

Name: Dan Cassidy

Class: CSCI-C 490, Mobile Application Development

Assignment: Homework 7 Part 3

Date: 2015-07-27

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="dancassidy.tictactoe" >
4
5      <application
6          android:allowBackup="true"
7          android:icon="@mipmap/ic_launcher"
8          android:label="@string/app_name"
9          android:theme="@style/AppTheme" >
10         <activity
11             android:name=".MainActivity"
12             android:label="@string/app_name"
13             android:screenOrientation="portrait" >
14             <intent-filter>
15                 <action android:name="android.intent.action.MAIN" />
16                 <category android:name="android.intent.category.LAUNCHER" />
17             </intent-filter>
18         </activity>
19     </application>
20
21 </manifest>
22
```

```
1  /*-----*/
2  * Author:      Dan Cassidy
3  * Date:        2015-07-27
4  * Assignment:  HW7-3
5  * Source File: MainActivity.java
6  * Language:    Java
7  * Course:      CSCI-C 490, Android Programming, MoWe 08:00
8  -----*/
9  package dancassidy.tictactoe;
10
11  import android.app.Activity;
12  import android.os.Bundle;
13  import android.view.View;
14  import android.widget.Button;
15  import android.widget.TextView;
16
17  /**
18   * Main activity class for the TicTacToe game.
19   *
20   * @author Dan Cassidy
21   */
22  public class MainActivity extends Activity {
23      TicTacToe theGame;
24      TextView statusTextView;
25      Button[] board;
26      Button resetButton;
27
28      /**
29       * Main method that runs on application start.
30       *
31       * @param savedInstanceState The saved instance state.
32       */
33      @Override protected void onCreate(Bundle savedInstanceState) {
34          super.onCreate(savedInstanceState);
35          setContentView(R.layout.activity_main);
36
37          theGame = new TicTacToe();
38          statusTextView = (TextView) findViewById(R.id.statusTextView);
39          board = new Button[]{
40              (Button) findViewById(R.id.xoButton1),
41              (Button) findViewById(R.id.xoButton2),
42              (Button) findViewById(R.id.xoButton3),
43              (Button) findViewById(R.id.xoButton4),
44              (Button) findViewById(R.id.xoButton5),
45              (Button) findViewById(R.id.xoButton6),
46              (Button) findViewById(R.id.xoButton7),
47              (Button) findViewById(R.id.xoButton8),
48              (Button) findViewById(R.id.xoButton9)};
49          resetButton = (Button) findViewById(R.id.resetButton);
50
51          // Set anonymous listeners for all the board buttons.
52          for (Button theButton : board)
53              theButton.setOnClickListener(new View.OnClickListener() {
54                  @Override public void onClick(View v) {
55                      int row = Integer.parseInt(v.getTag().toString()) / theGame.getColumns();
56                      int column = Integer.parseInt(v.getTag().toString()) % theGame.getColumns();
57
58                      theGame.playMove(row, column);
59                      ((Button) v).setText(theGame.getSpaceStringID(row, column));
60                      statusTextView.setText(theGame.getStatusStringID());
```

```
61         if (theGame.getStatus() != TicTacToe.Status.IN_PROGRESS)
62             for (Button theButton : board)
63                 theButton.setEnabled(false);
64     }
65 });
66
67 // Set anonymous listener for the reset button.
68 resetButton.setOnClickListener(new View.OnClickListener() {
69     @Override public void onClick(View v) {
70         theGame.reset();
71         MainActivity.this.reset();
72     }
73 });
74 }
75
76 /**
77  * Resets the status text and the board buttons to default.
78  */
79 private void reset() {
80     statusTextView.setText(R.string.status_x_turn);
81     for (Button theButton : board) {
82         theButton.setEnabled(true);
83         theButton.setText(R.string.blank);
84     }
85 }
86 }
87
```

```
1  /*-----*/
2  * Author:      Dan Cassidy
3  * Date:        2015-07-27
4  * Assignment:  HW7-3
5  * Source File: TicTacToe.java
6  * Language:    Java
7  * Course:      CSCI-C 490, Android Programming, MoWe 08:00
8  -----*/
9  package dancassidy.tictactoe;
10
11 /**
12  * Model for the Tic-Tac-Toe game.
13  * <p/>
14  * Can scale to an arbitrary board size and use an arbitrary winning sequence length.
15  *
16  * @author Dan Cassidy
17  */
18 public class TicTacToe {
19     public enum Mark {X, O}
20
21     public enum Status {IN_PROGRESS, X_WIN, O_WIN, DRAW}
22
23     private static final int DEFAULT_NUM_ROWS = 3;
24     private static final int DEFAULT_NUM_COLUMNS = 3;
25     private static final int DEFAULT_WIN_LENGTH = 3;
26
27     private final int NUM_ROWS;
28     private final int NUM_COLUMNS;
29     private final int WIN_LENGTH;
30     private final int MAX_SPACES;
31
32     private Mark[][] board;
33     private Mark turn;
34     private Status status;
35     private int usedSpaces;
36
37     /**
38     * Default constructor. Simply calls the 3-parameter constructor with the default values.
39     */
40     public TicTacToe() {
41         this(DEFAULT_NUM_ROWS, DEFAULT_NUM_COLUMNS, DEFAULT_WIN_LENGTH);
42     }
43
44     /**
45     * 3-parameter constructor. If there is a problem with an argument, the default value is used.
46     *
47     * @param rows      The number of rows on the game board. Should be >= 3.
48     * @param columns    The number of columns on the game board. Should be >= 3.
49     * @param winLength  The length of the sequence required to win. Should be >= 3 and <= the
50     *                  smaller of the number of rows and the number of columns.
51     */
52     public TicTacToe(int rows, int columns, int winLength) {
53         NUM_ROWS = (rows < DEFAULT_NUM_ROWS ? DEFAULT_NUM_ROWS : rows);
54         NUM_COLUMNS = (columns < DEFAULT_NUM_COLUMNS ? DEFAULT_NUM_COLUMNS : columns);
55         MAX_SPACES = NUM_ROWS * NUM_COLUMNS;
56         if (winLength < DEFAULT_WIN_LENGTH ||
57             winLength > (NUM_ROWS > NUM_COLUMNS ? NUM_COLUMNS : NUM_ROWS))
58             WIN_LENGTH = DEFAULT_WIN_LENGTH;
59         else
60             WIN_LENGTH = winLength;
```

```
61     reset();
62 }
63
64 // BEGIN GETTERS AND SETTERS -->
65 public int getColumns() {
66     return NUM_COLUMNS;
67 }
68
69 public int getRows() {
70     return NUM_ROWS;
71 }
72
73 public int getSpaceStringID(int row, int column) {
74     if (!validCoords(row, column) || board[row][column] == null)
75         return R.string.blank;
76     else
77         return (board[row][column] == Mark.X ? R.string.button_x : R.string.button_o);
78 }
79
80 public Status getStatus() {
81     return status;
82 }
83
84 public int getStatusStringID() {
85     switch (status) {
86         case IN_PROGRESS:
87             return (turn == Mark.X ? R.string.status_x_turn : R.string.status_o_turn);
88         case X_WIN:
89             return R.string.status_x_win;
90         case O_WIN:
91             return R.string.status_o_win;
92         case DRAW:
93             return R.string.status_draw;
94         default:
95             return R.string.status_error;
96     }
97 }
98
99 public Mark getTurn() {
100     return turn;
101 }
102
103 public int getWinLength() {
104     return WIN_LENGTH;
105 }
106 // <-- END GETTERS AND SETTERS
107
108 /**
109  * Play a single move at the given game board coordinates.
110  *
111  * @param row    The row where the mark should be placed.
112  * @param column The column where the mark should be placed.
113  */
114 public void playMove(int row, int column) {
115     // If the game had ended, no more moves are accepted.
116     if (status != Status.IN_PROGRESS)
117         return;
118
119     // Verify the row and column values.
120     if (!validCoords(row, column))
```

```
121         return;
122
123         // Verify that the destination is empty.
124         if (board[row][column] == null) {
125             usedSpaces++;
126             board[row][column] = turn;
127
128             // Can't be a winning move until at least (WIN_LENGTH * 2 - 1) spaces have been used.
129             if (usedSpaces >= WIN_LENGTH * 2 - 1)
130                 checkBoard();
131
132             turn = (turn == Mark.X ? Mark.O : Mark.X);
133         }
134     }
135
136     /**
137     * Discards the old game board and creates a new one in its place and sets the turn to X, the
138     * game status to in progress, and the number of used spaces to 0.
139     */
140     public void reset() {
141         board = new Mark[NUM_ROWS][NUM_COLUMNS];
142         turn = Mark.X;
143         status = Status.IN_PROGRESS;
144         usedSpaces = 0;
145     }
146
147     /**
148     * Checks the game board to see if there is a winner or a draw.
149     */
150     private void checkBoard() {
151         // Check for winning sequences.
152         if (checkWin())
153             status = (turn == Mark.X ? Status.X_WIN : Status.O_WIN);
154         // Check for a draw.
155         else if (usedSpaces == MAX_SPACES)
156             status = Status.DRAW;
157     }
158
159     /**
160     * Check for a winning sequence recursively in a given 'direction'. Upon first entry into the
161     * method (<b>numSequential</b> = 1), this method does several things to avoid unnecessary
162     * recursions so it can scale well to an arbitrary board size and winning sequence length.
163     * <ul><li>It verifies that the final row/column aren't going to be outside the bounds of the
164     * board.</li>
165     * <li>It checks the neighboring space in the direction of travel to make sure it matches.</li>
166     * <li>It checks the final destination space (that is, the space that this method will look at
167     * if it reaches the WIN_LENGTH'th depth) to make sure it matches.</li></ul>
168     *
169     * @param row          The row portion of the board space being looked at.
170     * @param column        The column portion of the board space being looked at.
171     * @param rowStepOffset The row offset applied each step.
172     * @param columnStepOffset The column offset applied each step.
173     * @param numSequential The number of sequential marks found thus far.
174     * @return boolean, indicating whether a winning sequence has been found (true) or not (false).
175     */
176     private boolean checkSequence(int row, int column, int rowStepOffset, int columnStepOffset,
177                                  int numSequential) {
178         // Perform initial checks. These are to cut down on the recursion that needs to happen.
179         if (numSequential == 1) {
180             int finalRow = row + rowStepOffset * (WIN_LENGTH - 1);
```

```
181         int finalColumn = column + columnStepOffset * (WIN_LENGTH - 1);
182
183         // Bounds check.
184         if (!validCoords(finalRow, finalColumn))
185             return false;
186
187         // Neighbor check.
188         if (board[row + rowStepOffset][column + columnStepOffset] != turn)
189             return false;
190
191         // Destination check.
192         if (board[finalRow][finalColumn] != turn)
193             return false;
194     }
195
196     // Verify that the sequence continues to match.
197     if (board[row][column] != turn)
198         return false;
199     // Check to see if the sequence is of winning length.
200     else if (numSequential == WIN_LENGTH)
201         return true;
202
203     // Move to the next spot in the sequence.
204     return checkSequence(row + rowStepOffset, column + columnStepOffset, rowStepOffset,
205         columnStepOffset, numSequential + 1);
206 }
207
208 /**
209  * Checks for a winning sequence on the game board. Wrapper for the recursive checkSequence
210  * method.
211  *
212  * @return boolean, indicating whether a winning sequence was found (true) or not (false).
213  */
214 private boolean checkWin() {
215     boolean win = false;
216
217     for (int row = 0; !win && row < NUM_ROWS; row++)
218         for (int column = 0; !win && column < NUM_COLUMNS; column++)
219             // Only need to check for a winning condition if the board space contains a mark
220             // that is the same as the current turn. E.g. - Only check for a winning condition
221             // if it is O's turn and the board contains an 'O' in the current space.
222             if (board[row][column] == turn)
223                 win = checkSequence(row, column, 0, 1, 1) || // Right.
224                     checkSequence(row, column, 1, 0, 1) || // Down.
225                     checkSequence(row, column, 1, 1, 1) || // Diagonal down right.
226                     checkSequence(row, column, -1, 1, 1); // Diagonal up right.
227
228     return win;
229 }
230
231 /**
232  * Checks the given row and column values to make sure they are valid (within bounds) for the
233  * current game board.
234  *
235  * @param row    The row value to check.
236  * @param column The column value to check.
237  * @return boolean, indicating whether the given coordinates are valid (true) or not (false).
238  */
239 private boolean validCoords(int row, int column) {
240     return row >= 0 && row < NUM_ROWS && column >= 0 && column < NUM_COLUMNS;
```



---

```
241         }  
242     }  
243
```

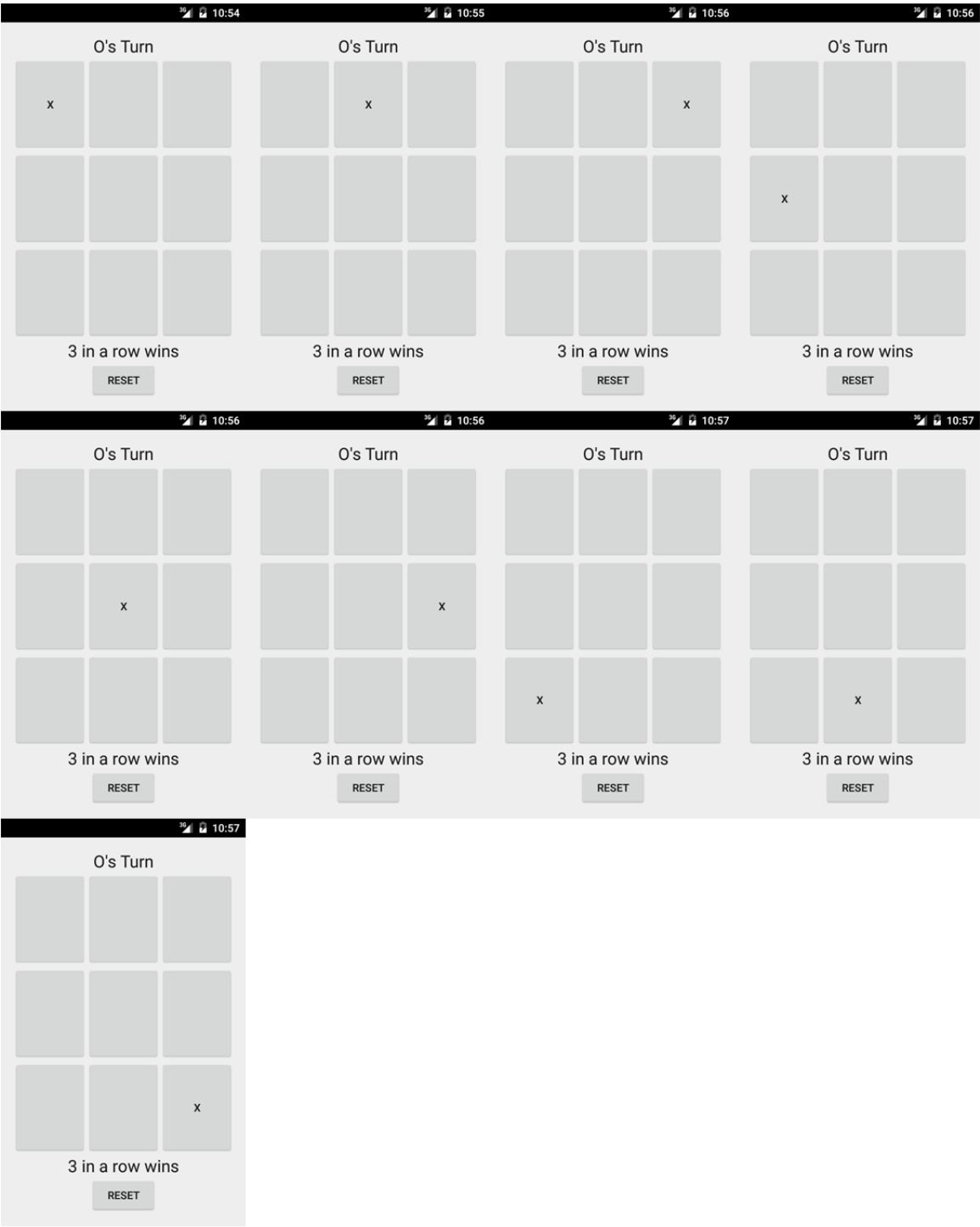
```
1  <RelativeLayout
2      xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6      android:paddingLeft="@dimen/activity_horizontal_margin"
7      android:paddingRight="@dimen/activity_horizontal_margin"
8      android:paddingTop="@dimen/activity_vertical_margin"
9      android:paddingBottom="@dimen/activity_vertical_margin"
10     tools:context=".MainActivity"
11     tools:ignore="NestedWeights,ButtonStyle">
12
13     <TextView
14         android:layout_width="wrap_content"
15         android:layout_height="wrap_content"
16         android:textAppearance="?android:attr/textAppearanceLarge"
17         android:id="@+id/statusTextView"
18         android:layout_alignParentTop="true"
19         android:layout_centerHorizontal="true"
20         android:text="@string/status_x_turn" />
21
22     <LinearLayout
23         android:orientation="vertical"
24         android:layout_width="match_parent"
25         android:layout_height="match_parent"
26         android:layout_below="@+id/statusTextView"
27         android:layout_centerHorizontal="true"
28         android:layout_above="@+id/winningConditionsTextView">
29
30         <LinearLayout
31             android:orientation="horizontal"
32             android:layout_width="match_parent"
33             android:layout_height="match_parent"
34             android:layout_weight="1">
35
36             <Button
37                 android:layout_width="match_parent"
38                 android:layout_height="match_parent"
39                 android:id="@+id/xoButton1"
40                 android:layout_weight="1"
41                 android:tag="0" />
42
43             <Button
44                 android:layout_width="match_parent"
45                 android:layout_height="match_parent"
46                 android:id="@+id/xoButton2"
47                 android:layout_weight="1"
48                 android:tag="1" />
49
50             <Button
51                 android:layout_width="match_parent"
52                 android:layout_height="match_parent"
53                 android:id="@+id/xoButton3"
54                 android:layout_weight="1"
55                 android:tag="2" />
56         </LinearLayout>
57
58     <LinearLayout
59         android:orientation="horizontal"
60         android:layout_width="match_parent"
```

```
61         android:layout_height="match_parent"
62         android:layout_weight="1">
63
64         <Button
65             android:layout_width="match_parent"
66             android:layout_height="match_parent"
67             android:id="@+id/xoButton4"
68             android:layout_weight="1"
69             android:tag="3" />
70
71         <Button
72             android:layout_width="match_parent"
73             android:layout_height="match_parent"
74             android:id="@+id/xoButton5"
75             android:layout_weight="1"
76             android:tag="4" />
77
78         <Button
79             android:layout_width="match_parent"
80             android:layout_height="match_parent"
81             android:id="@+id/xoButton6"
82             android:layout_weight="1"
83             android:tag="5" />
84     </LinearLayout>
85
86     <LinearLayout
87         android:orientation="horizontal"
88         android:layout_width="match_parent"
89         android:layout_height="match_parent"
90         android:layout_weight="1">
91
92         <Button
93             android:layout_width="match_parent"
94             android:layout_height="match_parent"
95             android:id="@+id/xoButton7"
96             android:layout_weight="1"
97             android:tag="6" />
98
99         <Button
100             android:layout_width="match_parent"
101             android:layout_height="match_parent"
102             android:id="@+id/xoButton8"
103             android:layout_weight="1"
104             android:tag="7" />
105
106         <Button
107             android:layout_width="match_parent"
108             android:layout_height="match_parent"
109             android:id="@+id/xoButton9"
110             android:layout_weight="1"
111             android:tag="8" />
112     </LinearLayout>
113 </LinearLayout>
114
115 <TextView
116     android:layout_width="wrap_content"
117     android:layout_height="wrap_content"
118     android:textAppearance="?android:attr/textAppearanceLarge"
119     android:id="@+id/winningConditionsTextView"
120     android:layout_above="@+id/resetButton"
```

```
121         android:layout_centerHorizontal="true"
122         android:text="@string/winning_condition_text" />
123
124     <Button
125         android:layout_width="wrap_content"
126         android:layout_height="wrap_content"
127         android:id="@+id/resetButton"
128         android:layout_alignParentBottom="true"
129         android:layout_centerHorizontal="true"
130         android:text="@string/button_reset" />
131
132 </RelativeLayout>
133
```

```
1  <resources>
2      <string name="app_name">TicTacToe</string>
3
4      <string name="button_reset">Reset</string>
5      <string name="button_x" translatable="false">X</string>
6      <string name="button_o" translatable="false">O</string>
7
8      <string name="winning_condition_text">3 in a row wins</string>
9
10     <string name="status_x_turn">X\'s Turn</string>
11     <string name="status_o_turn">O\'s Turn</string>
12     <string name="status_x_win">X Wins</string>
13     <string name="status_o_win">O Wins</string>
14     <string name="status_draw">Draw</string>
15     <string name="status_error">Error</string>
16
17     <string name="blank" translatable="false"/>
18 </resources>
19
```

Showing that X can appear anywhere.



Showing that O can appear anywhere.

X's Turn

o

x

3 in a row wins

RESET

X's Turn

o

x

3 in a row wins

RESET

X's Turn

o

x

3 in a row wins

RESET

X's Turn

o

x

3 in a row wins

RESET

X's Turn

o

x

3 in a row wins

RESET

X's Turn

o

x

3 in a row wins

RESET

X's Turn

o

x

3 in a row wins

RESET

X's Turn

o

x

3 in a row wins

RESET

X's Turn

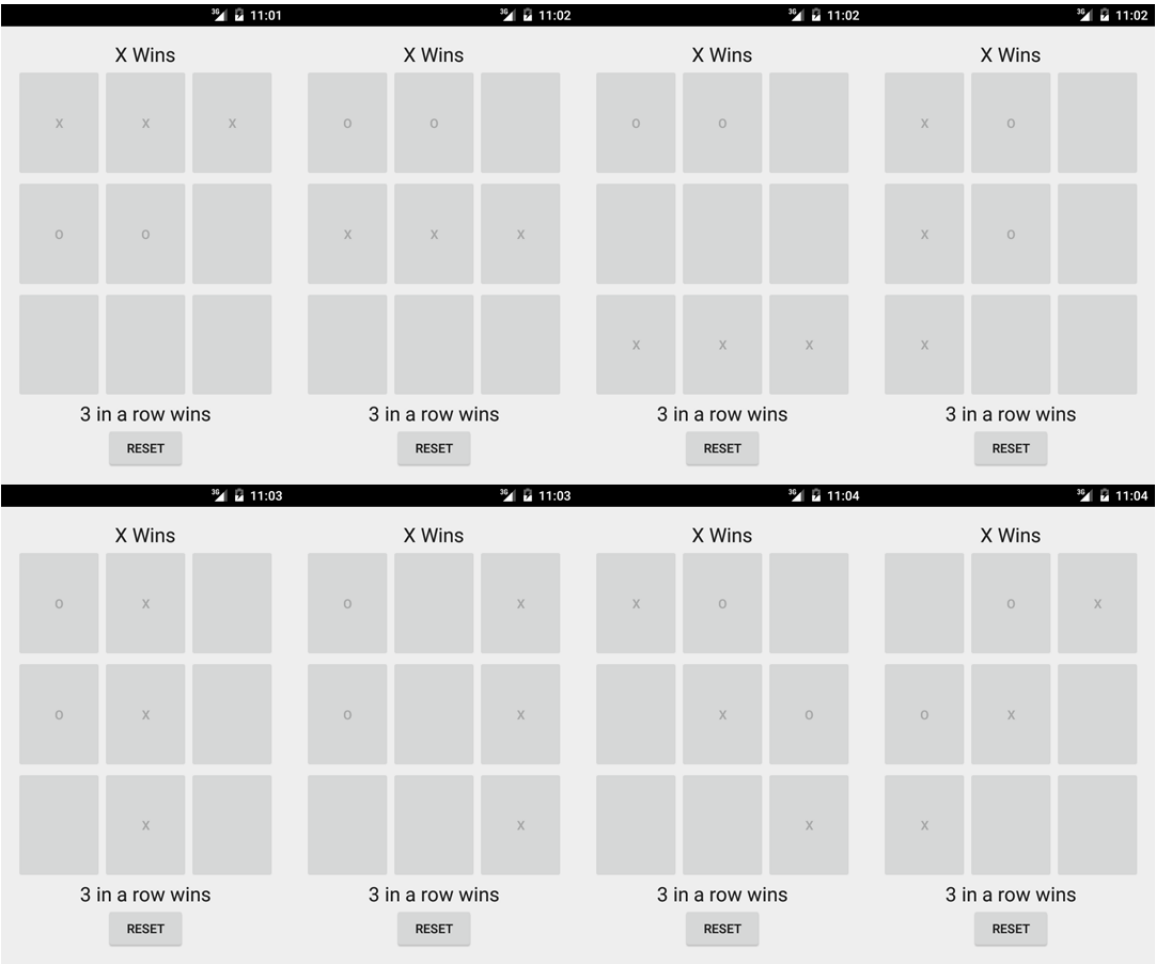
x

o

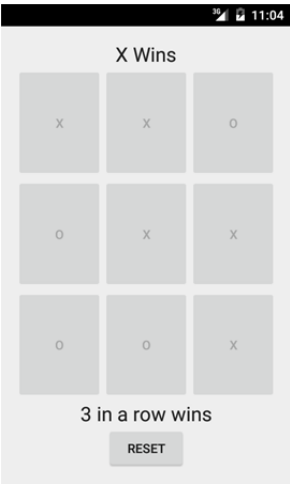
3 in a row wins

RESET

The different X wins.

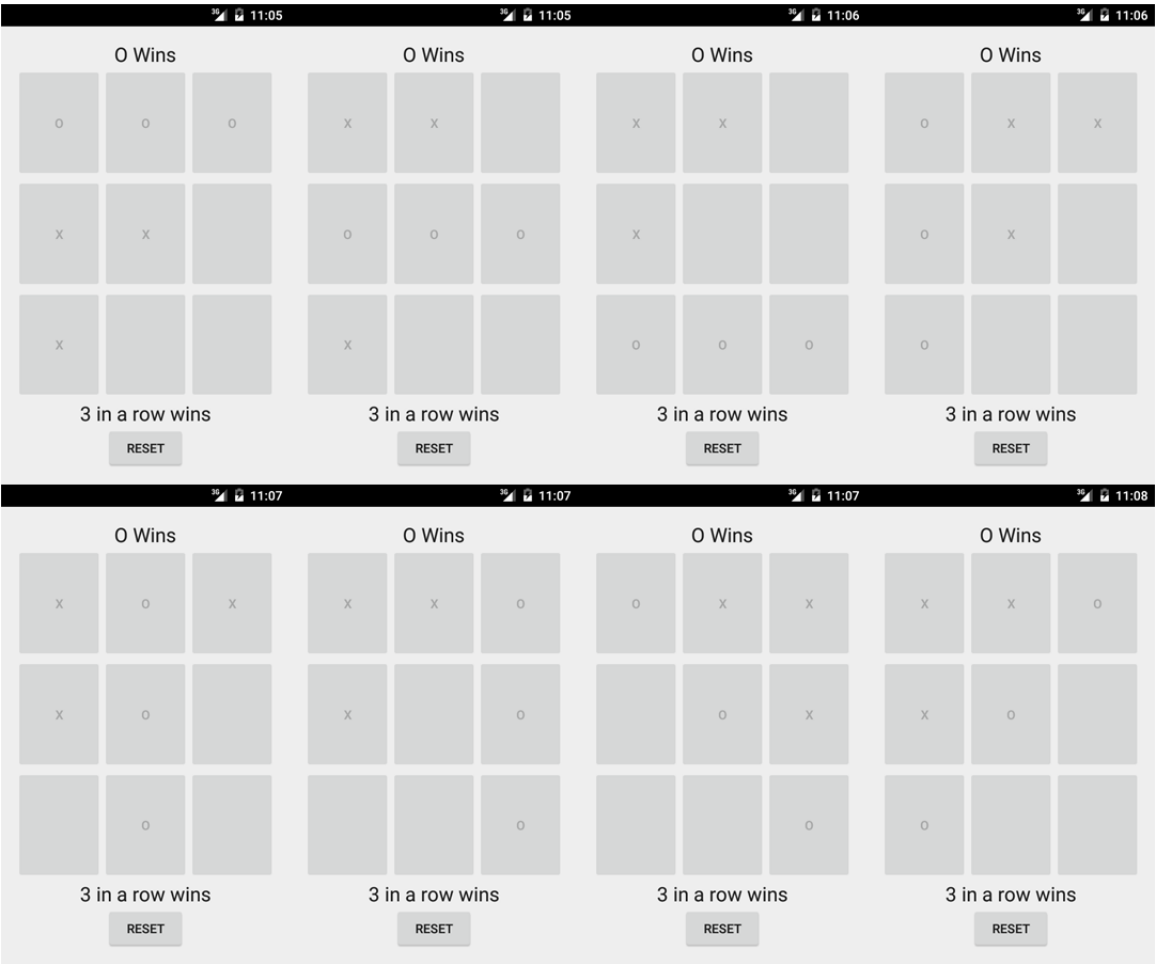


Last move win for X.





The different O wins.



Draw. => Reset.

