Create a networked tic-tac-toe game for Android

Build a networked, multiplayer tic-tac-toe game with PHP, XML, and the Android development kit

Jack D. Herrington
Senior Software Engineer
Fortify Software, Inc.

23 August 2011

Build the back end of a multiplayer, network-enabled tic-tac-toe game with a native Android front-end application in this article.

Networked multiplayer tic-tac-toe

Frequently used acronyms

- API: Application programming interface
- HTTP: HyperText Transfer Protocol
- IP: Internet protocol
- SDK: Software development kit
- SQL: Structured Query Language
- UI: User interface
- XML: Extensible Markup Language

Casual games are extremely popular and very lucrative, and it's easy to see why. Not everyone across all the age groups is interested in playing online, first-person shooters against hordes of preteens with lightening-speed reflexes. Sometimes, it's more interesting to play games where you have time to think and strategize or where the goal is to cooperate with each other to win the game.

The great thing about casual games from the developers' perspective is that they are much easier to build than the graphics-intensive, first-person shooters or sports games. So it's easier for a single developer, or a group of developers, to produce a first version of a novel new game.

In this article, we go through the basics of creating a casual, networked multiplayer tic-tac-toe game. The game server is a MySQL- and PHP-based web application with an XML interface. The front end is a native Android application that works on Android phones.

Building the back end

The back end starts with a simple MySQL database that has two tables. Listing 1 shows the schema for the database.

Listing 1. db.sql

```
DROP TABLE IF EXISTS games;
CREATE TABLE games(
     id INT NOT NULL AUTO_INCREMENT,
     primary key ( id ) );

DROP TABLE IF EXISTS moves;
CREATE TABLE moves(
     id INT NOT NULL AUTO_INCREMENT,
     game INT NOT NULL,
     x INT NOT NULL,
     y INT NOT NULL,
     color INT NOT NULL,
     primary key ( id ) );
```

The first of the two tables is the games table, which has just the unique ID of the game. In a production application, you likely have a users table, and the games table includes the user IDs of both players. To keep it simple, though, I forgo this approach to concentrate on the basics of storing the game data, communicating between the client and server, and building the front end.

The second table is the moves table, which includes the individual moves for the given game, so it has five columns. The first column is the unique ID of the move. The second column is the ID of the game this move applies to. Then come the x and y positions of the move. These values should be between 0 and 2 for both x and y given that you have a three-by-three grid. The last field is the "color" of the move, which is an integer that indicates X or O.

To build the database, first use mysqladmin to create it and then use the mysql command to run the db.sql script as shown here:

```
% mysqladmin --user=root --password=foo create ttt
% mysql --user=root --password=foo ttt < db.sql
```

This step creates a new database called "ttt," which has the tic-tac-toe schema.

Now that you have the schema, you need to create a way to start a game. For this, you have a script called start.php, as in Listing 2.

Listing 2. start.php

```
<?php
header( 'Content-Type:text/xml' );

$dd = new PDO('mysql:host=localhost;dbname=ttt', 'root', '');
$sql = 'INSERT INTO games VALUES ( 0 )';
$sth = $dd->prepare($sql);
$sth->execute( array() );
$qid = $dd->lastInsertId();

$doc = new DOMDocument();
$r = $doc->createElement( "game" );
$r->setAttribute( 'id', $qid );
$doc->appendChild( $r );

print $doc->saveXML();
?>
```

The script starts by connecting to the database. It then executes an INSERT statement against the games table and gets back the ID that was generated. From there it creates an XML document, adds the ID to a game tag, and exports the XML.

You need to run this script to get a game in the database because the simple Android application does not have an interface to create games. Here is the code:

```
$ php start.php
<?xml version="1.0"?>
<game id="1"/>
$
```

Now you have your first game. To see the list of games, use the games.php script that is in Listing 3.

Listing 3. games.php

```
<?php
header( 'Content-Type:text/xml' );

$dbh = new PDO('mysql:host=localhost;dbname=ttt', 'root', '');

$sql = 'SELECT * FROM games';

$q = $dbh->prepare( $sql );
$q->execute( array() );

$doc = new DOMDocument();
$r = $doc->createElement( "games" );
$doc->appendChild( $r );

foreach ( $q->fetchAll() as $row) {
  $e = $doc->createElement( "game" );
  $e->setAttribute( 'id', $row['id'] );
  $r->appendChild( $e );
}

print $doc->saveXML();
?>
```

This script, like the start.php script, starts by connecting to the database. After that it queries the games table to see what's available. And from there it creates a new XML document, adds a games tag, then adds game tags for each of the available games.

When you run this script from the command line, you see something like this:

```
$ php games.php
<?xml version="1.0"?>
<games><game id="1"/></games>
$
```

You can also run this script from the web browser to see the same output.

Excellent! With the games API out of the way, it's time to write the server code to handle the moves. This code starts with building a helper script called show_moves that gets the current moves for a given game and exports them as XML. Listing 4 shows the PHP code for this helper function.

Listing 4. show_moves.php

```
<?php
function show_moves( $dbh, $game ) {
  $sql = 'SELECT * FROM moves WHERE game=?';

  $q = $dbh->prepare( $sql );
  $q->execute( array( $game ) );

  $doc = new DOMDocument();
  $r = $doc->createElement( "moves" );
  $doc->appendChild( $r );

  foreach ( $q->fetchAll() as $row) {
    $e = $doc->createElement( "move" );
    $e->setAttribute( 'x', $row['x'] );
    $e->setAttribute( 'y', $row['y'] );
    $e->setAttribute( 'color', $row['color'] );
    $r->appendChild( $e );
  }

  print $doc->saveXML();
}
?>
```

The script takes a database handle and the game ID. From there it executes the SQL to get the list of moves. Then it creates an XML document with the moves for the given game.

You created this helper function because there are two scripts that use it; the first is a moves.php script that returns the current moves for the specified game. Listing 5 shows this script.

Listing 5. moves.php

```
<?php
require_once( 'show_moves.php' );

header( 'Content-Type:text/xml' );

$dbh = new PDO('mysql:host=localhost;dbname=ttt', 'root', '');

show_moves( $dbh, $_REQUEST['game'] );
?>
```

This simple script includes the helper function code, connects to the database, and then invokes the show_moves function with the specified game ID. To test this code, use the curl command to invoke the script on the server from the command line:

```
$ curl "http://localhost/ttt/moves.php?game=1"
<?xml version="1.0"?>
<moves/>
$
```

Sadly, you haven't made any moves yet, so it's not a particularly interesting output. To remedy that you need to add the final script to the server API. Listing 6 shows the move.php script.

Listing 6. move.php

```
<?php
require_once( 'show_moves.php' );

header( 'Content-Type:text/xml' );

$dbh = new PDO('mysql:host=localhost;dbname=ttt', 'root', '');
$sql = 'DELETE FROM moves WHERE game=? AND x=? AND y=?';
$sth = $dbh->prepare($sql);
$sth->execute( array(
  $_REQUEST['game'],
  $_REQUEST['x'],
  $_REQUEST['y']
) );

$sql = 'INSERT INTO moves VALUES ( 0, ?, ?, ?, ? )';
$sth = $dbh->prepare($sql);
$sth->execute( array(
  $_REQUEST['game'],
  $_REQUEST['x'],
  $_REQUEST['y'],
  $_REQUEST['color']
) );

show_moves( $dbh, $_REQUEST['game'] );
?>
```

This script starts by including the helper function and connecting to the database. It then executes two SQL statements. The first removes any move that might collide with the one being sent in. The second inserts a new row into the moves table for the specified move. The script then returns the list of moves to the client. This step saves the client from having to make two requests each time it makes a move. Bandwidth isn't cheap, so any time you can conglomerate requests you should.

To test that this all works you can make a move:

```
$ curl "http://localhost/ttt/move.php?game=1&x=1&y=2&color=1"
<?xml version="1.0"?>
<moves><move x="1" y="2" color="1"/></moves>
```

With the game server code complete, you can build the Android front end to this multiplayer networked game.

Building the Android front end

First, install the Android SDK, as well as some Android platform versions, and then finally Eclipse and the Android Eclipse plug-in. Thankfully, all this is well documented on the Android site (see Resources for links). In-depth coverage of how to set up your development environment would take up this whole article and more.

After you set up the development environment, launch Eclipse and start a new Android project. You should see something similar to Figure 1.

Figure 1. Creating the Android application in Eclipse

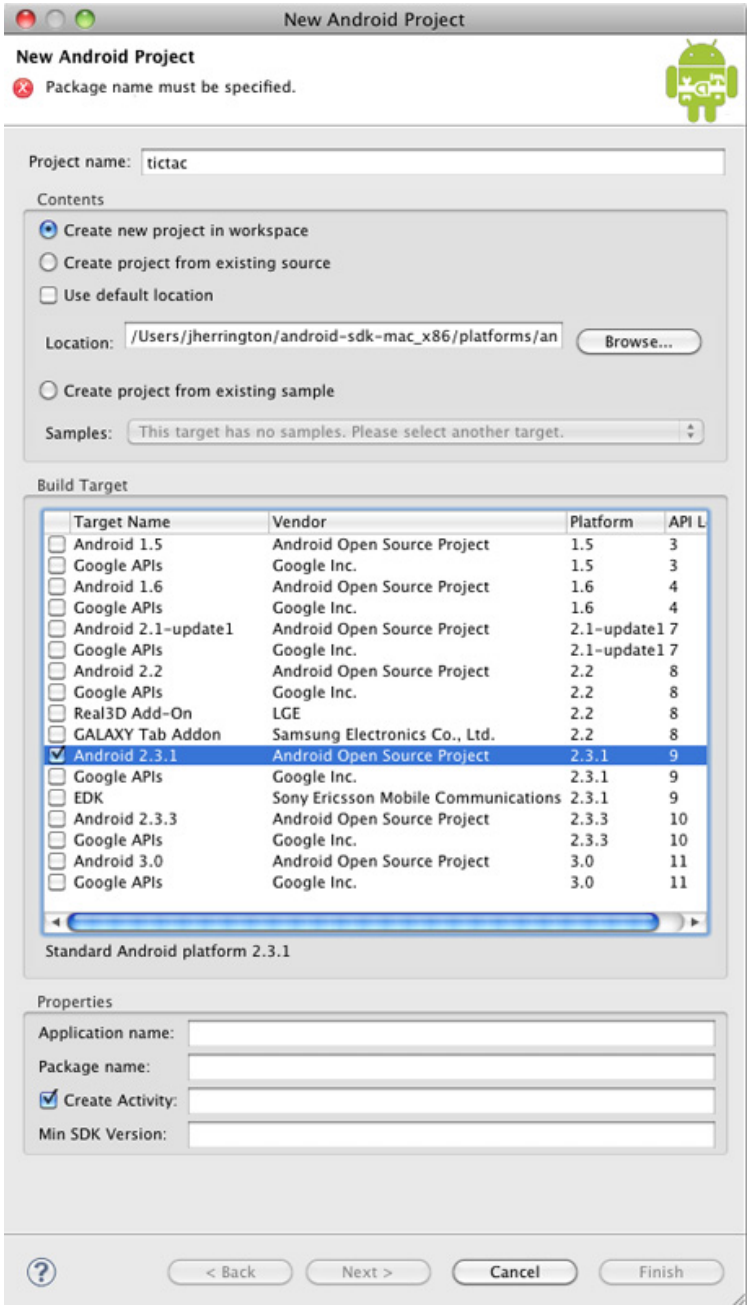


Figure 1 shows the project wizard for Android applications. Enter a project name, select the **Create new project in workspace** radio button and specify the location for code with the UI elements. In the Build Target checklist, select an Android platform. For this code, I use Android 2.3.1. The code is pretty simple so you can use any version that you prefer. If you don't see any platforms listed, then you need to download and install the platforms as noted in the Android SDK setup instructions. Be warned that downloading all of these platforms takes a long, long time.

In the **Properties** section, fill in the application name and the package name. I used "Tic Tac Toe" and "com.jherrington.tictactoe" in the respective fields. Then, select the **Create Activity** check box and enter a name for the activity. I used "TicTacToeActivity" as the activity name.

Click **Finish** to see a new project that resembles Figure 2.

Figure 2. The TicTacToe project files

Figure 2 shows the top-level directories and files for an Android application (the directories are src, gen, Android 2.3.1, and res and the files are assets, AndroidManifest.xml, default.properties, and proguard.cfg). The important items are:

- The res directory, which contains resources
- The src directory, which has the Java™ source
- The manifest file, which contains the biographical information about the application

Your first edit is to the manifest file. Most of the file is already correct, but you need to add the Internet permission so the application can make requests over the Internet. Listing 7 shows the completed manifest file.

Listing 7. AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      android:versionCode="1"
      android:versionName="1.0" package="com.jherrington.tictactoe">

    <uses-permission
        android:name="android.permission.INTERNET" />

    <uses-sdk android:minSdkVersion="5" />

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name="TicTacToeActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
</manifest>
```

The only change was to add the uses-permission tag at the top of the file.

Your next task is to design the UI. For this, tweak the layout.xml file, which is contained in the res/layout directory. Listing 8 shows the new contents for this file.

Listing 8. layout.xml

```
<?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"
    >
    <LinearLayout android:layout_height="wrap_content"
                android:layout_width="match_parent" android:id="@+id/linearLayout1">
        <Button android:text="Play X" android:id="@+id/playx"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"></Button>
        <Button android:text="Play O" android:id="@+id/playo"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"></Button>
    </LinearLayout>
    <com.jherrington.tictactoe.BoardView android:id="@+id/bview"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
    ></com.jherrington.tictactoe.BoardView>
</LinearLayout>
```

This is a straightforward layout. At the top is a set of two buttons wrapped in a linear layout with a horizontal orientation. These two buttons are the X and O buttons that the user uses to specify which color he or she is playing.

The rest of the code is filled with a BoardView class, which shows the Tic Tac Toe board with the current game. The code for the BoardView class is in Listing 11.

With the layout in hand, it's time to write some Java code for the application. This coding starts with the TicTacToeActivity class in Listing 9. Activities are the basic building blocks of Android applications. Each application has one or more activities that represent the various states of the application. As you navigate through the application you build a stack of activities that you can then pop out of by using the back button on the phone. The TicTacToe application has just a single activity.

Listing 9. TicTacToeActivity.java

```
package com.jherrington.tictactoe;

import java.util.Timer;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup.LayoutParams;
import android.widget.Button;
import android.widget.Gallery;
import android.widget.LinearLayout;

public class TicTacToeActivity extends Activity implements OnClickListener {
  @Override
    public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      Button playx = (Button)this.findViewById(R.id.playx);
      playx.setOnClickListener( this );
```

```
      Button playo = (Button)this.findViewById(R.id.playo);
      playo.setOnClickListener( this );

      Timer timer = new Timer();
      UpdateTimer ut = new UpdateTimer();
      ut.boardView = (BoardView)this.findViewById(R.id.bview);
      timer.schedule( ut, 200, 200 );
    }

    public void onClick(View v) {
      BoardView board = (BoardView)this.findViewById(R.id.bview);
      if ( v.getId() == R.id.playx ) {
        board.setColor( 2 );
      }
      if ( v.getId() == R.id.playo ) {
        board.setColor( 1 );
      }
    }
}
```

The activity has two methods. The first is the onCreate method, which builds the user interface, connects the onClick handler to the X and O buttons, and starts the update timer. The update timer is used to refresh the state of the game every 200 milliseconds. This feature allows both players to see when the other player moves.

The onClick handler sets the current color of the board based on whether the user clicks the X or O button.

The GameService class, in Listing 10, is a singleton class that represents the game server and the current state of the given game.

Listing 10. GameService.java

```
package com.jherrington.tictactoe;

import java.util.ArrayList;
import java.util.List;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

import android.util.Log;

public class GameService {
  private static GameService _instance = new GameService();

  public int[][] positions = new int[][] {
      { 0, 0, 0 },
      { 0, 0, 0 },
      { 0, 0, 0 }
  };
```

```
  public static GameService getInstance() {
    return _instance;
  }

  private void updatePositions( Document doc ) {
    for( int x = 0; x < 3; x++ ) {
      for( int y = 0; y < 3; y++ ) {
        positions[x][y] = 0;
      }
    }
    doc.getDocumentElement().normalize();
    NodeList items = doc.getElementsByTagName("move");
    for (int i=0;i<items.getLength();i++){
      Element me = (Element)items.item(i);
      int x = Integer.parseInt( me.getAttribute("x") );
      int y = Integer.parseInt( me.getAttribute("y") );
      int color = Integer.parseInt( me.getAttribute("color") );
      positions[x][y] = color;
    }
  }

  public void startGame( int game ) {
    HttpClient httpclient = new DefaultHttpClient();
    HttpPost httppost = new HttpPost("http://10.0.2.2/ttt/moves.php");

    try {
      List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(2);
      nameValuePairs.add(new BasicNameValuePair("game", Integer.toString(game)));
      httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));

      HttpResponse response = httpclient.execute(httppost);
      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
      DocumentBuilder db = dbf.newDocumentBuilder();
      updatePositions( db.parse(response.getEntity().getContent()) );
    } catch (Exception e) {
      Log.v("ioexception", e.toString());
    }
  }

  public void setPosition( int game, int x, int y, int color ) {
    HttpClient httpclient = new DefaultHttpClient();
    HttpPost httppost = new HttpPost("http://10.0.2.2/ttt/move.php");

    positions[x][y] = color;

    try {
      List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(2);
      nameValuePairs.add(new BasicNameValuePair("game", Integer.toString(game)));
      nameValuePairs.add(new BasicNameValuePair("x", Integer.toString(x)));
      nameValuePairs.add(new BasicNameValuePair("y", Integer.toString(y)));
      nameValuePairs.add(new BasicNameValuePair("color", Integer.toString(color)));
      httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));

      HttpResponse response = httpclient.execute(httppost);
      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
      DocumentBuilder db = dbf.newDocumentBuilder();
      updatePositions( db.parse(response.getEntity().getContent()) );
    } catch (Exception e) {
      Log.v("ioexception", e.toString());
    }
  }
}
```

This code is some of the most interesting code in the application. First, you have the updatePositions method, which takes the XML returned from the server and looks for the move

elements, then updates the positions array with the current set of moves. The positions array has a value for each position on the board; zero indicates an empty space, 1 represents "O," and 2 is for "X."

The other two functions, startGame and setPosition, are how you communicate with the server. The startGame method requests the current set of moves from the server and updates the list of positions. The setPosition method posts the move to the server by creating an HTTP post request and setting up the data for the post using an array of name-value pairs, which are then encoded for transport. It then parses the response XML to update the list of positions.

If you look closely, the IP used to connect to the server is really interesting. It's not "localhost" or "127.0.0.1"; it's "10.0.2.2," which is an alias for the machine that the emulator is running on. Because the Android phone is itself a UNIX® system, it has its own services on localhost. Fascinating, right? It's not often that it's so clear that the phone is not really a phone per se, but a fully fledged computer that fits in your palm and just happens to have a phone built into it.

So, where are we? You have the activity, which is the main component for the application; you have the UI layout set up; you have the Java code to connect to the server. Now you need to draw the game board. which is done by the BoardView class in Listing 11.

Listing 11. BoardView.java

```
package com.jherrington.tictactoe;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Rect;
import android.util.AttributeSet;
import android.view.MotionEvent;
import android.view.View;

public class BoardView extends View {
  private int _color = 1;

  public void setColor( int c ) {
    _color = c;
  }

  public BoardView(Context context) {
      super(context);
      GameService.getInstance().startGame(0);
  }

  public BoardView(Context context, AttributeSet attrs) {
      super(context,attrs);
      GameService.getInstance().startGame(0);
  }

  public BoardView(Context context, AttributeSet attrs, int defStyle) {
      super(context,attrs,defStyle);
      GameService.getInstance().startGame(0);
  }

  public boolean onTouchEvent( MotionEvent event ) {
    if ( event.getAction() != MotionEvent.ACTION_UP )
      return true;
```

```
    int offsetX = getOffsetX();
    int offsetY = getOffsetY();
    int lineSize = getLineSize();
    for( int x = 0; x < 3; x++ ) {
      for( int y = 0; y < 3; y++ ) {
        Rect r = new Rect( ( offsetX + ( x * lineSize ) ),
            ( offsetY + ( y * lineSize ) ),
            ( ( offsetX + ( x * lineSize ) ) + lineSize ),
            ( ( offsetY + ( y * lineSize ) ) + lineSize ) );
        if ( r.contains( (int)event.getX(), (int)event.getY() ) ) {
          GameService.getInstance().setPosition(0, x, y, _color);
          invalidate();
          return true;
        }
      }
    }
    return true;
  }

  private int getSize() {
    return (int) ( (float)
    ( ( getWidth() < getHeight() ) ? getWidth() : getHeight() ) * 0.8 );
  }

  private int getOffsetX() {
    return ( getWidth() / 2 ) - ( getSize( ) / 2 );
  }

  private int getOffsetY() {
    return ( getHeight() / 2 ) - ( getSize() / 2 );
  }

  private int getLineSize() {
    return ( getSize() / 3 );
  }

  protected void onDraw(Canvas canvas) {
    Paint paint = new Paint();
    paint.setAntiAlias(true);
    paint.setColor(Color.BLACK);
    canvas.drawRect(0,0,canvas.getWidth(),canvas.getHeight(), paint);

    int size = getSize();
    int offsetX = getOffsetX();
    int offsetY = getOffsetY();
    int lineSize = getLineSize();

    paint.setColor(Color.DKGRAY);
    paint.setStrokeWidth( 5 );
    for( int col = 0; col < 2; col++ ) {
      int cx = offsetX + ( ( col + 1 ) * lineSize );
      canvas.drawLine(cx, offsetY, cx, offsetY + size, paint);
    }
    for( int row = 0; row < 2; row++ ) {
      int cy = offsetY + ( ( row + 1 ) * lineSize );
      canvas.drawLine(offsetX, cy, offsetX + size, cy, paint);
    }
    int inset = (int) ( (float)lineSize * 0.1 );

    paint.setColor(Color.WHITE);
    paint.setStyle(Paint.Style.STROKE);
    paint.setStrokeWidth( 10 );
    for( int x = 0; x < 3; x++ ) {
      for( int y = 0; y < 3; y++ ) {
        Rect r = new Rect( ( offsetX + ( x * lineSize ) ) + inset,
            ( offsetY + ( y * lineSize ) ) + inset,
            ( ( offsetX + ( x * lineSize ) ) + lineSize ) - inset,
```

```
            ( ( offsetY + ( y * lineSize ) ) + lineSize ) - inset );
        if ( GameService.getInstance().positions[ x ][ y ] == 1 ) {
          canvas.drawCircle( ( r.right + r.left ) / 2,
                  ( r.bottom + r.top ) / 2,
                  ( r.right - r.left ) / 2, paint);
        }
        if ( GameService.getInstance().positions[ x ][ y ] == 2 ) {
          canvas.drawLine( r.left, r.top, r.right, r.bottom, paint);
          canvas.drawLine( r.left, r.bottom, r.right, r.top, paint);
        }
      }
    }
  }

}
```

Most of the work here is done in the onTouch method, which responds to the user touching a particular cell on the game board, and the onDraw method, which paints the game board using Android's painting mechanism.

The onTouch method uses the sizing functions to figure out a rectangle for each cell position. It then uses the contains method on the rectangle to see if the user clicked within the cell. If they did, it fires off a request to the game service to make the move.

The onDraw function uses the sizing functions to both draw the lines of the board and draw any played Xs and Os. The GameServer singleton is used for its positions array, which has the current state of each square on the game board.

The last class you need is the UpdateTimer, which uses the game service to update the board positions with their latest values. Listing 12 shows the code for the timer.

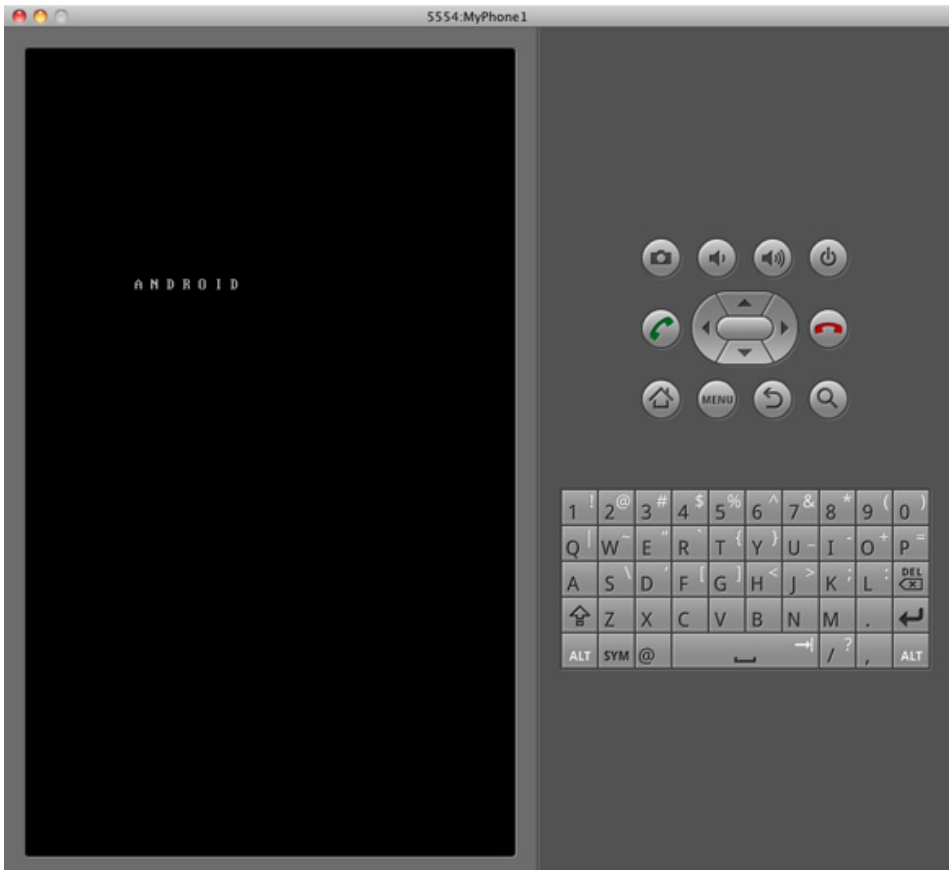
Listing 12. UpdateTimer.java

```
package com.jherrington.tictactoe;

import java.util.TimerTask;

public class UpdateTimer extends TimerTask {
  public BoardView boardView;

  @Override
  public void run() {
    GameService.getInstance().startGame( 0 );
    boardView.post(new Runnable(){ public void run(){ boardView.invalidate(); } });
  }
}
```

The timer is initialized by the TicTacToeActivity class when the application starts up. This timer is a polling mechanism. This is not the most efficient way to communicate between the client and server, but it is the simplest and most reliable. The most efficient way is to use the 1.1 version of the HTTP protocol to hold the connection open and to have the server send out updates to the client when moves are made. This approach is a lot more complex; it requires both the client and the server to support the 1.1 protocol, and it has scalability issues with the number of connections. That approach is outside the scope of this article. For simple demonstration games like this, a polling mechanism works just fine.

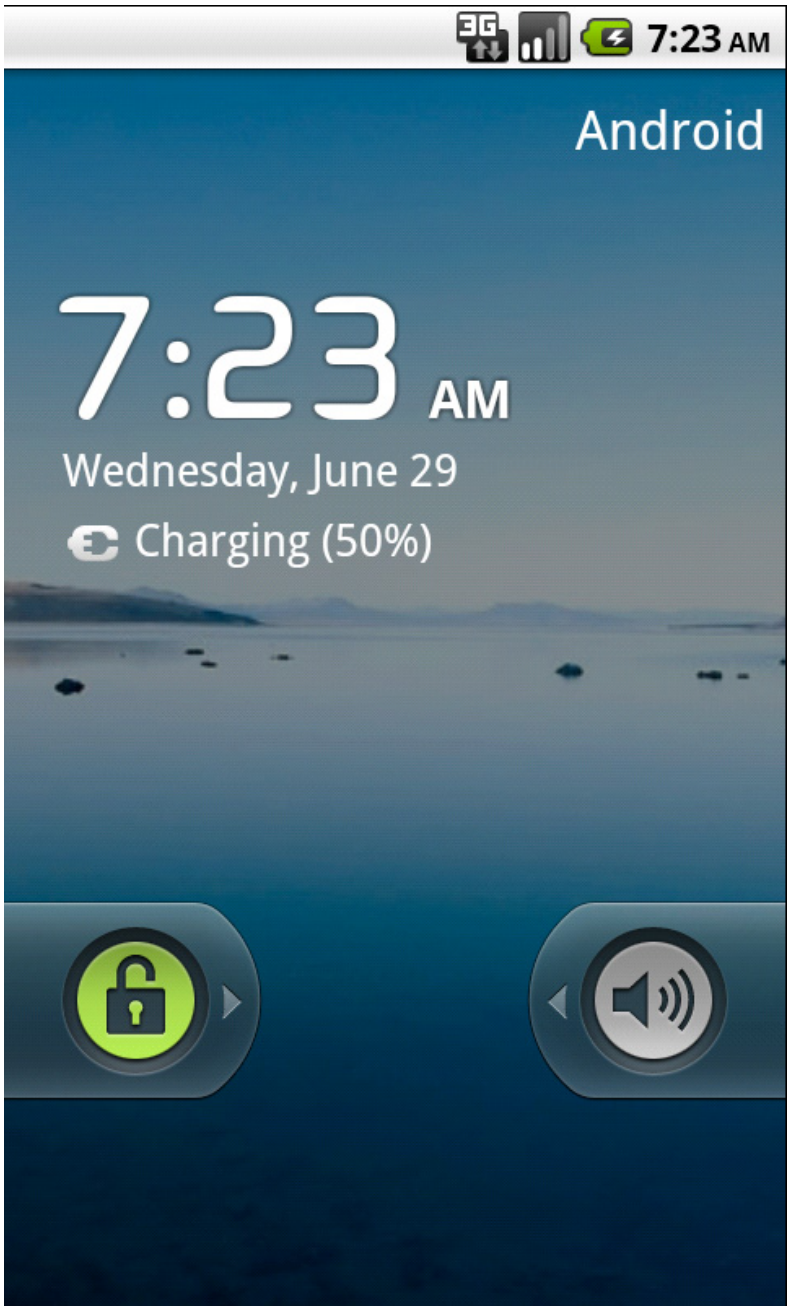
With the code done, you can test the application. That means starting up the emulator. You should see something like Figure 3 after startup.

Figure 3. Launching the Android emulator



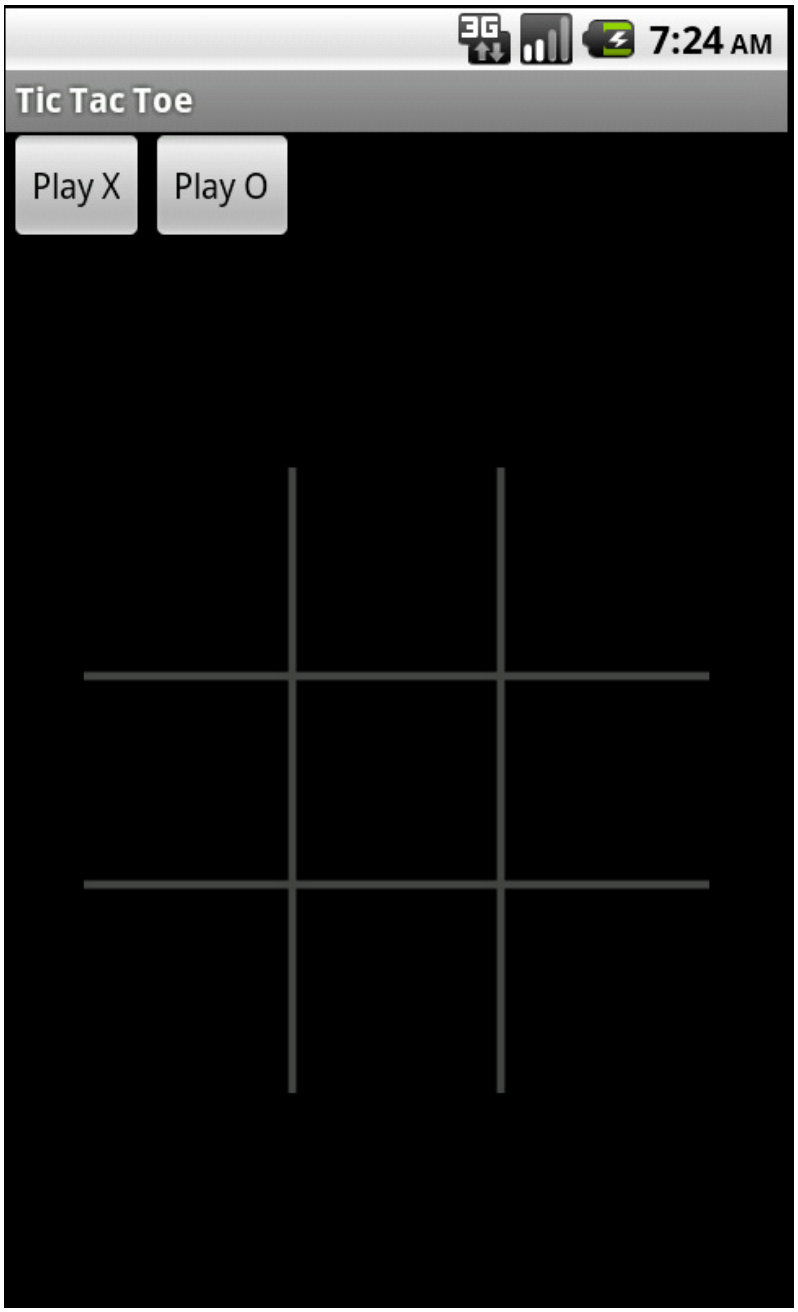
This is the emulator loading up a fantastic "A N D R O I D" interface, After it's loaded, you see the power-on screen in Figure 4.

Figure 4. The emulator launched and ready to go



To get into the phone, slide the lock icon to the right. That action gets you to the home screen and generally launches the application that you are debugging. In this case, this action displays the game screen in Figure 5.

Figure 5. The game before moves have been made



Depending on the state of your server, you either see or do not see any moves. In this case, the game was empty. The Play X and Play O buttons are at the top with the tic-tac-toe game board in the middle of the display. Next, click **Play X**, then click the center square to see something like Figure 6.

Figure 6. X takes center square, of course

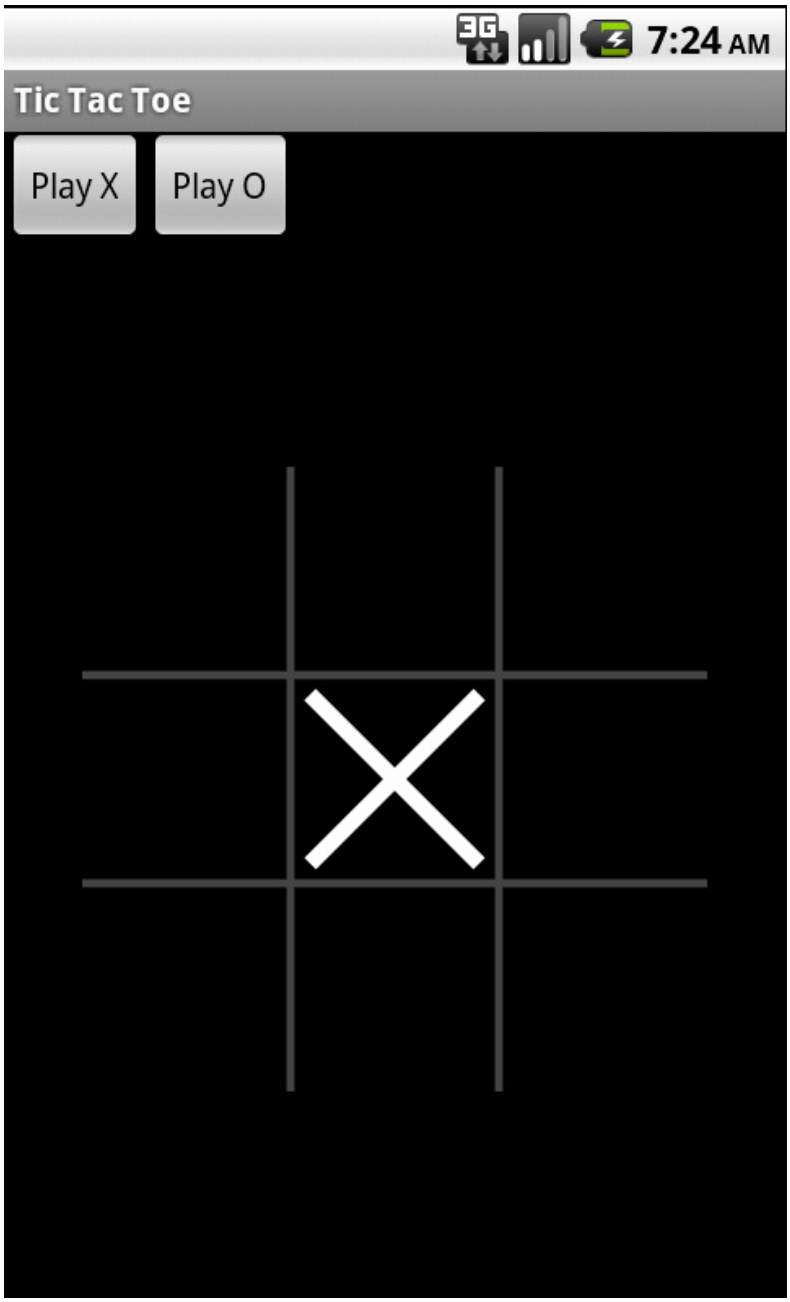
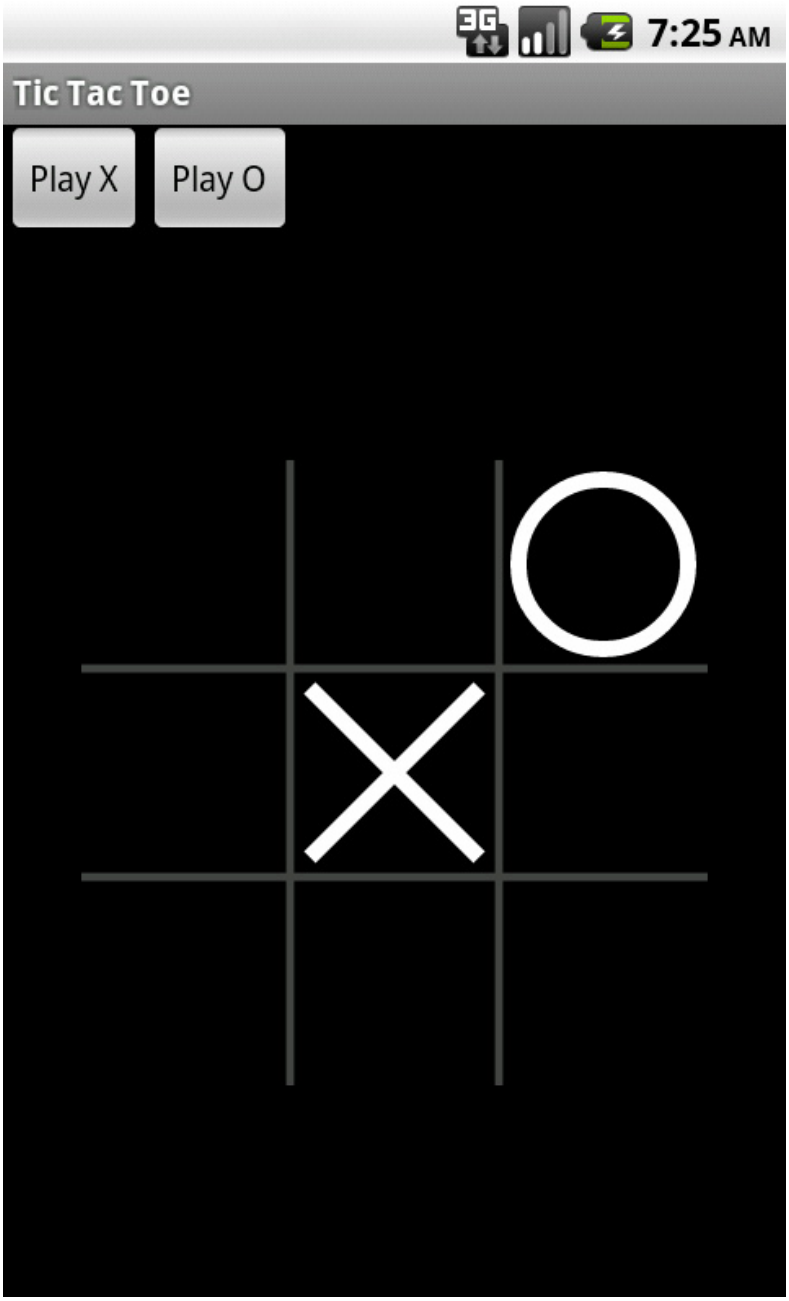


Figure 6 shows the display of the game board with an X now populating the center square. To verify that the server was connected, you can execute the curl command against the moves.php script on the server to get the most recent list of game moves.

To test that the Os work, click **Play O** and select a corner square as in Figure 7.

Figure 7. O takes a corner square



You can play both Xs and Os. The application connects to the server to hold the state of the game in a shared location. And because of the update timer, each user can see the moves made by the other.

Conclusion

Is this a complete game? Not really. There is no victory condition check, players can overwrite positions, and there is no turn check. But the basic technology pieces are present: a game server with shared stored state between the players and a native graphical application on a mobile device that connects to the game server to provide an interface to the game. You can use this game as

a starting point for your own game and build it out however you please. Just remember to keep it casual and fun, and you might have yourself the next Words With Friends or multiplayer Angry Birds.

Resources

Learn

- Eclipse: Learn more about the IDE used in this article to develop the Android application. Find Eclipse downloads and plugins, too.
- PHP Development Tools for Eclipse: Need an IDE for PHP? The Eclipse project has an extension for that plus other Eclipse plugins for just about everything.
- Android Market: After you write your Android networked multiplayer casual game, upload it to the Android marketplace. And let us know that you have done so in the comments section of this article.
- The PHP site: Explore the best reference for PHP that's available.
- The W3C: Visit a great site for standards, in particular the XML standard is relevant to this article.
- More articles by this author (Jack Herrington, developerWorks, March 2005-current): Read articles about Ajax, JSON, PHP, XML, and other technologies.
- New to XML? Get the resources you need to learn XML.
- XML area on developerWorks: Find the resources you need to advance your skills in the XML arena, including DTDs, schemas, and XSLT. See the XML technical library for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.
- IBM XML certification: Find out how you can become an IBM-Certified Developer in XML and related technologies.
- developerWorks technical events and webcasts: Stay current with technology in these sessions.
- developerWorks on Twitter: Join today to follow developerWorks tweets.
- developerWorks podcasts: Listen to interesting interviews and discussions for software developers.
- developerWorks on-demand demos: Watch demos ranging from product installation and setup for beginners to advanced functionality for experienced developers.

Get products and technologies

- The Android Developer site: Download the SDK and the Eclipse plug-in.
- IBM product evaluation versions: Download or explore the online trials in the IBM SOA Sandbox and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- XML zone discussion forums: Participate in any of several XML-related discussions.
- The developerWorks community: Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.

About the author

Jack D. Herrington

Jack Herrington is an engineer, author, and presenter who lives and works in the Bay Area. You can keep up with his work and his writing at http://jackherrington.com.