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

Encadrant : Tom Boumba

Fiche 4 : Coups

On souhaite à présent générer la liste des coups possibles dans une position donnée, et pouvoir exécuter chacun de ces coups sur l'échiquier.

Implémentation

Dans le fichier `coups.h`, on a défini un encodage (macros d'encodage et de décodage) permettant de représenter facilement n'importe quel coup sous la forme d'un entier, dont les 24 premiers bits représentent dans l'ordre :

- La case de départ du coup (6 bits)
- La case d'arrivée du coup (6 bits)
- Le type de la pièce qui se déplace (4 bits)
- Le type de pièce obtenue en cas de promotion de pion (4 bits)
- Le type de pièce capturée en cas de prise (4 bits)
- Si le coup est un déplacement d'un pion en avant de deux cases (1 bit)
- Si le coup est une prise en passant (1 bit)
- Si le coup est un roque (1 bit)

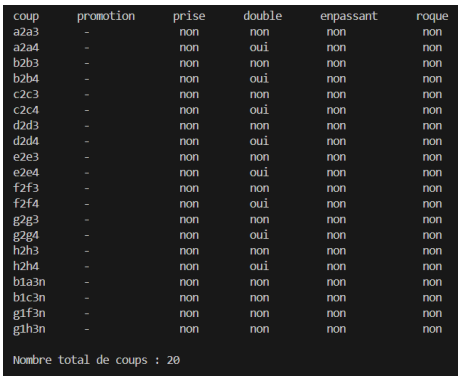
Remarque : La valeur 0 ne peut représenter aucun coup légal.

On a aussi déclaré une structure de données `coups` représentant la liste des coups possibles.

- Implémenter la fonction `void ajouter_coup(coups *coups, int coup)`, qui ajoute un nouveau coup à la liste fournie en entrée.
- Implémenter la fonction `void afficher_coup(int coup)`, qui affiche un coup au format `<case_de_depart><case_d_arrivee><type_de_piece_obtenue_par_promotion>` (ce format sera utile pour le protocole UCI).

- Implémenter la fonction `void afficher_coups(coups *coups)`, qui affiche tous les coups de la liste fournie en entrée. Cet affichage sera utilisé pour le débogage.
- Implémenter la fonction `int nb_bits(BB bb)`, qui renvoie le nombre de bits contenus dans le bitboard fourni en entrée (indication : on pourra calculer le résultat en autant d'opérations binaires qu'il y a de bits à 1 dans le bitboard fourni en entrée).
- Implémenter la fonction `int ls1b_index(BB bb)`, qui utilise la fonction `nb_bits` (on pourrait être encore plus efficient en utilisant l'approche De Bruijn) pour renvoyer l'indice du premier bit non nul du bitboard fourni en entrée, ou -1 si le bitboard est nul. Cette fonction sert à obtenir l'indice des cases de départ et d'arrivée des coups que l'on va générer en itérant sur les bits présents au sein des bitboards de présence actuelle des pièces sur l'échiquier ainsi que sur les masques d'attaques de ces pièces.
- Implémenter la fonction `void generer_coups(coups *coups)`, qui génère la liste des coups possibles dans l'état actuel de l'échiquier. On fera attention à bien inclure les coups spéciaux du pion (déplacement en avant, d'une ou de deux cases, déplacements avec promotion, prise en passant), et les coups de roque (on fera attention aux règles concernant la possibilité de roquer).
- Implémenter la fonction `int case_attaquee(int square, int couleur)`, qui renvoie 1 si la case en entrée est attaquée par les pièces de la couleur en entrée. Cette fonction servira à déterminer si un joueur est en échec. Indication : pour implémenter cette fonction, il est astucieux de remarquer qu'une case est attaquée par un type de pièce si une pièce de ce type est présente sur l'un des bits du masque d'attaque de ce type de pièce associé à la case considérée.
- Implémenter la fonction `int executer_coup(int coup)`, qui exécute le coup fourni en entrée sur l'échiquier s'il est légal, auquel cas la fonction renvoie 1, et 0 sinon (i.e si le coup joué met en échec le roi du joueur qui le joue). On sauvegardera l'état de l'échiquier avant l'exécution du coup et on le restaurera dans le cas où le coup est illégal à l'aide des macros définies dans le fichier `coups.h`. On fera attention à mettre à jour les bitboards d'occupation des pièces sur l'échiquier, ainsi que le trait. On fera aussi attention à mettre à jour la case prenable en passant dans le cas où le coup est un déplacement de pion en avant de deux cases. On fera également attention à mettre à jour les droits de roque. Pour cela, une variable globale `int droits_roque[64]` a été définie. Elle permet de mettre a jour les droits de roque en une opération binaire en fonction des cases de départ et d'arrivée du coup exécuté.
  - Vous testerez la génération de la liste des coups possibles dans une position donnée, par exemple :

```
  generer_masques_d_attaque_statiques();
  parse_fen(position_de_depart);
  coups *coups_possibles = (coups*)(malloc(sizeof(coups)));
  generer_coups(coups_possibles);
  afficher_coups(coups_possibles);
```



coup	promotion	prise	double	enpassant	roque
a2a3	-	non	non	non	non
a2a4	-	non	oui	non	non
b2b3	-	non	non	non	non
b2b4	-	non	oui	non	non
c2c3	-	non	non	non	non
c2c4	-	non	oui	non	non
d2d3	-	non	non	non	non
d2d4	-	non	oui	non	non
e2e3	-	non	non	non	non
e2e4	-	non	oui	non	non
f2f3	-	non	non	non	non
f2f4	-	non	oui	non	non
g2g3	-	non	non	non	non
g2g4	-	non	oui	non	non
h2h3	-	non	non	non	non
h2h4	-	non	oui	non	non
b1a3n	-	non	non	non	non
b1c3n	-	non	non	non	non
g1f3n	-	non	non	non	non
g1h3n	-	non	non	non	non

Nombre total de coups : 20

Figure 1: Liste des coups possibles dans la position initiale de l'échiquier.

```
int coup;
for (int i = 0; i < coups_possibles->nb_coups; i++) {
    coup = coups_possibles->tab_coups[i];
    copie_echiquier();
    if (!executer_coup(coup))
        continue;
    print_echiquier();
    restauration_echiquier();
}
```

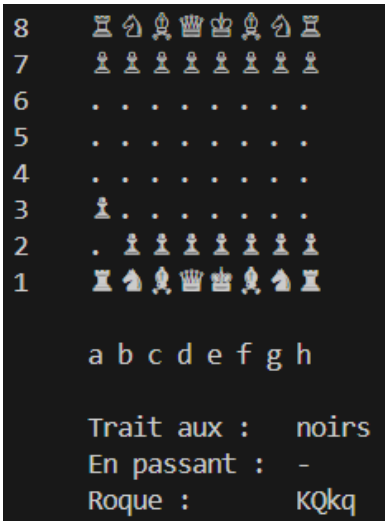


Figure 2: Affichage du résultat de l'exécution des coups possibles dans la position initiale de l'échiquier (la capture d'écran se limite au premier coup de la liste).