Rapport de projet

Échecs

BEAUJEAN Fabien
Année 2013/2014 L1
15/01/2014

Sommaire

Introduction

Le deuxième projet informatique que nous avons été amenés à réaliser dans le cadre des cours de programmation en langage C a été un jeu d'échecs.
Les principaux objectifs d'un tel programme ont été de gérer correctement un projet plus compliqué tant au niveau de la complexité des algorithmes mis en jeu qu'au niveau de la longueur du code.

Dans une première partie nous expliciterons l'organisation générale du projet ainsi que les choix effectués au début du développement. Nous étudierons ensuite les principaux algorithmes utilisés.

Description générale du projet

Les fonctionnalités implémentées

Ce programme d'échec comprend toutes les principales règles du jeu d'échec qui permettent au jeu d'être réellement jouable :

- Déplacement des pièces de manière générale
- Les coups spéciaux :
  - Prise en passant
  - Promotion
  - Roque
  - Echec
  - Echec et mat
  - Pat
  - Règle des 50 coups
  - Abandon
  - Match nul
- Création d'une interface graphique à l'aide de la SDL
- Ajout d'un menu
- Création d'une zone pour afficher les pièces capturées pendant la partie

Cependant, par manque de temps et du fait de contraintes telles que la validation de toutes les autres matières, certaines fonctionnalités telles que l'enregistrement ou le chargement de fichiers au format PGN, l'IA, la règles des trois coups, le problème des 8 dames, du cavalier, l'ajout d'animations pour les déplacements, le passage à la 3D, la mise en place de parties en réseau et la création de profils pour suivre sa progression qui étaient initialement prévues n'ont pas pu finalement être implémentées.

Découpage du code

Afin de saisir plus facilement la logique du code, de coder plus efficacement et d'alléger le poids des fichiers, il a été choisi de découper celui-ci en plusieurs fichiers représentant chacun un module bien particulier :

- Function.cpp, function.hpp : contient les définitions des variables ainsi que les fonctions d'initialisation et les fonctions régulièrement appelées, les defines, les énumérations.
- Engine.cpp engine.hpp : fonction principale du jeu qui appelle les bonnes fonctions en fonction de l'action demandée (nouvelle partie, enregistrement). Elle est assez courte mais c'est cette fonction qui est appelée en boucle et qui permet de pouvoir utiliser correctement les différents modules intégrés au jeu

- Menu.cpp menu.hpp : permet de gérer le menu principal du jeu, c'est-à-dire le menu apparaissant lors de l'appui sur Echap ou lors du clic sur « menu »
- Game.cpp, game.hpp : ces fichiers contiennent principalement les fonctions nécessaires à l'affichage des différents éléments du jeu. Elles affichent ainsi le plateau de jeu, les pièces sur l'échiquier, les pièces capturées et lors d'un tour, les cases jouables, les pièces capturables etc.
- Player.cpp, player.hpp : ces 2 fichiers contiennent la majorité du code source du programme. C'est cette partie qui se charge de contrôler les déplacements de toutes les pièces, les coups spéciaux (roque, prise en passant etc.), les fins de partie etc.

Méthode de gestion des données

Un jeu complet nécessite pour fonctionner un assez grand nombre de variables. Afin de pouvoir accéder à ces variables facilement, il a été choisi d'avoir recours aux structures. Ainsi, la principale structure du jeu est la structure **engine**. Cette structure contient elle-même une structure **menu** et **game**.

De même, la gestion de l'échiquier est faite à l'aide d'un tableau à deux dimensions de 8*8 de type **piece**, où piece est une structure.

Le principe est le même pour la plupart des éléments du jeu : menu, pièce etc.

De plus, afin de rendre le code source plus lisible, de nombreuses define et énumérations ont été créées pour par exemple indiquer si une case du jeu est un déplacement possible, si la case contient le roi et qu'il est en échec etc. Remplacer un simple chiffre par un terme plus verbeux mais plus compréhensible permet ainsi de simplifier la relecture du code.

Analyse des algorithmes mis en jeu

Échec

La mise en échec du roi est le principal élément lors du développement d'un jeu d'échec puisqu'on le retrouve absolument partout.
Dans mon implémentation du jeu d'échec, je distingue deux cas pour gérer la mise en échec : d'une part, le cas d'un déplacement simple, d'autre part le cas où le déplacement est un roque.

```
Si le roque est normalement possible, on vérifie que le roi n'est
pas mis en échec sur sa position initiale

Pour chacune des cases où l'on peut jouer, vérifier si en déplaçant
la pièce cela ne provoque pas une mise en échec du roi
```

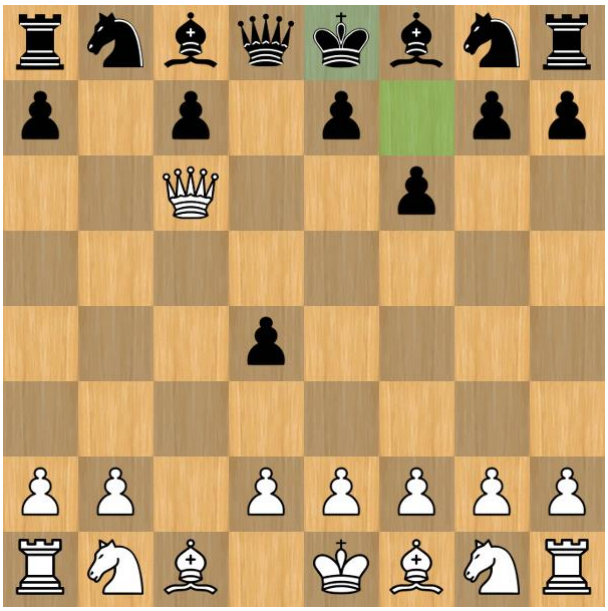
Pour cela, le principe est de simuler tous les déplacements possibles de la pièce souhaitée et de vérifier si ce déplacement n'entraîne pas la mise en échec du roi. Après la simulation du déplacement, on va boucler sur toutes les pièces ennemies et vérifier si une des pièces ne met pas le roi en échec. Dans ce cas, le déplacement possible est supprimé sinon, le déplacement est conservé.

Voici l'algorithme utilisé pour vérifier l'échec dans le cas simple

```
Pour (i de 0 à 7){
  Pour (j de 0 à 7){
    Si (echiquier[i][j] case jouable OU echiquier[i][j] case
traversée par le roque){
      simulationDeplacement();

      Si (la piece jouee est le roi){
        modificationPositionRoi();
      }

      //pour chaque pièce ennemie, on regarde si elle peut
      //attaquer le roi après le déplacement
      Pour (k de 0 à 7){
        Pour (l de 0 à 7){
           Si(echiquier[k][l].equipe =  ennemie){
             calculDeplacementPieceEnnemie(k,l);

             Si(le roi est mis en échec){
               annulationDeplacement();
             }
           }
        }
```

```
        }
      }
   }
}
```

Dans le cas du roque, la simple modification vient du fait que si le roi est en échec sur sa position initiale, alors tous les déplacements du roque sont annulés.



Mise en échec du roi noir

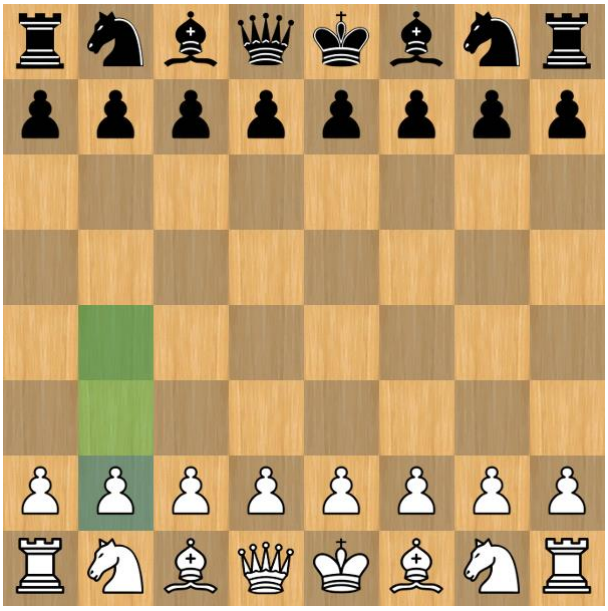
Déplacement générale des pièces

Déplacement du pion

Le déplacement du pion est l'un des plus simples à mettre en œuvre. Il consiste simplement à vérifier que les quelques positions jouables autour du pion le sont effectivement.

```
// [i][j] correspond à la position actuelle de la pièce
Si(les blancs jouent){
  Si(echiquier[i][j+1].type = vide){
    deplacementPossible([i],[j+1]);
  }

  Si(echiquier[i][j+1].type = vide ET echiquier[i][j+2.type = vide){
    deplacementPossible([i],[j+2]);
  }

  Si(echiquier[i+1][j+1].equipe = ennemie){
    deplacementPossible([i+1],[j+1]);
```

```
  }

  Si(echiquier[i-1][j+1].equipe = ennemi){
    deplacementPossible([i-1],[j+1]);
  }

}
Sinon{
  //même principe, seules les positions à tester changent
}
```



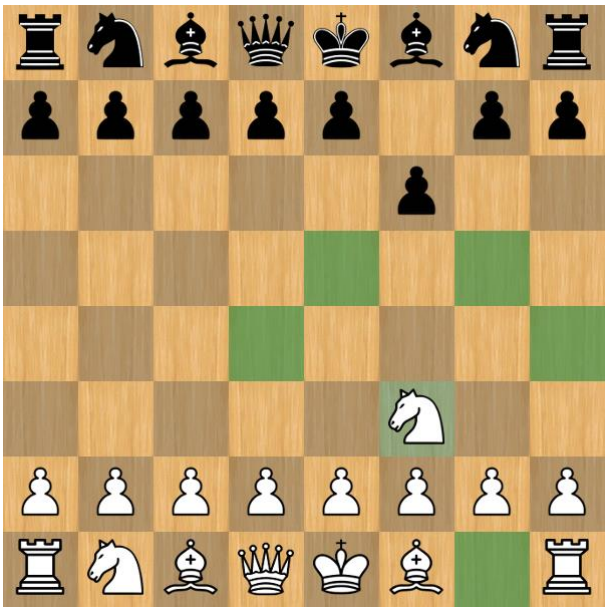
Déplacement d'un pion

Déplacement du cavalier

De la même façon que pour le pion, le déplacement du cavalier est assez facile à gérer puisqu'il suffit encore de tester certaines cases précises.

```
// [i][j] correspond à la position actuelle de la pièce
Si(echiquier[i-1][j-2].equipe = ennemie){
  deplacementPossible([i-1],[j-2]);
}

Si(echiquier[i-2][j-1] n'est pas de mon équipe){
  deplacementPossible([i-2],[j-1]);
}

Si(echiquier[i-2][j+1].equipe = ennemie){
  deplacementPossible([i-2],[j+1]);
}
```

```
Si(echiquier[i-1][j+2].equipe = ennemie){
  deplacementPossible([i-1],[j+2]);
}

Si(echiquier[i+1][j+2].equipe = ennemie){
  deplacementPossible([i+1],[j+2]);
}

Si(echiquier[i+2][j+1].equipe = ennemie){
  deplacementPossible([i+2],[j+1]);
}

Si(echiquier[i+2][j-1].equipe = ennemie){
  deplacementPossible([i+2],[j-1]);
}

Si(echiquier[i+1][j-2].equipe = ennemie){
  deplacementPossible([i+1],[j-2]);
}

Si(echiquier[i-2][j-1].equipe = ennemie){
  deplacementPossible([i-2],[j-1]);
}
```

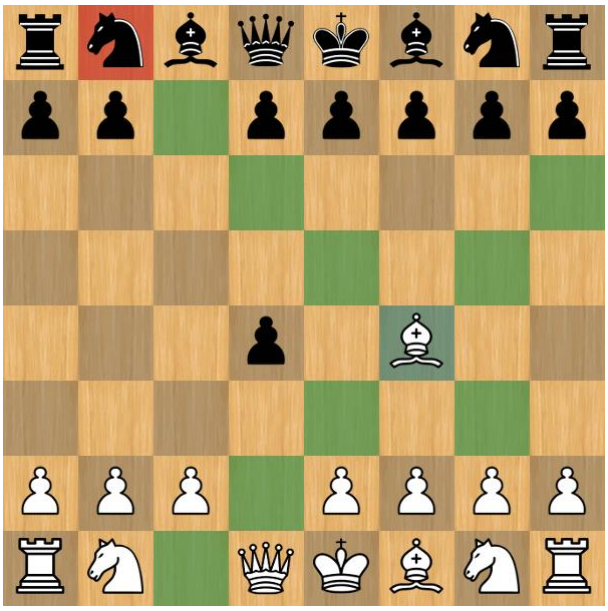


Déplacement d'un cavalier

Déplacement du fou

Pour gérer le déplacement du fou, il suffit de tester les diagonales bas-gauche, haut-gauche, haut-droite, bas-droite à l'aide de 4 boucles différentes en faisant bien attention à ce que les cases contenant une pièce de la même équipe bloque la trajectoire sans pouvoir être jouées et que les cases contenant une pièce ennemie bloque la trajectoire tout en pouvant être prises.
Voici l'algorithme pour le déplacement bas-gauche (les autre directions étant quasiment les mêmes)

```
// [i][j] correspond à la position actuelle de la pièce
i = i-1;
j = j-1;

Tant que (trajectoireBloquee = 0 ET i >= 0 && j >=0){
  Si(echiquier[i][j].equipe = ennemie){
    deplacementPossible([i],[j]);
  }
  Sinon{
    trajectoireBloquee = 1;
  }

  Si(echiquier[i][j].equipe = ennemie){
    trajectoireBloquee = 1;
  }

  i <- i-1;
  j <- j-1;
}
```

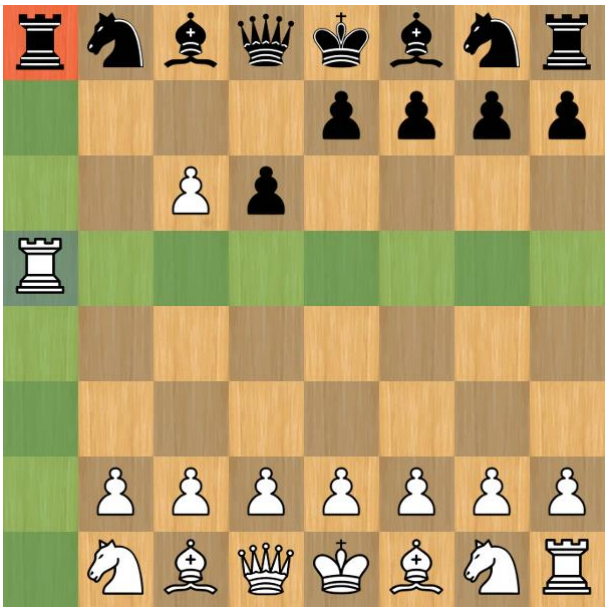


Déplacement d'un fou

Déplacement de la tour

Le déplacement d'une tour est similaire au déplacement d'un fou excepté que les déplacements ne sont plus en diagonale mais horizontales et verticales. Voici l'implémentation de l'algorithme pour le déplacement vers la gauche.

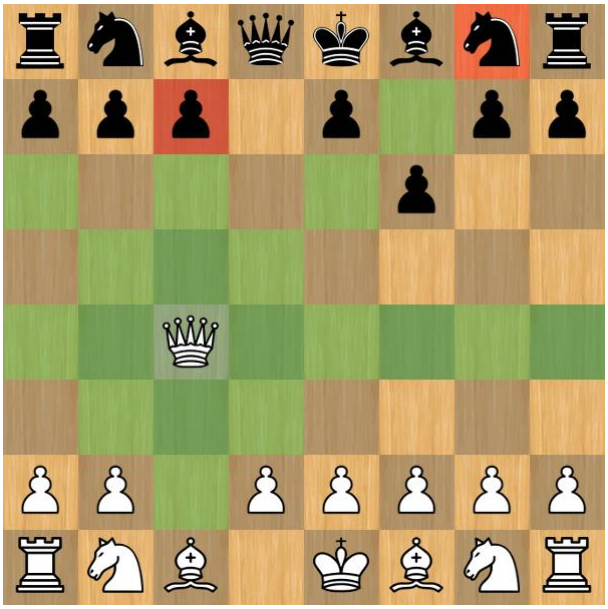
```
// [i][j] correspond à la position actuelle de la pièce
i = i-1;

Tant que (trajectoireBloquee = 0 ET i >= 0){
  Si(echiquier[i][j].equipe = ennemie){
    deplacementPossible([i],[j]);
  }
  Sinon{
    trajectoireBloquee = 1;
  }

  Si(echiquier[i][j].equipe = ennemie){
    trajectoireBloquee = 1;
  }

  i <- i-1;
}
```



Déplacement d'une tour

Déplacement de la reine

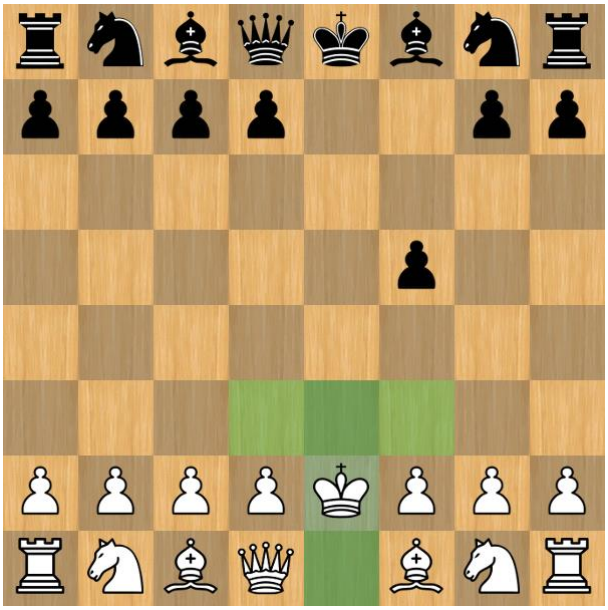
Le déplacement de la reine est extrêmement simple puisqu'il reprend le déplacement de la tour et du fou.



Déplacement d'une reine

Déplacement du roi

Le déplacement du roi est aussi extrêmement simple et très similaire à celui du pion ou du cavalier puisqu'il ne fait que tester les cases adjacentes.



Déplacement d'un roi

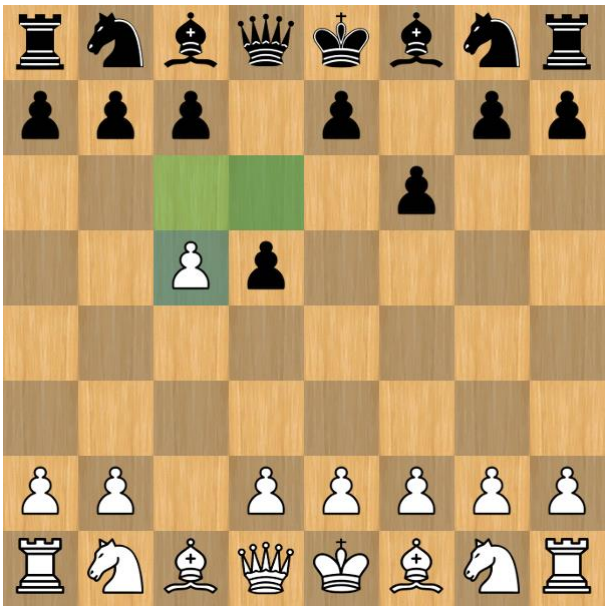
Gestion des coups spéciaux

D'après les conventions fixées au début du développement, mon programme calcule tout d'abord tous les coups possibles sans tenir compte des coups spéciaux et autre règles. Une fois tous les déplacements calculés, chacun des coups spéciaux va être appliqué à ces déplacements.

Prise en passant

La prise en passant désigne le fait que lorsqu'un pion se trouve sur la cinquième rangée et que l'adversaire avance un pion d'une colonne voisine de deux cases (les deux pions se retrouvent alors côte à côte sur la même rangée), le premier pion peut prendre le second.

```
// [i][j] correspond à la position actuelle de la pièce
//blancs : y = 5, noirs : y = 3 //on regarde si  la pièce est un
pion
Si((echiquier[i][j].equipe = blanc ET j = 5-1) OU
(echiquier[i][j].equipe = noir ET j = 4-1) ET echiquier[i][j].type =
pion){
    //on regarde si les cases à côté contiennent un pion
    //on regarde si le pion s'est déplacé une seule fois de 2 cases
    // si adverse
    Si(echiquier[i-1][j].type = pion ET echiquier[i-
1][j].tailleDernierDeplacement = 2 ET echiquier[i-1][j].equipe =
ennemie){
        si(echiquier[i][j].equipe = blanc ET dernierDeplacementNoirX
== i-1 ET dernierDeplacementNoirY == j){
            deplacementPossible(i-1,j+1);
            deplacementPossiblePriseEnPassant(i-1,j);
        }
        Sinon Si(echiquier[i][j].equipe = noir ET
dernierDeplacementNoirX == i-1 ET dernierDeplacementNoirY == j){
            deplacementPossible(i-1,j-1);
            deplacementPossiblePriseEnPassant(i-1,j);
        }
    }
    Sinon Si(echiquier[i+1][j].type = pion ET
echiquier[i+1][j].tailleDernierDeplacement = 2 ET echiquier[i-
1][j].equipe = ennemie){
        //presque pareil
    }
}
```

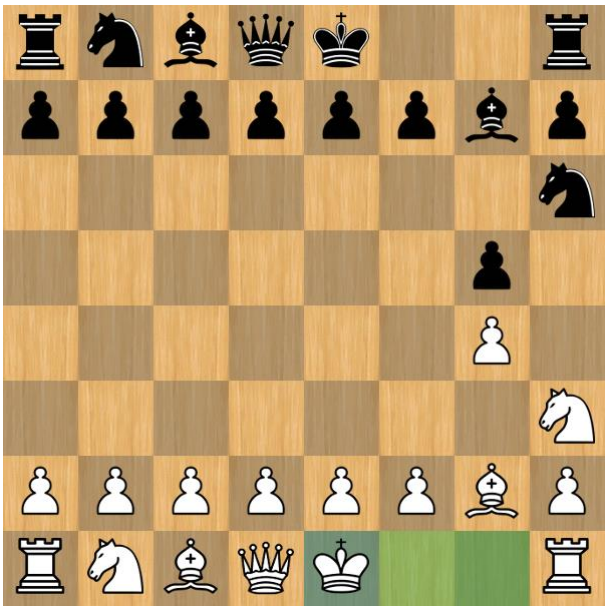


Prise en passant

Roques

Le roque est un déplacement spécial du roi et d'une des tours au jeu d'échecs. Le roque permet, en un seul coup, de mettre le roi à l'abri tout en centralisant une tour. C'est le seul coup légal permettant de déplacer deux pièces.

```
// [i][j] correspond à la position actuelle de la pièce
//on a bougé le roi pour la première fois, sur la même rangée
if(echiquier[i][j].type = ROI ET echiquier[i][j].nombreDeplacement =
0){
    //si la case finale est à côté de la tour de droite et que la
tour n'a pas encore bougé
    //si les cases entre sont libres
    Si(i+3 < 8
    ET echiquier[i+3][j].type = TOUR
    ET echiquier[i+3][j].nombreDeplacement = 0
    ET echiquier[i+1][j].type = EMPTY
    ET echiquier[i+2][j].type = EMPTY){
        deplacementPossible([i+2],[j]);
        deplacementPossibleRoque([i+3],[j]);
    }

    //le grand roque est très similaire au petit roque
    }
}
```



Petit roque du roi blanc

Échec et mat

L'échec et mat est une situation dans laquelle un roi est menacé de capture au prochain coup et pour laquelle aucune parade n'est possible quelle que soit la pièce jouée. La partie prend alors fin immédiatement (le roi n'est jamais capturé effectivement) et le joueur qui inflige le mat en est déclaré vainqueur.

```
Si(enEchec = 1){
    Pour (i de 0 à 7){
        Pour (j de 0 à 7){
            Si(echiquier[i][j].equipe = monEquipe){
                calculDeplacementPiece(i,j);

                Si(nombreDeplacementPossible > 0){
                    echecEtMat = 0;
                }
            }
        }
    }
}
```

Message de fin de partie lors d'un échec et mat

Gestion du pat

Le pat est assez similaire à l'échec et mat dans la pratique puisque bien que le roi d'un des deux camps ne soit pas en échec, aucun mouvement de lui-même ou des pièces de son équipe n'est possible. Dans ce cas, la partie s'arrête immédiatement et le match est déclaré comme nul.

```
Si(enEchec = 0){
    Pour (i de 0 à 7){
        Pour (j de 0 à 7){
            Si(echiquier[i][j].equipe = monEquipe){
                calculDeplacementPiece(i,j);

                Si(nombreDeplacementPossible > 0){
                    enPat = 0;
                }
            }
        }
    }
}
```

Message de fin de partie lors d'un pat

Règle des 50 coups

La règle des 50 coups stipule que si il s'est passé 50 coups depuis la dernière fois qu'une pièce a été capturée ou qu'un pion a été promu, alors, on doit demander aux joueurs s'ils veulent arrêter la partie et la considérer comme nulle.

Ainsi, on a une variable nommée **nombreCoupLastCapturePromotion** qui contient comme son nom l'indique le nombre de coups depuis la dernière capture ou la dernière promotion.
Ensuite, dans la boucle principale de jeu, celle qui permet de sélectionner des pièces, on effectue la condition suivante afin d'afficher le menu des 50 coups :

```
Si(nombreCoupLastCapturePromotion + 50 < nombreDeplacementTotal){
    menu50Coup();
}
```

Promotion

Lorsqu'un pion arrive sur le côté opposé de l'échiquier, le joueur a le droit de changer ce pion en une autre pièce : fou, cavalier, tour, reine.
L'algorithme utilisé pour ce coup est extrêmement simple :

```
//exemple d'implémentation pour les blancs. I et j, représente la
position finale de la pièce. 7 correspond à la dernière rangée des
pièces noires
Si(echiquier[i][j].equipe = monEquipe ET j = 7){
  menuPromotion() ;
```

```
}
```

Le menu promotion quant à lui va afficher les 4 pièces et comme c'est expliqué en annexe va permettre au jour de choisir laquelle des pièces il désire obtenir. Lorsque le joueur a choisi, le programme va changer le type du pion que le joueur avait déplacé.

Déplacement d'une pièce d'échec

Le déplacement d'une pièce est l'étape finale d'un tour de jeu ; après la sélection de la pièce puis de la position d'arrivée. Cette étape déplace effectivement la pièce voulue. Mais elle doit également mettre à jour les données du jeu, détruire les pièces qui doivent l'être, gérer certains des coups spéciaux notamment le pat.
Voici les différentes étapes remplies par ma fonction **deplacePiece() :**

- La case d'arrivée contenait-elle une pièce adverse ? Dans ce cas, on supprime la pièce adverse du plateau de jeu et on la stocke dans le tableau contenant les pièces capturées
- Dans le cas de la prise en passant, il faut supprimer la pièce adverse située juste en dessous de la position finale
- Dans le cas du roque, il faut déplacer à la fois le roi et la tour
- On met à jour la position de la pièce jouée
- On met à jour l'ancienne position puisqu'elle est maintenant vide
- On met à jour toutes les données propres à mon jeu et nécessaires à son fonctionnement

Déroulement d'un tour de jeu

Un tour commence lors de la sélection d'une pièce par le joueur. A ce moment-là, si le joueur est autorisé à sélectionner cette pièce, alors le programme va calculer tous les déplacements possibles sans tenir compte des coups spéciaux.
Une fois ceci fait, le programme va gérer tous les coups spéciaux. Cela va donc au choix soit supprimer certains déplacements auparavant possibles (mise en échec) soit en rajouter de nouveaux (roque, prise en passant).
Lorsque tous les coups possibles ont été calculés, le jeu affiche à l'écran les zones jouables et attend de l'utilisateur qu'il choisisse.
Lorsque l'utilisateur a choisi une case, le programme vérifie que la case est bien jouable et si c'est le cas, il appelle la fonction **deplacePiece(),** puis termine le tour.

Conclusion

Ce projet plus ambitieux et plus complexe m'a permis de mettre à profit les techniques apprises en cours et en TP, ainsi que d'adopter une plus grande rigueur dans l'élaboration du code source. Cela m'a également permis d'améliorer mes compétences en termes de correction d'erreurs.

Cependant, bien que le jeu puisse être considéré comme fini, il est toujours améliorable via l'ajout de nouvelles fonctionnalités ou l'optimisation de certains algorithmes.

Annexes

Technologies employées

La SDL (Simple DirectMedia Layer)

Cette bibliothèque a permis de gérer toute la partie graphique du programme : affichage de la fenêtre, des éléments de jeu, gestion des évènements de la souris et du clavier.

La SDL Image

Comme la SDL ne gère de base que les formats Bitmap (.bmp), cette librairie supplémentaire a été utilisée afin de pouvoir recourir à des formats tels que le PNG qui supportent la transparence d'une façon beaucoup moins contraignante que le bitmap.

La SDL ttf

Ce jeu affiche également quelques textes. Pour rendre cela possible, une troisième et dernière librairie a été employée : la SDL ttf qui gère de nombreux formats de polices de caractères.

Autres fonctionnalités gérées par le programme

Gestion du clic sur les cases et sur les menus

Afin de gagner du temps, la gestion du clic sur les différents éléments du jeu (cases, menu) a été mise au point afin d'être réutilisées au maximum. Le principe est simple : un bouton d'un menu est codé sous la forme d'une structure renseignant sa position et sa taille.
Ensuite, un menu complet peut être assimilé à un tableau de ces structures menu.

Ainsi, la fonction déterminant quel a été le menu cliqué est très simple : elle prend en paramètre les coordonnées du clic, le tableau de structures et le nombre d'éléments du tableau. Ensuite, elle parcourt le tableau en testant grâce à la position et à la taille de chaque menu lequel a été cliqué et retourne le numéro de la case du tableau correspondante.

Webographie

- Openclassrooms.com : cours sur la SDL

- Wikipédia.org : les règles du jeu d'échecs
- Stackoverflow : les questions purement techniques au sujet de certains problèmes vis-à-vis du code source