on appelle la fonction **IA*(voir 2.k)*** qui va renvoyer l'indice où il veut placer son pion.

Une fois l'indice du pion à placer récupéré dans la fonction IA, on va vérifier si l'IA a le droit de placer le pion grâce à la fonction **placement possible*(voir 2.g)***, on stocke la valeur renvoyé dans place_possible.

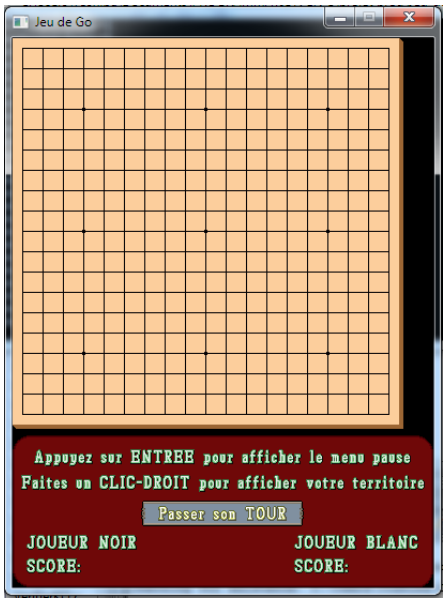
- On teste si place_possible=TRUE et si *passe_tour=FALSE :
  - ✦ on stocke 2 à tab[i]
  - ✦ on appelle la fonction **capture*(voir 2.f)***
  - ✦ cont_deplace prend la valeur FALSE sa veut dire que le pion a été placé
  - ✦ on reconvertit le tableau 1D en 2D avec la fonction **tab_1D_to_2D*(voir 2.a)***

Une fois le tableau 2D convertit récupéré on appelle la fonction refresh plateau pour mettre à jour le plateau.

- Sinon si place_possible=FALSE et *passe_tour==FALSE, on fait une boucle tant que place_possible=FALSE :
  - ✦ ici on voudrait pouvoir placer un pion si un autre indice du pion est possible
  - ✦ on test si l'indice *(libofp+1) est different de NULL si non il quitte la boucle en disant que place possible=TRUE et erreur_seg(erreur segmentation) =TRUE
  - ✦ si (*libofp+1) est different de NULL, on donne a libofp(donc l'adresse) libofp+1 pour accéder à l'indice suivant, on affecte placement possible à place_possible tant qu'il est égal à FALSE il va toujours tester l'indice suivant.
- si erreur_seg=FALSE place_possible=TRUE
- sinon place_possible=FALSE
- si *passe_tour=FALSE et place_possible=FALSE, cont_deplace=TRUE, i_test=i_IA+1, pour tester dans la boucle for dans la fonction i_IA à partir de i_IA+1.

- sinon si *passe_tour=TRUE, on affiche une image où y a marqué que le joueur blanc passe son tour, on attend 700 millisecondes avec le SDL_Delay(700) pour laisser le temps au joueur de voir que l'IA a passé son tour.

Le joueur peut à présent jouer contre l'IA, mais il ne peut pour l'instant pas passer son tour, c'est maintenant qu'intervient la case rouge sous le plateau dont on parlait au début de ce rapport, si le joueur clic sur la case Passer son TOUR il passe son tour



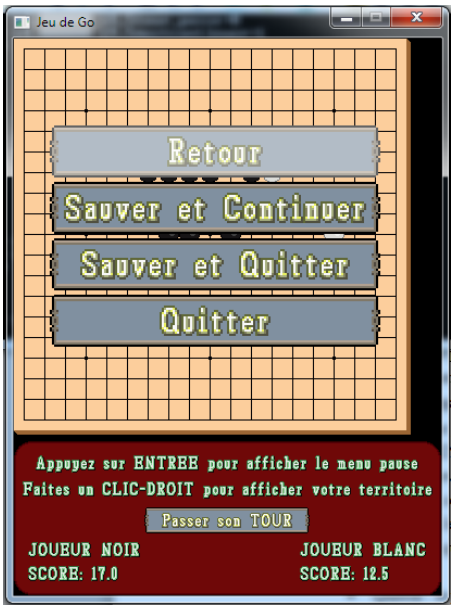
Et on affiche à l'écran « Noir passe », si c'est le joueur blanc qui passe sont tour on affiche « Blanc passe » dans le cas de l'IA et du joueur.

Dans le cahier des charges on doit permettre au joueur de sauvegarder on l'a fait, il a accès à la commande sauvegarder via une interface de menu_pause apparaissant quand le joueur clic sur ENTREE, cette interface est afficher avec la fonction **menu_pause*(voir 1.h).***

g) menu_pause

Cette fonction affiche un interface menu_pause quand le joueur appui sur ENTREE :

- Retour : on quitte le menu pause
- Sauver et continuer : on sauvegarde la partie et on continue
- Sauver et quitter : on sauvegarde la partie et on quitte le programme
- Quitter : on quitte le programme



Si le joueur appui sur Entree et que le « curseur » se trouve sur Sauver et continuer ou sauver et quitter il effectue la fonction suivante

h) affiche_score

void affiche_score(SDL_Surface *screen,float score,char joueur)

La fonction affiche le score en fonction d'une image dans le dossier Data, on commence par convertir la variable score en flottante via plusieurs opération dessus et à l'aide de la fonction sprintf, puis en fonction de la valeur de joueur, chaque caractère composant la chaine du score s'affiche progressivement à l'endroit prévu pour le joueur.

2. Fonctions n'utilisant pas la SDL

a) Fonction de conversion tableau

void tab_1D_to_2D(int tab_1D[19*19],char p[19][19])

Cette fonction sert à reconvertir le tableau 1D en 2D pour qu'il puisse être utilisé dans les fonctions manipulant un tableau 2D de caractere.

- on fait une boucle for(u=0 ;u<19*19 ;u++)
- i=u%19,
- j=(u-i)/19
- si tab[u]=1, p[i] [j]='B'
- si tab[u]=2, p[i] [j]='N'
- sinon p[i] [j]='0'

int *tab_2D_to_1D(char tab_2D[][19])

Cette fonction sert à convertir un tableau 2D en 1D en renvoyant l'adresse du tableau converti.

- On fait l'allocation dynamique d'un tableau d'entier à 1 dimension de taille 19*19
- On parcourt le tableau 2D avec j pour les lignes et i pour les colonnes
- Si tab_2D[i] [j]='N', tab_1D [i+19*j]=1, i+19*j → colonne+19*ligne.
- Si tab_2D[i] [j]='B', tab_1D [i+19*j]=2, i+19*j
- sinon tab_1D [i+19*j]=0
- on renvoie l'adresse du tableau_1D

b) test_existe

Cette fonction sert à vérifier si un élément existe dans un tableau, la fonction va parcourir tout le tableau et si elle trouve la valeur à tester on affecte à la variable test la valeur 1 sinon on le laisse a son valeur initiale qui est 0.

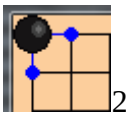
c) voisin

Cette fonction renvoie la liste des voisins d'un pion dont on a récupéré l'indice en paramètre. Dans cette fonction on va regarder l'intersection au-dessus du pion, l'intersection en dessous, à gauche et à droite et stocké chaque indices du voisin dans un pointeur qui stockera la liste de tous les voisins du pion.

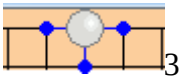
Un voisin est dit libre quand il n'est pas occupé par un pion(liberté)

En général un pion à 4 voisins sauf si il se trouve soit :

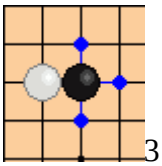
- sur une intersection dans un coin du plateau

 2 libertés(les points bleus représentent les libertés)

- sur une intersection sur la première ou dernière ligne ou colonnes.

 3 libertés

Prenons un exemple où le pion possède des voisins qui ne sont pas libres :

 3 voisins libres et 1 occupé par le pion blanc.

d) GetGroupe

Cette fonction sert à former une chaine de pion appartenant au même joueur, elle renvoie le nombre d'éléments dans la chaine, la fonction recherche le nombre de voisin occupé par un pion appartenant au même joueur que le premier pion testé avec la fonction test_existe.

e) Calculeliberte

Cette fonction sert à calculer le nombre de liberté d'une chaine, pour cela on commence par formé une chaine à partir de la fonction getGroupe et l'indice d'un pion qu'un joueur veut placé(mis en paramètre), cette fonction nous apporte le nombre d'élément dans la chaine ainsi que les indices de tous les pions contenu dans la chaine.

La méthode est de trouvé les voisins de la chaine en les testant pion par pion (indice par indice), si il trouve que le voisin d'un pion est égal à 0, il va vérifier que l'indice du voisin n'a pas déjà été testé avec la fonction test_exist, chaque fois il incrémente le compteur de 1 et va à la case suivante du tableau qui stocke les indices des libertés .

f) Capture

La fonction va effectuer ou non une capture à l'aide de deux fonctions :

- **getGroupe*(voir 2.d)***
- **Calculeliberte*(voir 3.e)***

La fonction va parcourir tout le tableau puis il alloue de la mémoire pour le pointeur *groupe qui est l'un des paramètres de **getGroupe** et pour le pointeur *libofp qui est un paramètre de **Calculeliberte** .La fonction **getGroupe** récupère la chaine à laquelle un pion choisi appartient, on regarde s'il a des voisins qui n'appartiennent pas au même joueur que le pion précédent, il calcule le nombre de liberté de la chaine si il est égale à 0 il parcourt tout le

tableau "groupe" et il remplace le contenu des cases du tableau correspondant à la chaine entouré par des pions de l'adversaire par 0, donc la chaine est capturé.

g) VerifierKO

Cette fonction vérifie premièrement si une capture d'un pion peut être effectué, dans ce cas elle vérifie si l'indice du pion qu'on veut placé est égale à l'indice de la seul liberté qu'a le pion puis on calcule le nombre de liberté qu'on a pour la chaine à laquelle il appartient à l'indice du pion qu'on veut placé si la liberté égale à 0 et le nombre d'élément du groupe égale à 1 (chaine formait d'un seul pion) on retourne l'indice de ce pion .

h) Placement possible

Cette fonction sert à interdire le placement d'un pion qui n'a pas de droit de se placer, la fonction renvoie la valeur 0 si le placement est impossible et 1 si le placement est possible la fonction vérifie les conditions suivant si le placement est impossible sinon le placement est possible:

- Si on veut placer un pion dans un indice qui contient pas 0 (c.à.d. il contient 1ou 2 (blanc ou noire)) et dans ce cas on retourne 0

- Si la fonction **vérifie KO** (p.....) ==p (avec p un indice) dans ce cas aussi on retourne 0

- on donne a data[p] joueur (1 ou 2) et on calcule la liberté du chaine

    - Si la liberté égale à 0
        - si data de l'un des voisin égale à la joueur adverse et la liberté de ce voisin égale à 0 on retourne 0 sinon on retourne 1
    - Si liberté égale à 1
        - on récupère l'indice de la liberté de pion et on vérifie si celui-ci forme une chaine de vide égale à 1 si oui la fonction retourne 0

- A l'aide de l'indice de la liberté on cherche la chaine à laquelle il appartient puis on s'assure qu'il est entouré du pion de appartenant au même joueur est ce ci en prenant deux variable qui test blanche et noire.

i) Calcule score

La fonction sert à calculer le score des deux joueurs et ceci en parcourant tout le plateau en augmentant le score du blanc de un si on trouve un pion blanc et en augmentant le score de noire si on trouve un pion noir puis on teste tous les territoires si on trouve un territoire on prend toutes ces cases vides et on les ajoute au nombre de pion appartenant au même joueur qui entoure les territoires

j) Load

fonction load(entree :caractere p[][19],entier charger,caractere *joueur,entier *choix) ;

Cette fonction sert à charger une partie sauvegardée au préalable, elle est effectué si charger=TRUE, elle prend en compte plusieurs paramètres :

- caractere p[][19] : le tableau 2D correspondant au plateau
- entier charger
- caractere *joueur : pour pouvoir changer dans le programme la valeur de joueur on met joueur en pointeur
- entier *choix : pour pouvoir changer dans le programme la valeur de choix, choix représente le choix du joueur adverse (joueur ou IA).

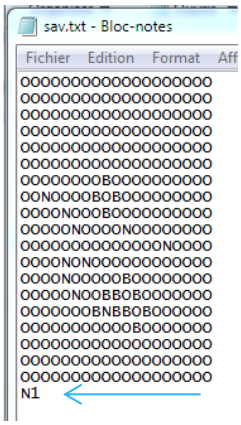
Pour charger une partie on utilise un fichier texte qui a été « rempli » au préalable.

- on lit ce fichier
- on récupère chaque caractère contenu dans le fichier avec la fonction fgetc dans une boucle faire…tant que(caractereactuel !=EOF), avec caractereactuel un entier qui stocke le caractère lu dans le fichier et EOF signifiant la fin du fichier.
- si ce caractere est different de '\n' et j different de 19, caractereactuel est stocké dans p[i][j] avec i=0 et j=0 au début de la boucle.
- si j=19 et i=0 *joueur prend la valeur de caractereactuel, cela permet de savoir qui était le joueur qui jouait quand la sauvegarde a été effectué
- sinon si j=19 mais que i est different de 0 (ce qui ne peut arriver par ce que i ne change pas quand j>19), *choix prendra la valeur :
  - ✦ 2 si caractereactuel=1 , ici le joueur blanc est un IA
  - ✦ 1 si caractereacutel=0 ,ici le joueur blanc est un joueur
- si i==19(on compte le '\n') une ligne du tableau a été lu, on passe à la suivante en incrémentant j de 1 et i est réinitialisé à 0.
- sinon si i<19(sans compté le '\n') on incrémente i de 1.

k) Save

void save(char p[][19],char joueur,int IA_active)

Ici on sauvegarde une partie dans un fichier texte sous ce format :



La difficulté pour la sauvegarde était de sauvegarder l'IA, le reste il suffit de suivre le tuto d'open classroom pour savoir comment écrire dans un fichier et afficher le tableau dedans comme on affiche en console.

Pour sauvegarder l'IA j'ai crée une variable caractère nommé IA qui prend la valeur '1' si IA_active est égal à 1 et '0' si IA_active est égal à 0 et le résultat est afficher en bas du fichier à coté du caractère correspondant au tour du joueur(voir l'image à coté et la fleche bleu)

l) IA

int IA(int data[maxSizeCarre],int nbcapt,int param,int i_test,int *passe_tour,int tour_passe,int*libofp,char joueur)

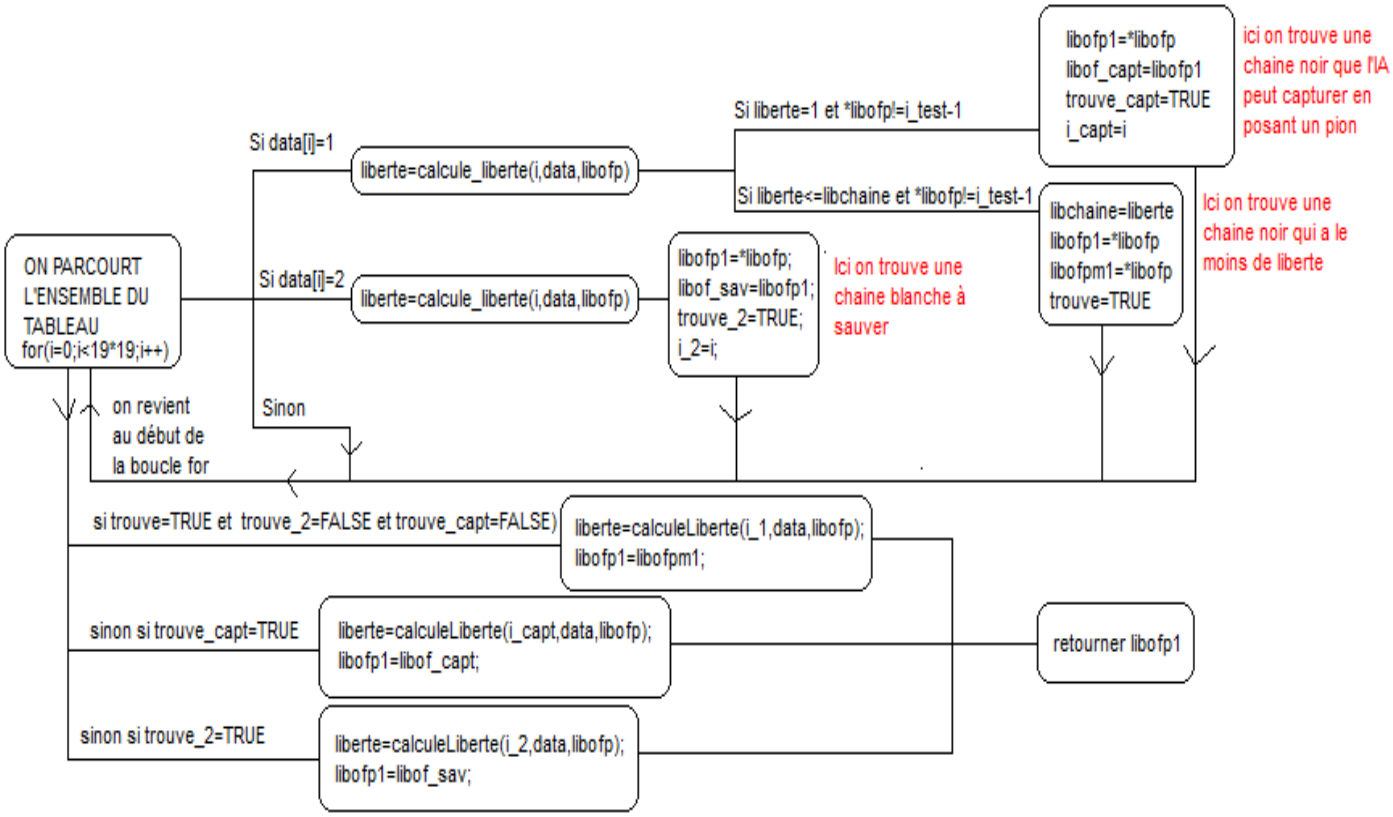
Cette fonction permet de retourner l'indice du pion que l'IA va placer et aussi de gérer si l'IA passe son tour grâce au paramètre *passe_tour et tour_passe, tour_passe égal TRUE si le noir a passé son tour on verra dans une autre fonction comment il va s'y prendre.

Avant de rentrer dans la fonction IA on a alloué 161 case mémoire pour libofp dynamiquement, on envoi libofp en argument.

- On refait la même allocation dynamique dans la fonction IA par ce que quand on ne l'a fait pas le programme nous dit qu'il y a une erreur de segmentation.
- on effectue la fonction calcul score pour calculer le score avec data en paramètre (data est le tableau 1D représentant le plateau), on affecte l'adresse du pointeur dont l'adresse a été allouée dans la fonction, à l'adresse du pointeur *score.
- on parcourt tout le tableau si la fonction placement possible(i,data,libofp) renvoie la valeur TRUE on arrête de parcourir le tableau et on affecte à une variable nommé place_possible la valeur TRUE sinon place_possible garde la valeur FALSE affecté au début de la fonction.
- si :
    - ✦ le contenu de score est inférieur au contenu de score+1
    - ✦ score différent de 361 car quand le joueur noir place son premier pion , il n'y a aucun pion blanc donc son territoire est en fait tout le plateau et place_possible=FALSE
    - ✦ tour_passe=TRUE (tour_passe= TRUE si le joueur noir a passé son tour).

  si toutes ces conditions sont vérifié *passe_tour=TRUE, si *passe_tour=TRUE l'IA va passer son tour.

- sinon si *passe_tour=FALSE, le programme va déterminé l'indice du pion blanc a placé sur le plateau, pour expliquer le fonctionnement je vais faire un schéma :
  i_test=0 la première fois que la fonction est effectué après dans place_pion elle change si le pion n'a pas pu être placé.
  libchaine=361, 361 est le nombre maximal de liberté d'une chaine.



On privilégie le cas où l'IA va capturer une chaine en plaçant un pion(libofp1=libof_capt), puis s'il ne peut pas il va regarder si il peut sauver une de ses chaines en plaçant son pion à l'indice de la dernière liberté de la chaine(libofp1=libof_sav) sinon il va tenter diminuer le nombre de liberté d'une chaine noir afin de pouvoir la capturer plus tard.
Puis on renvoie l'indice du pion à placé dans le tableau 1D.

3. *Le main*

Dans le main on appel la fonction **title_screen*(voir 1.a)*** pour afficher l'écran titre puis **menu*(voir 1.b)*, load*(voir 2.)*** et **choix_adversaire*(voir 1.c)*** si le joueur n'a choisi de continuer une partie.

Après que le joueur adverse est choisi, on change la taille de la fenêtre pour pouvoir afficher la « case » où s'afficheront les scores ainsi que des infos sur des touches :

Après on a la boucle événement « principal », c'est-à-dire la boucle tant que qui si la variable mustcontinue est égal à FALSE, fermera le programme.

Après les switch événements se trouve les fonctions qui vont faire que le joueur puisse jouer car jusqu'à maintenant il n'a fait que s'y préparé.

Dans le main, on appel 2 fois la fonction **place_pion*(voir 1.g)*** :

- une fois pour le tour du noir en mettant 'N' en paramètre.
- une deuxieme fois pour le tour du blanc en mettant 'B' en paramètre et en testant que les joueurs n'ont pas passé leur tour
- Après chaque appel de la fonction **place_pion*(voir 1.g)***,on appel la fonction **calculescore*(voir 2.)*** pour actualiser le score des deux joueurs puis on affiche le score des joueurs avec la fonction **affiche_score*(voir 1.)*** qui s'affiche dans la case rouge sous le nom du joueur correspondant :

- si *(score+1)<*(score), pass_blanc=TRUE et pass_noir=TRUE, mustContinue prendra la valeur FALSE, on affichera « VICTOIRE BLANC »

- si *(score+1)>*(score), pass_blanc=TRUE et pass_noir=TRUE, mustContinue prendra la valeur FALSE, on affichera « VICTOIRE NOIR »

- Sinon il ne se passe rien mustContinue est toujours égal à TRUE
- A plusieurs endroits j'ai fais ses deux test pour qu'il se fasse après le placement du pion noir ou le non placement du pion noir(donc si le noir passe son tour) et après celui du pion blanc qui s'affiche que si mustContinue=TRUE, passe_blanc=TRUE,passe_noir=TRUE.

Cette partie n'était pas présente dans le code durant la soutenance :

- Si après l'affichage du pion noir passe_noir=FALSE et pass_blanc=TRUE passe_blanc=FALSE car sinon le pion blanc sera placé mais pour le programme le joueur a passé son tour donc si le joueur noir a passé son tour après ce sera la fin de la partie alors le que le joueur blanc aura peut etre jouer à ce tour la.
- Si à la fin de la boucle tant que pass_noir=TRUE et pass_blanc=FALSE pass_noir=FALSE car sinon le pion noir sera placé mais pour le programme le joueur a passé son tour donc si le joueur blanc passe son tour après ce sera la fin de la partie alors le que le joueur noir aura peut etre jouer à ce tour là.

III. Les remarques sur le projet

- Quelques bugs dans le programme ont été constaté mais n'ont pu être corrigé :
  - La première fois qu'on lance le programme il cesse de fonctionner après le placement du pion noir
  - Quand on quitte le jeu, un message indique que le jeu de go à cesser de fonctionner
- Lors de la soutenance certains problèmes ont été rencontrés :
  - Pour nous le joueur pouvait placer un pion dans son territoire nous l'avons corrigé
  - Problème au niveau du passement de tour, si blanc passait son tour en premier, si noir passe son tour la partie devait se terminer mais le joueur blanc plaçait son pion puis termine la partie. Le problème a été corriger, la variable assigné au passement de tour de blanc n'était pas réinitialisé à 0 au cas où le joueur noir ne passait pas son tour donc la condition qu'on mettait sur l'affichage du pion blanc : si passe_blanc=FALSE et passe_noir=FALSE empecher le joueur de jouer et donc le programme cessait de fonctionner.
  - La règle du Seki pose problème car le sujet dit que le joueur ne peut pas capturer alors que lors de la soutenance on nous a dit le contraire, pour le prouver voici la citation où ce cas est géré.

    « Les pierres mortes sont les pierres qui ne peuvent être sauvées. A l'instar des pierres vivantes qui sont des pierres que l'on ne peut pas espérer capturer »

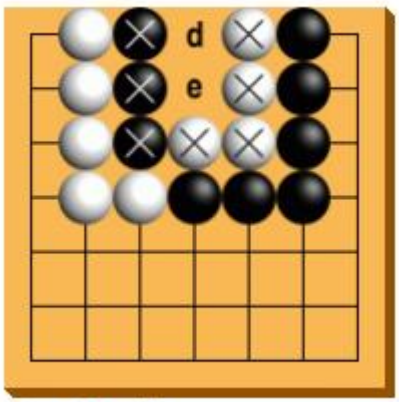


Figure 10 : seki

« La figure 10 montre un seki. Que noir ou blanc joue en d ou e, entrainera l'adversaire à jouer l'autre possibilité et ainsi prendre toutes les pierres. Aucun des 2 joueurs n'a donc intérêt à jouer d ou e. les pierres marquées d'un X sont donc dites vivantes par seki et les intersections d et e sont des intersections neutres. »

Le sujet dit bien que les pions marquées d'un X sont vivantes or des pions vivants sont des pions qu'on ne peut pas capturer.