Rapport de première soutenance

Projet libre
IA ECHEC

Etudiants

Guillaume ALVAREZ
Zakaria BEN ALLAL
Enzar SALEMI
Fabien TORRALBA

SOMMAIRE

INTRODUCTION

I. CAHIER DES CHARGES

II. SEGMENTATION DU PROJET

III. RECIT DE LA REALISATION DU PROJET

CONCLUSION

Lien Utiles

INTRODUCTION

L'histoire des machines joueuses d'échecs n'attend pas le développement de l'électronique et de l'informatique : la première fut l'automate turc inventée en 1769 par Johan Wolfgang von Kempelen1, qui joua notamment contre l'impératrice Catherine II et Napoléon Bonaparte. C'était en fait un homme de petite taille caché dans la machine.

Les premiers véritables programmes virent le jour dans les années 1950, avec le développement des ordinateurs. Ils passèrent en un demi-siècle de la connaissance des règles à un niveau au moins égal à celui des meilleurs joueurs humains.

L'un des pionniers fut Alan Turing qui, en mai 1952, écrit un programme de jeu d'échecs. Ne disposant pas d'un ordinateur assez puissant pour le faire tourner, il simule lui-même les calculs de la machine, mettant environ une demi-heure pour effectuer chaque coup. Une partie fut enregistrée, où le programme perdit contre Alick Glennie, un collègue de Turing.

En 1997, Deep Blue, un superordinateur spécialisé dans le jeu d'échecs par adjonction de circuits spécifiques, a battu le champion du monde Gary Kasparovdans un match à 6 parties. Deep Blue cherche 200 millions de positions par seconde, utilise une fonction d'évaluation très sophistiquée, et des méthodes non communiquées pour étendre quelques lignes de recherche jusqu'à 40 coups d'avance.
En 2017, Stockfish devient le meilleur logiciel d'Echecs. Son niveau est largement supérieur à celui des meilleurs joueurs humains, ce qui rend sa défaite impossible.

Comme vous l'aurez compris, le but de notre projet est de construire un programme d'échecs sur trois piliers que sont : l'interface graphique, l'algorithme de déplacement et une intelligence artificielle (IA).
Ce projet exécutable sous LINUX est effectué dans le cadre du projet libre de programmation, au second semestre de l'année pré-ingénierie.

Ce rapport se divisera ainsi en 4 chapitres relatant les attentes liées au projet, le travail réalisé, ainsi que l'avancement au regard du cahier des charges.

I. CAHIER DES CHARGES

1 / Cadre

Le projet est à réaliser en groupe de quatre personnes (et seulement quatre). Sa durée est d'environ cinq mois (de janvier à mai). Un cahier des charges présentera la nature du projet, les différentes parties qui le composent et les délais de réalisation (planning).

2 / Le Sujet

L'objectif de ce projet est de réaliser un jeu d'échecs doté d'une IA utilisant l'algorithme MinMax avec Elagage Alpha-Beta. Le jeu devra aussi être graphiquement représenté ainsi que l'affichage du score, du temps, etc...
Notre application sera un jeu d'Echec homme contre machine et le but est que l'IA soit la plus difficile à battre possible, pour cela nous ne disposons pas de plusieurs solutions. En effet, nous avons le choix entre faire un algorithme maison ou d'utiliser l'algorithme le plus connu et le plus utilisé MinMax crée en 1956 par John McCarthy. La limite de cet algorithme est qu'il cherche le meilleur coup dans le cas où l'adversaire jouerait le plus mal possible.

a. Description

Pour réaliser ce projet nous aborderons ce dernier suivant les étapes ci-dessous :

- Création du plateau de jeu
- Affichage du plateau
- Mécanismes et déplacements
- Contrôles pour le joueur humain
- Algorithme Min-Max
- Elagage Alpha-Bêta

b. Règles du jeu d'échecs

Au cours de sa longue histoire, les règles des échecs ont beaucoup évolué. Aujourd'hui, le jeu se pratique sur un plateau composé de 64 cases dont 32 de couleur blanche et 32 de couleur noire. La case inférieure droite, face aux joueurs, doit toujours être blanche. Notons également que la dame se trouve toujours sur la case de sa couleur.

Une partie d'échecs commence dans la position initiale. Les blancs jouent le premier coup puis les joueurs jouent à tour de rôle en déplaçant à chaque fois une de leurs pièces (deux dans le cas d'un roque). Chaque pièce se déplace de façon spécifique, il n'est pas possible de jouer sur une case occupée par une pièce de son propre camp. Lorsqu'une pièce adverse se trouve sur la case d'arrivée de la pièce jouée, elle est capturée et retirée de l'échiquier. Gagner du matériel (des pièces) est un moyen pour gagner la partie, mais ne suffit pas toujours pour y parvenir.
Il existe des règles spéciales liées au déplacement de certaines pièces :

- Le roque, qui permet le déplacement simultané du roi et de l'une des tours
- La prise en passant, qui permet une capture particulière des pions.
- La promotion des pions, qui permet de les transformer en une pièce maîtresse de son choix (sauf le roi) lorsqu'ils atteignent la dernière rangée de l'échiquier.

Lorsqu'un roi est menacé de capture, on dit qu'il est en échec. Si cette menace est imparable (on peut tenter de parer la menace en déplaçant le roi, en interposant une pièce ou en capturant la pièce attaquante) on dit qu'il y a échec et mat et la partie se termine sur la victoire du joueur qui mate. Il est interdit de mettre son propre roi en échec ou de le faire passer sur une ligne d'échec pendant le roque. Il est également interdit de roquer quand le roi est en échec sur sa case de départ. Si cela arrive (par inadvertance entre débutants) on doit reprendre le coup.
Si un camp ne peut plus jouer aucun coup légal (cela arrive par exemple avec un roi seul et l'ensemble de ses pions bloqués) et si son roi n'est pas en échec, on dit alors qu'il s'agit d'une position de pat. Quel que soit le matériel dont le camp adverse dispose, la partie est déclarée nulle, c'est-à-dire sans vainqueur.
Le but du jeu est donc d'infliger un échec et mat à son adversaire.

c. Mécanismes et directions

- Le Pion :

  Le pion est la pièce dont le mouvement est le plus compliqué à comprendre.
  - Il ne se déplace habituellement vers l'avant que d'une seule case et doit s'arrêter dès qu'il croise un obstacle. Il ne recule jamais.
  - Mais pour un pion n'ayant pas encore bougé, il est autorisé de le déplacer vers l'avant de deux cases ; cela n'est toutefois pas obligatoire. Un pion ayant déjà été déplacé n'est plus autorisé à se déplacer de deux cases à la fois.

  Le pion se différencie des autres pièces par le fait qu'il effectue la prise de manière distincte de son mouvement.
  - Le pion prend en diagonale (toujours vers l'avant) ; ce mouvement n'est autorisé que pour prendre une pièce adverse.
  - La prise n'est pas une obligation, rien ne l'empêche d'aller vers l'avant s'il n'y a pas d'obstacle.

L'objectif principal d'un pion est d'arriver à l'autre bout de l'échiquier. Tout pion réussissant cet exploit bénéficiera d'une promotion. C'est à dire que nous pourrons le remplacer par la pièce de notre choix parmi dame, tour, fou ou cavalier. Il est bien sûr interdit de se doter d'un second roi.

- Le Cavalier :

  - Le cavalier se déplace en formant un "L", c'est à dire deux cases verticalement et une case horizontalement (ou l'inverse).
  - Il a le droit de sauter par-dessus toutes les pièces, quelle que soit leur couleur, pour arriver sur une case vide.
  - Si la case d'arrivée est occupée par une pièce adverse, il prend sa place en éliminant cette pièce.
  - Si la case d'arrivée est occupée par une pièce de son camp, il ne peut en aucun cas s'y rendre.

- Le Fou :

  - De manière générale le fou se déplace en diagonale d'autant de cases qu'il souhaite s'il ne rencontre pas d'obstacle.
  - S'il rencontre un obstacle de couleur adverse, le fou n'a pas le droit de sauter par-dessus, cependant il est en droit de prendre la pièce en s'arrêtant sur la case où la pièce adverse se trouvait.
  - S'il rencontre un obstacle de même couleur que lui, il ne peut pas sauter par-dessus et est dans l'obligation de s'arrêter au plus loin dans la case adjacente.

- La Tour :

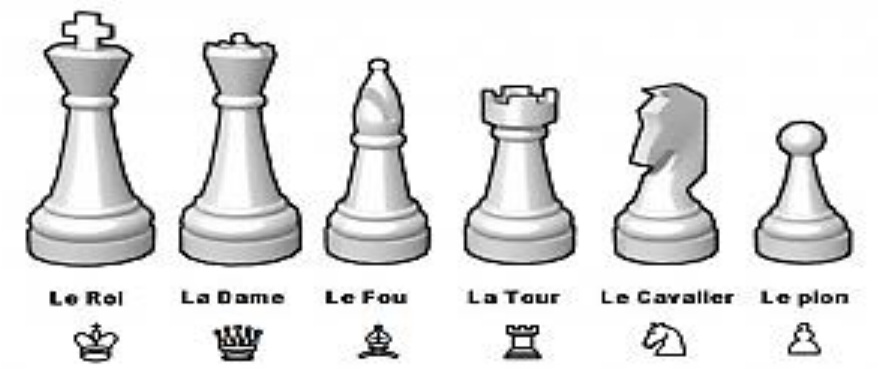
  - De manière générale la tour se déplace horizontalement ou verticalement d'autant de cases qu'elle souhaite si elle ne rencontre pas d'obstacle.
  - Si elle rencontre un obstacle de couleur adverse, la tour n'a pas le droit de sauter par-dessus, cependant elle est en droit de prendre la pièce en s'arrêtant sur la case où la pièce adverse se trouvait.
  - Si elle rencontre un obstacle de même couleur qu'elle, elle ne peut pas sauter par-dessus et est dans l'obligation de s'arrêter au plus loin dans la case adjacente.

➢ Le Roi :

- Le roi ne peut se déplacer que d'une seule case dans n'importe quel sens.
- Si une case est occupée par une pièce de son camp, le roi n'a pas le droit de s'y déplacer.
- Si une case est occupée par une pièce adverse, le roi peut prendre sa place en éliminant cette pièce.
- Le roi n'est autorisé à se déplacer sur une case que si celle-ci n'est pas attaquée par une pièce adverse. Des détails supplémentaires sont donnés dans la page but du jeu.

➢ La Dame :

- La dame peut se déplacer en diagonale, à l'horizontale ou à la verticale d'autant de cases qu'elle souhaite. Elle dispose en fait de la combinaison des pouvoirs du fou et de la tour.
- Si elle rencontre un obstacle de couleur adverse, la dame n'a pas le droit de sauter par-dessus, cependant elle est en droit de prendre la pièce en s'arrêtant sur la case où la pièce adverse se trouvait.
- Si elle rencontre un obstacle de même couleur qu'elle, elle ne peut pas sauter par-dessus et est dans l'obligation de s'arrêter au plus loin dans la case adjacente.



d. Min-Max et Elagage Alpha-Beta

- MinMax est un algorithme qui s'applique à la théorie des jeux, pour les jeux à deux joueurs à somme nulle (et à information complète), consistant à minimiser la perte maximum (c'est-à-dire dans le pire des cas). Il amène l'ordinateur à passer en revue toutes les possibilités pour un nombre limité de coups et à leur assigner une valeur qui prend en compte les bénéfices pour le joueur et pour son adversaire. Le meilleur choix est alors celui qui minimise les pertes du joueur tout en supposant que l'adversaire cherche au contraire à les maximiser (le jeu est à somme nulle).
  Il existe différents algorithmes basés sur MinMax permettant d'optimiser la recherche du meilleur coup en limitant le nombre de nœuds visités dans l'arbre de jeu, le plus connu est l'élagage alpha-bêta que nous utiliserons pour ce projet. En pratique, l'arbre est

souvent trop vaste pour pouvoir être intégralement exploré (comme pour le jeu d'échecs ou de go). Seule une fraction de l'arbre est alors explorée.

- L'**élagage alpha-bêta** (abrégé élagage αβ) est une technique en intelligence artificielle permettant de réduire le nombre de nœuds évalués par l'algorithme MinMax. Il est utilisé dans des programmes informatiques qui jouent à des jeux à 2 joueurs, comme pour notre jeu d'Echec.
  Notre algorithme effectue une exploration complète de l'arbre de recherche jusqu'à un niveau donné. L'élagage alpha-beta permet d'optimiser grandement l'algorithme minimax sans en modifier le résultat. Pour cela, il ne réalise qu'une exploration partielle de l'arbre. Lors de l'exploration, il n'est pas nécessaire d'examiner les sous-arbres qui conduisent à des configurations dont la valeur ne contribuera pas au calcul du gain à la racine de l'arbre. Dit autrement, l'élagage αβ n'évalue pas des nœuds dont on est sûr que leur qualité sera inférieure à un nœud déjà évalué.

3 / Les contraintes du projet

Pour que le développement de ce projet se passe le mieux possible, nous allons respecter les règles de Codage suivantes :

a. Pour le code :

- Notre code devra obligatoirement être correctement indenté.
- Tous les noms d'identifiants (fonctions, variables, constantes, macros, etc.) devront être en anglais.
- Nous découperons le code en plusieurs unités et chaque fichier .c devra être accompagné de son en-tête (fichier .h).
- Le code ne devra pas dépasser 80 caractères par ligne.
- Il ne doit y avoir d'espaces inutiles.
- Le code devra être compilé avec au moins les options suivantes :
  -Wall -Wextra -std=c99.
- Nous pourrons utiliser (au choix) les compilateurs gcc ou clang.
- Le code ne devra émettre aucun warning à la compilation.

b. Pour le projet :

Nous devrons utiliser le gestionnaire de versions git et être capable de montrer en soutenance l'évolution de notre projet et les différents apports de chacun. Pour la construction de notre projet, nous devrons fournir un Makefile permettant de compiler les différents fichiers et de produire les différents binaires. Notre Makefile devra également permettre de nettoyer le répertoire des produits de compilation (.o , exécutables, . . .).
Le Makefile devra, autant que possible, utiliser les règles de production implicites, il devra également ne compiler que les fichiers nécessaires en respectant les dépendances.
Le projet devra contenir au minimum un fichier README expliquant succinctement comment compiler et lancer le projet.

Figure 1 : Aperçu du Git repository du projet

II. SEGMENTATION DU PROJET

A. Représentation de l'échiquier et des pions

Afin de créer une interface graphique en langage C, nous avons opté pour l'utilisation de la SDL dans sa version 2.0.
La SDL, Simple Direct-Media Layer, est une bibliothèque destinée à permettre l'accès au matériel graphique pour faire, par exemple, des jeux en plein écran (ou en fenêtre), et ça de manière portable.
La SDL est simple d'emploi, mais particulièrement bas niveau, ce qui rend la prise en main assez difficile.
La SDL est disponible en licence libre LGPL. Cela veut dire que nous pouvons donc librement utiliser cette bibliothèque dans notre projet. La seule obligation est que la bibliothèque SDL doit rester en liaison dynamique.
L'implémentation de l'interface graphique se fera en 2 étapes principales représentés par 2 fonctions.

- Une première fonction nommée « init_screen » permettant d'afficher la fenêtre de notre jeu, avec les dimensions et les couleurs choisies

```
void init_screen()
{
        if(SDL_Init(SDL_INIT_VIDEO) == -1)
                errx(1,"Could not initialize SDL: %s.\n", SDL_GetError());
        /* Création de la fenêtre */
        pWindow = SDL_CreateWindow(
                "Chessmaster",                    // window title
                SDL_WINDOWPOS_CENTERED,           // initial x position
                SDL_WINDOWPOS_CENTERED,           // initial y position
                1920,                              // width, in pixels
                1080,                              // height, in pixels
                0
                );
```

Figure 2: exemple d'implémentation de la fonction init_screen

- Une deuxième fonction « DrawChessBoard » prenant comme paramètre notre table d'échecs précédemment affichée (map). Cette fonction permet de parcourir la map et d'afficher les éléments correspondants à chaque position. Ainsi Chaque case parmi les 64 affichées (8x8) recevra un code couleur (blanc ou noir), en plus d'un pion correspondant à l'état initial d'une table de jeu d'échecs.

```
void DrawChessBoard(t_map **map)
{
        size_t posx = 560;
        size_t posy = 140;
        for (size_t i = 0; i < 8; i++)
        {
                for (size_t j = 0; j < 8; j++)
                {
                        SDL_Rect rect = {posx, posy, 280, 120};
                        if (map[i][j].color == BLACK)
                                SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255);
                        else
                                SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255);
```

Figure 3 : exemple d'implémentation de la fonction DrawChessBoard

B. Implémentation des algorithmes de déplacements

Suivant les règles de déplacement citées dans le cahier des charges, un programme doit être implémenté pour traduire ses règles.
Parallèlement à l'implémentation de l'interface graphique, une partie du groupe va se pencher sur la programmation de la gestion des déplacement des pions sur la surface de la map. La fonction programmée a pour but de gérer le parcours de la map par chaque pièce, du début de la partie, jusqu'à la fin.
Cette fonction sera bien évidemment fusionnée avec l'implémentation de l'interface graphique afin que chaque déplacement soit représenté en temps réel au cours du déroulement d'une partie d'échecs.

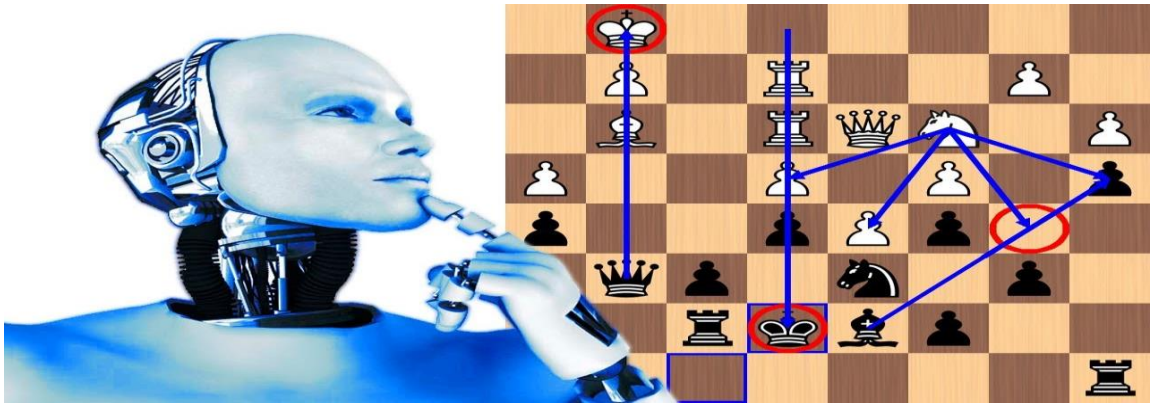
```
#include <stdlib.h>
#include <err.h>
#include <stdio.h>
#include <SDL2/SDL.h>
#include "chessmaster.h"
#include "rook.h"
#include "display.h"

void put_chessman(t_map **map)
{
        for (size_t i = 0; i < 8; i++)
        {
                for (size_t j = 0; j < 8; j++)
                {
                        if (i % 2 == 0)
                                map[i][j].color = (j % 2 == 0) ? BLACK : WHITE;
                        else
                                map[i][j].color = (j % 2 == 0) ? WHITE : BLACK;
                        map[i][j].target = NONE;
                        map[i][j].is_empty = true;
                        if (i == 0 || i == 7 || i == 1 || i == 6)
                        {
                                map[i][j].chessman = malloc(sizeof(t_chessman));
                                if (map[i][j].chessman == NULL)
                                        errx(84, "chessmaster: allocation error\n");
                                map[i][j].chessman->color = (i == 0 || i == 1) ? BLACK : WHITE;
                                if (j == 0 || j == 7) {
                                        map[i][j].chessman->type = ROOK;
                                        map[i][j].chessman->move = &move_rook;
                                }
                                if (j == 1 || j == 6) {
                                        map[i][j].chessman->type = KNIGHT;
                                        map[i][j].chessman->move = &move_knight;
```

Figure 4: exemple d'implémentation de la fonction de gestion de déplacement

C. Programmation de l'intelligence artificielle

Un échiquier dans la réalité est un objet palpable et concret. Il faut donc trouver une représentation pour que la machine puisse suivre le déroulement de la partie au cours du temps. Il faut pour cela que la machine à un instant donné du jeu ait en sa possession les mêmes informations que le joueur. C'est à dire l'état de toutes les cases de l'échiquier.

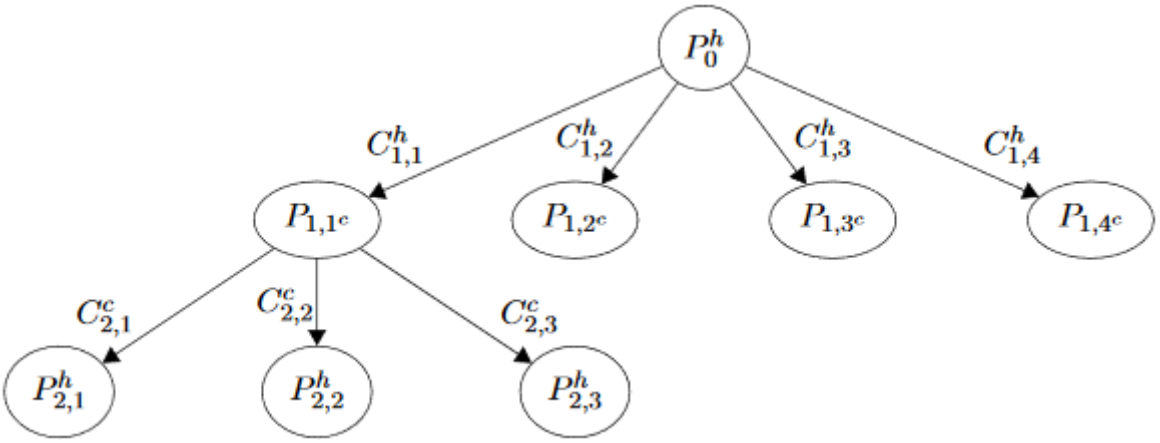


Le jeu d'échecs est un jeu où les deux joueurs jouent séquentiellement. Le but des deux joueurs est opposé et incompatible. Le joueur blanc veut mettre le roi noir mat tandis que le joueur noir veut mettre le roi blanc mat. Tour à tour, un joueur puis l'autre choisit parmi tous les coups possibles celui qui lui paraît le meilleur pour parvenir à son objectif. Il est donc possible de programmer une Intelligence Artificielle du jeu d'échecs qui va réfléchir à la conséquence de ses actes en regardant les mouvements possibles d'un joueur pour pouvoir jouer à sa place.

Pour notre projet on va donc suivre le principe de l'algorithme MinMax et de son amélioration principale, l'élagage AlphaBeta.

➢ Algorithme MinMax :

Untel jeu peut être vu comme une succession de positions, résultats des coups alternés des joueurs. Nous noterons P^h_i une position que doit jouer le joueur humain et P^c_i une position que doit jouer l'ordinateur. A un instant donné de la partie, supposons que le joueur humain doit jouer à partir de la positionPh0. Il a en général un nombre fini de coups possibles $C^h_{0,k}$ qui vont produire autant de positionsP^c_1,kque l'ordinateur va devoir jouer et ainsi de suite. Ce qui nous donne un arbre de toutes les possibilités d'évolution du jeu.



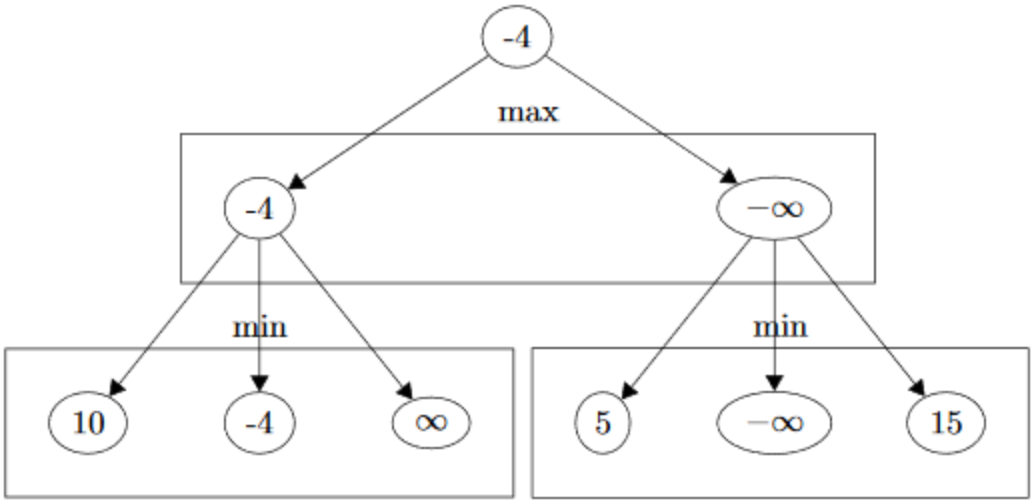
On définit maintenant une fonction valeur d'une position V(P).
La valeur d'une position gagnante pour le joueur humain vaut $+\infty$, d'une position perdante $-\infty$ et d'une position de match nul 0. La valeur d'une position autre sera calculée en fonction d'heuristiques liées au jeu. Par exemple aux échecs, on pourra prendre la différence des valeurs des pièces encore présentes sur l'échiquier entre les 2 joueurs à laquelle on ajoute la différence du nombre de cases contrôlées par les 2 joueurs. On définit alors la fonction MinMax d'une position quelconque P de la façon suivante :

$$\text{MinMax}(P) = \begin{cases} V(P) & \text{si } P \text{ est une position terminale} \\ \max\left(\text{MinMax}(P_1), \ldots, \text{MinMax}(P_k)\right) & \text{si } P \text{ est une position du joueur} \\ \min\left(\text{MinMax}(P_1), \ldots, \text{MinMax}(P_k)\right) & \text{si } P \text{ est une position de l'opposant} \end{cases}$$

P_1, ..., P_k désignant toutes les positions obtenues après 1 coup. L'algorithme du min-max consiste à calculer la fonction MinMax de toutes les positions issues du premier coup du joueur (humain dans l'exemple) et à jouer le coup qui donne une position maximale

➢ Elagage AlphaBeta

La variante$\alpha-\beta$consiste à stopper l'exploration d'une branche dès lors qu'à un niveau correspondant à une phase de minimisation, on trouve une valeur inférieure à une valeur min-max du niveau précédent. En effet, on peut abandonner l'exploration car on est sûr que le joueur cherchant à maximiser sa position ne jouera pas un coup produisant une valeur inférieure à celle qu'il a déjà trouvée. De même, on stoppera l'exploration d'une branche si en phase de maximisation, on trouve une valeur supérieure à une valeur min-max du niveau précédent. Dans l'exemple suivant :

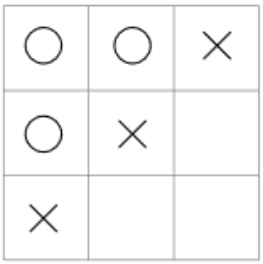


➢ Implémentation algorithme MinMax

Pour écrire d'un algorithme du min-max, il suffit de disposer d'une classe « Position » qui doit au moins fournir :

- ✓ Une fonction qui renvoie la valeur d'une position
- ✓ Un pointeur sur la position sœur
- ✓ Un pointeur sur la première position fille
- ✓ Un indicateur sur le joueur qui doit jouer cette position
- ✓ Un générateur de toutes les positions suivantes à une profondeur donnée
- ✓ Un destructeur de toutes les positions filles d'une position (par récursivité, toutes les positions seront détruites)

On pourra valider l'algorithme sur un jeu très simple, par exemple le tic-tac-toe (premier `a aligner3 pions dans un carré 3x3) :



D. Agencement du projet

Par souci d'efficacité et de clarté, chaque partie du projet a été référencée séparément par le biais de différents dossier :

- Un dossier AI qui contiendra toute l'implémentation de la partie intelligence artificielle.
- Un dossier Game contenant toute la gestion de déplacements des pièces et déroulement de la partie.
- Un dossier Display dans lequel figurent tous les fichiers permettant l'affichage de l'interface graphique

Conformément au cahier des charges, notre répertoire de projet comporte également un Makefile ainsi qu'un fichier README.txt.

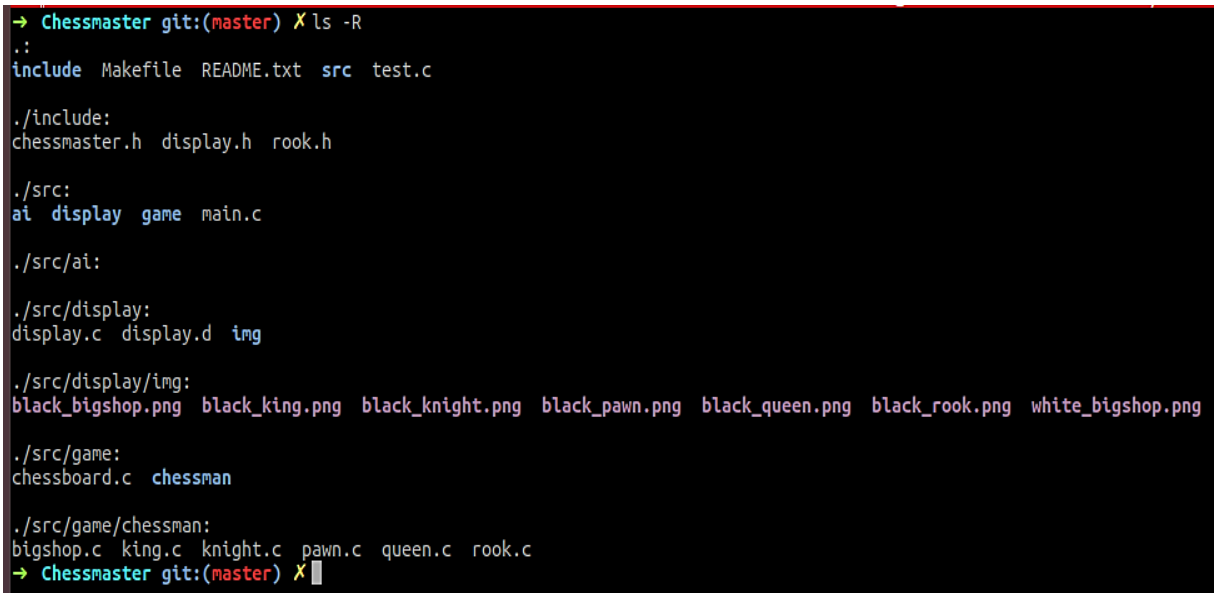


Figure 5 : Répertoire de notre projet

III. Récit de la réalisation du projet

A. Programmation

1. Interface graphique (gérée par Zakaria Ben Allal et Guillaume Alvarez)

Afin d'implémenter l'interface graphique, la première étape à été d'installer la bibliothèque SDL 2.0. Cette étape n'a pas été des plus simples. En effet, après avoir suivi plusieurs tutoriels d'installation sur internet, plusieurs erreurs de compilation subsistaient. Il a fallu télécharger des paquets spécifiques en plus et rajouter des flags propres à la bibliothèque SDL 2.0, afin de corriger les erreurs de compilation. N'ayant pas une grande expérience dans l'installation de bibliothèques, le processus fut assez frustrant et couteux en termes de temps.
Une fois la bibliothèque installée, il fallait se documenter sur les différentes fonctions de la bibliothèque permettant d'afficher une fenêtre de la manière la plus adéquate pour notre projet.
Après s'être accommodé aux méthodes d'implémentation de la SDL, nous avons réussi à afficher la fenêtre du jeu d'échecs ainsi que sa table.

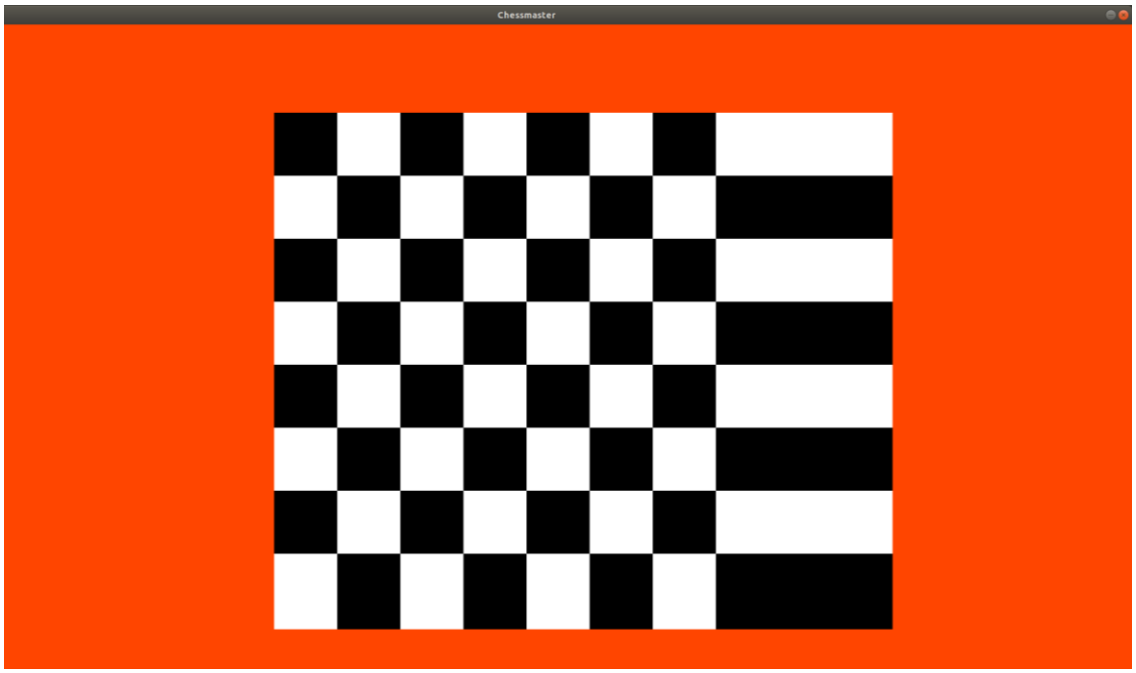


Figure 6:exemple d'interface graphique obtenue

Comme vous l'aurez remarqué, notre table souffrait d'une sorte de mal formation de naissance.

En effet, il a fallu travailler à corriger la boucle d'allocation de cases afin d'avoir une map constituée de 64 carrés de même taille. Grace à l'aide d'Enzar Salemi, ce problème a pu être réglé.

Il fallait après cela rajouter les pions sur la map. La SDL étant une bibliothèque très simple, elle ne propose à la base que le chargement d'images de type « bitmap » (extension.bmp). Or, nous sommes habitués aux formats d'images « compressés » comme le PNG, le GIF et le JPEG. Il est en fait possible de rajouter des extensions à la SDL. Ce sont des bibliothèques qui ont besoin de la SDL pour fonctionner. On peut voir ça comme des add-ons et SDL_Image est l'une d'entre elles.
Nous avons pu implémenter l'affichage de chaque pièce dans une fonction load_images à partir d'un dossier image composé d'images au format .png de chaque pièce.

```
5   #include <SDL2/SDL_image.h>
6   #include "chessmaster.h"
7   #include "display.h"
8
9   SDL_Window *pWindow;
10  SDL_Renderer *renderer;
11  SDL_Surface *surface;
12
13  void load_images(t_map **map)
14  {
15          SDL_Surface* black_king = IMG_Load("src/display/img/black_king.png");
16          SDL_Surface* white_king = IMG_Load("src/display/img/white_king.png");
17          SDL_Surface* black_pawn = IMG_Load("src/display/img/black_pawn.png");
18          SDL_Surface* white_pawn = IMG_Load("src/display/img/white_pawn.png");
19          SDL_Surface* black_queen = IMG_Load("src/display/img/black_queen.png");
20          SDL_Surface* white_queen = IMG_Load("src/display/img/white_queen.png");
21          SDL_Surface* black_rook = IMG_Load("src/display/img/black_rook.png");
22          SDL_Surface* white_rook = IMG_Load("src/display/img/white_rook.png");
23          SDL_Surface* black_bigshop = IMG_Load("src/display/img/black_bigshop.png");
24          SDL_Surface* white_bigshop = IMG_Load("src/display/img/white_bigshop.png");
25          SDL_Surface* black_knight = IMG_Load("src/display/img/black_knight.png");
26          SDL_Surface* white_knight = IMG_Load("src/display/img/white_knight.png");
```

Figure 7: implémentation d'affichage des pièces du jeux

Ainsi, notre fichier display.c dans lequel est implémentée l'interface graphique, est constitué de :

- Une fonction de création de la fenêtre du jeu : init_screen ()
- Une fonction de création de la table de jeu d'échecs : DrawChessBoard ()
- Une fonction d'affichage des pièces sur la map : load_images ()

2. Implémentation des déplacements (gérée par Enzar Salemi et Fabien Torralba)

La gestion des déplacements a été la partie la plus difficile à implémenter. Elle requiert une prise en compte minutieuse des déplacements de chaque pièce conformément aux règles du jeu développées exhaustivement dans le cahier des charges.
Tout d'abord il a fallu programmer l'ensemble des coups possibles spécifiques à chaque type de pièce, et chaque déplacement sur la map. Chaque pièce possède donc son propre fichier comportant les possibilités de déplacement. Ces fichiers figure dans le dossier « chessman » du répertoire « game ». Ceci afin de pouvoir afficher au joueur les éventuelles possibilités de frappes. Ces dernières seront représentées par une coloration verdâtre des cases concernées. Puis il a fallu intégrer ces déplacements dans l'affichage SDL afin d'avoir un rendu en temps réel du déroulement du jeu. Cette implémentation s'est faite dans le fichier chessboard.c du répertoire « game ».

Figure 8: Exemple d'une phase de jeu avec possibilité de frappe du cavalier

B. Difficultés

Tout au long de l'accomplissement de ce projet nous avons rencontré plusieurs obstacles :

- Installation de la SDL :

A cause de plusieurs bugs machine et du manque d'expérience du binôme s'occupant de la partie interface graphique, cette dernière a connu un démarrage assez long.

- Hétérogénéité du groupe :

Compte tenu des différences de niveau en programmation entre les membres du groupe, il a fallu moduler le temps imparti pour l'implémentation des différentes parties du projet en fonction de celui ou ceux qui s'en chargeaient. Un important travail de communication a dû être fait afin d'assurer au mieux le respect des délais. De plus, à maintes reprises, Enzar Salemi a dû mettre l'implémentation de sa partie en suspens afin de promulguer son aide sur la partie SDL.

- Sous-effectif :

Compte tenu de l'indisponibilité prolongée de Fabien Torralba, nous avons dû conduire le projet en sous-effectif pendant plusieurs semaines. Ainsi, Enzar s'est retrouvé seul sur sa partie.

Pour toutes ces raisons, nous n'avons pas pu aborder les prémices de l'implémentation de l'IA.
Ce retard n'est pas irrattrapable et sera probablement comblé d'ici la prochaine soutenance.

CONCLUSION

Ce projet s'avère être formateur sur plusieurs plans. Il nous permet de travailler sur nos faiblesses ou nos lacunes en programmation et en algorithmie. Un tel projet nous pousse aussi à faire preuve d'ingéniosité et d'adaptation dans le cadre d'utilisation de nouvelles notions.

Le travail fourni à ce jour nous a permis de construire les piliers sur lesquels se basera notre rendu final. D'ici la prochaine soutenance, notre quête sera d'entamer l'implémentation de l'IA, et d'améliorer l'interface graphique construite jusque-là.

Dans les jours à venir, le groupe devra être plus soudé et plus efficace.
Notre but est de réaliser un jeu fonctionnel qui pourrait être utilisé régulièrement par n'importe quel joueur d'échecs, du débutant grâce à l'aide au déplacement, au joueur avancé grâce à la gestion de timers de jeu ou de règles complexes comme la prise en passant.
Pour conclure, bien que le nombre d'heures passées sur ce projet soit important, il reste une quantité importante de travail à mener. Le rendez vous est donc pris pour la prochaine soutenance.

Liens utiles

- Gestion des events : https://alexandre-laurent.developpez.com/tutoriels/sdl-2/les-evenements
- Premiers pas avec SDL : https://openclassrooms.com/fr/courses/19980-apprenez-a-programmer-en-c/17117-installation-de-la-sdl
- Guide de migration SDL 1.2 vers 2.0 : https://jeux.developpez.com/tutoriels/sdl-2/guide-migration/
- Guide programmation de jeu : http://lazyfoo.net/tutorials/SDL/
- Librairies SDL 2.0 : https://wiki.libsdl.org/SDL_CreateSoftwareRenderer