Projet de langage C – 2014/2015

Réalisation d'un jeu d'échec à l'aide de la bibliothèque SDL

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

Département Electronique et Informatique Industrielle – 3ème année.

Clément MIQUEU
Nicolas Monnier
Alexis TASSE
Alexandre VIGNAL

Sommaire

Introduction

Pour ce projet de programmation en langage C, nous avons choisi de réaliser un jeu d'échec, l'idée initiale étant de réaliser une première version sans intelligence artificielle, puis éventuellement une seconde version avec intelligence artificielle selon le temps et les capacités en réserve. L'objectif de ce projet est donc d'appliquer nos connaissances acquises durant le module de langage C afin de mettre en œuvre un programme ludique, fonctionnel et stable.

En plus des connaissances acquises en cours, nous aurons besoin d'outils complémentaires à l'image de la bibliothèque SDL, afin de pouvoir gérer les graphismes du jeu.

1. Présentation générale du projet.

Initialement, nous avions prévu de réaliser deux grands modes de jeu : un mode solo, avec donc une intelligence artificielle, et un mode multijoueur, où deux joueurs s'affrontent sur le même ordinateur. Toutefois, par une réelle contrainte de temps, nous avons reconsidéré notre idée initiale pour finalement choisir de développer un mode différent du mode solo : le mode entrainement. Dans ce mode, les joueurs peuvent placer leurs pièces à leur guise sur le plateau afin de s'entrainer à partir de situations précises et ainsi élaborer une stratégie pour les parties ultérieures.

Avant de commencer à coder, nous avions établi une hiérarchie, très simple, de l'idée que nous nous faisions du jeu. Cette hiérarchie était la suivante.

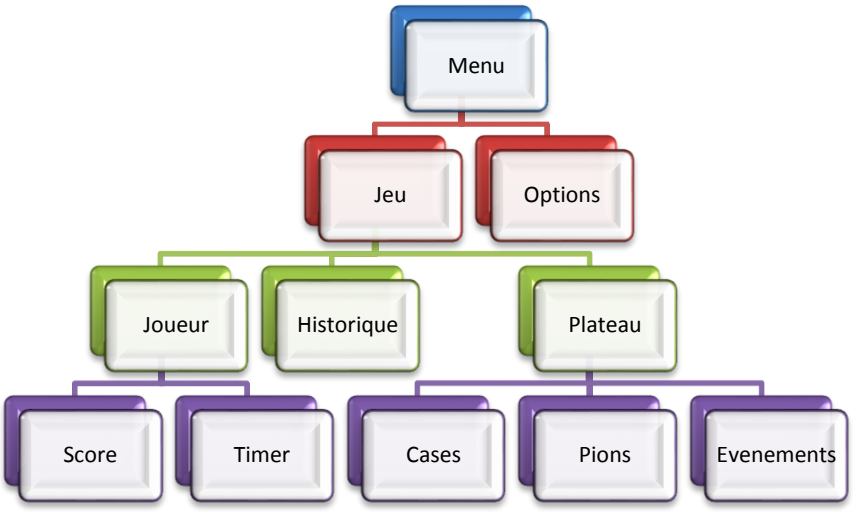


Figure 1 Structure générale de notre jeu d'échec

Toutefois, le projet étant en réalité plus complexe, nous avons créé un diagramme UML (annexe 1) à partir de cette ébauche de structure, nous permettant d'envisager toutes les structures nécessaires à la bonne réalisation du jeu, les méthodes et les points difficiles. Aussi, il s'est rapidement avéré que pour un tel projet, un langage de programmation orientée objet aurait peut-être été plus adapté. Toutefois, le jeu n'en reste pas moins parfaitement réalisable en langage C.

Voici donc le cahier des charges de notre projet :

CAHIER DES CHARGES	
Un mode multijoueur	Mode de jeu classique, pour 2 personnes sur le même ordinateur
Un mode entrainement	Mode de jeu permettant de positionner des pièces à sa guise sur le plateau et de lancer la partie dans une configuration précise
Des options réglables	Temps de jeu : illimité ou 15, 30, 45, 60 min par partie et par joueur
	Surbrillance : Mise en surbrillance des déplacements possibles
	Possibilité de retour en arrière après un coup joué
Un historique des déplacements effectués	
La possibilité d'entrer un pseudo : le jeu indique alors quelle est la personne qui doit jouer	
L'affichage des défausses de pièces déjà prises par l'adversaire	
L'affichage des évènements importants du jeu : échec, échec et mat et pat.	
L'affichage du chronomètre de chacun des joueurs durant leur tour de jeu	
Une partie Règles permettant d'expliquer succinctement les règles du jeu d'échec	
Une interface soignée et intuitive	

La création de ce jeu se décompose ainsi en deux grandes parties:

- Une partie graphique permettant de visualiser les éléments du plateau et les menus
- Une partie traitant les évènements, les calculs de déplacements, les vérifications d'échec, et les sauvegardes d'historiques, partie permettant donc l'interactivité homme/jeu.

2. Outil de collaboration pour le développement du projet

Afin de gérer au mieux le partage du code source et puisque nous avons choisi d'utiliser l'environnement Visual Studio pour développer notre code source, nous avions initialement choisi de partager notre projet grâce à un répertoire Visual Studio stocké en ligne, le logiciel permettant ensuite de se connecter à un projet d'équipe sur ce type de serveur.

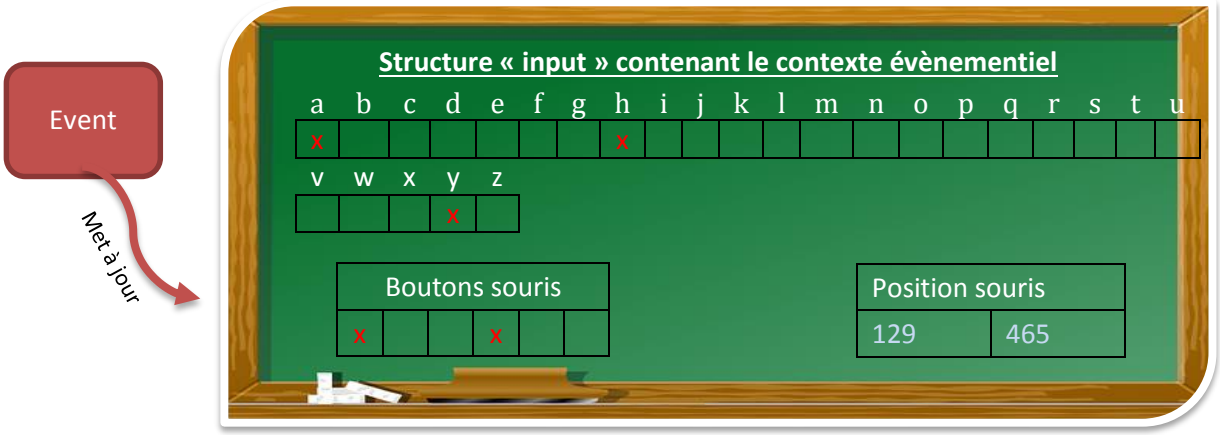
Toutefois, nous avons rencontré énormément de problèmes de conflits et de synchronisation, et avons donc décidé d'utiliser le logiciel et les serveurs de GitHub pour partager ensemble notre code. Ce logiciel nous aura permis un gain de temps énorme, tant par sa facilité d'utilisation que par sa puissance de fonctionnement et les services qu'il nous a apporté. Le code de notre projet est donc désormais open source et disponible sur le site github.com

3. Gestion des évènements

La bibliothèque SDL est une bibliothèque relativement complète et simple d'utilisation, et notamment un nombre important de fonctionnalités quant à la gestion des évènements. Voici donc une brève explication de la manière dont nous avons décidé d'utiliser ces fonctionnalités.

La SDL permet de stocker le contexte évènementiel dans une variable de nommée « event ». Plutôt que d'utiliser directement le contenu de cette variable dans la fonction main de notre programme pour effectuer nos tests et faire réagir le programme en conséquence, nous avons choisi de l'utiliser pour mettre à jour, à chaque passage dans notre boucle principale, une structure que nous avons défini.

Le fonctionnement de cette structure, nommée « input », est assez simple à comprendre. Elle se comporte comme une sorte de tableau qui contient les touches clavier pressées, la dernière position de la souris, les boutons de la souris enfoncés et est constamment mise à jour grâce à la fonction « mettreAJourEvents »… Le schéma suivant permet de bien comprendre le fonctionnement de cette structure.



Enfin, afin de faciliter la compréhension du code, nous avons décidé de prédéfinir un grand nombre de test dans un fichier header à l'aide de directives de préprocesseurs, à l'exemple de :

```
#define CLIC _SOURIS_INTERIEUR_MENU_GAUCHE (in.clicSouris.x > 0 && in.clicSouris.x < OFFSET_PLATEAU_GAUCHE)
```

4. Gestion du Timer

Dans notre cahier des charges, nous avons décidé d'implémenter un timer, permettant d'attribuer à chaque joueur un chronomètre, dont le temps de départ est modifiable grâce au menu Options. A chaque tour, ce chronomètre se vide un peu plus, et s'il atteint 0 pour l'un des deux joueurs, celui-ci perd la partie.

Nous avons donc créé une structure Timer comprenant notamment le temps de début de jeu, le temps accordé à chaque joueur pour une partie, le temps actuel et un booléen reaffichageNécessaire qui prend la valeur vraie toutes les secondes pour déclencher un réaffichage du chronomètre. Ces différents paramètres auront nécessité l'utilisation de la librairie standard <time.h>

5. Modélisation du plateau et gestion des déplacements

Afin de pouvoir gérer au mieux les déplacements et l'affichage, nous avons défini une structure « PlateauDeJeu » contenant les différents éléments du plateau. Ces éléments sont : L'échiquier, les défausses et les bordures du plateau.

L'échiquier est quant à lui modélisé par deux tableaux à deux dimensions (8x8) : un tableau de cases, et un tableau de pièces. Ces deux tableaux sont initialisés en début de partie et évoluent au fur et à mesure que les joueurs jouent. Par ailleurs ces deux tableaux interviennent lors du calcul des déplacements possibles pour une pièce.

En effet, lorsqu'un joueur clique sur une pièce, le programme calcule et affiche les déplacements possibles pour celle-ci. Pour ce faire, une matrice 8x8, représentant les cases de l'échiquier, est mise à jour à chaque appel de la fonction calculerDeplacementPossible. Si le déplacement est interdit sur une case, la case correspondante de la matrice prend la valeur 0, s'il est autorisé car la case est vide, elle prend la valeur 1, et si le déplacement est autorisé à condition de prendre une pièce, elle prend la valeur 2. Ainsi, en fonction des valeurs de cette matrice, on met en surbrillance certaines cases du tableau de cases de l'échiquier (vert si valeur 1, rouge si valeur 2, pas de surbrillance si 0).

Les déplacements sont calculés selon le type de la pièce sélectionnée. En effet, à chaque type de pièce correspondent des vecteurs de déplacement possibles, relatifs à la position de la pièce. Ces vecteurs sont stockées dans un tableau 2 dimensions de taille 2xNbVecteursPossibles. Ainsi pour le roi, on l'exemple suivant (8 déplacements possibles sur 2 dimensions → tableau 2x8) : {{1,1}, {-1,-1}, {0,1}…}, soit 8 déplacements possibles, relatifs à la position du roi, de valeur {x+1, y+1}, {x-1, y-1}, {x+0, y+1}…

Chaque déplacement est enregistré dans une liste chaînée, afin de pouvoir annuler 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 que le programme est fermé.

6. Gestion de l'affichage

Nous avons retenu l'idée de 4 grands menus : un menu entrainement avec son échiquier attribué, un menu 2 joueurs avec son propre échiquier aussi (→ deux parties simultanées possibles), un menu options permettant de régler les paramètres de jeu, et enfin un menu Règles permettant, au travers de plusieurs pages, d'avoir un bon aperçu des différentes règles du jeu d'échec. Un écran d'accueil permet de sélectionner le menu escompté. Voici l'aperçu de quelques menus.

Figure 2 : Ecran d'accueil

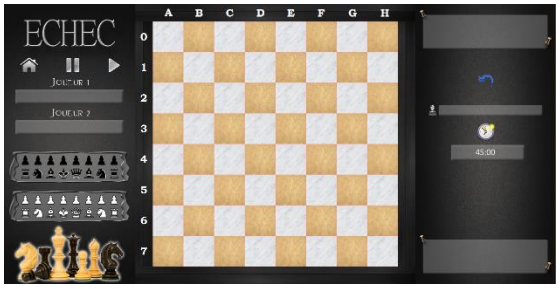


Figure 3 : Mode entrainement

Figure 4 : Mode 2 joueurs

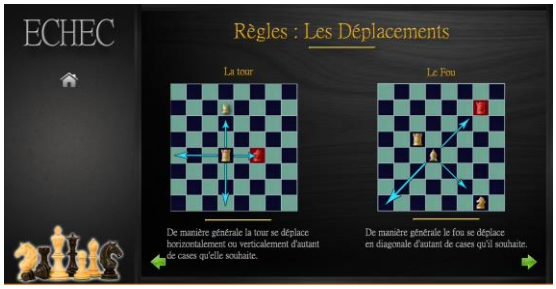


Figure 5 : Menu Règles

7. Gestion des erreurs

Dans l'optique de gérer, stocker et comprendre rapidement les erreurs renvoyées par notre code, nous avons décidé de créer un fichier log.txt. Trois fonctions permettent d'utiliser ce fichier de log : l'une permet de l'initialiser, l'autre de le modifier, la troisième de le fermer. Aussi, la fonction logPrint, permettant de le modifier, est la plus intéressante. Cette fonction prend deux paramètres :

- Le type de message, défini dans un enum : INFO, AVERTISSEMENT ou ERREUR qui nous donne une indication sur la nature du message et l'indique dans le fichier texte.
- Une chaîne de caractères contenant le message en question.

On appelle cette fonction à chaque évènement notable de notre programme, ou dans les parties susceptibles de générer des erreurs. Cela nous donne un fichier de cette forme :

```
INFO : Création de l'objet Vecteur Deplacement
INFO : Création de la fenêtre contenenant les règles du jeu
ERREUR : Impossible de charger l'image de la page de règle
INFO : Nom de l'image de page regle chargée ci dessous :
INFO : Regles/0.png
ERREUR : Impossible de charger l'image de la page de règle
```

Figure 6 Exemple de fichier log généré

Cette bonne gestion des erreurs nous aura souvent permis de débugger facilement notre programme, grâce à un simple coup d'œil dans le fichier log.txt.

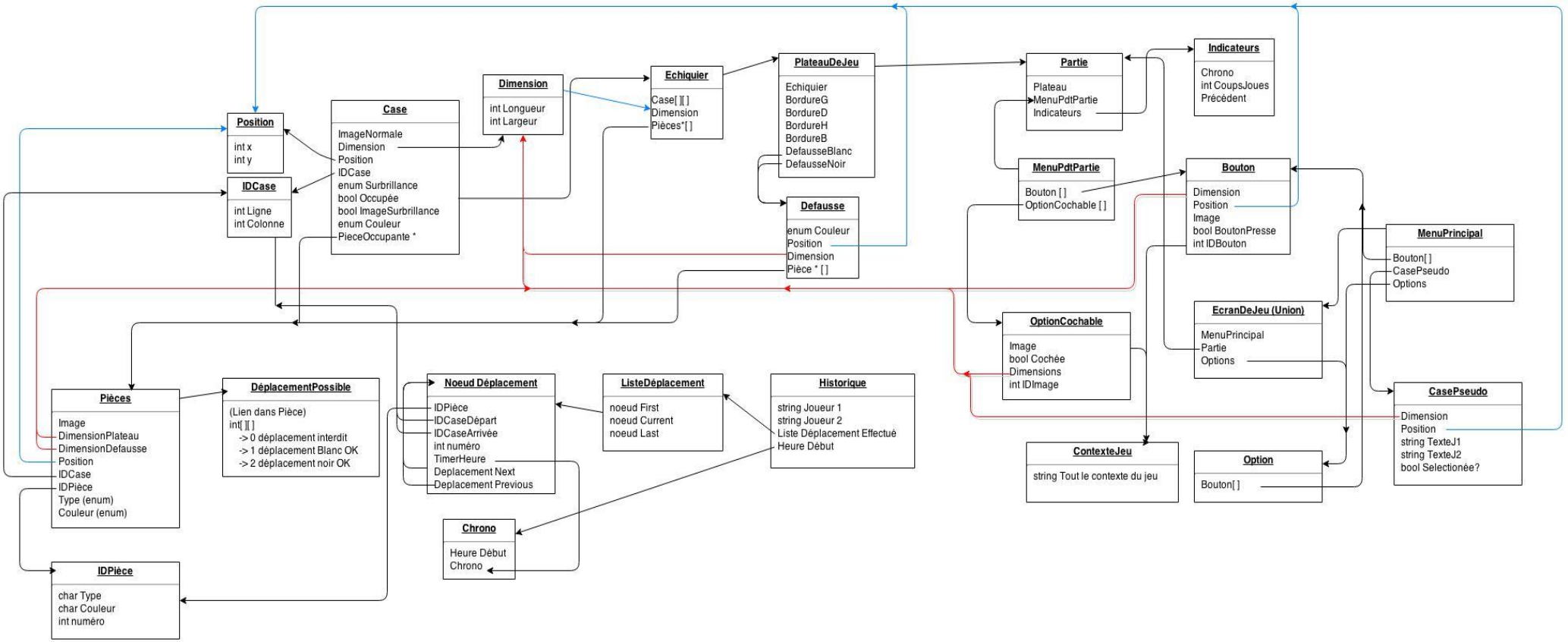
Conclusion :

En conclusion, il nous semble que le cahier des charges que l'on avait initialement émis aura été dans l'ensemble bien respecté. Nous avec cependant rapidement apporté quelques modifications à l'idée de projet initiale, comme l'abandon de la « pseudo-3D », car nous l'avons jugé trop coûteuse en terme de temps par rapport aux bénéfices apportés au jeu. L'idée de départ était en effet principalement d'obtenir un jeu intuitif, stable et surtout fonctionnel. D'autre part, comme nous pouvons le voir en comparant les deux diagrammes UML en annexe, les structures définies avant de commencer à coder auront été dans l'ensemble bien respectées, même si quelques structures supplémentaires auront été implémentées. Il reste cependant encore plusieurs pistes d'améliorations pour ce projet, comme le développement d'une intelligence artificielle permettant de jouer seul.

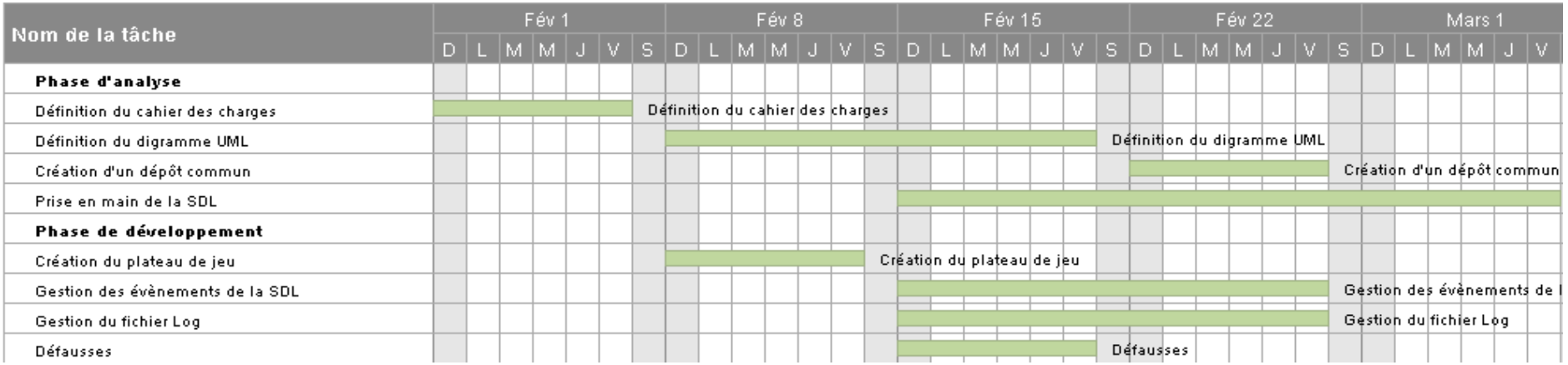
Sur le plan technique, ce projet nous a permis d'appréhender la librairie SDL avec la gestion des différents évènements, de l'affichage… Nous aussi découvert des logiciels de création de diagramme UML, et des logiciels de collaboration comme GitHub.

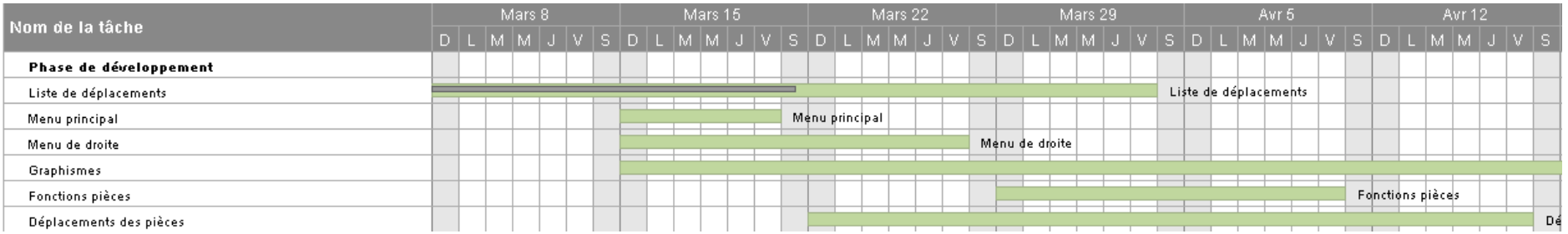
Enfin, sur le plan humain, la réalisation de ce jeu d'échec nous ara permis de mieux appréhender le travail en équipe, de définir grâce à des réunions quotidiennes un grand nombre de « deadlines », et d'établir un planning de réalisation des différentes taches du projet au travers d'un diagramme de Gantt. Ces différentes aptitudes nous semblent plus que primordiales pour envisager notre futur métier d'ingénieur.

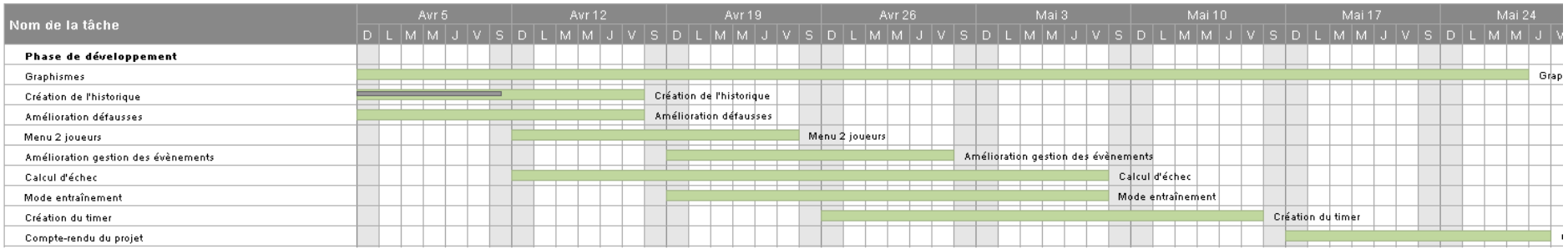
Annexe 1 : Diagramme UML simplifié initial



Annexe 2 : Avancée du projet







Annexe 3 : Diagramme UML simplifié final