Projet IN104 (2025) - Programmation d'une IA pour le jeu d'échecs en C

Encadrant : Tom Boumba

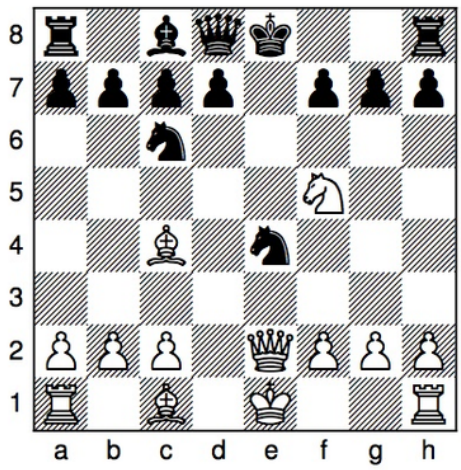
Fiche 3 : Masques d'attaque

Au cours d'une partie, tout type de coup joué par un joueur peut être une prise d'une pièce adverse si la case de destination est occupée, sauf deux :

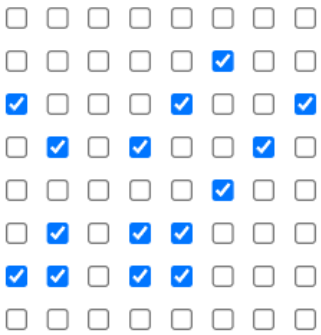
- Le déplacement d'un pion en avant
- Le roque

Pour ces deux types de coup, la case de destination doit être vide. Tous les autres types de coup peuvent constituer la prise d'une pièce adverse si la case de destination est occupée, c'est pourquoi on les appelle des *attaques*.

Dans l'optique de lister l'ensemble des coups possibles dans l'état actuel de l'échiquier, on souhaite d'abord déterminer l'ensemble des attaques au moyen de *masques d'attaque*. Un masque d'attaque pour un type de pièce donné est un bitboard dont les bits représentent la potentialité d'attaque du type de pièce donné sur l'échiquier dans la position actuelle (voir figure 1a).



(a) Position de l'échiquier



(b) Masque d'attaque des fous blancs : les cases cochées représentent des bits à 1, les cases non cochées des bits à 0.

Figure 1: Figure obtenue à l'aide de l'outil Bitboard Viewer.

Remarque : les masques d'attaques contiennent aussi les prises de pièces alliées, que l'on peut retirer à l'aide d'une simple opération binaire, par exemple :

```
attaques = attaques & ~occupations[blancs];
```

Pour les pions, les cavaliers, et les rois, les masques d'attaque dépendent uniquement de la présence du type de pièce considéré sur l'échiquier. On peut donc pré-calculer l'ensemble des masques d'attaque associés à la présence de chacun de ces types de pièce sur chaque case de l'échiquier, qui resteront statiques.

Pour les fous, les tours, et les dames, qui sont des pièces dites *glissantes*, les masques d'attaque dépéndent aussi des autres pièces présentes sur l'échiquier, qui peuvent *bloquer* l'attaque. On les représente sous la forme d'un *masque de blocage*, un bitboard dont les bits à 1 représentent la présence de pièces situées sur l'une des trajectoires de mouvement de la pièce considérée. On ne pourra donc pas précalculer ces masques d'attaque (à moins d'utiliser des Magic Bitboards). On calculera ces masques de manière dynamique.

Implémentation

Dans le fichier `masques.h`, on a déclaré les variables globales :

```
BB attaques_pions[2][64];
BB attaques_cavaliers[64];
BB attaques_rois[64];
```

qui contiendront l'ensemble des masques d'attaque statiques. On a aussi déclaré les fonctions :

```
BB masque_d_attaque_dynamique_fou(int square, BB bloqueurs);
BB masque_d_attaque_dynamique_tour(int square, BB bloqueurs);
BB masque_d_attaque_dynamique_dame(int square, BB bloqueurs);
```

qui permettront de calculer dynamiquement les masques d'attaques des pièces glissantes.

- Implémenter la fonction `void generer_attaques_pion()`, qui pré-calcule l'ensemble des masques d'attaque de pion. Pour gérer les masques d'attaque aux bords droit et gauche de l'échiquier, vous pourrez utiliser les variables `BB pas_colonne_lettre`, que vous aurez préalablement définies au moyen de la fonction `void generer_bb_pas_colonne()`. En effet, pour gérer les masques d'attaques aux bords de l'échiquier, il vous suffira de cette manière de ne pas ajouter au masque d'attaque résultant les cases correspondant au bord opposé de l'échiquier au moyen de ces variables `pas_colonne_lettre`.
- Implémenter la fonction `void generer_attaques_cavalier()`. Pour gérer les masques d'attaque aux bords droit et gauche de l'échiquier, vous pour-

rez utiliser les variables `BB pas_colonne_lettres` de la même façon que dans la fonction `void generer_attaques_pion()`.

- Implémenter la fonction `void generer_attaques_roi()`.
- Implémenter la fonction `void generer_masques_d_attaque_statiques()`, qui pré-calcule les masques d'attaque statiques.
- Implémenter la fonction `BB masque_d_attaque_dynamique_fou(int square, BB bloqueurs)`, qui calcule le masque d'attaque d'un fou présent sur la case `square` lorsque les pièces représentées par le bitboard `bloqueurs` entravent ses trajectoires.
- Implémenter la fonction `BB masque_d_attaque_dynamique_tour(int square, BB bloqueurs)`, qui calcule le masque d'attaque d'une tour présente sur la case `square` lorsque les pièces représentées par le bitboard `bloqueurs` entravent ses trajectoires.
- Implémenter la fonction `BB masque_d_attaque_dynamique_dame(int square, BB bloqueurs)`, qui calcule le masque d'attaque d'une dame présente sur la case `square` lorsque les pièces représentées par le bitboard `bloqueurs` entravent ses trajectoires.
  - Vous testerez vos masques d'attaque, par exemple :

```
generer_masques_d_attaque_statiques();
print_bitboard(attaques_cavaliers[e4]);
```

```
8   0 0 0 0 0 0 0 0
7   0 0 0 0 0 0 0 0
6   0 0 0 1 0 1 0 0
5   0 0 1 0 0 0 1 0
4   0 0 0 0 0 0 0 0
3   0 0 1 0 0 0 1 0
2   0 0 0 1 0 1 0 0
1   0 0 0 0 0 0 0 0

    a b c d e f g h
```

Figure 2: Masque d'attaque d'un cavalier en e4.

```
BB bloqueurs = 0ULL;
set_bit(bloqueurs, g6);
set_bit(bloqueurs, c6);
set_bit(bloqueurs, c2);
set_bit(bloqueurs, f3);
print_bitboard(masque_d_attaque_dynamique_fou(e4, bloqueurs));
```

```
8   0 0 0 0 0 0 0 0
7   0 0 0 0 0 0 0 0
6   0 0 1 0 0 0 1 0
5   0 0 0 1 0 1 0 0
4   0 0 0 0 0 0 0 0
3   0 0 0 1 0 1 0 0
2   0 0 1 0 0 0 0 0
1   0 0 0 0 0 0 0 0

    a b c d e f g h
```

Figure 3: Masque d'attaque d'un fou en e4 pour un masque de bloqueurs donné.