```java
package Game;

import Pieces.Bishop;
import Pieces.King;
import Pieces.Knight;
import Pieces.Pawn;
import Pieces.Piece;
import Pieces.PieceType;
import Pieces.Queen;
import Pieces.Rook;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * This class represents the chess board
 *
 * @author Ethan Palser, Param Jansari
 */
public class Board {

    private Piece board[][]; // the chess board
    private int heuristicVal; // used to check value later without calculating again
    private File log; // used to log moves

    public Board(Board copy) {
        board = copy.board;
        heuristicVal = copy.heuristicVal;
        log = copy.log;
    }

    public Board() {
        log = new File("log.txt");

        //Empties file for new log
        PrintWriter pW;
        try {
            pW = new PrintWriter(log);
            pW.close();
        } catch (FileNotFoundException ex) {
            Logger.getLogger(Board.class.getName()).log(Level.SEVERE, null, ex);
        }

        board = new Piece[8][8];
        //Initializ White Pieces
        board[7][0] = new Rook(Colour.White); // Bottom left square black
        board[7][1] = new Knight(Colour.White);
        board[7][2] = new Bishop(Colour.White);
        board[7][3] = new Queen(Colour.White);
        board[7][4] = new King(Colour.White); // On black square
        board[7][5] = new Bishop(Colour.White);
        board[7][6] = new Knight(Colour.White);
        board[7][7] = new Rook(Colour.White); // Bottom right square white
        for (int i = 0; i < 8; i++) {
            board[6][i] = new Pawn(Colour.White);
        }

        //initailize Black Pieces
        board[0][0] = new Rook(Colour.Black); // Top left square white
        board[0][1] = new Knight(Colour.Black);
        board[0][2] = new Bishop(Colour.Black);
        board[0][3] = new Queen(Colour.Black);
```

```
70          board[0][4] = new King(Colour.Black); // On white squre
71          board[0][5] = new Bishop(Colour.Black);
72          board[0][6] = new Knight(Colour.Black);
73          board[0][7] = new Rook(Colour.Black); // Top right square black
74          for (int i = 0; i < 8; i++) {
75              board[1][i] = new Pawn(Colour.Black);
76          }
77      }
78
79      /**
80       * This method determines the board value of a player (colour) As in how
81       * well does the board look for the player
82       *
83       * @param playerColour
84       * @return
85       */
86      public int heristic(Colour playerColour) {
87          Piece piece;
88          int result = 0;
89          for (int i = 0; i < 8; i++) {
90              for (int j = 0; j < 8; j++) {
91                  piece = board[i][j];
92                  if (piece != null && piece.colour == playerColour) {
93                      result += board[i][j].heuristic(this, i, j);
94                  }
95              }
96          }
97          heuristicVal = result;
98          return result;
99      }
100
101     /**
102      * This method output the board to console
103      */
104     public void printBoard() {
105         System.out.println("   a   b   c   d   e   f   g   h");
106         System.out.println(" +---+---+---+---+---+---+---+---+");
107         for (int i = 0; i < board[0].length; i++) {
108             System.out.print((8-i) + "|");
109             for (int j = 0; j < board[0].length; j++) {
110                 if (board[i][j] == null) {
111                     System.out.print("   |");
112                 } else {
113                     String piece = "\u2006\u2006" + board[i][j].printToBoard() +
                        "\u2006\u2006\u2006|";
114                     System.out.print(piece);
115                 }
116             }
117             System.out.println("\n +---+---+---+---+---+---+---+---+");
118         }
119     }
120
121     /**
122      * This method prints the moves to log
123      *
124      * @param piece
125      * @param nextR
126      * @param nextC
127      * @param actions
128      * @param promotionTo
129      */
130     public void printToLog(Piece piece, int nextR, int nextC, ArrayList<Action>
        actions, Piece promotionTo) {
131         FileWriter fR;
132         // Hopefully ensures that the move is valid before logging
133         if (piece == null) {
134             return; // could log invalid move, but actual games do not
135         }
136         try {
```

```
137                 //Write to log
138                 fR = new FileWriter(log, true);
139                 String s = "";
140 
141                 for (Action action : actions) {
142                     switch (action) {
143                         case Move:
144                             s += piece.printToLog() + indexToBoardR(nextR) +
                                indexToBoardC(nextC);
145                             break;
146                         case Capture:
147                             s += piece.printToLog() + "x" + indexToBoardR(nextR) +
                                indexToBoardC(nextC);
148                             break;
149                         case Promotion:
150                             s += "=" + promotionTo.printToLog();
151                             break;
152                         case CastleKingSide:
153                             s += "0-0";
154                             break;
155                         case CastleQueenSide:
156                             s += "0-0-0";
157                             break;
158 //                        case EnPassant:
159 //                            s += "e.p.";
160 //                            break;
161                         case Check:
162                             s += "+";
163                             break;
164                         case Checkmate:
165                             s += "#";
166                             break;
167                         default:
168                             s = "Unknown Move";
169                             break;
170 
171                     }
172                 }
173                 fR.write(s);
174                 fR.close();
175             } catch (IOException ex) {
176                 Logger.getLogger(Board.class.getName()).log(Level.SEVERE, null, ex);
177             }
178         }
179 
180         /**
181          * This method prints the final outcome to log
182          *
183          * @param winner
184          */
185         public void printToLogfinalOutcome(Colour winner) {
186             FileWriter fR;
187             try {
188                 //Write to log
189                 fR = new FileWriter(log, true);
190                 String s = "";
191 
192                 if (null == winner) {
193                     s = "1/2 - 1/2";
194                 } else {
195                     switch (winner) {
196                         case White:
197                             s = "1-0";
198                             break;
199                         case Black:
200                             s = "0-1";
201                             break;
202                         default:
203                             s = "unknown outcome";
```

```
                        break;
                    }
                }
                fR.write(s);
                fR.close();
            } catch (IOException ex) {
                Logger.getLogger(Board.class.getName()).log(Level.SEVERE, null, ex);
            }

        }

        /**
         * This method output the log to console
         */
        public void printLog() {
            BufferedReader bR;
            try {
                bR = new BufferedReader(new FileReader(log));
                String line = bR.readLine();
                String output = "";
                while (line != null) {
                    output += line;
                    line = bR.readLine();
                }
                bR.close();
            } catch (Exception e) {
            }

        }

        /**
         * This method returns the board
         *
         * @return
         */
        public Piece[][] getBoard() {
            return board;
        }

        /**
         * This method converts the board row index to code index
         *
         * @param boardR
         * @return
         */
        public static int boardToIndexR(int boardR) {
            switch (boardR) {
                case 1:
                    return 7; // Bottom of Board in White's persepective
                case 2:
                    return 6;
                case 3:
                    return 5;
                case 4:
                    return 4;
                case 5:
                    return 3;
                case 6:
                    return 2;
                case 7:
                    return 1;
                case 8:
                    return 0; // Top of Board in White's persepctive
                default:
                    return -1;
            }
        }

        /**
```

```
273         * This method converts the board column to code index
274         *
275         * @param boardC
276         * @return
277         */
278        public static int boardToIndexC(char boardC) {
279            switch (boardC) {
280                case 'a':
281                    return 0; // Left of Board in White's persepective
282                case 'b':
283                    return 1;
284                case 'c':
285                    return 2;
286                case 'd':
287                    return 3;
288                case 'e':
289                    return 4;
290                case 'f':
291                    return 5;
292                case 'g':
293                    return 6;
294                case 'h':
295                    return 7; // Right of Board in White's persepctive
296                default:
297                    return -1;
298            }
299        }
300
301        /**
302         * This method convert the code index value to row on board
303         *
304         * @param indexR
305         * @return
306         */
307        public static int indexToBoardR(int indexR) {
308            switch (indexR) {
309                case 0:
310                    return 8; // Top of Board in White's persepective
311                case 1:
312                    return 7;
313                case 2:
314                    return 6;
315                case 3:
316                    return 5;
317                case 4:
318                    return 4;
319                case 5:
320                    return 3;
321                case 6:
322                    return 2;
323                case 7:
324                    return 1; // Bottom of Board in White's persepctive
325                default:
326                    return -1;
327            }
328        }
329
330        /**
331         * This method convert the code index input to column on board
332         *
333         * @param indexC
334         * @return
335         */
336        public static char indexToBoardC(int indexC) {
337            switch (indexC) {
338                case 0:
339                    return 'a'; // Left of Board in White's persepective
340                case 1:
341                    return 'b';
```

```
342                 case 2:
343                     return 'c';
344                 case 3:
345                     return 'd';
346                 case 4:
347                     return 'e';
348                 case 5:
349                     return 'f';
350                 case 6:
351                     return 'g';
352                 case 7:
353                     return 'h'; // Right of Board in White's persepctive
354                 default:
355                     return '-';
356             }
357         }
358     }
359
```