SupMti

Projet de fin d'année

Jeux D'échec

Réalisé par :

Oussama Essadaoui

Encadré par :

Mr. Khelloufi

Année : 2014/2015

SOMAIRE

Dédicace et Remerciement :

Nous tenons à exprimer notre reconnaissance à Mr Khelloufi qui a bien voulu diriger ce projet.

Nous lui présentons nos vifs remerciements pour sa disponibilité et ses conseils pertinents qui ont aidé de façon très significative à l'amélioration de ce travail.

Nous remercions l'ensemble des professeurs d'école SupMti supérieure de management, d'informatique et de télécommunication Beni Mellal.

Nous Présentons aussi nos remerciements à toutes les personnes qui ont contribué de près ou de loin à la réalisation de ce travail.

I-Introduction

Ces dernières années le marché du jeu vidéo a explosé, à tel point qu'il est de nos jours plus important que celui du grand écran.

Les jeux attirent un public de plus en plus large, mais séduisent également de plus en plus de développeurs. Malheureusement, la programmation de jeu est souvent méconnue et beaucoup imaginent que ce sera aussi amusant que de jouer. Le fait est que c'est totalement faux, en vérité le jeu vidéo demande beaucoup d'investissement ainsi que des connaissances théoriques et pratiques assez poussées, que la plupart des codeurs n'ont pas forcément.

Dans ce projet nous essayerons de programmer un jeu d'échecs, Donc D'abord nous devons donner une petite définition d'échecs :

Les échecs, c'est un jeu qui se joue entre deux adversaires sur les côtés opposés d'une planche contenant 64 cases de couleurs alternées. Chaque joueur possède 16 pièces: un roi, une reine, 2 tours, 2 fous, 2 cavaliers et 8 pions. Le but du jeu est de mater le roi adverse. L'échec et mat survient quand le roi est en position pour être capturé (en échec) et qu'il ne peut pas s'échapper sur une autre case.

Le présent document est structuré comme suit: La première partie présente en la programmation Structurée et en C en générale, puis le deuxième chapitre fournie l'analyse, la conception et la réalisation, Finalement, une conclusion

II- programmation structurée

La **programmation structurée** peut être vue comme un sous-ensemble, ou une branche, de la programmation impérative, un des paradigmes majeurs de la programmation.

Elle est célèbre pour son **combat** pour la suppression de l'instruction *goto* ou du moins pour la **réduction** de son usage.

Il est en fait possible de faire de la **programmation structurée** dans n'importe quel langage de programmation procédural, mais depuis que, vers 1970, la programmation structurée est devenue une technique populaire, la plupart des nouveaux langages de programmation procéduraux ont intégré des mécanismes qui encouragent la programmation structurée (et ont quelquefois abandonné ceux qui facilitaient une programmation déstructurée). Parmi les langages de programmation structurée les plus connus, on trouve Pascal et Ada.

Pour l'écriture de fragments assez courts, la programmation structurée recommande une organisation hiérarchique simple du code. On peut le faire dans la plupart des langages de programmation modernes par l'utilisation de structures de contrôles *while*, *repeat*, *for*, *if .. then.. Else*. Il est également recommandé de n'avoir qu'un point d'entrée pour chaque boucle (et un point de sortie unique dans la programmation structurée originelle), et quelques langages l'imposent.

Les programmeurs doivent décomposer leur code en petits sous-programmes (appelés **fonctions** et procédures dans certains langages), assez petits pour être facilement compris. En général les programmes doivent éviter d'utiliser des variables globales ; au lieu de cela, les sous-programmes doivent utiliser des variables locales et agir sur des arguments fournis explicitement en paramètre, par valeur ou par référence. Ces techniques aident à créer des petits morceaux de code, faciles à comprendre isolément sans avoir à étudier l'ensemble du programme.

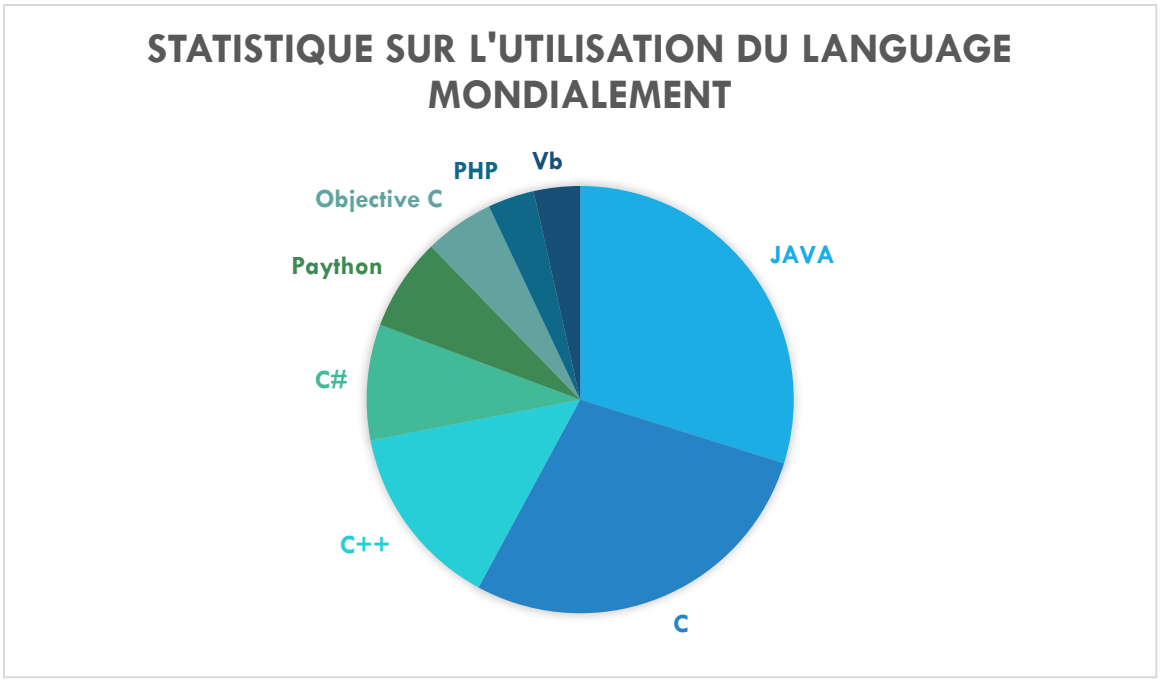
La programmation structurée est souvent confondue avec la méthodologie de développement (*top-down design*). Dans cette approche les programmeurs décomposent la structure, à large échelle, d'un programme en terme d'opérations plus petites, implémentent et testent ces petites opérations, et les assemblent pour réaliser le programme.

À la fin du 20e siècle la plupart des programmeurs ont adopté la programmation structurée. Certains affirment que leurs collègues peuvent comprendre les programmes structurés plus facilement, ce qui améliore la fiabilité et simplifie la maintenance.

III-Choix Du langage du programmation

a-Histoire du langage **C**

Entre les années 1980 et 2000, aucun langage de programmation n'a pu se vanter d'une croissance en popularité comparable à celle de **C** et de son frère **C++**. Ce n'est que vers l'an 2000 où le langage **C** doit passer sa première place au langage Java - l'un de ses petits-fils. Même en 2011, **C** occupe toujours la seconde place dans le **classement de la popularité**, entouré de Java, C++, C# et PHP qui sont tous en quelque sorte des dérivés de **C**.
Le langage **C** trouve ses sources en 1972 dans les 'Bell Laboratoires': Pour développer une version portable du système d'exploitation UNIX, Dennis M. Ritchie a conçu ce langage de programmation



b-Avantages

Le grand succès du langage C s'explique par les avantages suivants; C'est un langage:

1. **universel :**

C n'est pas orienté vers un domaine d'applications spéciales, comme par exemple FORTRAN (applications scientifiques et techniques) ou COBOL (applications commerciales ou traitant de grandes quantités de données).

2. **compact :**

C'est basé sur un noyau de fonctions et d'opérateurs limité, qui permet la formulation d'expressions simples, mais efficaces.

3. **moderne :**

C'est un langage structuré, déclaratif et récursif; il offre des structures de contrôle et de déclaration comparables à celles des autres grands langages de ce temps (FORTRAN, ALGOL68, PASCAL).

4. **près de la machine :**

Comme C a été développé en premier lieu pour programmer le système d'exploitation UNIX, il offre des opérateurs qui sont très proches de ceux du langage machine et des fonctions qui permettent un accès simple et direct aux fonctions internes de l'ordinateur (p.ex.: la gestion de la mémoire).

5. **rapide :**

Comme C permet d'utiliser des expressions et des opérateurs qui sont très proches du langage machine, il est possible de développer des programmes efficients et rapides.

6. **indépendant de la machine :**

Bien que **C** soit un langage près de la machine, il peut être utilisé sur n'importe quel système en possession d'un compilateur **C**. Au début **C** était surtout le langage des systèmes travaillant sous UNIX, aujourd'hui **C** est devenu le langage de programmation standard dans le domaine des micro-ordinateurs.

7. **portable :**

il est possible d'utiliser le même programme sur tout autre système (autre hardware, autre système d'exploitation), simplement en le recompilant.

8. **extensible :**

C ne se compose pas seulement des fonctions standard; le langage est animé par des bibliothèques de fonctions privées ou livrées par de nombreuses maisons de développement.

c-Désavantages

Evidemment, rien n'est parfait. Jetons un petit coup d'œil sur les désavantages:

1-En C, nous avons la possibilité d'utiliser des expressions compactes et efficientes. D'autre part, nos programmes doivent rester compréhensibles pour nous-mêmes et pour d'autres.

2- C est un langage près de la machine, donc dangereux et bien que C soit un langage de programmation structuré, il ne nous force pas à adopter un certain style de programmation (comme p.ex. Pascal). Dans un certain sens, tout est permis et la tentation de programmer du 'code spaghetti' est grande. (Même la commande 'goto', si redoutée par les puristes ne manque pas en C). Le programmeur a donc beaucoup de libertés, mais aussi des responsabilités: il doit veiller lui-même à adopter un style de programmation propre, solide et compréhensible.

Conclusions

La programmation efficiente en C nécessite beaucoup d'expérience et n'est pas facilement accessible à des débutants.

Sans commentaires ou explications, les programmes peuvent devenir incompréhensibles, donc inutilisables.

VI-Analyse, conception et réalisation :

Dans un but de simplicité, ce projet se concentre uniquement sur le moteur du jeu, non sur une interface graphique.

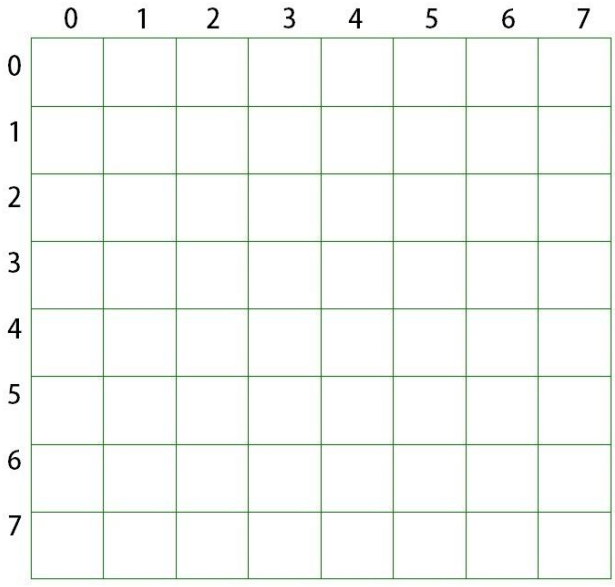
Alors pour programmer jeu, On peut décomposer un programme d'échecs aux éléments suivants :

- un échiquier;
- des pièces d'échecs;
- une interface.
- un moteur de jeu;

L'échiquier :

Il existe plusieurs façons de représenter l'échiquier. Etant composé de 64 cases, le plus simple en programmation est d'utiliser un tableau de 64 éléments (2 dimension).

```
board[8][8] ;
```



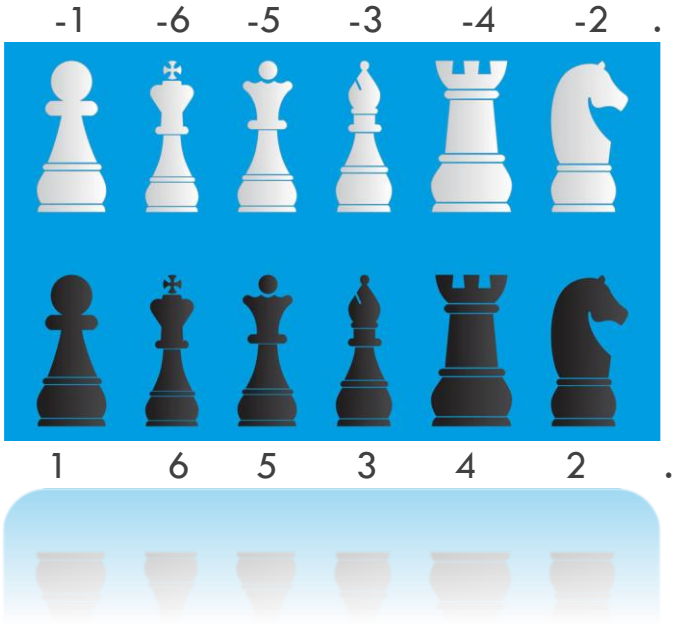
Les index de ce tableau seront numérotés de 0 à 7 et représente les cordonnée de chaque case dans l'échiquier.

Pièces d'échecs :

Comme vous le savez les pièces sont : roi, dame, tour, cavalier, fou, pion. Certaines sont plus importantes que d'autres. On peut donc attribuer une valeur à chaque case pour mettre une pièce ou pas.

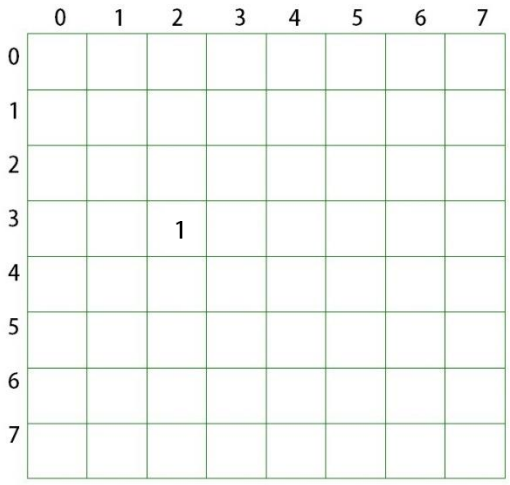
Valeurs positive pour les pièces noire et négative pour les blancs, par exemple 5 pour la dame Noire, -1 pour le pion blanc, 0 pour Les cases vides. Etc.

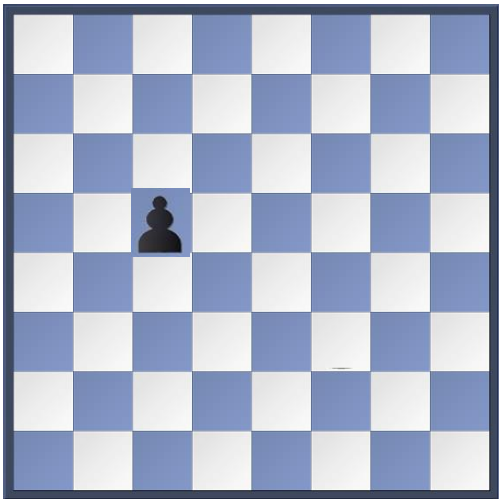
Vide 0



Alors pour mettre un pion noir dans la case 2,3 on peut donc faire :

```
board[2][3]=1;
```

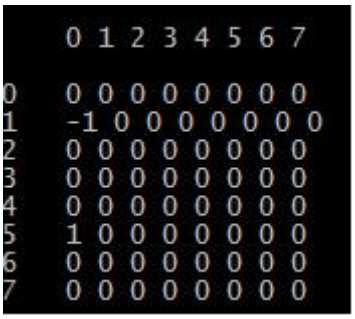




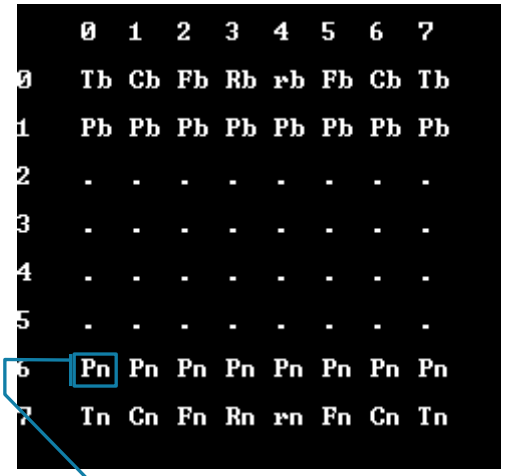
L'Interface :

Pour créer une interface nous devons afficher chaque valeur dans le tableau board[8][8] dans un ordre pour qu'il ressemble à un échiquier et même afficher les coordonnée .

Ce qui donne quelque chose comme ça :



Alors pour afficher un propre échiquier on a créé une fonction pour convertir les nombre par des caractères qui représente Les pièces :



Pn

Premier lettre de la pièce — couleur de la pièce

Exemple d'affichage :

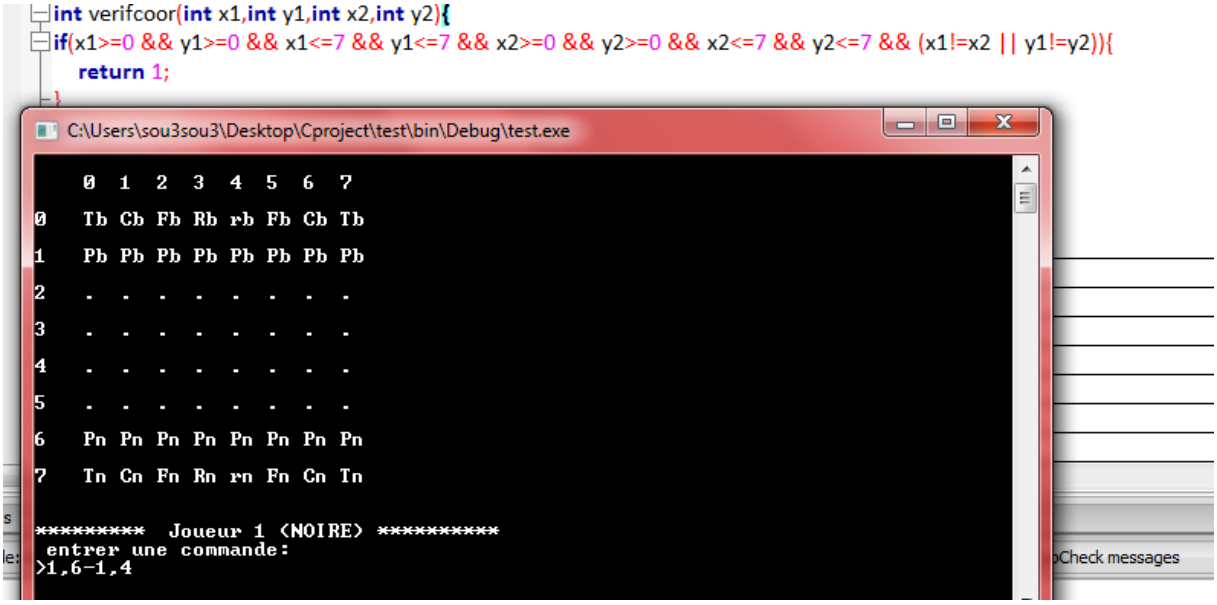
```
switch(board[i][j]){//conversion des nombres par des caractères
    case 0://case vide
        printf(".  ");break;
    case 1://pion
        printf("Pn ");break;
```

Moteur du jeu :

1- Déplacement Des Pièces :

On peut les déplacer d'une case à une autre selon les règles propres à chacune. On va donc créer les fonctions permettant de trouver la case de départ d'une pièce et la case de destination.

Vérification des coordonnées :



Alors et pour ne pas donner des coordonnées hors l'échiquier on vérifie des conditions : Le X1, Y1 (cordonnée de la pièce) et X2, Y2 (Destination) son entre 7 et 0. Et Les Coordonnées du départ ne doivent pas être les mêmes de la destination.

N.B : on peut entrer Les Coordonnées de cette façon : x1,y1-x2,y2

Vérification de la pièce et du tour :

Il faut que les coordonnées du départ correspondent à une pièce alors si la case est vide Ou si la couleur de la pièce ne correspond pas à la couleur qui a le tour le programme doit afficher une erreur.

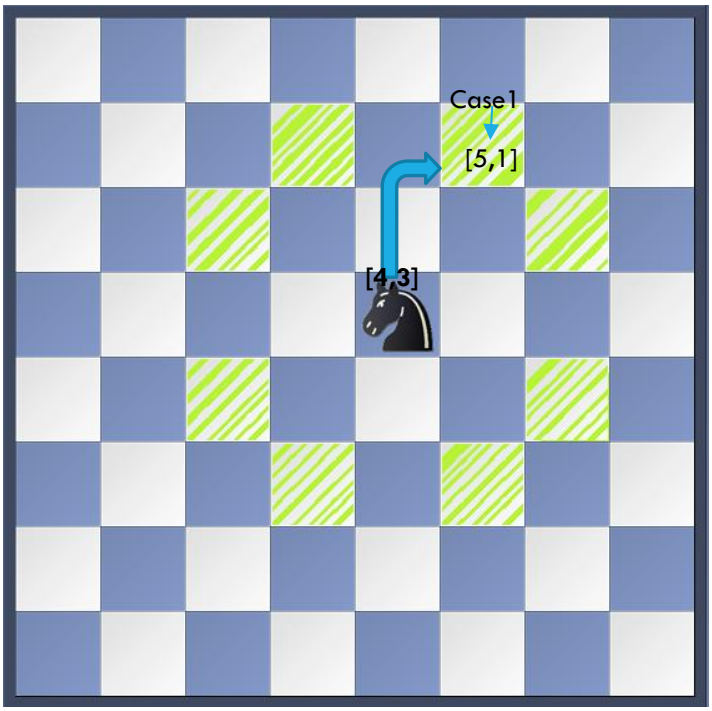
```
if(tour==0 && board[x1][y1]>0){
//Pièce noire
}
```

Vérification des règles d'échec :

Pour savoir si un déplacement est possible on doit vérifier la règle de chaque pièce (Pion-Cavalier-Tour…)

Par exemple :

- **Le Cavalier :**



Les conditions pour que le cavalier se déplace :

1-la case de la destination doit être vide ou a une pièce de la couleur opposée [Blanc pour notre cas (valeur négative)]

2- Les cavaliers se déplacent d'une manière très différente de celle des autres pièces - passent par deux cases dans une direction, puis une case de plus à un angle de 90 degrés, tout comme la forme d'un «L». Les cavaliers sont aussi les seules pièces qui peuvent se déplacer par-dessus d'autres pièces. Par exemple si on veut déplacer à la case num1 : on doit vérifier que x2=x1+1 et y2=y1-2

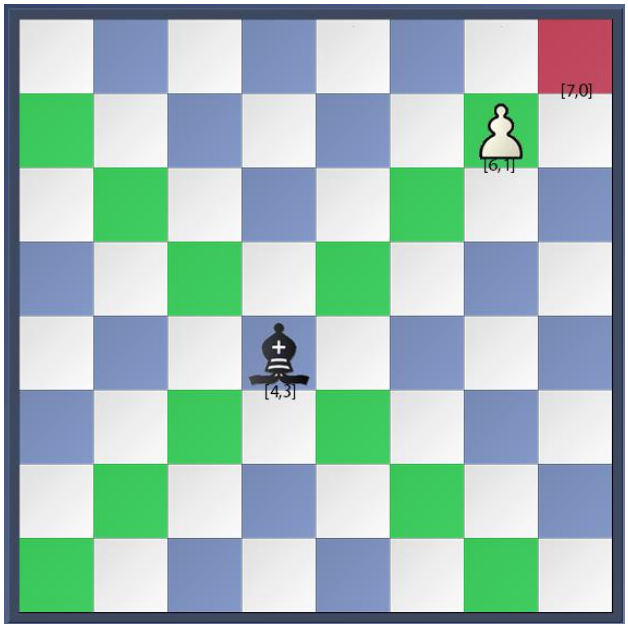
Ce qui donne en **C:**

1 **2**

```
if(board[x2][y2]<= 0 && (x2==x1+1 && y2==y1-2)) {
```

Nous répétons cette opération jusqu'à ce que nous couvrions tous les autres possibilités (8cases).

- **Le fou :**



Les conditions pour que le fou se déplace :

1- Le fou peut aller loin qu'il le souhaite, mais seulement en diagonale, alors on doit vérifier que la destination est en diagonale.
2- Le fou ne se déplacer pas par-dessus d'autres pièces, alors on doit vérifier s'il y a une pièce dans le chemin à la destination. (exemple : il ne peut pas se déplacer à la case rouge)

Réalisation :

1- Pour la première condition on cherche La relation entre un x1et x2, y1 et y2 qui se trouve dans le même diagonal ce qui donne x1-x2=y1-y2 ou x2-x1=y1-y2 ou y2-y1=x1-x2
2- Pour la deuxième condition on cherche case par case dans le diagonale jusqu'à la destination s'il y a d'autres pièces

Fonction en C :

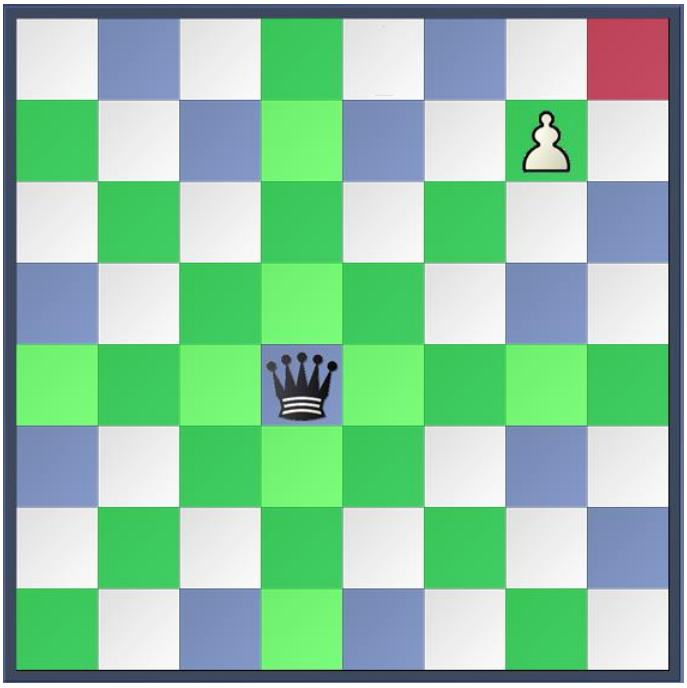
```
int verifdiagohd(int board[8][8],int x1,int y1,int x2,int y2){

    int k=1; int i=1;
    while(k==1 && !(x1-i==x2 && y1+i==y2)){

        if(board[x1-i][y1+i]==0){
           i++;
        }
        else{
           k=0;
        }
        }
```

Le code final :

```
//******************** Le fou (Bichop)
else if (board[x1][y1]==3){//Le fou
if(x1-x2==y1-y2  || x2-x1==y1-y2||y2-y1==x1-x2  && board[x2][y2]<=0){//la destination est dans les diagonal du fou
if(VerifDiago(board,x1,y1,x2,y2)){//une piece est dans le chemain du fou
    board[x1][y1]=0;
    board[x2][y2]=3;
    tour=1;
}
```

NB : VerifDiago() c'est la fonction qui vérifie s'il n'y a pas de pièces dans le chemin du fou

- **La reine**



La reine est la pièce la plus puissante. Elle peut se déplacer dans n'importe quelle direction en ligne droite - en avant, en arrière, sur les côtés ou en diagonale - aussi loin que possible tant qu'elle ne se déplace pas au travers de ses propres pièces. Et, comme avec toutes les pièces, si la reine capture une pièce adverse, son mouvement est terminé.

N.b : pour terminer un mouvement on change le tour.

Réalisation :

C'est un mélange de deux algorithmes : tour et le fou.

```
else if(board[x1][y1]==5){//La REINE Noire
   if( x1-x2==y1-y2  || x2-x1==y1-y2 || y2-y1==x1-x2 &&  board[x2][y2]<=0){
      if( VerifDiago(board,x1,y1,x2,y2)){//vérifier le Diagonale
        board[x1][y1]=0;
         board[x2][y2]=5;
         tour=1;
      }
      else{ printf("Ce deplacement est impossible");}
   }
   else if(y1==y2 || x1==x2 && board[x2][y2]<=0 ){
         if(VerifLine(board,x1,y1,x2,y2)){// vérifier les lignes
            board[x1][y1]=0;
            board[x2][y2]=5;
            tour=1;
         }
         }
   else {printf("Erreur deplacement ");}
```

Algo du fou

Algo du tour

2. Menue (sauvegarde…)

Pour avoir accès au menu nous devons entrer **« 8 »** comme une commande

Si le programme à détecter que le premier coordonnée est 8 il va afficher le menue :

```
else if(y1==8){//********MENUE
   printf("*********Menue***********\n");
   printf("1-Continuer la partie\n");
   printf("2-Nouvelle partie\n");
   printf("3-sauvegarder partie\n");
   printf("4-ouvrir partie\n");
   printf("5-sortir\n>");
   scanf("%d",&r);
} switch(r){//menue core
  case 1:
     break;
  case 2:
     initialisation(&board);
     break;
  case 3://sauvegarde
     printf("entrer le nom du partie >");
scanf("%s",nomparti);
```

```
**********  Joueur 1 (NOIRE) **********
 entrer une commande:
>8
**********Menue***********
1-Continuer la partie
2-Nouvelle partie
3-sauvegarder partie
4-ouvrir partie
5-sortir
>
```

1- **Continuer la partie :**

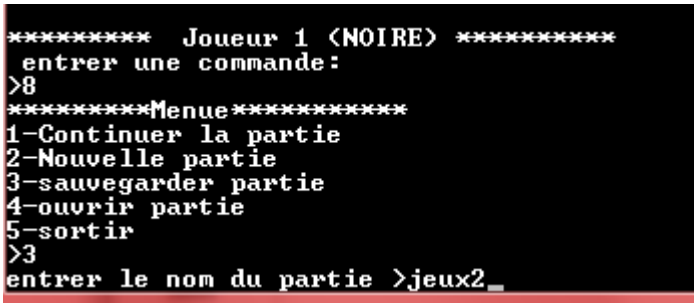
Le programme va tout simplement continuer la partie sans rien faire

2- **Nouvelle partie :**

Appeler la fonction initialisation() qui initialiser les positions des pièces

3- Sauvegarder partie :

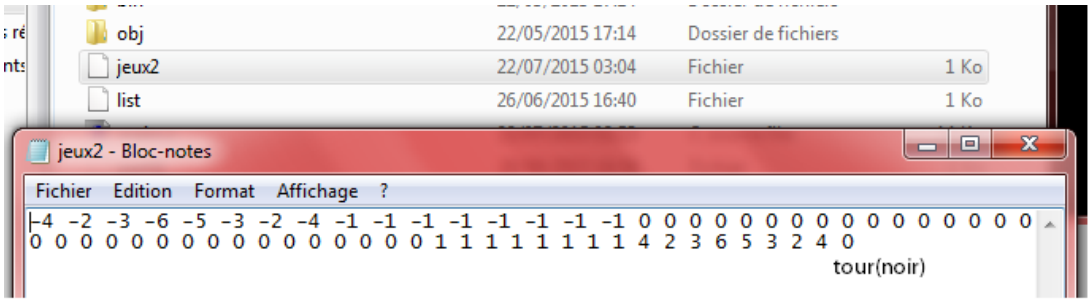
Pour sauvegarder une partie on écrit les valeurs du tableaux board[8][8] séparer par un espace puis on ajoute le tour(1/0) à la fin dans un fichier que l'utilisateur donne lui un nom (c'est le même nom de la partie) :



Réalisation En C :

```
 case 3://sauvegarde
    printf("entrer le nom du partie >");
scanf("%s",nomparti);
FILE *p=fopen(nomparti,"w");
  for(i=0;i<8;i++){
        for(j=0;j<8;j++){
  fprintf(p,"%d ",board[i][j]);
   }
   }
   fprintf(p,"%d",tour);

fclose(p);

printf("sauvegardee!");
```

Le ficher sauvegarder :



4- Ouvrir une partie :

Pour ouvrir une partie le programme remplace les valeurs dans le tableau par les valeurs lit depuis un fichier choisi par l'utilisateur (nom de la partie)

5- Sortir :

Fermer le programme par sortir de la boucle principale (while)

V-Conclusion :

La réalisation de ce projet m'a beaucoup aidé à développer mon analyse et esprit de programmation en particulier en matière de circuit logique on utilisant et on simplifiant beaucoup d'équation logique et de mettre en pratique nos connaissances théoriques dans ce cours. Aussi nous a t-elle permit de maîtriser le langage C et de l'utiliser d'une façon approfondie.

Références :

http://Chess.com [les bases]

http://Wikipedia.com

http://www.developpez.com [Statistique utilisation des langages de programmation]

http://jeffprod.com

Fretz 13 [Les figures]

Code Source :

https://goo.gl/OtOjjC

https://www.dropbox.com/home/Projet%20findannee?preview=Last.c