```java
package engine.game.displayChess;

import chess.PieceType;
import chess.PlayerColor;
import engine.game.board.Move;
import engine.game.board.Piece;
import engine.game.board.Vector;
import engine.game.chess.Chess;
import engine.game.chess.ChessColor;

import java.util.Objects;

/**
 * Synchronize GUI with the engine
 * @author Alen Bijelic
 * @author Nelson Jeanrenaud
 */
public class DisplayChess extends Chess {
    private final Controller controller;

    boolean areGUIPromptsDisable() {
        return guiPromptsDisabled;
    }

    private boolean guiPromptsDisabled;

    /**
     * DisplayChess constructor
     * @param controller Concerned controller
     */
    public DisplayChess(Controller controller) {
        super();
        this.controller = Objects.requireNonNull(controller, "controller must be non null");
        guiPromptsDisabled = false;
    }

    /**
     * Init rules
     */
    @Override
    protected void initRules() {
        super.initRules();
    }

    /**
     * Move a piece
     * @param fromX Start X value
     * @param fromY  Start Y value
     * @param toX Destination X value
     * @param toY Destination Y value
     * @return Either the piece can move or not
     */
    public boolean move(int fromX, int fromY, int toX, int toY) {
        return move(new Vector(fromX, fromY), new Vector(toX, toY));
    }

    /**
     * Check if the King is in check
     * @param defendingColor Concerned color
     * @return Either the King is in check or not
     */
    @Override
    public boolean check(ChessColor defendingColor) {
        if(super.check(defendingColor)){
            displayCheck();
            return true;
        }
        return false;
    }

    /**
     * Display winner
     * @param winner Winner color
     */
    @Override
    protected void endGame(ChessColor winner) {
        super.endGame(winner);
```

```
        displayWinner(winner);
    }

    /**
     * Set piece to a given position
     * @param piece Piece to set at given position
     * @param position Position to set the piece
     * @return The moved piece
     */
    @Override
    public ChessPiece setPieceAtPosition(Piece<Chess> piece, Vector position) {
        super.setPieceAtPosition(piece, position);
        if(piece != null)
            controller.getView().putPiece(getPieceType(((ChessPiece)piece).getPieceType()),
            getPlayerColor(((ChessPiece)piece).getColor()), position.getI(), position.getJ());
        return (ChessPiece)piece;
    }

    /**
     * Remove piece at a given position
     * @param position Position to remove the piece
     * @return The removed piece
     */
    @Override
    public ChessPiece removePieceAtPosition(Vector position) {
        Objects.requireNonNull(position, "position must be non null");
        ChessPiece piece = super.removePieceAtPosition(position);
        controller.getView().removePiece(position.getI(), position.getJ());
        return piece;
    }

    @Override
    public boolean doesMoveCheck(ChessPiece piece, Vector start, Vector destination, Move<Chess>
    moveType) {
        guiPromptsDisabled = true;
        boolean status = super.doesMoveCheck(piece, start, destination, moveType);
        guiPromptsDisabled = false;
        return status;
    }

    /**
     * Get promoted piece
     * @return Engine promoted piece
     */
    public Chess.ChessPiece getPromotedPiece() {
        if(!areGUIPromptsDisable())
            return askUserPromotion();
        return super.getPromotedPiece();
    }

    /**
     * Ask user to which Piece does he want to promote his pawn
     * @return Promoted piece
     */
    private ChessPiece askUserPromotion() {
        Piece[] possibilities = {
                new Queen(getTurn(), this),
                new Knight(getTurn(), this),
                new Rook(getTurn(), this),
                new Bishop(getTurn(), this)
        };

        return (ChessPiece) controller.getView().askUser("You can promote your piece !", "In what will
        your piece promote to ?", possibilities);
    }

    /**
     * Display Check message
     */
    public void displayCheck(){
        controller.getView().displayMessage("Check");
    }

    /**
     * Display winner when checkmate
     */
    protected void displayWinner(ChessColor winner) {
```

```
        controller.getView().displayMessage("Checkmate ! " + Objects.requireNonNull(winner, "winner
        must be non null") + " won the game !");
    }

    /**
     * Get the player color
     * @param color Chess color
     * @return Player color
     */
    private PlayerColor getPlayerColor(ChessColor color){
        switch (Objects.requireNonNull(color, "color must be non null")){
            case WHITE:
                return PlayerColor.WHITE;
            case BLACK:
                return PlayerColor.BLACK;
        }
        throw new IllegalArgumentException(color + " is not handled by GUI");
    }

    /**
     * Get enum Piece type
     * @param type Engine piece type
     * @return Enum Piece type
     */
    private PieceType getPieceType(ChessPieceType type){
        switch (Objects.requireNonNull(type, "type must be non null")) {
            case PAWN:
                return PieceType.PAWN;
            case ROOK:
                return PieceType.ROOK;
            case KNIGHT:
                return PieceType.KNIGHT;
            case BISHOP:
                return PieceType.BISHOP;
            case QUEEN:
                return PieceType.QUEEN;
            case KING:
                return PieceType.KING;
        }
        throw new IllegalArgumentException(type + " is not handled by GUI");
    }
}
```