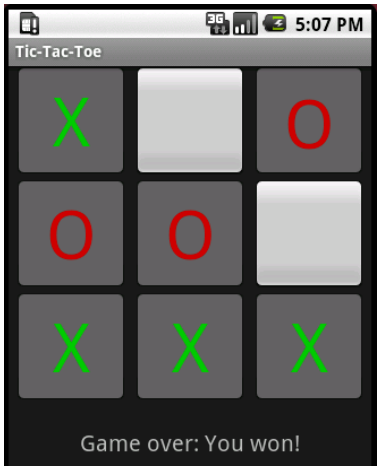
Android Application Programming

Tutorial 2: Tic-Tac-Toe App

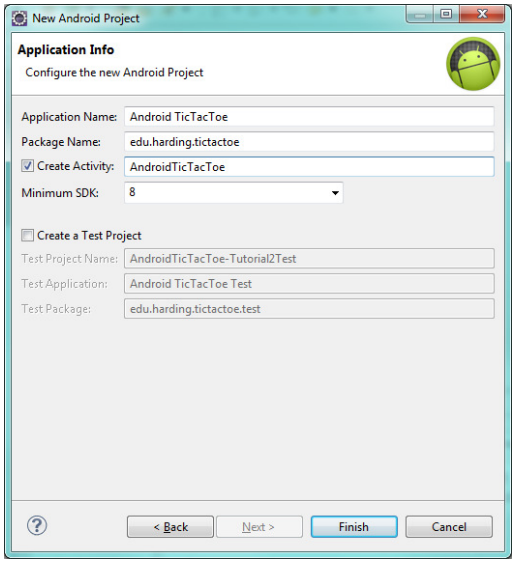
Introduction



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 them 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.

You should have already installed Eclipse, the Android SDK, and the ADT plugin for Eclipse as directed in the first tutorial and be comfortable running an Android app in an emulator.

Creating the Android Project



Start Eclipse, select **File → New → Project…** When the new Project dialog appears, select the **Android** folder, select **Android Project**, and click **Next**. When the New Android Project dialog appears, enter the following values:

- Project name: **AndroidTicTacToe-Tutorial2**
- Build Target: **Android 2.2**
- Application Name: **Android Tic-Tac-Toe**
- Package Name: **edu.harding.tictactoe**
- Activity Name: **AndroidTicTacToeActivity**

Click the **Finish** button, and your project will be created.

Start the Emulator

In order to run an Android project in Eclipse, you need to have a running Android Virtual Device, aka: an enumaltor. We'll use an emulator that runs the Android 2.2 SDK since at the time of writing it is one of the most popular platforms on many mobile phones and some tablets.

Click the  icon from the Eclipse toolbar menu which will start the **Android Virtual Device Manager**. You can also start the Manager from Eclipse's Window menu. If you have not already created a device with a 2.2 platform or above, you will need to create a new device by clicking **New** which will launch a dialog box. Give the new device the name **MyDevice**, select a target of **Android 2.2**, and click **Create AVD**.

Now that you have created this device, you won't need to perform this step again unless you want a device which runs a more recent version of Android.

Start the emulator by selecting **MyDevice** from the list of existing Android Virtual Devices, click **Start…**, and press the **Launch** button on the dialog box that appears. It will take the emulator a minute or two to load. Once the emulator has finished loading, you may need to swipe the unlock button on the emulator so it is ready to go. Now return to Eclipse and minimize or close the Android Virutal Device Manager dialog box.

Now let's run the Android project we have create in Eclipse. Press **Ctrl-F11**, select **Android Application**, and pressing **OK**. When your application finally runs, you will see a blank screen with the words, "Hello World, AndroidTicTacToeActivity!". The application does nothing useful yet.

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

Run this program and 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 **AndroidTicTacToeActivity.java** and all the game logic in **TicTacToeGame.java**.

1. Add **TicTacToeGame.java** to your Android project by clicking on **File → New → Class** the package should be set to **edu.harding.tictactoe**. 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 also change 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 `AndroidTicTacToeActivity`. 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 generater

initialization. You should also remove the `getUserMove()` and `main()` functions 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 functions 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.

```
/** 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()
```

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

```
public class AndroidTicTacToeActivity 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 convensions 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.

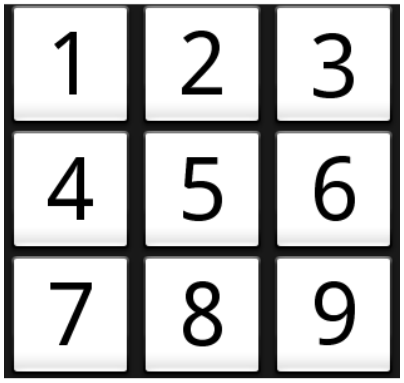
Creating a Game Board

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.

1. From your project, navigate to the res/layout folder and double-click on **main.xml**. The file will likely load into Graphical Layout mode which gives us a preview of how the View will look in the emulator. Click the **main.xml** tab at the bottom of the window to change to the XML view of the file.

2. 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.



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`.

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

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

      <TableRow android:gravity="center_horizontal">
          <Button android:id="@+id/one"                    android:layout_width="100dp"
                  android:layout_height="100dp"            android:text="1"
                  android:textSize="70dp" />

          <Button android:id="@+id/two"                    android:layout_width="100dp"
                  android:layout_height="100dp"            android:text="2"
                  android:textSize="70dp" />

          <Button android:id="@+id/three"                  android:layout_width="100dp"
                  android:layout_height="100dp"            android:text="3"
                  android:textSize="70dp" />
      </TableRow>
   </TableLayout>

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

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 `fill_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 `fill_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 dp is usually best practice.

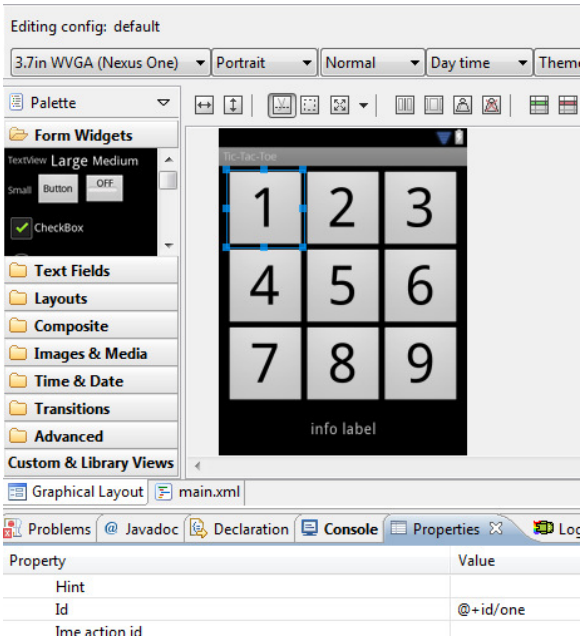
The `TextView` uses these additional properties:

- **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 main.xml, click the **Graphical Layout** tab to display the view in a WYSIWYG editor. In the Graphical Layout editor, you can click on a widget and see its attribute values in the **Properties** tab. 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 main.xml.

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



Accessing the Widgets

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

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

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

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

2. At this point, you'll likely encounter an error in Eclipse indicating that the `Button` and `TextView` classes "cannot be resolved to a type." When you encounter these types of syntax errors, press **Ctrl-Shift-O** to make Eclipse fix all your import statements. Pressing Ctrl-Shift-O at this point will add the following import statements and remove the error messages:

```
import android.widget.Button;
import android.widget.TextView;
```

3. In the `onCreate()` method, instantiate the array and use the `findViewById()` method to attach each button in main.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 main.xml. You'll also need to instatiate `mGame` so our Activity will be able to access the tic-tac-toe game logic.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mBoardButtons = new Button[TicTacToeGame.BOARD_SIZE];
    mBoardButtons[0] = (Button) findViewById(R.id.one);
    mBoardButtons[1] = (Button) findViewById(R.id.two);
    mBoardButtons[2] = (Button) findViewById(R.id.three);
    mBoardButtons[3] = (Button) findViewById(R.id.four);
    mBoardButtons[4] = (Button) findViewById(R.id.five);
    mBoardButtons[5] = (Button) findViewById(R.id.six);
    mBoardButtons[6] = (Button) findViewById(R.id.seven);
    mBoardButtons[7] = (Button) findViewById(R.id.eight);
    mBoardButtons[8] = (Button) findViewById(R.id.nine);

    mInfoTextView = (TextView) findViewById(R.id.information);

    mGame = new TicTacToeGame();
}
```

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.

```
// 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 set their text to an empty string. We also need to enable each button (we will disable them later when a move is placed), and we to create an event listener for each button by setting each button's `OnClickListener` to a new `ButtonClickListener`, a class which we will define shortly. We pass to `ButtonClickListener`'s constructor the button's number (0-8) which will be used to identify *which* button was actually clicked on. Add the code below to the `startNewGame()` function.

```
    // Reset all buttons
    for (int i = 0; i < mBoardButtons.length; i++) {
        mBoardButtons[i].setText("");
        mBoardButtons[i].setEnabled(true);
        mBoardButtons[i].setOnClickListener(new ButtonClickListener(i));
    }
```

3. Finally, we will indicate that the human is to 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 fix this later. Add these lines to the bottom of `startNewGame()`.

```
    // Human goes first
    mInfoTextView.setText("You go first.");

}    // End of startNewGame
```

4. You need to call `startNewGame()` when the App first loads, so add this call on the last line of `onCreate()`:

```
@Override
public void onCreate(Bundle savedInstanceState) {
...

    startNewGame();
}
```

5. Now let's add the `ButtonClickListener` that was used earlier when resetting the buttons. It needs to implement the `android.view.View.OnClickListener` interface which means it needs to implement an `onClick()` method. We also need to create a constructor that takes an integer parameter, the button number, since we need to know which button was clicked on. Without this information, we won't 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. We will write a `setMove()` function 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 function 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 appropriately.

```
// Handles clicks on the game board buttons
private class ButtonClickListener implements View.OnClickListener {
    int location;

    public ButtonClickListener(int location) {
        this.location = location;
    }

    public void onClick(View view) {
        if (mBoardButtons[location].isEnabled()) {
            setMove(TicTacToeGame.HUMAN_PLAYER, location);

            // If no winner yet, let the computer make a move
            int winner = mGame.checkForWinner();
            if (winner == 0) {
               mInfoTextView.setText("It's Android's turn.");
             int move = mGame.getComputerMove();
             setMove(TicTacToeGame.COMPUTER_PLAYER, move);
             winner = mGame.checkForWinner();
            }

            if (winner == 0)
              mInfoTextView.setText("It's your turn.");
            else if (winner == 1)
                mInfoTextView.setText("It's a tie!");
            else if (winner == 2)
                mInfoTextView.setText("You won!");
            else
                mInfoTextView.setText("Android won!");
        }
    }
}
```

6. Below is the `setMove()` function which is called from the `ButtonClickListener`. Place this function in your `AndroidTicTacToeActivity` 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.

```
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));
}
```

7. 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.

8. 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 `AndroidTicTacToeActivity`. 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 appopriate modifications.

Adding a Menu

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

1. In `AndroidTicTacToeActivity`, override the `onCreateOptionsMenu()` method which is called only once, when the Activity's option menu is shown the first time. The code below creates a single menu option labelled **New Game**. The function must return `true` for the menu to be displayed.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
     super.onCreateOptionsMenu(menu);
     menu.add("New Game");
     return true;
}
```

2. 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. In a later tutorial we will learn how to process different menu selections.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    startNewGame();
    return true;
}
```

3. 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, it 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.

1. Locate **strings.xml** and double-click on it to view its contents in Eclipse.

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);
```

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.

Extra 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:

References

Jason Houle's Tic Tac Toe for Android tutorial was helpful in producing this tutorial. I used a similar UI layout and borrowed some of his code. http://www.intelliproject.net/articles/showArticle/index/Android_TicTacToe