CIS 163 – Computer Science II
Project 2: Tic Tac Toe

89/100

Student Name Aaron Teague	
Due Date 10/1	

Graded Item	Points	Points Secured and Comments
• Javadoc Comments and Coding Style/Technique • (http://www.cis.gvsu.edu/studentsupport/javaguide) • Code Indentation (auto format source code in IDE) • Naming Conventions (see Java style guide) • Proper access modifiers for fields and methods • Use of helper (private) methods • Using good variable names • Header/class comments • Every method uses @param and @return • Every method uses a /********** separator • Overall layout, readability, No text wrap • Using /** … / for each Instance variable • Has many inner "inner" comments	10	CHECK DIAGONAL NEEDS WORK -1
Steps 1 – 4.	5	
TicTacToePanel class (Step 5)	25	
TicTacToeGame class (Step 6)	30	
Step 7: Additional functionality • User enters the board size • Who starts first • Labels to display number of wins and loses	 5 5 5	
Step 8: Additional functionality • Undo and Load/Save	 15	-8
Total	**100**	

Additional Comments:

7 CANCEL BUTTON (-1)
OPEN / SAVE (-1)

NOT PASSED VIA MAIN

NOT RUNNING ON EOS

```java
/**********************************************************************
 *Creates a Super Tic Tac Toe game.
 *
 *Author: Aaron Teague
 **********************************************************************/
package package1;

import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;

public class SuperTicTacToe {

    /**********************************************************************
     * Creates the board.
     **********************************************************************/
    public static void main(String[] args) {
        try {

        int size;
        String input;
        String inputTwo;
        int player = 0;

        try {
        //Gets the size of the game from the user
        input = JOptionPane.showInputDialog(null, "Enter in the size "
                + "for the Tic Tac Toe board: ");
        size = Integer.parseInt(input);

        //Checks if the board is a valid
        while((size < 3)  || (size > 15)){
        input = JOptionPane.showInputDialog(null, "Invalid number" +
        " please try again, or press cancel.");
            size = Integer.parseInt(input);
        }


        }

        catch(NumberFormatException e){
        input = JOptionPane.showInputDialog(null, "Please enter a" +
        " number, or press cancel.");
            size = Integer.parseInt(input);
        }
            //Asks player who is playing first
            inputTwo = JOptionPane.showInputDialog(null, "Who Starts"
                    + " first? X or O");

            //Displays if X is first
            if(inputTwo.equals("X")){
                player = 1;
                JOptionPane.showMessageDialog(null,"X will start.");
```

```
56              }
57
58              //Displays if O is first
59              else if(inputTwo.equals("O")){
60                  player = 2;
61                  JOptionPane.showMessageDialog(null,"O will start.");
62              }
63
64              //Displays if X or O isn't selected
65              else
66              {
67              JOptionPane.showMessageDialog(null,"An invalid value " +
68              "has been entered, X will start.");
69                  player = 1;
70              }
71
72
73
74
75          JFrame frame = new JFrame("Super TicTacToe");
76          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
77
78          SuperTicTacToePanel panel =
79                  new SuperTicTacToePanel(size, player);
80
81          frame.getContentPane().add(panel);
82
83          frame.pack();
84          frame.setSize(600, 600);
85          frame.setVisible(true);
86          }
87
88          //If every thing fails this message will display
89          catch(Exception e){
90              JOptionPane.showMessageDialog(null,"Super Tic Tac Toe" +
91          " has unexpectedly stopped working.");
92          }
93
94      }
95
96  }
97
```

```java
package package1;

import javax.swing.*;

import java.awt.*;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.util.ArrayList;
import java.util.Scanner;

public class SuperTicTacToeGame {

    /**The status of each game button **/
    private Cell[][] board;

    /**Array list of player turns **/
    private ArrayList<Integer> playerList;

    /**The status of each game **/
    private GameStatus status;

    /**Stores whose turn it is **/
    private int player;

    /**Stores the size of the board **/
    private int size;

    /**Total number of X wins **/
    private int xWin;

    /**Total number of O wins **/
    private int oWin;

    /**Stores the player that started the game **/
    private int startPlayer;

    /**Array list that stores each move made **/
    private ArrayList<Point> undoList;

    /*****************************************************************
     * Constructor that creates the game.
     * @param int x
     * @param int player
     * @return none
     *****************************************************************/
    public SuperTicTacToeGame(Integer x, int player){
        undoList = new ArrayList();
        playerList = new ArrayList();
        size = x;
        this.player = player;
        startPlayer = player;
        //The game starts with a status of progress
        status = GameStatus.IN_PROGRESS;
```

```
56          board = new Cell[size][size];
57          //All cells start with the value of empty
58          for(int row = 0; row < size; row++)
59              for(int col = 0; col < size; col++)
60                  board[row][col] = Cell.EMPTY;
61      }
62
63      /*****************************************************************
64       * Returns the size of the board.
65       * @param none
66       * @return size
67       *****************************************************************/
68      public int getSize(){
69
70          return size;
71      }
72
73      /*****************************************************************
74       * Changes the status according to the button pushed.
75       * @param int row
76       * @param int col
77       * @return none
78       *****************************************************************/
79      public void select(int row, int col){
80
81          //Adds the current player to the playerList
82          playerList.add(0, this.player);
83          //Checks which cell to change according to the button pushed
84          if(board[row][col] == Cell.EMPTY)
85          {
86              if(player == 1){
87                  board[row][col] = Cell.X;
88                  this.nextPlayer();
89                  return;
90              }
91
92              if(player == 2){
93                  board[row][col] = Cell.O;
94                  this.nextPlayer();
95                  return;
96              }
97          }
98
99          //Informs the player that they are clicking an occupied button.
100         if(board[row][col] != Cell.EMPTY){
101             JOptionPane.showMessageDialog(null, "Sorry " +
102                     "please try again.");
103             return;
104         }
105     }
106
107     /*****************************************************************
108      * Returns the board
109      * @param none
110      * @return board
```

```
111      ********************************************************************/
112     public Cell[][] getBoard(){
113
114         return board;
115     }
116
117     /********************************************************************
118      *Resets the game to at the starting point
119      *@param none
120      *@return none
121      ********************************************************************/
122     public void reset(){
123     //Resets the player to who started
124         player = startPlayer;
125         //All cells are set to empty
126         for(int row = 0; row < size; row++)
127             for(int col = 0; col < size; col++)
128                 board[row][col] = Cell.EMPTY;
129         //The status is change to in progress
130         status = GameStatus.IN_PROGRESS;
131     }
132
133     /********************************************************************
134      * Return the current player
135      * @param none
136      * @return the current player
137      ********************************************************************/
138     public int getCurrentPlayer(){
139
140         return player;
141     }
142
143     /********************************************************************
144      * Switches to the next the player
145      * @param none
146      * @return returns the current player
147      ********************************************************************/
148     public int nextPlayer(){
149
150         if(player == 1)
151             player = 2;
152         else
153             player = 1;
154
155         return player;
156
157     }
158
159     /********************************************************************
160      * Sets the game status
161      * @param none
162      * @return none
163      ********************************************************************/
164     public void setGameStatus(){
165
```

```
166         //Checks if there is a winner across
167         this.checkAcross();
168         //Checks if there is a winner down
169         this.checkDown();
170         //Checks if there is a winner diagonally
171         this.checkDiagonal();
172         //Checks if the game is full
173         this.checkFull();
174         //Check if there is a winner wrap
175         if(size > 3)
176         this.checkWrap();
177     }
178
179     /*************************************************************
180      * Looks for a winner from a wrap
181      * @param none
182      * @return none
183      *************************************************************/
184     private void checkWrap(){
185             for(int row = 0; row < board.length;  row++){
186             for(int column = 0; column < board[row].length; column++){
187                 if(row == 0){
188                     if((board[0][column] == Cell.X) &&
189                        (board[board.length-1][column] == Cell.X) &&
190                        (board[board.length-2][column] == Cell.X)){
191                          xWin++;
192                          oWin--;
193                          this.status = GameStatus.X_WON;
194                          return;
195                     }
196
197                     if((board[0][column] == Cell.O) &&
198                        (board[board.length-1][column] == Cell.O) &&
199                        (board[board.length-2][column] == Cell.O)){
200                          xWin--;
201                          oWin++;
202                          this.status = GameStatus.O_WON;
203                          return;
204                     }
205
206                     if((board[0][column] == Cell.X) &&
207                        (board[1][column] == Cell.X) &&
208                        (board[board.length-1][column] == Cell.X)){
209                          xWin++;
210                          oWin--;
211                          this.status = GameStatus.X_WON;
212                          return;
213                     }
214
215                     if((board[0][column] == Cell.O) &&
216                        (board[1][column] == Cell.O) &&
217                        (board[board.length-1][column] == Cell.O)){
218                           xWin--;
219                           oWin++;
220                           this.status = GameStatus.O_WON;
```

```
221                         return;
222                      }
223
224
225                   }
226
227
228               if((board[row][0] == Cell.X) &&
229                  (board[row][1] == Cell.X) &&
230                  (board[row][board[row].length-1] == Cell.X)){
231                     xWin++;
232                     oWin--;
233                     this.status = GameStatus.X_WON;
234                     return;
235               }
236
237               if((board[row][0] == Cell.X) &&
238                  (board[row][board[row].length-2] == Cell.X) &&
239                  (board[row][board[row].length-1] == Cell.X)){
240                    xWin++;
241                    oWin--;
242                    this.status = GameStatus.X_WON;
243                    return;
244               }
245
246               if((board[row][0] == Cell.O) &&
247                  (board[row][1] == Cell.O) &&
248                  (board[row][board[row].length-1] == Cell.O)){
249                       xWin--;
250                       oWin++;
251                       this.status = GameStatus.O_WON;
252                       return;
253               }
254
255               if((board[row][0] == Cell.O) &&
256                  (board[row][board[row].length-2] == Cell.O) &&
257                  (board[row][board[row].length-1] == Cell.O)){
258                       xWin--;
259                       oWin++;
260                       this.status = GameStatus.O_WON;
261                       return;
262               }
263
264
265           }
266       }
267   }
268
269
270
271   /*****************************************************************
272    * Looks for a winner across increments and decrements the number
273    * of wins for each player respectively
274    * @param none
275    * @return none
```

```
276      ****************************************************************/
277     private void checkAcross(){
278
279         //Holds the total of X spaces
280         int acrossX = 0;
281         //Holds the total of O spaces
282         int acrossO = 0;
283
284         //Loops through the board if 3 X or O spaces are found across
285         //a winner is declared.
286         for(int row = 0; row < board.length;  row++){
287
288             acrossX = 0;
289             acrossO = 0;
290
291             for(int column = 0; column < board[row].length; column++){
292
293                 if(board[row][column]== Cell.X)
294                     acrossX++;
295                 else
296                     acrossX = 0;
297
298                 if(acrossX==3){
299                     xWin++;
300                     oWin--;
301                     this.status = GameStatus.X_WON;
302                     return;
303                 }
304
305                 if(board[row][column]== Cell.O)
306                     acrossO++;
307                 else
308                     acrossO = 0;
309
310                 if(acrossO==3){
311                     oWin++;
312                     xWin--;
313                     this.status = GameStatus.O_WON;
314                     return;
315                 }
316             }
317         }
318
319     }
320
321     /****************************************************************
322      * Check if there is three matching spaces down. Increments and
323      * decrements the number wins for each player respectively
324      * @param none
325      * @return none
326      ****************************************************************/
327     private void checkDown(){
328
329         int downX = 0;
330         int downO = 0;
```

```
331         int rowDown = 0;
332 
333         //Loops down each column and checks if three places have
334         //the same value
335         for(int columnDown = 0; columnDown <
336             board[rowDown].length; columnDown++){
337 
338             while(rowDown < board.length){
339                 if(board[rowDown][columnDown] == Cell.X)
340                     downX++;
341                 else
342                     downX = 0;
343 
344                 if(downX==3){
345                     xWin++;
346                     oWin--;
347                     this.status = GameStatus.X_WON;
348                     return;
349                 }
350 
351                 if(board[rowDown][columnDown] == Cell.O)
352                     downO++;
353 
354                 else
355                     downO = 0;
356 
357                 if(downO==3){
358                     oWin++;
359                     xWin--;
360                     this.status = GameStatus.O_WON;
361                     return;
362                 }
363 
364                 rowDown++;
365             }
366 
367             rowDown = 0;
368             downX = 0;
369             downO = 0;
370 
371         }
372 
373     }
374 
375     /*****************************************************************
376      * Check if there is three matching spaces down diagonally.
377      * Increments and decrements the number wins for each player
378      * respectively.
379      * @param none
380      * @return none
381      *****************************************************************/
382     private void checkDiagonal(){
383 
384         for(int row = 0; row < board.length;  row++){
385             for(int column = 0; column < board[row].length; column++){
```

```
386
387             if(((row + 1) < board.length) &&
388                (((column + 1) < board[row].length) &&
389                ((column - 1)>=0) && (row - 1)>=0)){
390
391                     if(board[row][column] == Cell.X)
392                     if(board[row+1][column+1] == Cell.X)
393                     if(board[row-1][column-1] == Cell.X){
394                         xWin++;
395                         oWin--;
396                         this.status = GameStatus.X_WON;
397                         return;
398                     }
399
400
401                     if(board[row][column] == Cell.O)
402                     if(board[row+1][column+1] == Cell.O)
403                     if(board[row-1][column-1] == Cell.O){
404                             oWin++;
405                             xWin--;
406                             this.status = GameStatus.O_WON;
407                             return;
408                     }
409
410                     if(board[row][column] == Cell.X)
411                     if(board[row+1][column-1] == Cell.X)
412                     if(board[row-1][column+1] == Cell.X){
413                         xWin++;
414                         oWin--;
415                         this.status = GameStatus.X_WON;
416                         return;
417                     }
418
419                     if(board[row][column] == Cell.O)
420                     if(board[row+1][column-1] == Cell.O)
421                     if(board[row-1][column+1] == Cell.O){
422                             oWin++;
423                             xWin--;
424                             this.status = GameStatus.O_WON;
425                             return;
426                     }
427
428
429                     if(((row + 2) < board.length) &&
430                       ((column + 2) < board[row].length) &&
431                       ((column - 2)>=0) && ((row - 2)>=0)){
432
433                         if(board[row][column] == Cell.X)
434                         if(board[row+1][column+1] == Cell.X)
435                         if(board[row+2][column+2] == Cell.X){
436                             xWin++;
437                             oWin--;
438                             this.status = GameStatus.X_WON;
439                             return;
440                         }
```

```
441
442                         if(board[row][column] == Cell.O)
443                         if(board[row+1][column+1] == Cell.O)
444                         if(board[row+2][column+2] == Cell.O){
445                             oWin++;
446                             xWin--;
447                             this.status = GameStatus.O_WON;
448                             return;
449                         }
450
451                         if(board[row][column] == Cell.X)
452                         if(board[row+1][column-1] == Cell.X)
453                         if(board[row+2][column-2] == Cell.X){
454                             xWin++;
455                             oWin--;
456                             this.status = GameStatus.X_WON;
457                             return;
458                         }
459
460                         if(board[row][column] == Cell.O)
461                         if(board[row+1][column-1] == Cell.O)
462                         if(board[row+2][column-2] == Cell.O){
463                             oWin++;
464                             xWin--;
465                             this.status = GameStatus.O_WON;
466                             return;
467                         }
468
469                         if(board[row][column] == Cell.X)
470                         if(board[row-1][column+1] == Cell.X)
471                         if(board[row-2][column+2] == Cell.X){
472                             xWin++;
473                             oWin--;
474                             this.status = GameStatus.X_WON;
475                             return;
476
477                         }
478
479                         if(board[row][column] == Cell.O)
480                         if(board[row-1][column+1] == Cell.O)
481                         if(board[row-2][column+2] == Cell.O){
482                             oWin++;
483                             xWin--;
484                             this.status = GameStatus.O_WON;
485                             return;
486
487                         }
488
489                     }
490                 }
491             }
492         }
493
494     }
495
```

```
496
497     /*****************************************************************
498      * Checks if the board is full
499      * @param none
500      * @return none
501      *****************************************************************/
502     private void checkFull(){
503
504         //Loops through the board to find empty spaces
505         for(int row = 0; row < board.length;  row++)
506             for(int column = 0; column < board[row].length; column++)
507                 if(board[row][column] == Cell.EMPTY)
508                   return;
509
510         this.status = GameStatus.CATS;
511
512     }
513
514     /*****************************************************************
515      * Returns the game status
516      * @param none
517      * @return The game status
518      *****************************************************************/
519     public GameStatus getGameStatus(){
520
521         return status;
522     }
523
524     /*****************************************************************
525      * Returns the number of X wins
526      * @param none
527      * @return none
528      *****************************************************************/
529     public int getXwin(){
530
531         return xWin;
532     }
533
534     /*****************************************************************
535      * Returns the number of O wins
536      * @param none
537      * @return none
538      *****************************************************************/
539     public int getOwin(){
540
541         return oWin;
542     }
543
544     /*****************************************************************
545      * Adds player's move to the undoList
546      * @param Point x
547      * @return none
548      *****************************************************************/
549     public void addundoList(Point x){
550
```

```
551         undoList.add(0, x);
552     }
553 
554     /**********************************************************************
555      * Sets the cell according to the Point from the first element in
556      * the undoList. The first element in the UndoList and the next
557      * player is set.
558      * @param Point x
559      * @return oWin, number of O wins
560      *********************************************************************/
561     public void undo(){
562 
563     //Sets the cell according to the Point in undoList
564     board[(int)undoList.get(0).getX()][(int)undoList.get(0).getY()] =
565             Cell.EMPTY;
566     //The first element is removed from the undoList
567     undoList.remove(0);
568     //nexPlayer is called
569     this.nextPlayer();
570     }
571 
572     /**********************************************************************
573      * Saves the player and the player's move to a file of the player's
574      * choice
575      * @param none
576      * @return none
577      *********************************************************************/
578     public void save(){
579 
580         String line;
581 
582         JFileChooser chooser = new JFileChooser();
583         chooser.showOpenDialog(null);
584 
585             try
586             {
587                 File file = chooser.getSelectedFile();
588                 FileWriter filewriter = new FileWriter(file);
589 
590                 for(int i = 0 ; i < this.undoList.size() ; i++){
591                 //Creates the line from the playerList and undoList
592                 line = String.valueOf(this.playerList.get(i)) + "," +
593                 String.valueOf((int)undoList.get(i).getX()) +
594                 "," + String.valueOf((int)undoList.get(i).getY()+"\n");
595                     //Writes the line to the file
596                     filewriter.write(line);
597 
598                     this.nextPlayer();
599                 }
600 
601                 filewriter.close();
602             }
603 
604             catch(Exception ex)
605             {
```

```
606                 ex.printStackTrace();
607             }
608
609     }
610
611     /*****************************************************************
612      * Loads the the save game.
613      * @param none
614      * @return none
615      *****************************************************************/
616     public void load() throws FileNotFoundException
617     {
618         //String array holds the string in three different parts
619         String[] parts;
620         JFileChooser chooser = new JFileChooser();
621         chooser.showOpenDialog(null);
622
623
624             File file = chooser.getSelectedFile();
625             Scanner scan = new Scanner(file);
626
627             String info = "";
628             while(scan.hasNext())
629             {
630                 info = scan.nextLine();
631                 //Splits the line that is read by a comma
632                 parts = info.split(",");
633                 //Sets the player
634                 this.player = Integer.parseInt(parts[0]);
635                 //Selects the space
636                 this.select(Integer.parseInt(parts[1]),
637                         Integer.parseInt(parts[2]));
638
639             }
640
641             this.nextPlayer();
642     }
643 }
644
645
```

```java
package package1;
import java.awt.*;
import java.awt.event.*;
import java.io.FileNotFoundException;
import java.util.*;

import javax.swing.*;

public class SuperTicTacToePanel extends JPanel
{
    /**The quit button **/
    private JButton quitButton;
    /**The reset button **/
    private JButton resetButton;
    /**The undo button **/
    private JButton undoButton;
    /**The save button **/
    private JButton saveButton;
    /**The load button **/
    private JButton loadButton;
    /**The top pane **/
    private JPanel top;
    /**The middle pane **/
    private JPanel middle;
    /**Array of buttons that make up board **/
    private JButton[][] board;
    /**Corresponding cell values for each button **/
    private Cell[][] iboard;
    /**The X graphic **/
    private ImageIcon xIcon;
    /**The O graphic **/
    private ImageIcon oIcon;
    /**The empty graphic **/
    private ImageIcon emptyIcon;
    /**The Tic Tac Toe game **/
    private SuperTicTacToeGame game;
    /**Label X wins **/
    private JLabel xLabel;
    /**Label O wins **/
    private JLabel oLabel;
    /**The point, coordinates of a button **/
    private Point point;


    public SuperTicTacToePanel(Integer size, int player)
    {

        ButtonListener listener = new ButtonListener();

        game = new SuperTicTacToeGame(size, player);

        top = new JPanel();
        middle = new JPanel();

        xIcon = new ImageIcon("x.png");
```

```
56          oIcon  = new ImageIcon("o.jpg");
57          emptyIcon = new ImageIcon("empty.png");
58
59          quitButton = new JButton("Quit");
60          quitButton.addActionListener(listener);
61          top.add(quitButton);
62
63          resetButton = new JButton("Reset");
64          resetButton.addActionListener(listener);
65          top.add(resetButton);
66
67          undoButton = new JButton("Undo");
68          undoButton.addActionListener(listener);
69          top.add(undoButton);
70
71          saveButton = new JButton("Save");
72          saveButton.addActionListener(listener);
73          top.add(saveButton);
74
75          loadButton = new JButton("Load");
76          loadButton.addActionListener(listener);
77          top.add(loadButton);
78
79          xLabel = new JLabel("X Wins: ");
80          top.add(xLabel);
81          oLabel = new JLabel("O Wins: ");
82          top.add(oLabel);
83
84          middle.setLayout(new GridLayout(size,size));
85
86          //Creates the board buttons
87          board = new JButton[size][size];
88          for(int row = 0; row < size; row++)
89              for(int col = 0; col < size; col++)
90              {
91                  board[row][col] = new JButton ("", emptyIcon);
92                  board[row][col].addActionListener(listener);
93                  middle.add(board[row][col]);
94              }
95
96          //Creats the cells for the board
97          iboard = new Cell[size][size];
98          for(int row = 0; row < size; row++)
99              for(int col = 0; col < size; col++)
100                 iboard[row][col] = Cell.EMPTY;
101
102
103         setLayout(new BorderLayout ());
104         add(top, BorderLayout.NORTH);
105         add(middle, BorderLayout.CENTER);
106
107     }
108
109     /*************************************************************
110      * Displays the board
```

```
111      * @param none
112      * @return none
113      *****************************************************************/
114     private void displayBoard(){
115
116         iboard = game.getBoard();
117
118         for(int row = 0; row < game.getSize(); row++)
119             for(int col = 0; col < game.getSize(); col++){
120                 if(iboard[row][col] == Cell.O)
121                     board[row][col].setIcon(oIcon);
122                 if(iboard[row][col] == Cell.X)
123                     board[row][col].setIcon(xIcon);
124                 if(iboard[row][col] == Cell.EMPTY)
125                     board[row][col].setIcon(emptyIcon);
126             }
127     }
128
129
130
131     private class ButtonListener implements ActionListener
132     {
133
134         /*************************************************************
135          * Perform the correct action
136          *************************************************************/
137         public void actionPerformed(ActionEvent event) {
138
139             JComponent comp = (JComponent) event.getSource();
140             //if the quit button is clicked the game exits
141             if(comp == quitButton)
142                 System.exit(1);
143             //if the reset button is clicked the game resets
144             if(comp == resetButton)
145                 game.reset();
146             //if the undo button is clicked the move is undone
147             if(comp == undoButton)
148                 game.undo();
149             //if the save button is clicked the game is saved
150             if(comp == saveButton)
151                 game.save();
152             //if the load button is clicked the game is loaded
153             if(comp == loadButton)
154                 try {
155                     game.load();
156                 } catch (FileNotFoundException e) {
157                     // TODO Auto-generated catch block
158                     e.printStackTrace();
159                 }
160
161             //When a board is selected the game selects the button
162             for(int row = 0; row < game.getSize(); row++)
163                 for(int col = 0; col < game.getSize(); col++){
164                     if(board[row][col] == comp){
165                        game.select(row, col);
```

```
166                     //a point is created from the row and column
167                     //that is selected
168                     point = new Point(row, col);
169                     //point is added to the game undo list
170                     game.addundoList(point);
171                 }
172 
173             }
174           //the board is displayed
175           displayBoard();
176           //the game's status is set
177           game.setGameStatus();
178 
179           //if x has won a pane is displayed to show the win
180           if(game.getGameStatus() == GameStatus.O_WON){
181 
182               JOptionPane.showMessageDialog(null, "O won and X" +
183               " lost!\n The game will reset");
184               game.reset();
185               displayBoard();
186 
187           }
188 
189           //if o has won a pane is displayed to show the win
190           if(game.getGameStatus() == GameStatus.X_WON){
191 
192               JOptionPane.showMessageDialog(null, "X won and O" +
193               " lost!\n The game will reset");
194               game.reset();
195               displayBoard();
196 
197           }
198 
199           //if there are no empty spaces the game is declared a draw
200           if(game.getGameStatus() == GameStatus.CATS){
201 
202               JOptionPane.showMessageDialog(null, "The Game has " +
203               "ended in a draw. And will reset.");
204               game.reset();
205               displayBoard();
206 
207           }
208 
209           //displays the number of X wins
210           xLabel.setText("X Wins: " + game.getXwin());
211           //displays the number of O wins
212           oLabel.setText("O Wins: " + game.getOwin());
213 
214 
215       }
216   }
217 }
```

GameStatus.java

```
1 package package1;
2
3 /**
4  * Creates the enum gStatus type, possible values {X_WON, O_WON, CATS,
5  * IN_PROGRESS
6  */
7
8 public enum GameStatus
9
10     {X_WON, O_WON, CATS, IN_PROGRESS}
11
12
13
```

```java
package package1;

/**
 * Creates the enum cellStatus type, possible values X,O,EMPTY
 */

public enum Cell
    {X, O, EMPTY}


```