

```
1 package engine;
2
3 import chess.ChessController;
4 import chess.ChessView;
5 import engine.utils.BoardDimensions;
6 import engine.pieces.*;
7 import engine.utils.Coordinates;
8
9 public class GameManager implements ChessController {
10     protected ChessView view;
11     protected Board board;
12
13     public GameManager() {
14         this.board = new Board(BoardDimensions.WIDTH.getValue(),
15                                 BoardDimensions.HEIGHT.getValue());
16     }
17
18     /**
19      * Update the message displayed by the view
20      */
21     protected void updateMessage() {
22         if (view == null || board == null) return;
23
24         StringBuilder message = new StringBuilder();
25
26         // Add the current turn
27         message.append("Turn ").append(board.getTurn()).append(" : ");
28
29         // If the king is in checkmate
30         if (board.isCheckMate()) {
31
32             message.append("(Checkmate) ");
33             message.append(board.getOpponentPlayer()).append(" player wins");
34
35             // If the king is in stalemate
```

```

36     } else if(board.isStaleMate()){
37         message.append("(Stalemate) ");
38         message.append("Draw");
39     }
40     }else {
41         // Add the current player
42         message.append(board.getCurrentPlayer()).append(" player's turn ");
43     }
44     // If the king is in check
45     message.append(board.isCheck() ? "(Check)" : "");
46     }
47
48     // Update the message
49     view.displayMessage(message.toString());
50 }
51
52 /**
53  * Initialize the listeners
54  */
55 private void initListeners() {
56
57     // Add the listener to add pieces
58     board.setAddPieceListener((piece, cell) -> {
59         if (view != null) {
60             view.putPiece(piece.getType(), piece.getColor(), cell.getX(),
61                 cell.getY());
62         }
63     });
64
65     // Add the listener to remove pieces
66     board.setRemovePieceListener((piece, cell) -> {
67         if (view != null) {
68             view.removePiece(cell.getX(), cell.getY());
69         }
70     });

```

```

71
72 // Add the listener to promote pawns
73 board.setPromotePawnListener((pawn, cell) -> {
74     if (view != null) {
75         // Ask the user which piece he wants
76         ChessView.UserChoice choice = view.askUser("Promotion",
77             "Choose a piece to promote your pawn",
78             () -> "Queen",
79             () -> "Rook",
80             () -> "Bishop",
81             () -> "Knight");
82
83         Coordinates coordinates = new Coordinates(cell.getX(),
84             cell.getY());
85         // Set the new piece
86         switch (choice.textValue()) {
87             case "Queen":
88                 board.setPiece(new Queen(pawn.getColor(), coordinates);
89                 break;
90             case "Rook":
91                 board.setPiece(new Rook(pawn.getColor(), coordinates);
92                 break;
93             case "Bishop":
94                 board.setPiece(new Bishop(pawn.getColor(),
95                     coordinates);
96                 break;
97             case "Knight":
98                 board.setPiece(new Knight(pawn.getColor(),
99                     coordinates);
100                 break;
101         }
102     }
103 });
104 }
105

```

```
106 /**
107  * Start the game
108  *
109  * @param view The view to use
110  */
111 @Override
112 public void start(ChessView view) {
113     this.view = view;
114
115     // Start the view
116     view.startView();
117
118     // Initialize the piece listeners
119     initListeners();
120
121     // Update the message
122     updateMessage();
123 }
124
125 /**
126  * Move a piece
127  *
128  * @param fromX The X coordinate of the piece to move
129  * @param fromY The Y coordinate of the piece to move
130  * @param toX The X coordinate of the destination
131  * @param toY The Y coordinate of the destination
132  * @return True if the movement is valid, false otherwise
133  */
134 @Override
135 public boolean move(int fromX, int fromY, int toX, int toY) {
136
137     // Check if the movement is valid
138     boolean valid = board.doMovement(fromX, fromY, toX, toY);
139
140     // Update the message
```

```
141     updateMessage();
142
143     return valid;
144 }
145
146 /**
147  * Start a new game
148  */
149 @Override
150 public void newGame() {
151
152     // Reset the board
153     board.reset();
154
155     // Put the pieces on the board
156     board.initialize();
157
158     // Update the message
159     updateMessage();
160 }
161 }
```