

```

1 package Project3;
2
3 import java.util.*;
4 import java.awt.*;
5 import java.awt.event.*;
6 import javax.imageio.ImageIO;
7 import javax.swing.*;
8
9 /*****
10  * CIS 163 Section 01
11  * Project 3: Chess Game
12  * ChessPanel Class
13  *
14  * This class is a public JPanel which handles the GUI for the chess
15  * game. Extends JPanel and implements ActionListener.
16  *
17  * @author George Fayette
18  * @version 3/23/2019
19  *****/
20 public class ChessPanel extends JPanel implements ActionListener {
21
22     /**
23      * Private JButton array representing the chess board.
24      */
25     private JButton[][] board;
26
27     /**
28      * Private JButtons for undo, PvP mode, PvAI mode, strobe, and make
29      * AI move.
30      */
31     private JButton undoButton, pvpButton, pvaiButton, aiMoveButton,
32         strobeButton;
33
34     /**
35      * Private JLabel for showing the number of moves.
36      */
37     private JLabel movesLabel;
38
39     /**
40      * Private ChessModel representing the game.
41      */
42     private ChessModel model;
43
44     /**
45      * Private ImageIcon arrays for storing the images for white and
46      * black pieces.
47      */
48     private ImageIcon[] whitePieces, blackPieces;
49
50     /**
51      * Private TileStrobes for the move to and move from locations.
52      */
53     private TileStrobe strobe, flash;
54
55     /**
56      * Private booleans representing the move from location being
57      * selected, whether the game is being played against the AI, and
58      * whether or not the strobe graphics are enabled.
59      */
60     private boolean firstTurnFlag, vsAI, strobeOn;

```

```

61
62     /**
63      * Private ints representing the coordinates that a chess piece is
64      * moving from.
65      */
66     private int fromRow, fromCol;
67
68     /**
69      * Private final String array for storing the the game piece names.
70      */
71     private final String[] pieces =
72         {"Pawn", "Rook", "Knight", "Bishop", "Queen", "King"};
73
74     /**
75      * Private final String array for storing the white piece images
76      */
77     private final String[] wFiles =
78         {"resources/wPawn.png", "resources/wRook.png",
79          "resources/wKnight.png", "resources/wBishop.png",
80          "resources/wQueen.png", "resources/wKing.png"};
81
82     /**
83      * Private final String array for storing the black piece images
84      */
85     private final String[] bFiles =
86         {"resources/bPawn.png", "resources/bRook.png",
87          "resources/bKnight.png", "resources/bBishop.png",
88          "resources/bQueen.png", "resources/bKing.png"};
89
90     /*****
91      * Public default constructor.
92      *****/
93     public ChessPanel() {
94         model = new ChessModel();
95         firstTurnFlag = true;
96         vsAI = false;
97         strobeOn = false;
98         createIcons();
99
100         JPanel boardPanel = new JPanel();
101         boardPanel.setLayout(
102             new GridLayout(model.numRows(), model.numColumns(), 1,
103                             1));
104         boardPanel.setPreferredSize(new Dimension(600, 600));
105         board = new JButton[model.numRows()][model.numColumns()];
106         for (int r = 0; r < model.numRows(); r++) {
107             for (int c = 0; c < model.numColumns(); c++) {
108                 createButton(r, c);
109                 setBackgroundColor(r, c);
110                 boardPanel.add(board[r][c]);
111             }
112         }
113         strobe = new TileStrobe(0, 0, 0);
114         flash = new TileStrobe(0, 0, 0);
115
116         JPanel buttonPanel = new JPanel();
117         buttonPanel.setLayout(new GridLayout(2, 4, 1, 1));
118         undoButton = new JButton("Undo");
119         undoButton.addActionListener(this);
120         pvpButton = new JButton("PvP");

```

```

121     pvpButton.addActionListener(this);
122     pvaiButton = new JButton("PvAI");
123     pvaiButton.addActionListener(this);
124     aiMoveButton = new JButton("AI Move");
125     aiMoveButton.addActionListener(this);
126     strobeButton = new JButton("Strobe");
127     strobeButton.addActionListener(this);
128     movesLabel = new JLabel("Moves: " + model.numMoves());
129     buttonPanel.add(pvpButton);
130     buttonPanel.add(pvaiButton);
131     buttonPanel.add(aiMoveButton);
132     buttonPanel.add(undoButton);
133     buttonPanel.add(strobeButton);
134     buttonPanel.add(movesLabel);
135
136     add(new JLabel("CHESS"), BorderLayout.NORTH);
137     add(boardPanel, BorderLayout.CENTER);
138     add(buttonPanel, BorderLayout.SOUTH);
139 }
140
141 // Sets the background color for the board
142 private void setBackgroundColor(int r, int c) {
143     if ((c % 2 == 1 && r % 2 == 0) || (c % 2 == 0 && r % 2 == 1)) {
144         board[r][c].setBackground(Color.LIGHT_GRAY);
145     } else {
146         board[r][c].setBackground(Color.WHITE);
147     }
148 }
149
150 // Creates the JButtons for the board
151 private void createButton(int r, int c) {
152     if (model.pieceAt(r, c) == null) {
153         board[r][c] = new JButton(null, null);
154     } else {
155         for (int i = 0; i < pieces.length; ++i) {
156             if (model.pieceAt(r, c).type().equals(pieces[i])) {
157                 if (model.pieceAt(r, c).player() == Player.WHITE) {
158                     board[r][c] = new JButton(null, whitePieces[i]);
159                 } else if (model.pieceAt(r, c).player() ==
160                     Player.BLACK) {
161                     board[r][c] = new JButton(null, blackPieces[i]);
162                 }
163             }
164         }
165     }
166     board[r][c].addActionListener(this);
167 }
168
169 // Reads image files and stores in ImageIcon arrays
170 private void createIcons() {
171     whitePieces = new ImageIcon[pieces.length];
172     blackPieces = new ImageIcon[(pieces.length)];
173     try {
174         for (int i = 0; i < pieces.length; ++i) {
175             whitePieces[i] = new ImageIcon(ImageIO.read(
176                 getClass().getResource(wFiles[i])));
177             blackPieces[i] = new ImageIcon(ImageIO.read(
178                 getClass().getResource(bFiles[i])));
179         }
180     } catch (Exception e) {

```

```

181         System.out.println("Error creating icons");
182     }
183 }
184
185 // method that updates the board
186 private void displayBoard() {
187     for (int r = 0; r < 8; r++) {
188         for (int c = 0; c < 8; c++) {
189             if (model.pieceAt(r, c) == null) {
190                 board[r][c].setIcon(null);
191             } else {
192                 for (int i = 0; i < pieces.length; ++i) {
193                     if (model.pieceAt(r, c).type()
194                         .equals(pieces[i])) {
195                         if (model.pieceAt(r, c).player() ==
196                             Player.WHITE) {
197                             board[r][c].setIcon(whitePieces[i]);
198                         } else if (model.pieceAt(r, c).player() ==
199                             Player.BLACK) {
200                             board[r][c].setIcon(blackPieces[i]);
201                         }
202                     }
203                 }
204             }
205         }
206     }
207     movesLabel.setText("Moves: " + model.numMoves());
208     repaint();
209 }
210
211 /*****
212  * This method handles ActionEvents from the GUI elements.
213  * @param event An ActionEvent from the GUI.
214  *****/
215 public void actionPerformed(ActionEvent event) {
216     for (int r = 0; r < model.numRows(); r++) {
217         for (int c = 0; c < model.numColumns(); c++) {
218             if (board[r][c] == event.getSource()) {
219                 // First click
220                 if (firstTurnFlag) {
221                     if (model.pieceAt(r, c) != null &&
222                         model.pieceAt(r, c).player() ==
223                             model.currentPlayer()) {
224                         fromRow = r;
225                         fromCol = c;
226                         firstTurnFlag = false;
227                         strobe = new TileStrobe(r, c, -1);
228                     }
229                     // If another piece is selected
230                 } else if (model.pieceAt(r, c) != null &&
231                     model.pieceAt(r, c).player() ==
232                         model.currentPlayer()) {
233                     fromRow = r;
234                     fromCol = c;
235                     strobe.stop();
236                     strobe = new TileStrobe(r, c, -1);
237                     // Try the move
238                 } else {
239                     attemptMove(new Move(fromRow, fromCol, r, c));
240                 }
241             }
242         }
243     }
244 }

```

```

241         }
242     }
243 }
244
245 if (undoButton == event.getSource()) {
246     if (vsAI) {
247         model.undo();
248         checkStatus();
249         model.undo();
250     } else {
251         model.undo();
252     }
253
254     firstTurnFlag = true;
255     strobe.stop();
256     displayBoard();
257     checkStatus();
258 }
259
260 if (pvpButton == event.getSource()) {
261     vsAI = false;
262 }
263
264 if (pvaiButton == event.getSource()) {
265     vsAI = true;
266     if (model.currentPlayer() == Player.BLACK) {
267         model.AI();
268         firstTurnFlag = true;
269         strobe.stop();
270         displayBoard();
271         checkStatus();
272     }
273 }
274
275 if (aiMoveButton == event.getSource()) {
276     model.AI();
277     firstTurnFlag = true;
278     strobe.stop();
279     displayBoard();
280     checkStatus();
281     if (model.currentPlayer() == Player.BLACK && vsAI) {
282         model.AI();
283         displayBoard();
284         checkStatus();
285     }
286 }
287
288 if (strobeButton == event.getSource()) {
289     if (strobeOn) {
290         strobeOn = false;
291         for (int r = 0; r < model.numRows(); r++) {
292             for (int c = 0; c < model.numColumns(); c++) {
293                 setBackgroundColor(r, c);
294             }
295         }
296     } else {
297         strobeOn = true;
298     }
299 }
300 }

```

```

301
302
303     // This method attempts to make a move chosen by the player
304     private void attemptMove(Move m) {
305         if (model.tryMove(m)) {
306             firstTurnFlag = true;
307             flash = new TileStrobe(m.toRow, m.toColumn, 51);
308             strobe.stop();
309             displayBoard();
310             checkStatus();
311
312             if (vsAI && model.currentPlayer() == Player.BLACK) {
313                 model.AI();
314                 displayBoard();
315                 checkStatus();
316             }
317         }
318     }
319
320     // This method checks the current game status and informs the
321     // player or asks for input as necessary
322     private void checkStatus() {
323         model.updateStatus();
324
325         if (model.GUIcode() == GUIcodes.UPGRADE) {
326             if (vsAI && model.currentPlayer() == Player.WHITE) {
327                 model.upgradePawn("Queen");
328             } else {
329                 String upgrade = JOptionPane.showInputDialog(null,
330                     "Enter promotion type.\n" +
331                     " R = Rook\nK = Knight\nB = " +
332                     "Bishop\nDefault is Queen");
333                 if (upgrade == null) {
334                     upgrade = "";
335                 }
336
337                 upgrade = upgrade.toLowerCase();
338                 if (upgrade.equals("r")) {
339                     model.upgradePawn("Rook");
340                 } else if (upgrade.equals("k")) {
341                     model.upgradePawn("Knight");
342                 } else if (upgrade.equals("b")) {
343                     model.upgradePawn("Bishop");
344                 } else {
345                     model.upgradePawn("Queen");
346                 }
347             }
348             displayBoard();
349             model.updateStatus();
350         }
351
352         if (model.GUIcode() == GUIcodes.CHECKMATE) {
353             flash.stop();
354             flashBoard(153);
355
356             if (model.currentPlayer() == Player.BLACK) {
357                 JOptionPane.showMessageDialog(null,
358                     "CheckMate! White Wins!", "Hooray!",
359                     JOptionPane.INFORMATION_MESSAGE,
360                     whitePieces[5]);

```

```

361         } else {
362             JOptionPane.showMessageDialog(null,
363                 "CheckMate! Black Wins!", "Hooray!",
364                 JOptionPane.INFORMATION_MESSAGE,
365                 blackPieces[5]);
366         }
367     } else if (model.GUIcode() == GUIcodes.DRAW) {
368         flash.stop();
369         flashBoard(153);
370
371         JOptionPane
372             .showMessageDialog(null, "It's a Draw!!", "Draw!",
373                 JOptionPane.INFORMATION_MESSAGE,
374                 whitePieces[5]);
375
376     } else if (model.GUIcode() == GUIcodes.IN_CHECK) {
377         flash.stop();
378         flashBoard(50);
379
380         if (model.currentPlayer() == Player.BLACK) {
381             JOptionPane
382                 .showMessageDialog(null, "Black is in check!",
383                     "Yikes!", JOptionPane.WARNING_MESSAGE,
384                     blackPieces[0]);
385         } else {
386             JOptionPane
387                 .showMessageDialog(null, "White is in check!",
388                     "Yikes!", JOptionPane.WARNING_MESSAGE,
389                     whitePieces[0]);
390         }
391     }
392 }
393
394 // This method flashes all tiles on the board
395 private void flashBoard(int ticks) {
396     for (int r = 0; r < model.numRows(); ++r) {
397         for (int c = 0; c < model.numColumns(); ++c) {
398             flash = new TileStrobe(r, c, ticks);
399         }
400     }
401 }
402
403 /*****
404  * CIS 163 Section 01
405  * Project 3: Chess Game
406  * TileStrobe Class
407  *
408  * This class rapidly changes the color of a tile on the board to
409  * produce a strobe effect
410  *
411  * @author George Fayette
412  * @version 3/23/2019
413  *****/
414 private class TileStrobe extends TimerTask {
415     java.util.Timer timer;
416     JButton StrobeButton;
417     int bRow;
418     int bCol;
419     int tickCounter;
420     int numTicks;

```

```

421
422     // Default constructor, strobes the tile at location r,c for
423     // given number of ticks
424     private TileStrobe(int r, int c, int ticks) {
425         bRow = r;
426         bCol = c;
427         tickCounter = 0;
428         numTicks = ticks;
429         StrobeButton = board[bRow][bCol];
430         timer = new java.util.Timer(true);
431         timer.scheduleAtFixedRate(this, 0, 15);
432     }
433
434     /*****
435     * This method is executed every time the TimerTask is called.
436     *****/
437     public void run() {
438         if (numTicks < 0 || tickCounter < numTicks) {
439             if (strobeOn) {
440                 StrobeButton.setBackground(
441                     new Color(tickCounter * 5 % 256,
442                             tickCounter * 5 % 256,
443                             tickCounter * 5 % 256));
444                 ++tickCounter;
445             }
446             else {
447                 stop();
448             }
449         }
450
451         // This method stops the strobe effect and resets the tile
452         // background.
453         private void stop() {
454             timer.cancel();
455             this.cancel();
456             setBackground(bRow, bCol);
457         }
458     }
459 }

```