Insa Rennes

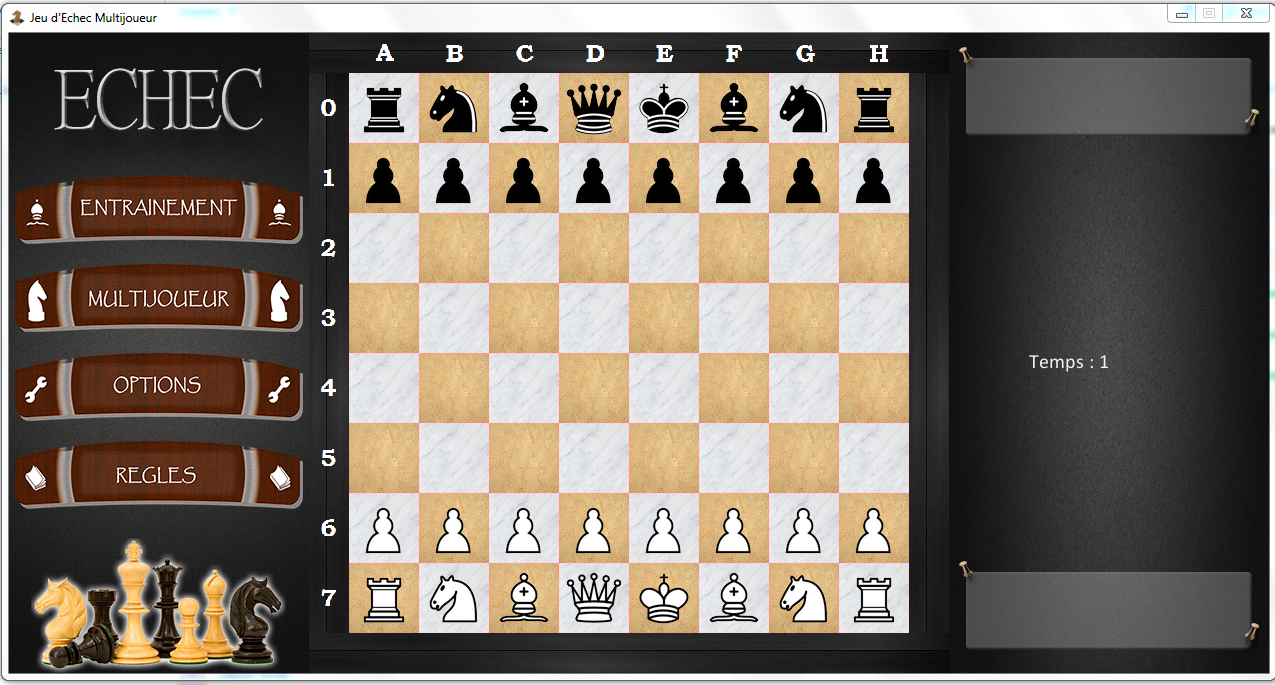
Département Electronique

et Informatique Industrielle

Année 2014/0215

Projet de Programmation en C

Jeu d’Echec



MIQUEU Clément 3EII

MONNIER Nicolas Enseignant référent :

TASSE Alexis M. Laurent BEDAT

VIGNAL Alexandre

Table des matières

[Introduction 3](#_Toc420360146)

[1. Présentation générale du projet. 3](#_Toc420360147)

[2. Gestion des déplacements 4](#_Toc420360148)

[3. Gestion de l’affichage 4](#_Toc420360149)

[4. Gestion des erreurs 5](#_Toc420360150)

[5. Gestion des évènements 5](#_Toc420360151)

[6. Gestion du timer 5](#_Toc420360152)

[Conclusion : 6](#_Toc420360153)

[Annexe 1 : Diagramme UML initial 7](#_Toc420360154)

[Annexe 2 : Avancée du projet 8](#_Toc420360155)

[Annexe 3 : Diagramme UML final 9](#_Toc420360156)

# Introduction

Pour ce projet de programmation en langage C, nous avons choisi de réaliser un jeu d’échec en « pseudo-3D », c’est-à-dire en 3D isométrique, où il n’y a qu’un seul angle de vue. Nous développerons d’abord une première version sans intelligence artificielle, puis éventuellement une seconde version avec intelligence artificielle selon le temps et les capacités en réserve. L’objectif du projet est d’appliquer nos connaissances acquises durant le module de langage C.

Nous aurons cependant besoin d’outils complémentaires comme la bibliothèque SDL, afin de pouvoir gérer la partie graphique du jeu.

## Présentation générale du projet.

Nous sommes partis dans l’idée de faire deux modes de jeu : un « multijoueur » où deux joueurs vont s’affronter et un « entraînement » où on peut placer ses pièces dans une situation donnée afin d’élaborer une stratégie pour une partie ultérieure.

Avant de commencer à coder, nous avons établi un cahier des charges qui nous a donné la structure générale suivante :

Figure 1 Structure générale de notre jeu d'échec

Nous avons ensuite créé un diagramme UML (annexe 1) à partir de cette ébauche de structure, ce qui a révélé plus de structures et de complexité que ce que nous avions imaginé.

Un timer sera activé à chaque début de partie afin de pouvoir fixer un temps par coup ainsi que pour permettre de sauvegarder la partie dans un historique si un des deux joueurs voulait revoir la partie. Il y aurait également une mise en surbrillance des cases afin de visionner les déplacements possibles lorsque l’on clique sur une pièce, ainsi que l’affiche de messages, lors d’une mise en échec par exemple.

La création de ce jeu se découpe en deux parties évidentes :

* Une graphique afin de visualiser le plateau de jeu
* Une concernant la gestion de jeu et des évènements, avec notamment les déplacements et la sauvegarde des parties.

Afin de gérer au mieux le partage du code source, nous avons choisi d’utiliser GitHub. En effet, sa prise en main est relativement aisée et permet un bon suivi du code source.

## Gestion des déplacements

Afin de pouvoir gérer au mieux les déplacements, on a assimilé l’échiquier à une matrice à deux dimensions (8\*8). On affecte 0 aux cases occupées et donc sur lesquelles la pièce sélectionnée ne peut se déplacer, et 1 aux cases libres. Lorsque les cases sont à 1 et qu’on a sélectionné une pièce, les cases correspondant aux mouvements possibles se mettent en surbrillance.

Les déplacements sont calculés selon le type de pièce sélectionnée grâce à un *switch/case.* On remet à chaque fois la matrice de déplacements possibles à 0, puis on calcule les déplacements grâce à des deltas prédéfinis. Dans le cas de la dame, il suffit de combiner les déplacements de la tour et du fou .

Tous ces déplacements sont enregistrés dans une liste chaînée, afin notamment, de pouvoir défaire un coup joué. On vient sauvegarder cette liste dans un fichier texte « Historique.txt » pour pouvoir conserver un aperçu de la partie, même lorsqu’on a fini la partie et fermé l’exécutable.

## Gestion de l’affichage

Nous sommes partis sur 2 écrans principaux : un écran d’accueil et un écran en mode jeu comme on peut le voir ci-dessous :



Figure 2 Aperçu de l'écran d'accueil

Figure 3 Aperçu de l'écran en mode jeu

A COMPLETER !!!

## Gestion des erreurs

Dans l’optique de gérer et stocker les erreurs renvoyées par notre code, nous avons décidé de créer une fonction logPrint spécifique. Cette fonction ouvre un fichier texte en mode écriture et prend deux paramètres :

* Le type de message : INFO, AVERTISSEMENT ou ERREUR qui nous donne ainsi une indication sur la nature de l’erreur produite par le code.
* Une chaîne de caractère contenant le message en question.

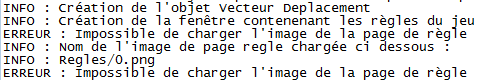
Cela nous donne un fichier de cette forme :

Figure 2 Exemple de fichier log généré

## Gestion des évènements

La bibliothèque SDL possède un nombre très important de fonctionnalités. Au cours de notre projet, nous avons eu besoin de gérer tous les évènements résultant d’une interaction entre le joueur et le programme.

Afin de faciliter le développement, nous avons défini à l’aide des #define des raccourcis pour tous les évènements susceptibles d’arriver dans le jeu. En effet, le code pour la gestion des évènements avec la SDL peut s’avérer lourd et difficile à gérer tant les instructions peuvent être longues. Cette gestion s’effectue par le biais de la fonction mettreAJourEvent du fichier evenement.c.

## Gestion du timer

Dans notre cahier des charges, nous avons décidé d’implémenter un timer qui nous servira à enregistrer la partie dans un historique et aussi à limiter le temps à chaque coup. Pour cela, nous avons dû installer une bibliothèque supplémentaire, SDL\_TTF.

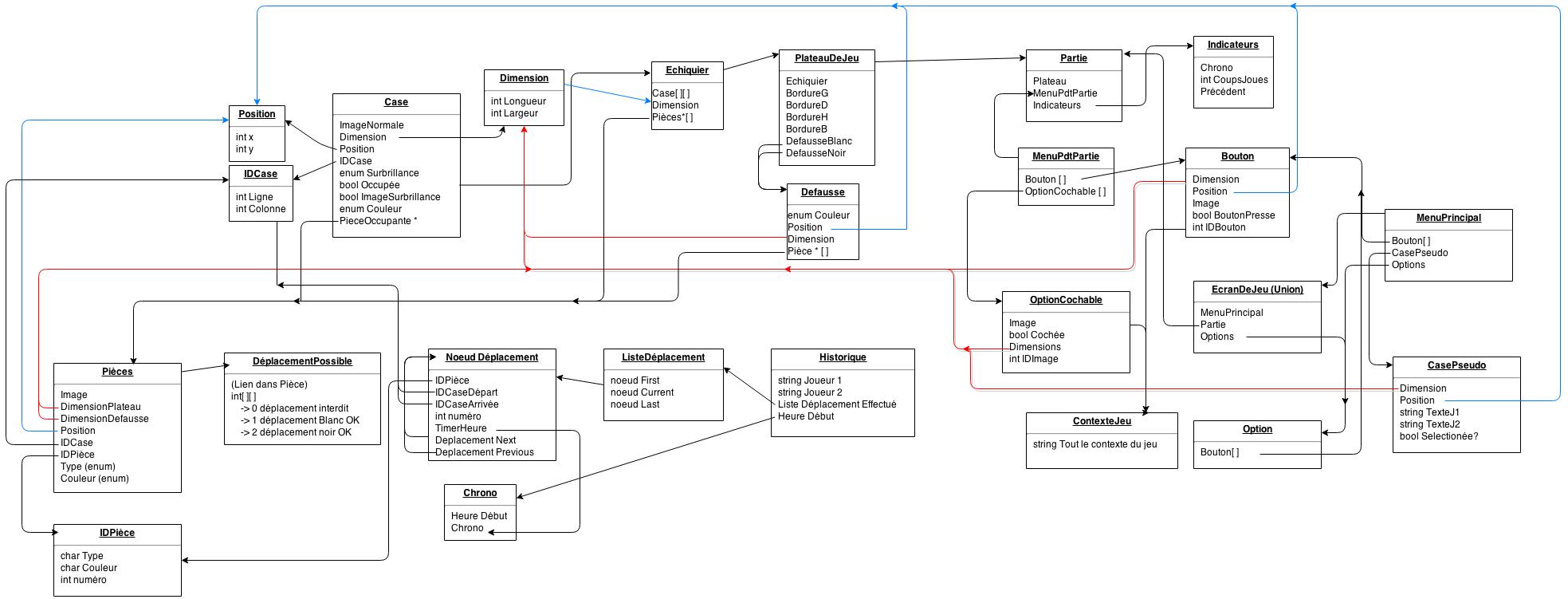
Nous avons créé une structure Timer comprenant notamment un buffer qui…….., les différents temps nécessaires au chronométrage de la partie et un booléen reaffichageNécessaire qui passe à 1 lorsqu’une erreur de timer apparaît. Afin d’afficher le temps actuel, on utilise la fonction SDL\_GetTicks présente dans la librairie SDL\_TTF qui donne le temps écoulé depuis la dernière initialisation de la SDL en ms. Il faut toutefois faire attention au format retourné par cette fonction, il s’agit d’un Uint32 qui permet d’avoir un format fixe quel que soit le système utilisé.

# Conclusion :

En conclusion, le cahier des charges a été respecté, avec cependant quelques modifications de développement comme nous pouvons le voir en comparant les deux diagrammes UML. Il reste encore plusieurs pistes d’améliorations comme le développement d’une intelligence artificielle permettant de jouer en solo, ou encore l’implémentation de toutes les combinaisons possibles, en plus du roque.

Sur le plan technique, ce projet nous a permis d’appréhender la librairie SDL avec la gestion des différents évènements (saisie au claver, clic souris…) ainsi que la gestion du temps avec le timer. Nous avons également appris à faire des diagrammes UML afin de structurer le projet dès le départ et à gérer le partage du code source grâce à GitHub. Enfin, il nous a permis d’apprendre à gérer un projet, avec notamment des réunions hebdomadaires, aptitude qui sera mise à contribution lors de notre carrière d’ingénieurs.

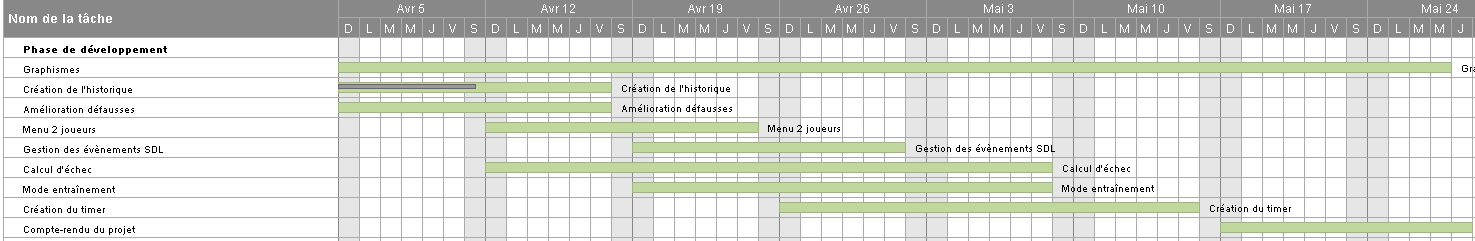
## Annexe 1 : Diagramme UML initial



## Annexe 2 : Avancée du projet







## Annexe 3 : Diagramme UML final

