```java
package project2;

import java.awt.*;
import java.util.ArrayList;
import java.util.Random;

/**********************************************************************
 * This program builds a GUI for SuperTicTacToe. SuperTicTacToe is a game
 * where the user chooses the size of the board and the number of
 * connections needed to win. It then allows the user to play with another
 * person or against an AI. The program will tell the user when either
 * side has won, lost, or tied.
 *
 * @author Justin Von Kulajta Winn & Nick Layman
 * @version 1.8
 **********************************************************************/

public class SuperTicTacToeGame {

    /** this is the board the that an X or O is 'placed' onto */
    private Cell[][] board;

    /** This is a variable that tells the program if one side has won or tied */
    private GameStatus status;

    /** This is a variable that determines who's turn it is */
    private Cell turn;

    /** This is the number of connections needed to win */
    private int connections;

    /** This is a variable that determines if X's or O's go first */
    private Cell starter;

    /** This is the width of the board */
    private int width;

    /** This is the height of the board */
    private int height;

    /** This is the number of board tiles that have an X or O in them */
    private int numSelected;

    /** This defines whether the AI is on or off */
    private static final boolean AI = true;

    /** This is an Array list that holds the location of previous plays made */
    private ArrayList<Point> backup = new ArrayList<Point>();

    /**********************************************************************
     * This constructor is the basic constructor. It sets the board to it's
     * default settings
     **********************************************************************/
    public SuperTicTacToeGame() {
        this(3, 3, 3, Cell.X, false);
    }
```

```
 57
 58      /**************************************************************************
 59       * This constructor sets the width, height, and number of connections
 60       * needed to win for the current board. It also determines if X or O
 61       * is to go first
 62       * @param pHeight is the height of the board
 63       * @param pWidth is the width of the board
 64       * @param pConnections is the number of connections needed to win the game
 65       * @param pStarter holds the cell that lets the program know who goes
 66       *                 first
 67       * @param pAI is whether or not the AI is on or off
 68       **************************************************************************/
 69      public SuperTicTacToeGame(int pHeight, int pWidth, int pConnections,
 70                                Cell pStarter, boolean pAI) {
 71          height = pHeight;
 72          width = pWidth;
 73          connections = pConnections;
 74          starter = pStarter;
 75          turn = starter;
 76          numSelected = 0;
 77          backup = new ArrayList<Point>();
 78
 79          status = GameStatus.IN_PROGRESS;
 80          board = new Cell[height][width];
 81          reset();
 82
 83          for (int row = 0; row < height; row++)
 84              for (int col = 0; col < width; col++)
 85                  board[row][col] = Cell.EMPTY;
 86      }
 87
 88      /**************************************************************************
 89       * This function returns the current board
 90       * @return is the current board with the currently marked X's and O's
 91       **************************************************************************/
 92      public Cell[][] getBoard() {
 93          return board;
 94      }
 95
 96      /**************************************************************************
 97       * This function selects a row and column on the board and fills it
 98       * with the respective X or O which is dependent on who's turn it is.
 99       * It then checks the status of the game and switches who's turn it is
100       * @param col is the y coordinate that is being selected
101       * @param row is the x coordinate that is being selected
102       **************************************************************************/
103      public void select(int row, int col) {
104          if (board[row][col] != Cell.EMPTY)
105              return;
106
107          board[row][col] = turn;
108          numSelected++;
109
110          turn = (turn == Cell.O) ? Cell.X : Cell.O;
111          status = isWinner();
112          backup.add(new Point (row, col));
```

```
113     }
114
115     /**********************************************************************
116      * This function goes back by 1 turn. It removes the last icon placed,
117      * which is either an X or O. It removes the point saved in the arrayList,
118      * Backup. It then switches who's turn it is and lowers the number
119      * of X's and O's by 1.
120      **********************************************************************/
121     public void undo(){
122         Point lastMove = backup.get(backup.size() - 1);
123         int r = (int) lastMove.getX();
124         int c = (int) lastMove.getY();
125         board[r][c] = Cell.EMPTY;
126         backup.remove(backup.size() - 1);
127         turn = (turn == Cell.O) ? Cell.X : Cell.O;
128         numSelected--;
129     }
130
131     /**********************************************************************
132      * This function determines if there has been a winner or not of the
133      * game. It checks if X or O has won vertically, horizontally, or
134      * diagonally. If the board is completely full, it declares the game
135      * a CATS game. IF none of these conditions are true, it returns the
136      * game status 'IN_PROGRESS."
137      * @return the current GameStatus of the board which will
138      *         be either IN_PROGRESS, X_WON, O_WON, OR CATS
139      **********************************************************************/
140     private GameStatus isWinner() {
141         int xCount = 0;
142         int oCount = 0;
143         for (int r = 0; r < height; r++)
144             for (int c = 0; c < width; c++) {
145                 // horizontal win
146                 if (c <= width - connections) {
147                     xCount = 0;
148                     oCount = 0;
149
150                     for (int i = 0; i < connections; i++) {
151                         if (board[r][c + i] == Cell.X)
152                             xCount++;
153                         if (board[r][c + i] == Cell.O)
154                             oCount++;
155                     }
156
157                     if (xCount == connections)
158                         return GameStatus.X_WON;
159                     else if (oCount == connections)
160                         return GameStatus.O_WON;
161                 }
162
163                 // vertical win
164                 if (r <= height - connections) {
165                     xCount = 0;
166                     oCount = 0;
167
168                     for (int i = 0; i < connections; i++) {
```

```
169                         if (board[r + i][c] == Cell.X)
170                             xCount++;
171                         if (board[r + i][c] == Cell.O)
172                             oCount++;
173                     }
174
175                     if (xCount == connections)
176                         return GameStatus.X_WON;
177                     else if (oCount == connections)
178                         return GameStatus.O_WON;
179                 }
180
181                 // major diagonal win
182                 if (r <= height - connections && c <= width - connections) {
183                     xCount = 0;
184                     oCount = 0;
185
186                     for (int i = 0; i < connections; i++) {
187                         if (board[r + i][c + i] == Cell.X)
188                             xCount++;
189                         if (board[r + i][c + i] == Cell.O)
190                             oCount++;
191                     }
192
193                     if (xCount == connections)
194                         return GameStatus.X_WON;
195                     else if (oCount == connections)
196                         return GameStatus.O_WON;
197                 }
198
199                 // minor diagonal win
200                 if (r >= connections - 1 && c <= width - connections) {
201                     xCount = 0;
202                     oCount = 0;
203
204                     for (int i = 0; i < connections; i++) {
205                         if (board[r - i][c + i] == Cell.X)
206                             xCount++;
207                         if (board[r - i][c + i] == Cell.O)
208                             oCount++;
209                     }
210
211                     if (xCount == connections)
212                         return GameStatus.X_WON;
213                     else if (oCount == connections)
214                         return GameStatus.O_WON;
215                 }
216             }
217
218         if (numSelected >= height * width)
219             return GameStatus.CATS;
220
221         return GameStatus.IN_PROGRESS;
222     }
223
224     /*****************************************************************
```

```
225      * This function returns the current status of the game. It will
226      * either be IN_PROGRESS, X_WON, O_WON, OR CATS
227      * @return the current status of the game
228      **********************************************************************/
229     public GameStatus getGameStatus() {
230         return isWinner();
231     }
232 
233     /**********************************************************************
234      * This function resets the board. It resets the arrayLIst backup,
235      * clears the board, clears the number of X's and O's, and lets the AI
236      * go if it was desired to go first
237      **********************************************************************/
238     public void reset() {
239         for (int i = backup.size(); i > 0; i--){
240             backup.remove(i - 1);
241         }
242 
243         for (int r = 0; r < height; r++)
244             for (int c = 0; c < width; c++)
245                 board[r][c] = Cell.EMPTY;
246 
247         numSelected = 0;
248         if (starter == Cell.O)
249             AI();
250 
251         turn = starter;
252     }
253 
254     /**********************************************************************
255      * This function is the AI that plays against a user. Its first goal is
256      * to win. If it cannot win, it blocks X from winning. It will be placed
257      * randomly otherwise
258      **********************************************************************/
259     public void AI(){
260         numSelected++;
261         turn = Cell.X;
262 
263         //try to win
264         for (int r = 0; r < height; r++){
265             for (int c = 0; c < width; c++){
266                 if (board[r][c] == Cell.EMPTY){
267                     board[r][c] = Cell.O;
268                     if (isWinner() != GameStatus.O_WON)
269                         board[r][c] = Cell.EMPTY;
270                     else {
271                         backup.add(new Point(r, c));
272                         return;
273                     }
274                 }
275             }
276         }
277 
278         //try to block
279         for (int r = 0; r < height; r++){
280             for (int c = 0; c < width; c++){
```

```
281                 if (board[r][c] == Cell.EMPTY){
282                     board[r][c] = Cell.X;
283                     if (isWinner() != GameStatus.X_WON)
284                         board[r][c] = Cell.EMPTY;
285                     else {
286                         board[r][c] = Cell.O;
287                         backup.add(new Point(r, c));
288                         return;
289                     }
290                 }
291             }
292         }
293 
294         //otherwise, random
295         Random rand = new Random();
296         int randR;
297         int randC;
298         do {
299             randR = rand.nextInt(height);
300             randC = rand.nextInt(width);
301         } while (board[randR][randC] != Cell.EMPTY);
302         backup.add(new Point(randR, randC));
303         board[randR][randC] = Cell.O;
304     }
305 }
```