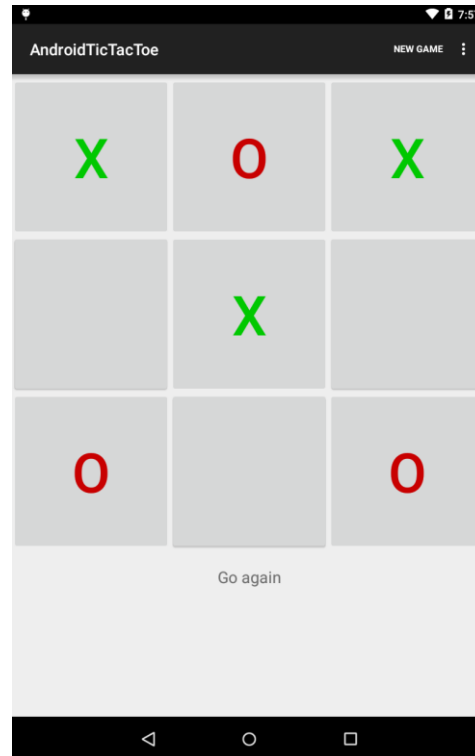


# Android Application Programming

## Tutorial 2: Tic-Tac-Toe App

### Introduction

The goal of this assignment is to create a simple tic-tac-toe game for Android as illustrated on the right. You will use buttons to represent the game board and a text control to display the status of the game at the bottom of the screen. The buttons will have no text on them when the game starts, but when the user clicks on a button, it will display a green X. The computer will then move and turn the appropriate button text to a red O. The text control will indicate whose turn it is and when the game is over.

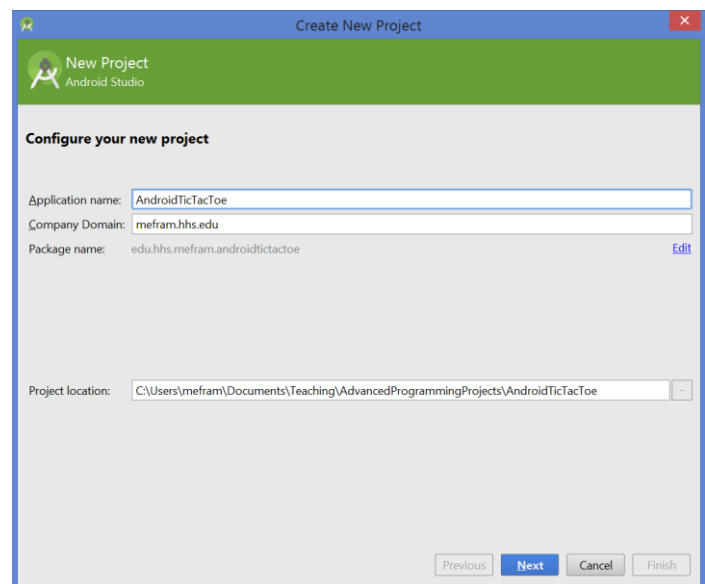


### Creating the Android Project

Start Android Studio, select **File** → **New** → **New Project**. When the New Android Project dialog appears, enter the following values:

- Application Name: **AndroidTicTacToe**
- Company Name: **<yourname>.hhs.edu**
- Project location: **<on your flash drive!!!>**

Click the **Next** button. Select **Phone and Tablet** and **Minimum SDK** to be **API 15**. Click **Next**. Choose **Blank Activity** and click **Next** again. Give the **Activity** the name **TicTacToe**. This should then populate the **Layout Name**, **Title** and **Menu Resource Name**. If you want to change the names of the last three you are welcome to do so. Click **Finish**.



Original work: Frank McCown at Harding University.

Modified by Mike Efram at Healdsburg High School to suit needs of Advanced Programming Projects class.

## App Design

Rather than starting from scratch, a fully functional tic-tac-toe game that runs in a console window has been provided for you at: <http://cs.harding.edu/fmccown/android/TicTacToeConsole.java>

Take a look at the source code to familiarize yourself with how it works. It uses a char array to represent the game board with X representing the human and O representing the computer. The program implements a simple AI for the computer: it will win the game if possible, block the human from winning, or make a random move. The game has all the logic to switch between the two players and to check for a winner.

Our goal is to convert this console game into an Android game. We have two basic choices when porting this game to Android: we could merge the game logic with the user interface (UI) code, or we could keep the two separated as much as possible. The second approach is preferred since it allows us to make changes to the UI, like changing the layout of the board, without having to modify code that just deals with game logic. Also if we decide to port our game to another platform like the BlackBerry or iPhone, having our UI and game logic cleanly separated will minimize the amount of code needing to be modified and make the port much easier to perform.

Our goal then is to place all the UI logic in **TicTacToe.java** and all the game logic in **TicTacToeGame.java**.

1. Add **TicTacToeGame.java** to your Android project by clicking on **File → New → Java Class**. Give it the name **TicTacToeGame** and click **Finish**. You will now need to copy and paste the tic-tac-toe code from the console game into TicTacToeGame.java. You should change the class name and the constructor name from TicTacToeConsole to TicTacToeGame.
2. There are some modifications we'll need to make to the TicTacToeGame class in order to expose the game logic to the AndroidTicTacToe. First, it's no longer necessary that we place the numbers 1-9 into the board character array to represent free positions, so let's create a constant called `OPEN_SPOT` that uses a space character to represent a free location on the game board. We'll continue to use X and O to represent the human and computer players.

```
// Characters used to represent the human, computer, and open spots
public static final char HUMAN_PLAYER = 'X';
public static final char COMPUTER_PLAYER = 'O';
public static final char OPEN_SPOT = ' ';
```

3. You should remove all the code in the TicTacToeGame constructor that is no longer needed for playing in the console. The only line you need to leave in the constructor is the random number generator initialization. You should also remove the `getUserMove()`, `displayBoard()`, and `main()` methods which are no longer needed.
4. You will need to create several public methods that can be used to manipulate the game board and determine if there is a winner. Below is a listing of the public methods you will need to craft the existing code into. **It is left to you to implement these methods.** All other methods in the TicTacToeGame class should be private since we do not want to expose them outside the class.



Original work: Frank McCown at Harding University.

Modified by Mike Efram at Healdsburg High School to suit needs of Advanced Programming Projects class.

```

/** Clear the board of all X's and O's by setting all spots to OPEN_SPOT. */
public void clearBoard()

/** Set the given player at the given location on the game board.
 * The location must be available, or the board will not be changed.
 *
 * @param player - The HUMAN_PLAYER or COMPUTER_PLAYER
 * @param location - The location (0-8) to place the move
 */
public void setMove(char player, int location)

/** Return the best move for the computer to make. You must call setMove()
 * to actually make the computer move to that location.
 * @return The best move for the computer to make (0-8).
 */
public int getComputerMove()

/**
 * Check for a winner and return a status value indicating who has won.
 * @return Return 0 if no winner or tie yet, 1 if it's a tie, 2 if X won,
 * or 3 if O won.
 */
public int checkForWinner()

```

- For the game logic to be accessible to the AndroidTicTacToe, create a class-level variable for the TicTacToeGame class in TicTacToe.java:

```

public class AndroidTicTacToe extends Activity {

// Represents the internal state of the game
private TicTacToeGame mGame;

```

We'll instantiate this object in a later step in the onCreate() method. The Activity can now use mGame to start a new game, set moves, and check for a winner. Note that Android programming conventions use a lowercase "m" at the beginning of all member variables to distinguish themselves from local parameters and variables; we'll use the same naming conventions throughout the tutorials.

## Creating a Game Board

We will represent the tic-tac-toe game board using buttons or ButtonView widgets. We will follow best-practices and create our screen layout using XML rather than hard-code the creation of the buttons in our Java code.

- From your project, navigate to the res/layout folder and double-click on **activity\_tic\_tac\_toe.xml**. The file will be viewable in **Design** or **Text** mode by clicking on the tabs at the bottom of the page. **Design** mode gives us a preview of how the View will look in the emulator. **Text** mode allows us to see the real XML behind the scenes.
- There are several different ways we could lay out the buttons on the screen. We are going to combine the [LinearLayout](#) which displays child view elements horizontally or vertically, and the [TableLayout](#) which displays elements in rows and columns.

The image below is our goal where each button is numbered 1 through 9.

1	2	3
4	5	6
7	8	9

The XML below places buttons 1 through 3 into a TableRow and a TextView below the grid. The TextView will be used later to indicate whose turn it is. **It is left to you to define buttons 4 through 9** which should be placed in their own TableRows. Don't worry about xml warnings about the buttons for now.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal" >

    <TableLayout
        android:id="@+id/play_grid"                android:layout_width="match_parent"
        android:layout_height="wrap_content"      android:layout_marginTop="5dp" >

        <TableRow android:gravity="center_horizontal">
            <Button android:id="@+id/one"                android:layout_width="200dp"
                android:layout_height="200dp"          android:text="1"
                android:textSize="70sp" />

            <Button android:id="@+id/two"                android:layout_width="200dp"
                android:layout_height="200dp"          android:text="2"
                android:textSize="70sp" />

            <Button android:id="@+id/three"              android:layout_width="200dp"
                android:layout_height="200dp"          android:text="3"
                android:textSize="70sp" />
        </TableRow>
    </TableLayout>

    <TextView android:id="@+id/information"            android:layout_width="match_parent"
        android:layout_height="wrap_content"          android:text="info"
        android:gravity="center_horizontal"           android:layout_marginTop="20dp"
        android:textSize="20sp" />

```

Notice that each Button has several attributes:

- **id** – a unique ID which will be used in our Java code to access the widget. The @ tells the XML parser to expand the rest of the ID string and identify it as an ID resource. The + means this is a new resource name that must be added to the app's resources in the R.java file.
- **layout\_width** – the width of the widget in *density-independent pixels* (dp). These allow the buttons to scale appropriately for Android devices with different resolutions. The TextView uses a layout\_width value of match\_parent which means the widget should fill the entire parent. Another acceptable value



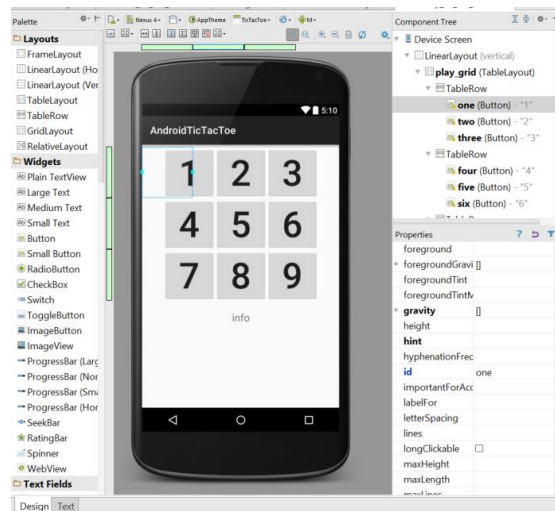
would be `wrap_content` which makes the widget just big enough to hold the content.

- **layout\_height** – the height of the widget in pixels or `match_parent` or `wrap_content`.
- **text** – the text that should be displayed in the button.
- **textSize** – the font size of the text. This can be defined with pixels (px), inches (in), millimeters (mm), points (pt), density-independent pixels (dp), or scale-independent pixels (sp – like dp but scaled by the user's font size preference). Using sp is usually best practice.

The `TextView` uses these additional properties:

- **gravity** – specifies how the text should be placed within its container. The `center_horizontal` value centers the text horizontally.
- **layout\_marginTop** – how much of a margin should be left on the top side of the widget.

Once you have typed all the XML into `activity_tic_tac_toe.xml`, click the **Design** tab to display the view in a WYSIWYG editor. In the **Design** editor, you can click on a widget and see its attribute values in the **Properties** pane. In the figure on the right, you can see the Id of button 1 is set to `"@+id/one"`. You can edit any of the values in this list, and they will be automatically updated in `activity_tic_tac_toe.xml`.



The Graphical Layout editor is useful for dropping widgets onto your View and sizing them. However, many Android programmers prefer to edit the XML directly.

## Accessing the Widgets

We now need to connect the buttons and text widgets defined in `activity_tic_tac_toe.xml` with our Activity.

1. In `TicTacToe.java`, declare a `Button` array and `TextView` member variables:

```
// Buttons making up the board
private Button mBoardButtons[];

// Various text displayed
private TextView mInfoTextView;
```

2. In the `onCreate()` method, instantiate the array and use the `findViewById()` method to attach each button in `activity_tic_tac_toe.xml` with a slot in the array. Note that the ID's are found in `R.java` and match precisely the name assigned to each button in `activity_tic_tac_toe.xml`. You'll also need to instantiate `mGame` so our Activity will be able to access the tic-tac-toe game logic.

 Original work: Frank McCown at Harding University.

Modified by Mike Efram at Healdsburg High School to suit needs of Advanced Programming Projects class.

## Adding Game Logic

Let's now add some logic to start a new game.

1. First, to start a new game, we'll need to clear the game board of any X's and O's from the previous game by calling `mGame.clearBoard()`. This only clears the internal representation of the game board, not the game board we are seeing on the screen. Create a private method in **TicTacToe.java** to do the initialization each time a new game has started.

```
// Set up the game board.  
private void startNewGame() {  
  
    mGame.clearBoard();  
}
```

2. To clear the visible game board, we need to loop through all the buttons representing the board and we need to
  - set their text to an empty string.
  - enable each button (we will disable them later when a move is placed), and
  - set the Button's `onClick` listener to this. (will cause a syntax error for now)

We will need to implement the `OnClickListener` interface later so that when a Button is clicked, we will get notified. Add the code to the `startNewGame()` method. (Try using a for-each loop!!)

3. Finally, we will indicate that the human is to go first. Set the text for your `TextView` object to "You go first." Note that the "You go first." text should normally not be hard-coded into the Java source code for a number of reasons, but you will fix this later.
4. You need to call `startNewGame()` when the App first loads, so add this call on the last line of `onCreate()`:
5. Now let's implement the `OnClickListener` interface for this Activity. It needs to implement the `android.view.View.OnClickListener` interface which means it needs to implement an `onClick()` method. We need to identify which Button was clicked on by comparing with each Button in the array. With this information, we will be able to easily determine which button was selected and where the X should be placed.

The `onClick()` method is called when the user clicks on a button. We should only allow the user to click on enabled buttons which represent valid locations the user may place an X. If the button is enabled, make a call to a method named `setMove()`. We will write a `setMove(char, int)` method shortly which will place the X or O at the given location and disable the button. After displaying the human's move, we need to see if the human has won or tied by calling `checkForWinner()`. If this method returns back 0, then the game isn't over, so the computer must now make its move. Once it moves, we must again check to see if the game is over and update the information text (your `TextView`) appropriately.

6. Below is the `setMove()` method which is called from the `ButtonClickListener`. Place this method in your `AndroidTicTacToe` class. It updates the board model, disables the button, sets the text of the button to X or O, and makes the X green and the O red.



Original work: Frank McCown at Harding University.

Modified by Mike Efram at Healdsburg High School to suit needs of Advanced Programming Projects class.

```

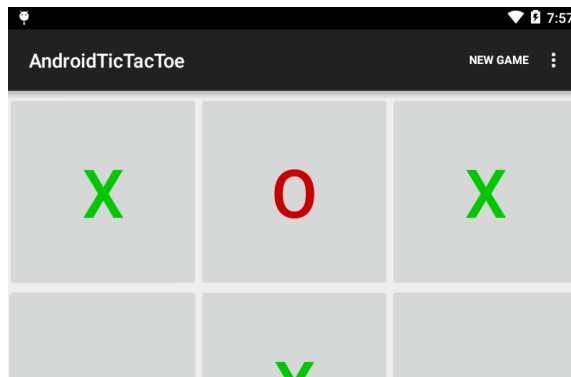
private void setMove(char player, int location) {
    mGame.setMove(player, location);
    mBoardButtons[location].setEnabled(false);
    mBoardButtons[location].setText(String.valueOf(player));
    if (player == TicTacToeGame.HUMAN_PLAYER)
        mBoardButtons[location].setTextColor(Color.rgb(0, 200, 0));
    else
        mBoardButtons[location].setTextColor(Color.rgb(200, 0, 0));
}

```

- Now run your app and play a game. You should see the a message at the bottom of the board telling you when it's your turn, but the computer moves so quickly that you will likely never see the message indicating it's the computer's turn. When the game is over, you won't be able to play another game unless you restart the app. This is a problem we'll fix in the next section.
- Another problem is that when the game is over, the *human can continue making moves!* You will need to fix this problem by introducing a class-level boolean `mGameOver` variable to `AndroidTicTacToe`. When the game starts, `mGameOver` should be set to `false`. When the user is clicking on a button, the variable should be checked so a move isn't made when the game is over. And `mGameOver` needs to be set to `true` when the game has come to an end. It's left to you to make the appropriate modifications.

## Adding a Menu (ActionBar)

In order to start a new game, let's add an option menu so that when the user clicks on the device's ActionBar, the **New Game** option appears as pictured below.



- In `AndroidTicTacToe`, the `onCreateOptionsMenu()` method is called only once, when the Activity's option menu is shown the first time. Add an item to the xml file in the menu folder so that it says "New Game" if there is room. The method must return `true` for the menu to be displayed.
- Next, override the `onOptionsItemSelected()` method which will be called when the user clicks on the **New Game** menu item. Since there's only one menu item that can be selected, we will call `startNewGame()` and return `true`, indicating that we processed the option here.
- Run the Tic-tac-toe app again, and play a complete game. You should now be able to click the emulator's Menu option which will display the New Game option which, when clicked, clears the board and starts a new game.

## String Messages

Several times we have hard-coded UI string messages into our Java source code. For example, when the user wins a game, we have hard-coded the response “You win!” This is a problem for a couple of reasons. First, we may want to change the contents of this message at a later date, and it would be helpful if we did not have to dig through our source code and recompile just to change this message. Secondly, if we wanted our game to use messages in other languages (localization), hard-coding other languages in the source code is problematic.

We should instead follow best-practices and place all our UI messages in **strings.xml**, a file that is located in the project’s `res/values` directory.

If you have not done so already:

1. Locate **strings.xml** and double-click on it to view its contents.
2. Add the following strings to be used in your application:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Tic-Tac-Toe</string>
  <string name="first_human">You go first.</string>
  <string name="turn_human">Your turn.</string>
  <string name="turn_computer">Android\'s turn.</string>
  <string name="result_tie">It\'s a tie.</string>
  <string name="result_human_wins">You won!</string>
  <string name="result_computer_wins">Android won!</string>
</resources>
```

A public, static, final identifier will be created for each of these messages in `R.java`. Note that a single apostrophe (“It’s”) in your string messages must be escaped by placing a backslash in front of it.

3. Now locate where you have hard-coded specific messages in your Java code and modify it to use the messages from `strings.xml`. For example, to display the “You won!”:

```
mInfoTextView.setText(R.string.result_human_wins);
```

4. Lastly, there are many warnings in your layout xml file due to “hardcoding” text strings for the Buttons and TextView widgets. Add a string resource for an “open spot”...a space into `strings.xml` and then rid yourself of all the hardcoding by replacing the hardcoded strings with “@string/open\_spot” (that is if you named your string resource `open_spot`). You should have no warnings when you are finished.

Run your app again and verify that the messages are being displayed properly. Although [localization](#) is not covered in this tutorial, it is rather straight-forward to support it by creating `strings.xml` files for other locals.



## Challenge

As currently implemented, the human always goes first. Make the game fairer by alternating who gets to go first. Also keep track of how many games the user has won, the computer has won, and ties. You should display this information using TextView controls under the game status TextView. Use the [RelativeLayout](#) class to position the TextView controls next to each other like the example on the right:

Human: 0    Ties: 1    Android: 0

Except as otherwise noted, the content of this document is licensed under the Creative Commons Attribution 3.0 License <http://creativecommons.org/licenses/by/3.0>



Original work: Frank McCown at Harding University.

Modified by Mike Efram at Healdsburg High School to suit needs of Advanced Programming Projects class.