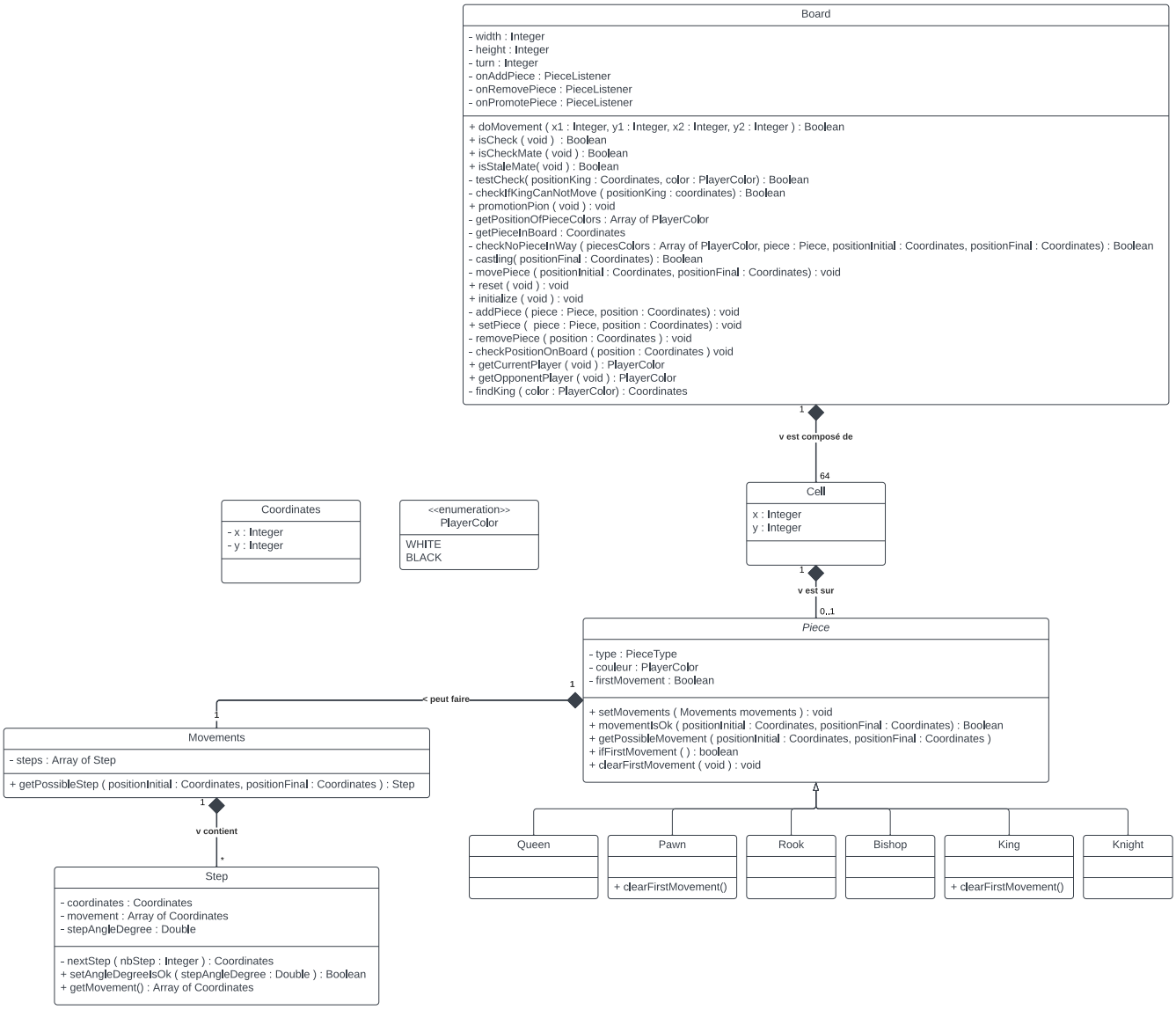
Laboratoire 8 - Jeu d'échecs

Groupe : L08GrK

Étudiants : Rodrigo Lopes dos Santos, Urs Behrmann

Modélisation UML



Choix de conception

GameManager

Le GameManager est la classe qui gère le jeu. Il contient les méthodes pour faire le liens entre l'UI et l'engine. Il hérite de la classe 'ChessController' qui est une classe abstraite qui contient les méthodes pour gérer le jeu. Elle contient la méthode 'initListenres' qui initialise les listeners qui transforme l'UI dans l'engine, lorsque l'utilisateur veut bouger une pièce sur la plateau à travers la fonction 'move'. On a aussi les fonctions 'newGame' qui reset le plateau et l'UI.

La 'updateMessage' est une fonction qui met à jour le message de l'UI. Elle est appelée à chaque fois qu'un mouvement est fait. Elle écrit qui qui doit jouer, si il y a un "check", "checkmate" ou "stalemate".

Plateau ('Board')

Le plateau est une classe qui contient les pièces et les méthodes pour les manipuler. Elle contient un Array de cellule qui représente le plateau affiché sur l'UI.

Elle contient les fonction qui servent à bouger les pièces, à les supprimer et à les ajouter. Elle contient aussi les fonctions qui servent à vérifier si un mouvement est légal ou pas, si le roi est en "check", "checkmate" ou "stalemate".

Elle est aussi responsable de déterminer quel joueur doit jouer et aussi le nombre de tour jouer.

Mouvement

Fichiers et Leur Rôle

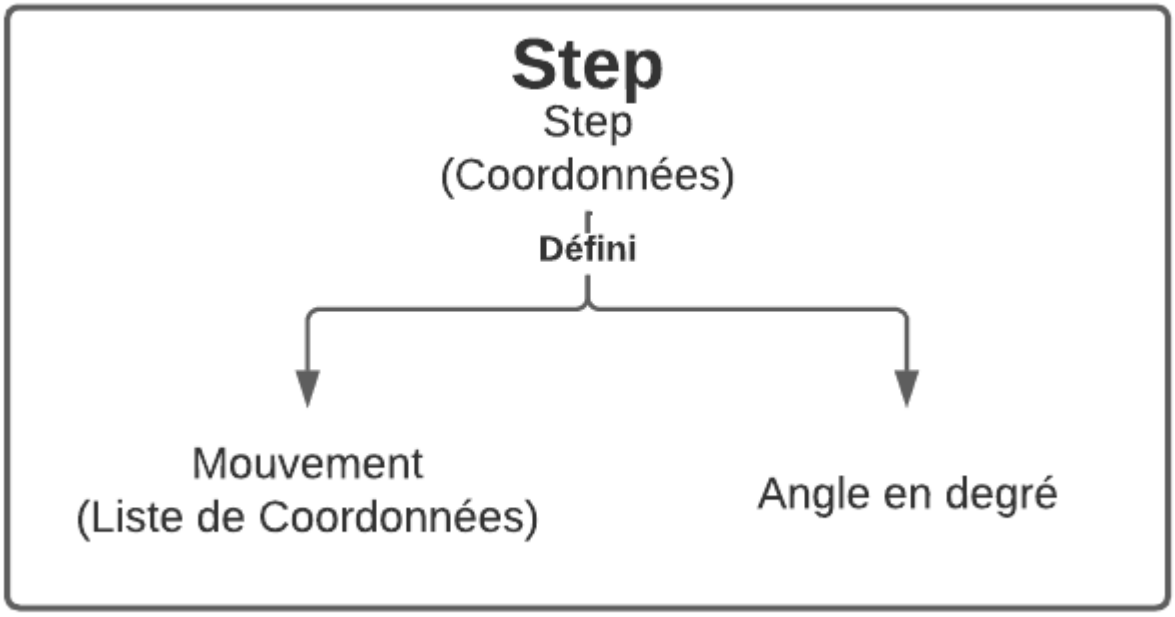
`Step.java`

- Définit un pas de mouvement.
- Selon le maxStep, il définie un mouvement via des les coordonnées :
  - [Pas de mouvement, Pas de mouvement + 1 pas, Pas de mouvement + 2 pas, ..., Pas de mouvement + maxStep]
- Contient les coordonnées d'un pas, une liste de coordonnées du mouvement selon maxStep et l'angle de déplacement.

- Il est possible d'inverser l'axe x et y. Cela est intéressant pour les pièces noir car, par exemple, avancer en avant est un step de (0, -1) et non pas de (0, 1)

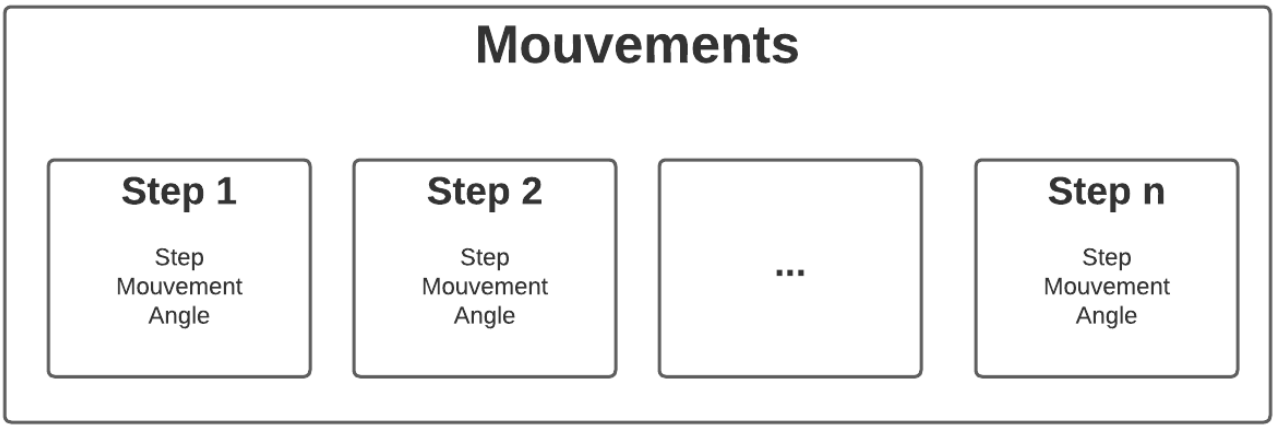
```
public Step(Coordinates step, int maxStep, boolean invertOrdinateAxis, boolean invertAbsc

public class X extends Piece {
    public X(PlayerColor color) {
        super(PieceType.X, color, new Movements(new Step[]{
                new Step(new Coordinates(1,0), 1, color == PlayerColor.BLACK, color == Pl
                // new Step(....), ...
                }));
    }
}
```



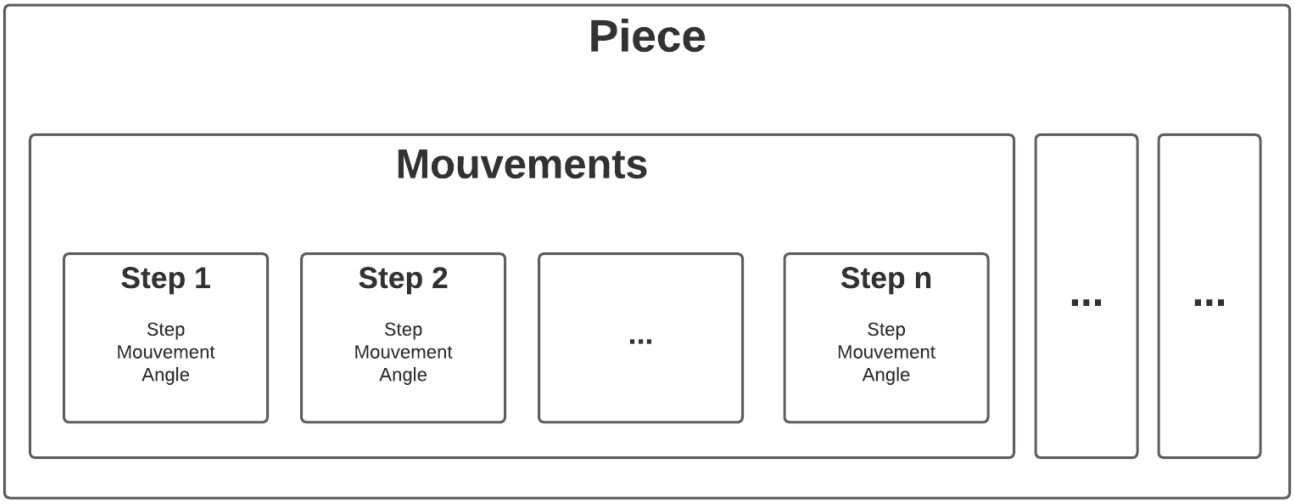
`Movements.java`

- Gère les mouvements des pièces.
- Contient une liste de `Step` et une méthode pour obtenir `Step` possible en fonction des positions initiale et finale.



`Piece.java`

- Contient une méthode `Coordinates[] getPossibleMovement( ... )` qui via les `movements` et les positions initiale et finale détermine le step possible et retourne son mouvement.
- Contient une méthode `boolean movementIsOk( ... )` qui via les positions initiale, finale et `Coordinates[] getPossibleMovement( ... )` détermine le mouvement possible et cherche dans celui-ci si la position final si trouve.
  - Si oui : Le mouvement de la position initiale à finale est possible
  - Sinon : Le mouvement de la position initiale à finale est impossible



Fonctionnement Général

Les mouvements des pièces sont gérés de manière modulaire, où chaque pièce a ses propres règles de mouvement définies dans des classes spécifiques (par exemple, `Knight.java` , `Pawn.java`). Ces classes utilisent la classe `Movements` pour gérer les déplacements possibles.

La classe `Step` joue un rôle clé en définissant la nature d'un pas de mouvement, tandis que `Coordinates` est utilisée pour représenter et manipuler les positions sur le plateau.

Exemple d'utilisation:

```
public class King extends Piece {
    public King(PlayerColor color) {
        super(PieceType.KING, color, new Movements(new Step[]{
                //Horizontal
                new Step(new Coordinates(1,0), 1, color == PlayerColor.BLACK, color == Pl
                new Step(new Coordinates(-1,0), 1, color == PlayerColor.BLACK, color == P

                //Vertical
                new Step(new Coordinates(0,1), 1, color == PlayerColor.BLACK, color == Pl
                new Step(new Coordinates(0,-1), 1, color == PlayerColor.BLACK, color == P

                //Diagonal
                new Step(new Coordinates(1,1), 1, color == PlayerColor.BLACK, color == Pl
                new Step(new Coordinates(1,-1), 1, color == PlayerColor.BLACK, color == P
                new Step(new Coordinates(-1,1), 1, color == PlayerColor.BLACK, color == P
                new Step(new Coordinates(-1,-1), 1, color == PlayerColor.BLACK, color ==
        }));
    }
}
```

Tests

On a crée un GameManagerTest et un BoardTest qui peuvent lancer plusieurs scènarios de test différent :

- test pawn
- test rook
- test knight
- test queen
- test king
- test bishop
- test pawn promotion
- test check

- test checkmate
- test stalemate
- test castling

test pawn

On a que deux pions sur le plateau pour tester le mouvement des pions et leur attack "en passant" et en diagonal.

test rook

On a que deux tours sur le plateau pour tester le mouvement des tours et leur attack.

On peut aussi tester le mouvement des tours quand il y a des pièces entre la tour et la position finale.

test knight

On a que deux cavaliers sur le plateau pour tester le mouvement des cavaliers et leur attack.

test queen

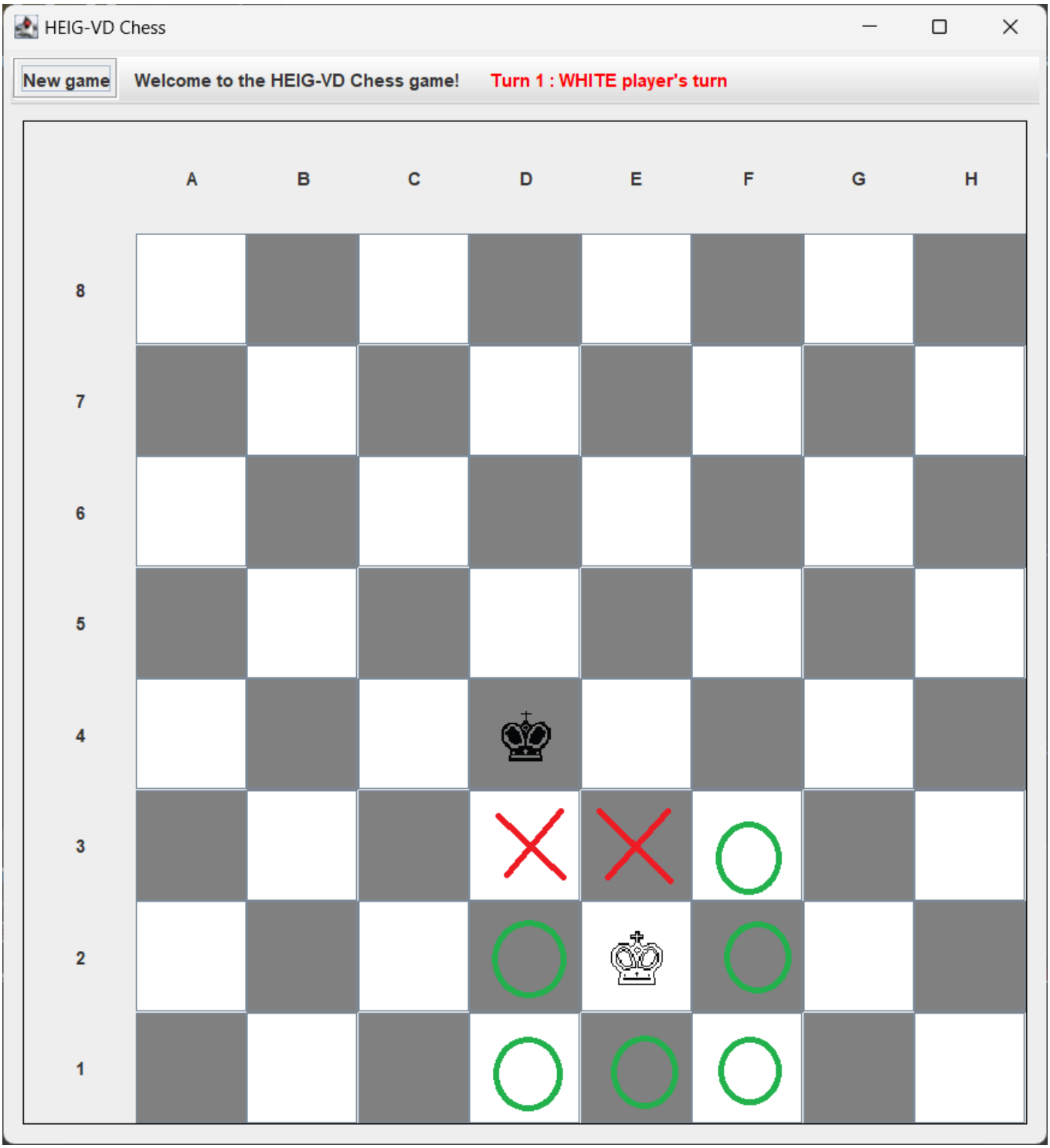
On a que deux reines sur le plateau pour tester le mouvement des reines et leur attack.

On peut aussi tester le mouvement des reines quand il y a des pièces entre la reine et la position finale.

test king

On a que deux rois sur le plateau pour tester le mouvement des rois et leur attack.

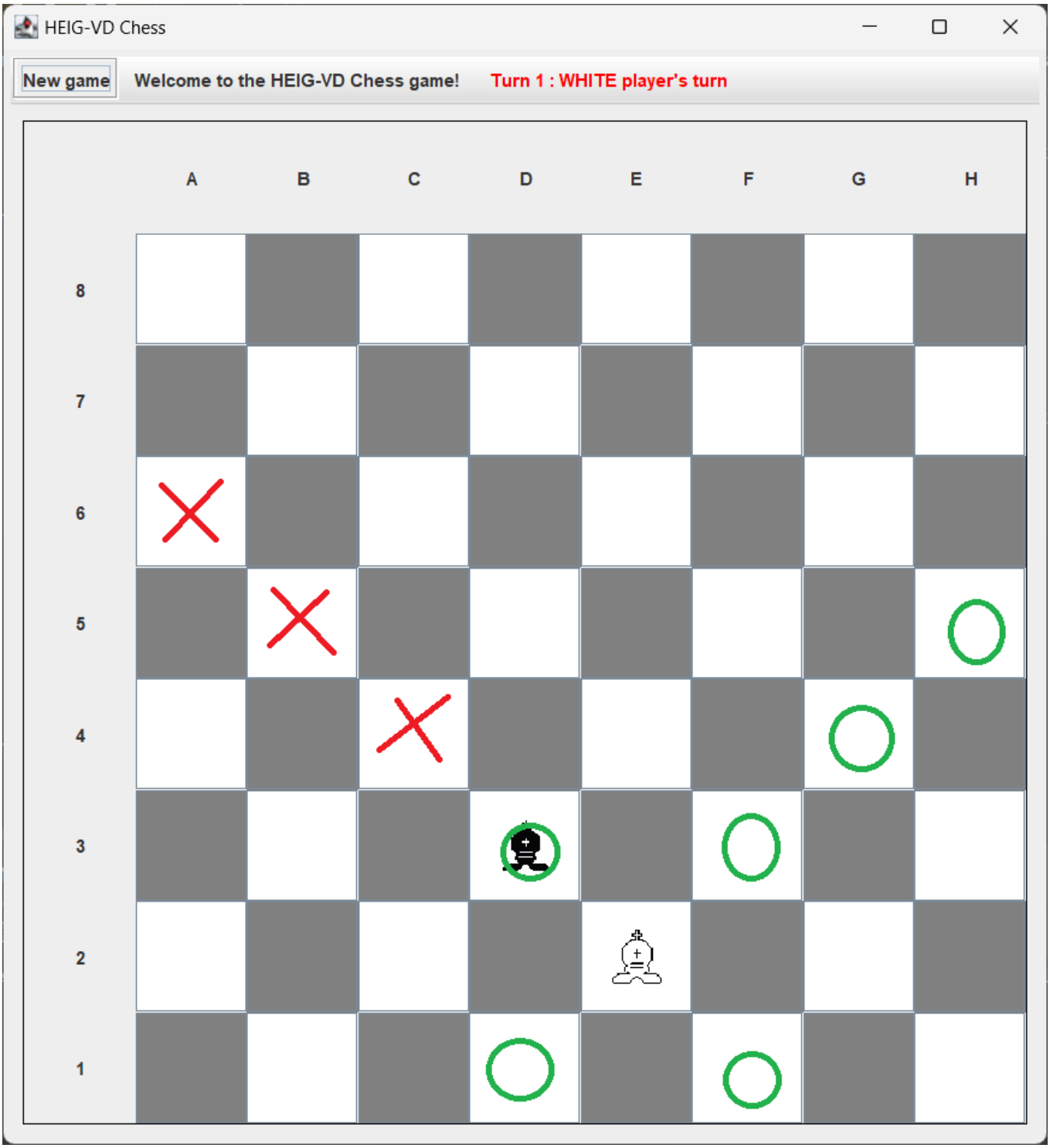
On peut aussi tester le mouvement des rois quand il y a une pièce qui peut attaquer le roi et est donc un mouvement illégal.



test bishop

On a que deux fous sur le plateau pour tester le mouvement des fous et leur attack.

On peut aussi tester le mouvement des fous quand il y a des pièces entre le fou et la position finale.

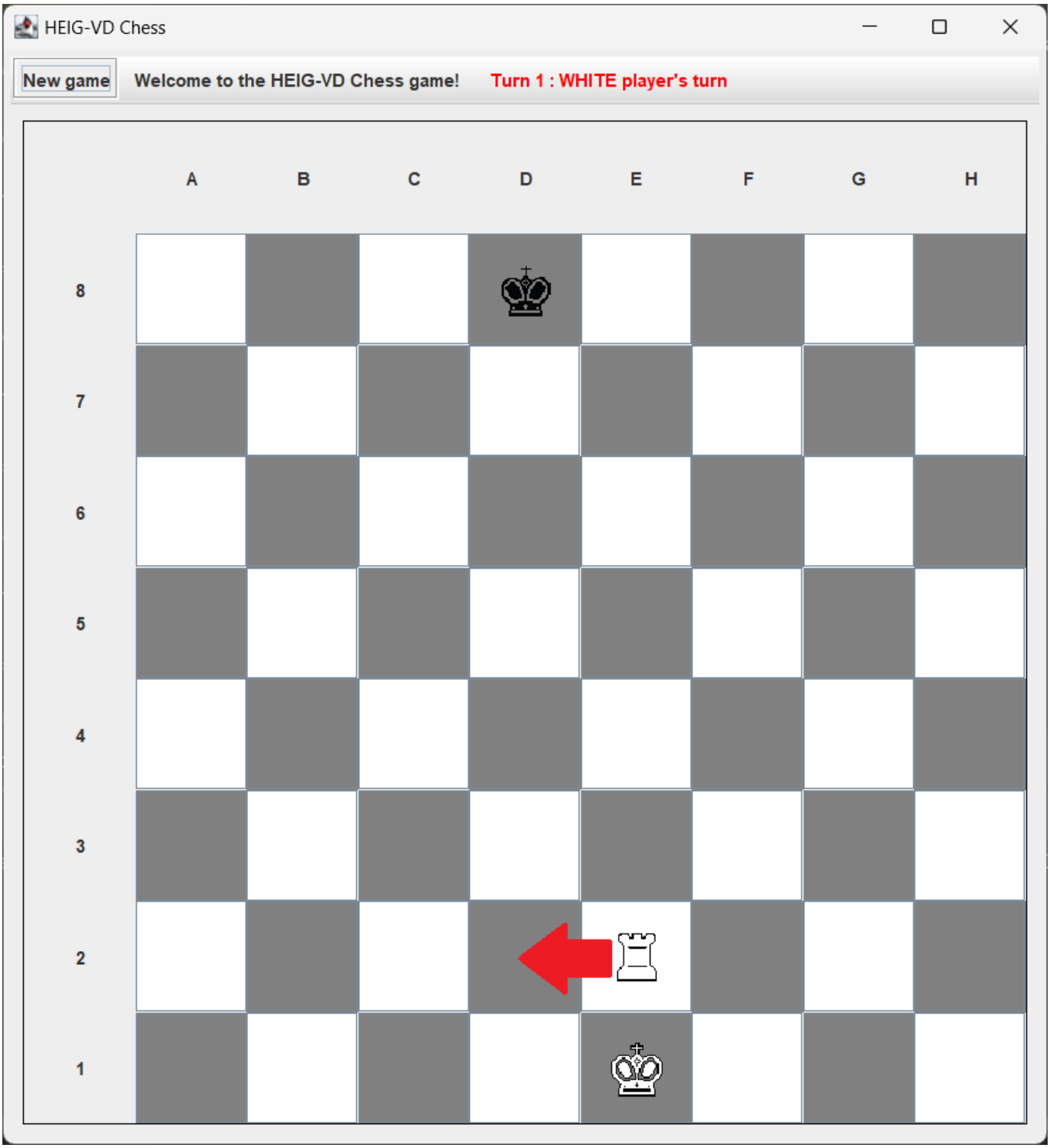


test pawn promotion

On a que deux pions sur le plateau pour tester la promotion des pions.

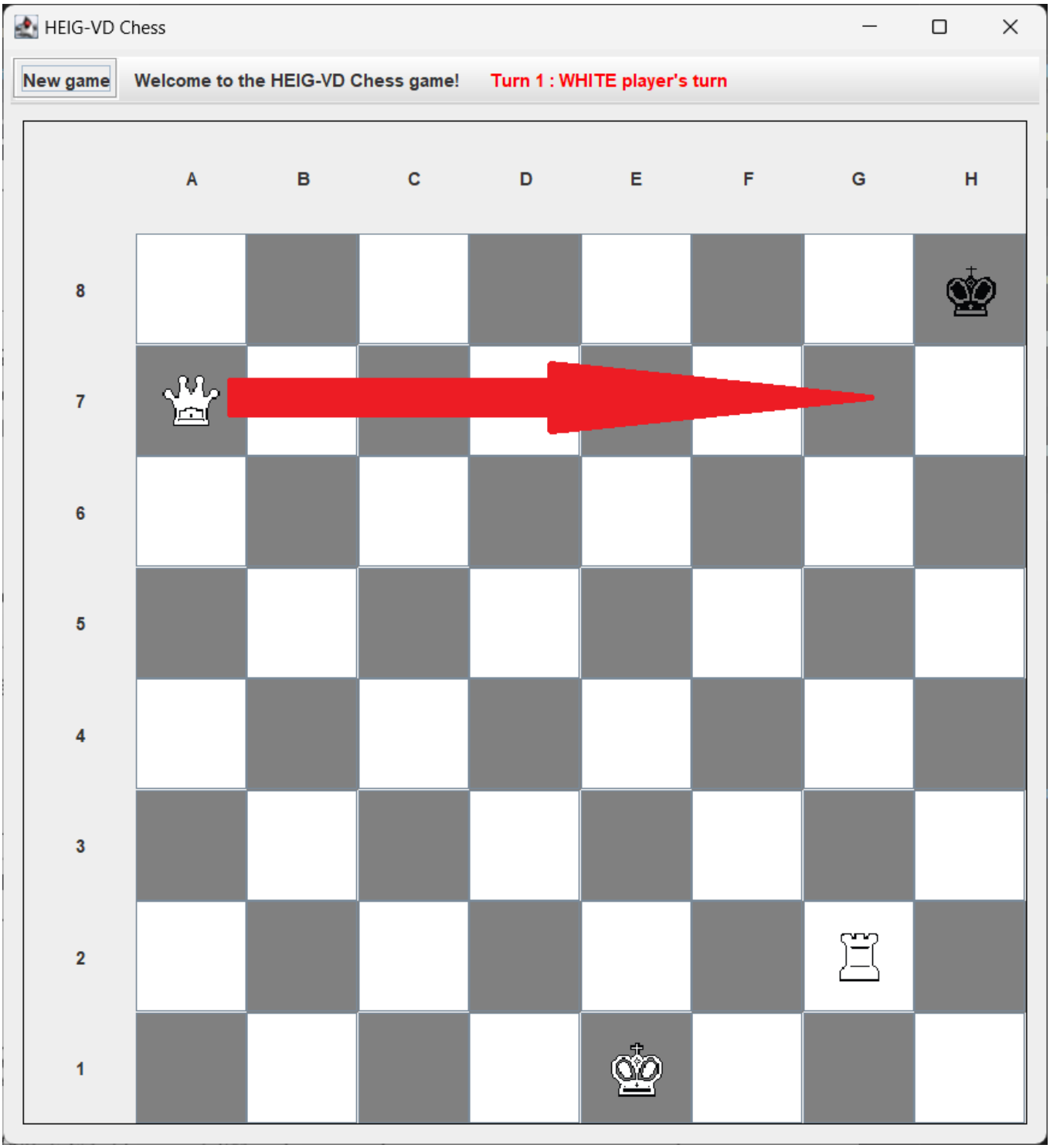
test check

On a quelque pièces sur le plateau pour tester le "check" pour noir.



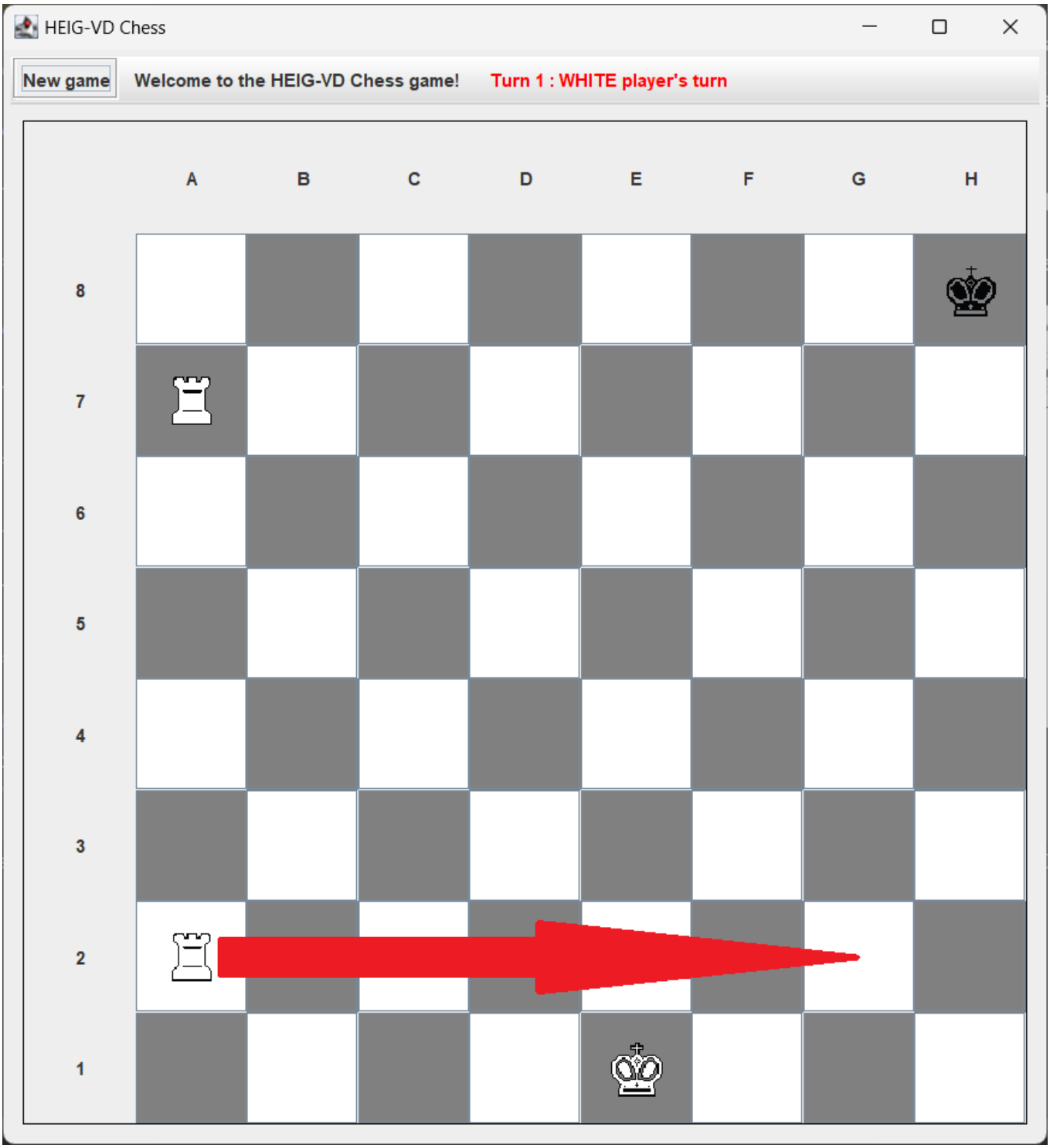
test checkmate

On a quelque pièces sur le plateau pour tester le "checkmate" pour noir.



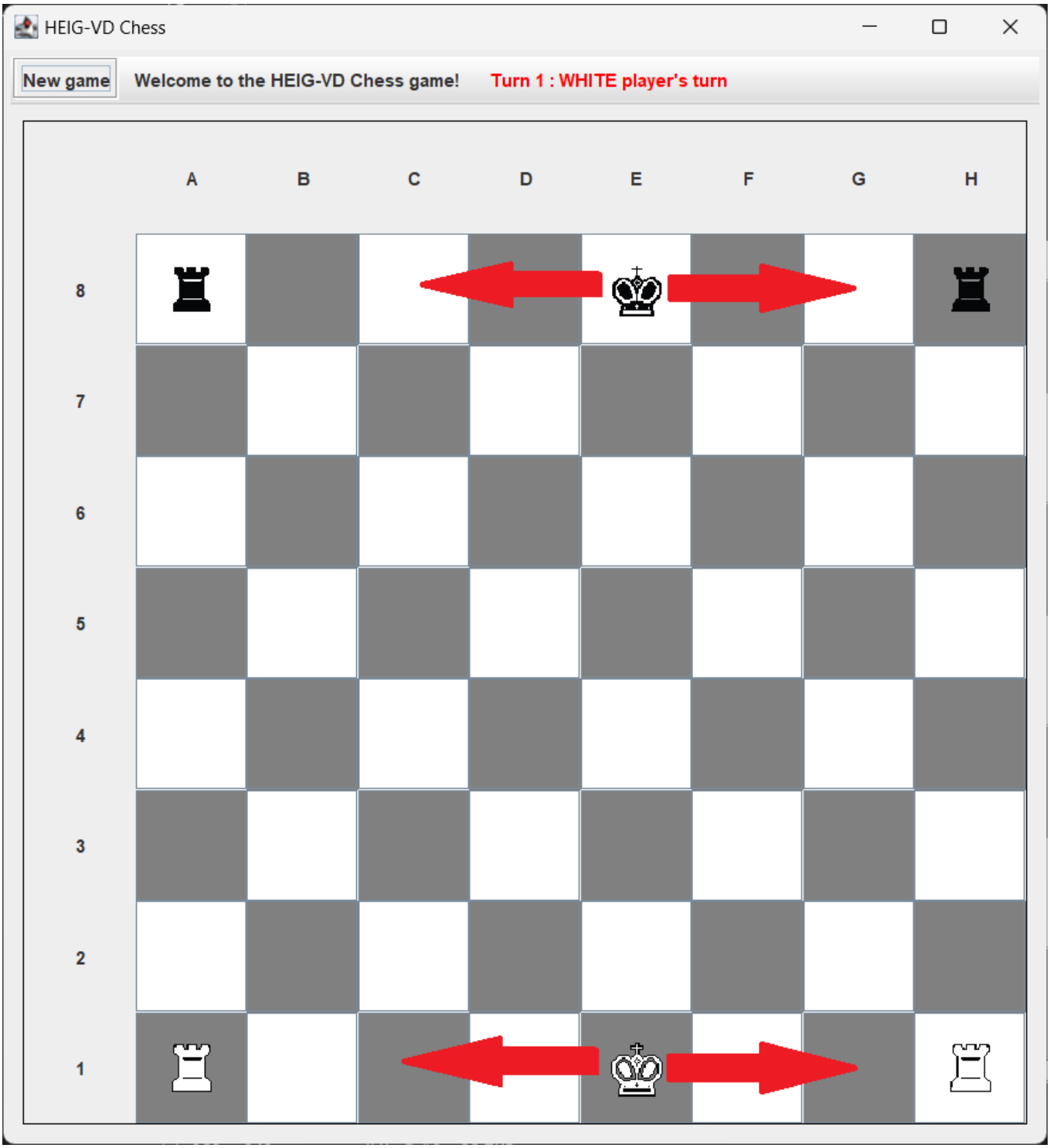
test stalemate

On a quelque pièces sur le plateau pour tester le "stalemate" pour noir.



test castling

On a les deux rois et les quatre tours sur le plateau pour tester les deux "castling" pour les deux couleurs.



Points bonus

L'échec, l'échec et mat et la pat sont implémentés. On a fait plusieurs tests pour vérifier si les fonctions marchent correctement, mais on n'a pas testé toutes les possibilités probablement.

L'implémentation du test si le roi est en échec et mat n'est pas très efficace. Car on essaie de bouger toutes les pièces de la couleur qui doit jouer et on vérifie si le roi est toujours en échec. Si oui, alors c'est un échec et mat. Si non, alors c'est un échec. Ceci fait que beaucoup de mouvements sont faits pour rien. On pourrait améliorer cette implémentation.

Listing du code