```java
package engine;

import chess.ChessController;
import chess.ChessView;
import engine.utils.BoardDimensions;
import engine.pieces.*;
import engine.utils.Coordinates;

public class GameManager implements ChessController {
    protected ChessView view;
    protected Board board;

    public GameManager() {
        this.board = new Board(BoardDimensions.WIDTH.getValue(),
                BoardDimensions.HEIGHT.getValue());
    }

    /**
     * Update the message displayed by the view
     */
    protected void updateMessage() {
        if (view == null || board == null) return;

        StringBuilder message = new StringBuilder();

        // Add the current turn
        message.append("Turn ").append(board.getTurn()).append(" : ");

        // If the king is in checkmate
        if (board.isCheckMate()) {

            message.append("(Checkmate) ");
            message.append(board.getOpponentPlayer()).append(" player wins");

            // If the king is in stalemate
        } else if(board.isStaleMate()){
            message.append("(Stalemate) ");
            message.append("Draw");

        }else {
            // Add the current player
            message.append(board.getCurrentPlayer()).append(" player's turn ");

            // If the king is in check
            message.append(board.isCheck() ? "(Check)" : "");
        }

        // Update the message
        view.displayMessage(message.toString());
    }

    /**
     * Initialize the listeners
     */
    private void initListeners() {

        // Add the listener to add pieces
        board.setAddPieceListener((piece, cell) -> {
            if (view != null) {
                view.putPiece(piece.getType(), piece.getColor(), cell.getX(),
                        cell.getY());
            }
        });

        // Add the listener to remove pieces
        board.setRemovePieceListener((piece, cell) -> {
            if (view != null) {
                view.removePiece(cell.getX(), cell.getY());
            }
        });

        // Add the listener to promote pawns
        board.setPromotePawnListener((pawn, cell) -> {
            if (view != null) {
                // Ask the user which piece he wants
                ChessView.UserChoice choice = view.askUser("Promotion",
                        "Choose a piece to promote your pawn",
                        () -> "Queen",
                        () -> "Rook",
                        () -> "Bishop",
                        () -> "Knight");

                Coordinates coordinates = new Coordinates(cell.getX(),
                        cell.getY());
                // Set the new piece
                switch (choice.textValue()) {
                    case "Queen":
                        board.setPiece(new Queen(pawn.getColor()), coordinates);
                        break;
                    case "Rook":
                        board.setPiece(new Rook(pawn.getColor()), coordinates);
                        break;
                    case "Bishop":
                        board.setPiece(new Bishop(pawn.getColor()),
                                coordinates);
                        break;
                    case "Knight":
                        board.setPiece(new Knight(pawn.getColor()),
                                coordinates);
                        break;
                }
            }
        });
    }

    /**
     * Start the game
     *
     * @param view The view to use
     */
    @Override
    public void start(ChessView view) {
        this.view = view;

        // Start the view
        view.startView();

        // Initialize the piece listeners
        initListeners();

        // Update the message
        updateMessage();
    }

    /**
     * Move a piece
     *
     * @param fromX The X coordinate of the piece to move
     * @param fromY The Y coordinate of the piece to move
     * @param toX   The X coordinate of the destination
     * @param toY   The Y coordinate of the destination
     * @return True if the movement is valid, false otherwise
     */
    @Override
    public boolean move(int fromX, int fromY, int toX, int toY) {

        // Check if the movement is valid
        boolean valid = board.doMovement(fromX, fromY, toX, toY);

        // Update the message
        updateMessage();

        return valid;
    }

    /**
     * Start a new game
     */
    @Override
    public void newGame() {

        // Reset the board
        board.reset();

        // Put the pieces on the board
        board.initialize();

        // Update the message
        updateMessage();
    }
}
```