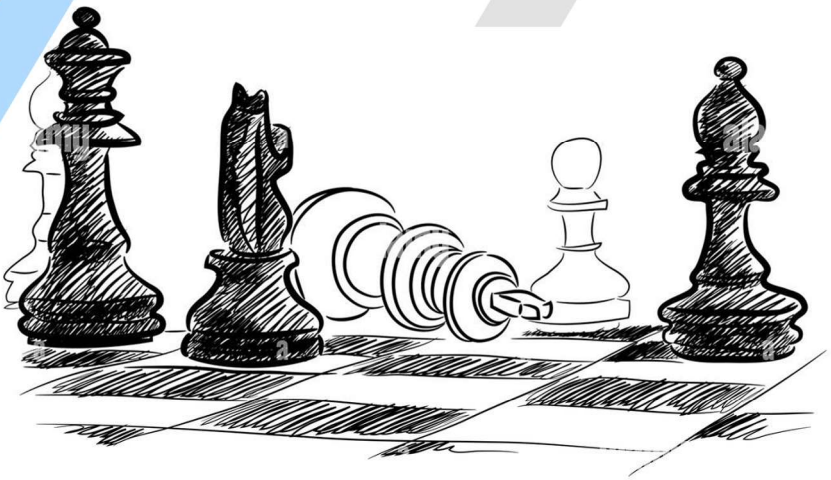
Projet StockMeat

Site web: https://yanisbayle5.wixsite.com/stockmeat
09/03/2023

RAPPORT DE SOUTENANCE



Antoine JOUY – Maxence Gatard
Evan Reynaud – Yanis Bayle

TABLE DES MATIERES

INTRODUCTION

Rappel des tâches

Nous avions pour but d'avoir commencé la structure de base du jeu d'échec, l'interface ainsi que le site web. En parallèle, des recherches complémentaires sur l'algorithme Min-Max devaient être effectuées.
Nous avons un peu de retard sur le développement de l'interface dû à la complexité de la librairie utilisée, à savoir GTK.
Pour le moment, seule la structure de base a été codée, cependant nous allons détailler nos recherches et nos réflexions.

Tâches	Semaine du 20/02 1ère soutenance	Semaine du 11/04 2ème soutenance	Semaine du 29/05 3ème soutenance
Composantes de l'algorithme Min-Max	Pas commencé	En cours	Terminé
Interface graphique	En cours	Terminé	Terminé
Fonctionnement d'un jeu d'échec	En cours	Terminé	Terminé
Site Web	En cours	Terminé	Terminé
Autres	Pas commencé	En cours	Terminé

Voici les deadlines que nous nous étions fixées dans le cahier des charges (d'où le décalage des dates des soutenances).

STRUCTURE DU JEU

(Antoine JOUY)

Les pièces

Les pièces sont des structures avec différents éléments permettant de savoir le rôle, la couleur, les coups possibles sur le tour, la valeur de la pièce, les coordonnées sur le tableau ainsi que l'index dans la liste des pièces (listes détaillées plus tard).
Les valeurs sont les suivantes :

Les rôles et les couleurs sont définis par deux énumérations avec des valeurs attribuées pour chaque rôle/couleur et les pièces sont définies par une structure "pièce" :

```
enum Role {
    EMPTY = 0,
    PAWN = 1,
    BISHOP = 2,
    KNIGHT = 3,
    ROOK = 4,
    QUEEN = 5,
    KING = 6
};
enum Color{
    BLACK = 1,
    WHITE = -1,
    NONE = 0

};
```

```
struct piece {
    int x;
    int y;
    enum Role role;
    enum Color color;
    int value;
    struct list *possibleMoves;
    int index;
};
```

Les coups possibles sont stockés dans une structure classique de liste avec une sentinelle. Ces coups sont définis par un algorithme que nous détaillerons plus loin.

La variable index permet d'identifier la pièce dans un tableau regroupant toutes les pièces encore en jeu. Ce tableau nous servira pour accéder plus simplement aux pièces et appliquer l'algorithme Min-Max.

STRUCTURE DU JEU

(Antoine JOUY)

L'échiquier

L'échiquier est un tableau de pièce, la différence avec le premier tableau mentionné plus tôt est que l'échiquier comporte des cases vides (des pièces avec la couleur "NONE", le rôle "EMPTY", une liste vide pour les coups possibles, un index et une valeur négatifs).
Pour pouvoir initialiser un tableau avec ses pièces, nous avons codé la fonction initBoard() :

```
void initBoard(struct piece **board, enum Color colorPlayer, struct piece** listOfPieces);
```

La variable "board" correspond à l'échiquier et sa mémoire est dynamiquement allouée au préalable.
La variable "colorPlayer" indique la couleur choisie par le joueur afin d'orienter l'échiquier correctement.
La variable "listOfPieces" correspond à la liste des pièces en jeu et sa mémoire est dynamiquement allouée au préalable.

Cette fonction va permettre d'avoir un échiquier initialisé avec toutes les pièces. Pour ce faire, on appelle la fonction newpiece() qui avec une pièce crée au préalable, et selon les coordonnées renseignées va lui assigner le bon rôle, la bonne couleur etc. Chaque pièce est un pointeur dynamiquement alloué.

Voici un exemple de l'échiquier après initialisation, si le joueur choisit la couleur blanche. :

```
4  3  2  6  5  2  3  4
1  1  1  1  1  1  1  1
0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
1  1  1  1  1  1  1  1
4  3  2  5  6  2  3  4
        --------
-1  -1  -1  -1  -1  -1  -1  -1
-1  -1  -1  -1  -1  -1  -1  -1
0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
1  1  1  1  1  1  1  1
1  1  1  1  1  1  1  1
```

Affichage des rôles

Affichages des couleurs

STRUCTURE DU JEU

(Antoine JOUY)

Les coups

La fonction principale, getMoves() permet d'obtenir les coups possibles d'une pièce.

```
void getMoves(struct piece** board, struct piece* piece, int playercolor);
```

Cette fonction effectue un switch() et selon le rôle de la pièce, la fonction correspondante est appelée.
Il a été long d'effectuer chaque fonction, car il faut respecter les déplacements propres aux pièces : le fou ne suit que les diagonales, la tour ne se déplace que verticalement ou horizontalement etc.

L'idée générale est donc de regarder les cases atteignables par une pièce selon ses règles de déplacement. Si la case est vide, on ajoute sa position à la liste et on continue, s'il y a une pièce adverse, on ajoute la position de la case à la liste mais on arrête de regarder cette direction et s'il y a une pièce de la même couleur, on arrête de regarder dans cette direction.

Ces fonctions ne sont pas encore parfaites, les mouvements sont bien calculés mais nous ne prenons pas encore en compte si la pièce défend le roi d'un échec (ce qui est très important et est en cours d'implémentation).

Détection d'un échec

Pour pouvoir savoir s'il y a un échec, il faut regarder pour chaque pièce si dans les coups possibles, il y a la position du roi en question. Si oui, alors il y a échec et il faut que le joueur concerné protège son roi.

Déplacement

Nous avons commencé la fonction move() permettant de déplacer une pièce. Nous n'avons qu'un prototype car il nous faut l'information de l'échec éventuel que nous n'avons pas encore implémenté.

```
int move(struct piece** board, struct piece* piece, int x , int y);
```

Cependant, l'idée générale de fonctionnement serait de vérifier si la pièce a dans ses coups possibles, les coordonnées données en entrée et s'il n'y a pas une mise en échec de son propre roi si elle se déplace.

Algorithme Min-Max

Pour cela nous allons construire une fonction d'évaluation qui prend comme donnée une position et renvoie une évaluation chiffrée ainsi qu'une autre fonction qui va chercher le meilleur coup à jouer dans la position. Cette seconde fonction va évaluer les différentes positions potentielles après les différents coups possibles pour l'adversaire selon l'algorithme mix-max. Une analyse avec une profondeur plus élevée donnera donc des résultats plus performants.

Afin de pouvoir atteindre une profondeur plus élevée, il serait donc utile d'optimiser la rapidité de ce procédé en ne perdant pas de temps à évaluer des positions qui ne rapporteraient aucun résultat.

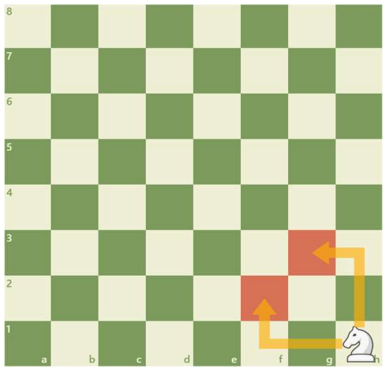
Alpha-Beta Pruning

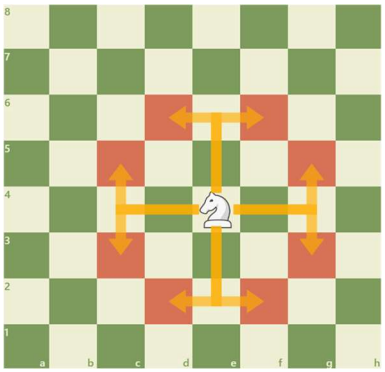
Ainsi, si l'on a déjà trouvé une réponse adverse à un possible coup qui lui permet d'obtenir une meilleure position que sa meilleure option par suite d'un autre coup précédemment évalué, il n'est pas la peine d'évaluer les autres réponses adverses pour savoir que ce n'est pas le bon coup à jouer.
Le procédé sera également d'autant plus efficace que l'on évaluera les bons coups en premier. On va donc pouvoir pousser l'algorithme dans la bonne direction en lui suggérant que certains types de coups ont généralement beaucoup plus de chances d'être des bons coups et seront donc dans les premières idées à examiner. Par exemple, capturer une pièce adverse puissante avec une pièce alliée plus faible est souvent une bonne idée. De plus, une bonne façon d'attaquer est de jouer des coups dit « forcés » tels que de mettre l'adversaire en échec ou de capturer une pièce (pour lancer une chaîne de capture ou pour causer une faiblesse). Il est donc également intéressant de suggérer à notre programme d'examiner en premier ces coups à potentiel élevé.

Fonction d'évaluation

La fonction d'évaluation doit prendre en compte des informations de base, la valeur des pièces et les position des pièces etc.
En effet, certaines pièces sont plus puissantes quand elles sont placées à certains endroits de l'échiquier.

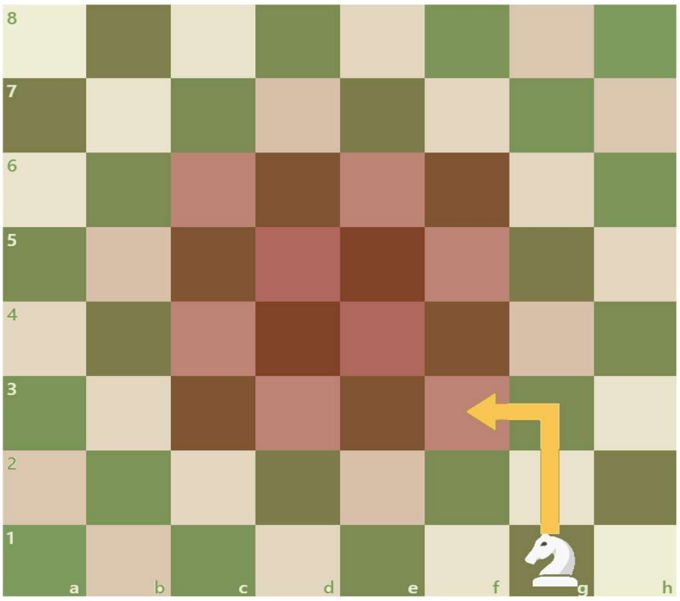
Par exemple, un cavalier bloqué dans le coin ne peut voir que deux cases près du bord et ne servira donc pas à grand-chose.
Au contraire, un cavalier posé plus près du centre va pouvoir voir huit cases réparties dans les deux camps et avoir un plus grand impact sur la position.





Afin que notre algorithme puisse comprendre cela, nous allons lui donner plusieurs grilles pour chacune des pièces. Celles-ci seront prises en compte pour la fonction d'évaluation et elles pourront lui servir de base afin de savoir comment améliorer la position de ses pièces s'il ne voit pas d'autres opportunités plus intéressantes.

Voici la grille de notre cavalier, une teinte rouge foncé plus prononcée indique une meilleure case pour mettre un cavalier.

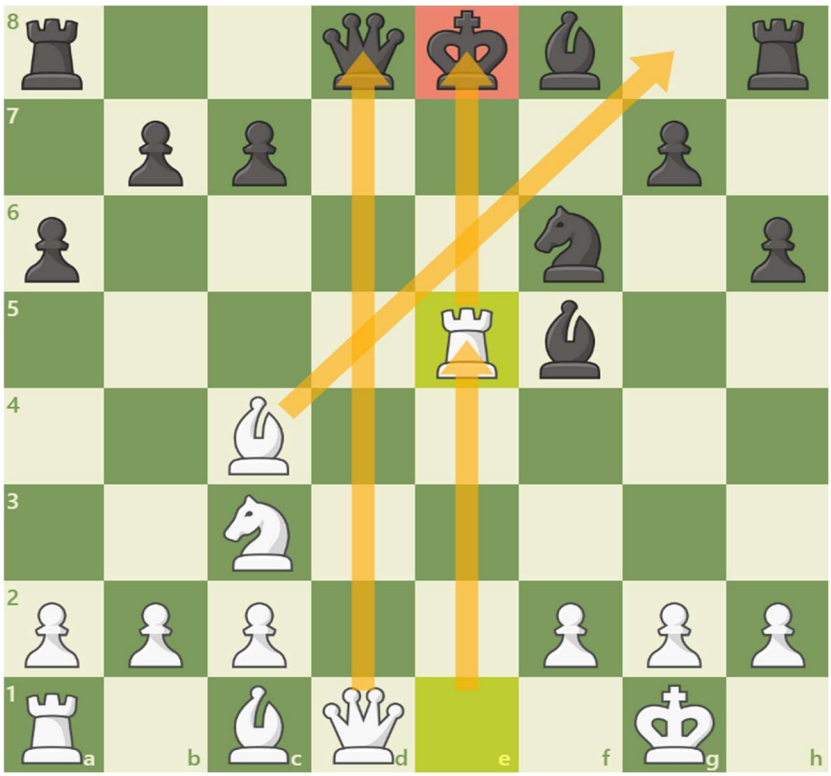


EVALUATION ET MIN-MAX

(Evan Reynaud)

Un autre critère à prendre en compte pour l'évaluation est la sécurité du roi. Il est généralement conseillé d'effectuer le roque durant la phase d'ouverture de la partie afin de ne pas laisser celui-ci vulnérable.

Par exemple dans la position ci-dessous, le roi blanc est bien protégé grâce au roque, mais le roi noir, qui lui n'a pas roqué, est resté vulnérable. Les blancs ont pu profiter de cette faiblesse pour lancer une attaque dévastatrice sur toutes les cases autour du roi noir et vont remporter la partie.



Enfin, nous comptons également faire peser les structures de pions dans l'évaluation. Les pions qui forment des chaînes et se protègent ensemble sont solides alors que des pions isolés ou doublés sur une même colonne sont souvent des cibles faciles.

Dans cet exemple, même si la position pourrait sembler équitable, les noirs ont en fait un grand avantage grâce à la très mauvaise structure des pions blancs. À cause de celle-ci, ils ne pourront pas défendre leurs pions vulnérables face au cavalier noir, alors que les pions noirs pourront résister au cavalier blanc en s'entre-aidant. Notre algorithme saura identifier ce genre de défauts positionnels.

Par la suite, si nous en avons le temps, il serait possible d'améliorer encore le niveau de jeu de notre intelligence artificielle et sa connaissance en ouvertures en lui fournissant par exemple des données issues de parties de maîtres. S'il reconnaît que la position actuelle est une position qu'il a déjà rencontré dans l'une de ces parties, cela pourrait lui permettre de jouer de manière plus riche et plus précise, surtout dans le début de partie.

INTERFACE

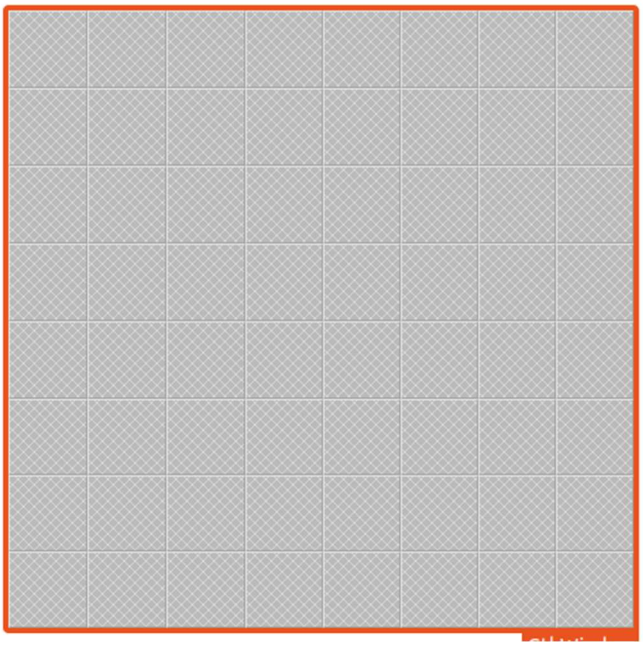
(Maxence Gatard)

Pour l'interface graphique, nous avons utilisé la librairie GTK que nous avions appris à utiliser lors d'un de nos TPs de programmation.

Nous avons donc réfléchi à deux méthodes différentes pour ce qui concerne l'interface graphique :

Une première qui se veut être la plus simple car le principe est de tout simplement se servir des GTK Grid, de mettre des boutons dans chacun des emplacements et de faire une grille d'un plateau simple 8x8.

Elle a un avantage majeur qui nous permettrait de déplacer nos pions plus facilement car nous aurions juste à mettre un bouton dans chacun des emplacements de la grille et cela rendrait les pions cliquables. Mais elle a un gros inconvénient qui nous dérange beaucoup, c'est son côté pas du tout esthétique. En effet, le plateau serait essentiellement d'une seule couleur (donc difficile pour voir la différence entre deux cases même s'il y aurait un contour), les pièces seraient difficilement visibles et nous aurions pas forcément tout le temps un carré parfait.



La deuxième méthode à laquelle nous avions pensé est l'utilisation d'une zone de dessin avec GTK. Elle consiste à mettre l'image du plateau sur le fond de cette zone de dessin, puis de positionner chacun des pions sur le plateau.

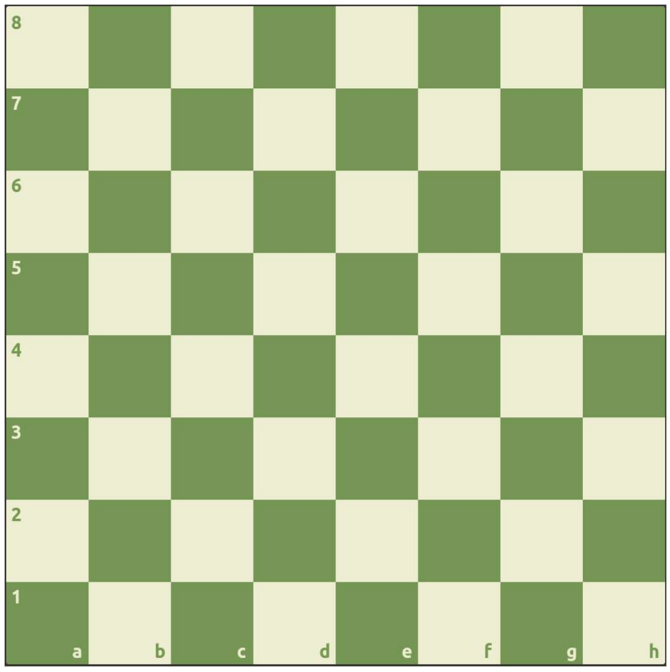
INTERFACE

(Maxence Gatard)

Elle a un inconvénient majeur, c'est le fait compliqué de pouvoir déplacer les pions, en effet ce n'est qu'une zone de dessin et c'est par conséquent non cliquable mais nous avons aussi pensé au fait de pouvoir mettre à côté de cette zone de dessin un espace ou nous pourrions choisir quelle pièce nous bougeons à tel endroit. (Exemple : A2 - A3)

Une autre façon de faire bouger les pièces que nous pourrions essayer aussi, est de récupérer, dès que le joueur clique sur le plateau les coordonnées de la souris au moment où il y a eu un clique gauche et donc nous permettre d'en déduire quelle pièce il souhaite faire bouger.

L'avantage de cette méthode est qu'elle va nous permettre énormément de liberté sur le design de l'application. Par exemple, nous pourrions avoir un menu au départ pour sélectionner la couleur du plateau de jeu et en fonction de la couleur du plateau choisi, changer l'image de fond. Mais aussi d'avoir différents styles de pièces pour le jeu et enfin de pouvoir par exemple activer une fonction qui permettrait à un débutant de savoir où il peut déplacer ses pièces en mettant des flèches dans la direction où le pion est autorisé à aller.



Nous avons donc choisi la deuxième méthode et elle sera, conformément au cahier des charges, prête pour la seconde soutenance.

CONCLUSION

Prévision

Pour la prochaine soutenance, nous avons pour objectif de finaliser la structure ainsi que l'interface. Il nous faudra avoir commencé l'implémentation de l'algorithme Min-Max et refaire le site web en HTML et CSS.
Ainsi, Antoine continuera la structure et finalisera le site, Maxence l'interface, et tout le groupe se concentrera sur l'algorithme Min-Max. Notre but étant de n'avoir qu'à optimiser ce dernier pour la soutenance finale.

Pour conclure, nous sommes satisfaits du travail fournit en une semaine, nous nous sommes suffisamment renseignés sur la théorie, il ne nous reste plus qu'à l'appliquer.