Addressing Non-Functional Requirements in Mobile Applications

CSE 5236: Mobile Application Development
Instructor: Adam C. Champion
Course Coordinator: Dr. Rajiv Ramnath
Outline

• Non-Functional Requirements
• Optimize Performance with Profiler
• Maximize Battery Life
• Optimize for Responsiveness
• Improve App Security
• Testing
Non-Functional Requirements (NFRs)

- AKA quality/design requirements
  - Building the app right (as opposed to the “right app” from functional requirements)

- Typical NFRs:
  - Performance
  - Availability
  - Scalability
  - Usability
  - Security
  - Modifiability
  - Maintainability and testability
  - Cost

- (Almost) always involves tradeoffs
  - Some combinations are aligned (e.g. security and availability)
Key NFRs for Mobile Devices

- Performance
- Responsiveness *(not the same as performance)*
- Energy *(not covered here, see PowerManager class)*
  
  

- Security
Systematic Steps Towards Meeting NFRs

• “Quantify” for the app
• Make appropriate architectural decisions: often pre-determined by the underlying architecture of the implementation framework (e.g., the Android SDK)
• Optimize tactically based on real measurements
Architectural Decisions in Tic-Tac-Toe

• Java – reduced cost of development
• Data storage tactics:
  – Preferences: cost of development
  – SQLite: Reliability, queries faster than inserts suited for login use case.
• Data transfer in JSON
• 2-D graphics for speed
Tactical Optimizations Used in Tic-Tac-Toe

• Used variables to cache data retrieved from collections (e.g. arrays)
• Avoided internal use of getters and setters
• Reduced heap access: avoid creating unnecessary objects (see use of Singleton for X, O and Blank symbols)
• Used `static final` for constants (allows inlining of constants)
• Leveraged optimizations in framework libraries
Outline

• Non-Functional Requirements
• **Optimize Performance with Profiler**
• Maximize Battery Life
• Optimize for Responsiveness
• Improve App Security
• Testing
Optimize Performance Using the Profiler

1. Click “Android Device Monitor” icon
2. Start the application
3. Navigate to start of profiling scenario
4. Start Method Profiling
5. Execute several profiling scenarios
6. Stop Method Profiling
Traceview Window

Note contribution of Board.onDraw(...) and getBitmapForSymbol(...)

[Diagram showing thread activity with emphasis on Board.onDraw and getBitmapForSymbol]
for(int i = 0; i < GameGrid.SIZE; i++) {
    for(int j = 0; j < GameGrid.SIZE; j++) {
        Bitmap symSelected =
            getBitmapForSymbol(grid.getValueAtLocation(i, j));
        offsetX = (int)(((width - symSelected.getWidth())/2) + (i * width));
        offsetY = (int)(((height - symSelected.getHeight())/2) + (j * height));
        canvas.drawBitmap(symSelected, offsetX, offsetY, ditherPaint);
    }
}
public Bitmap getBitmapForSymbol(Symbol aSymbol) {
    Resources res = getResources();
    Bitmap symX = BitmapFactory.decodeResource(res, R.drawable.x);
    Bitmap symO = BitmapFactory.decodeResource(res, R.drawable.o);
    Bitmap symBlank = BitmapFactory.decodeResource(res, R.drawable.blank);

    Bitmap symSelected = symBlank;
    if (aSymbol == Symbol.SymbolXCreate())
        symSelected = symX;
    else if (aSymbol == Symbol.SymbolOCreate())
        symSelected = symO;
    return symSelected;
}
Optimizing `getBitmapForSymbol()`

```java
static Bitmap symX=null, symO=null, symBlank=null;
static boolean bitMapsInitialized=false;

public Bitmap getBitmapForSymbol(Symbol aSymbol){
    if (!bitMapsInitialized){
        Resources res = getResources();
        symX = BitmapFactory.decodeResource(res, R.drawable.x);
        symO = BitmapFactory.decodeResource(res, R.drawable.o);
        symBlank = BitmapFactory.decodeResource(res, R.drawable.blank);
        bitMapsInitialized=true;
    }
    Bitmap symSelected = symBlank;
    if (aSymbol == Symbol.SymbolXCreate())
        symSelected = symX;
    else if (aSymbol == Symbol.SymbolOCreate())
        symSelected = symO;
    return symSelected;
}
```
## After Optimization

<table>
<thead>
<tr>
<th>Name</th>
<th>Incl %</th>
<th>Inclusive</th>
<th>Excl %</th>
<th>Exclusive</th>
<th>Calls + Recur Calls / Total</th>
<th>Time/Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (toplevel)</td>
<td>100.0%</td>
<td>13848.253</td>
<td>0.5%</td>
<td>81.828</td>
<td>4+0</td>
<td>3462.063</td>
</tr>
<tr>
<td>1 android/os/Handler.dispatchMessage (Android/os/Message;)V</td>
<td>95.5%</td>
<td>13227.367</td>
<td>0.1%</td>
<td>19.075</td>
<td>210+0</td>
<td>62.987</td>
</tr>
<tr>
<td>2 android/view/ViewRoot.handleMessage (Android/os/Message;)V</td>
<td>87.2%</td>
<td>12081.056</td>
<td>0.2%</td>
<td>31.974</td>
<td>173+0</td>
<td>69.833</td>
</tr>
<tr>
<td>3 android/view/ViewRoot.performTraversals ()V</td>
<td>47.8%</td>
<td>6614.546</td>
<td>0.3%</td>
<td>46.002</td>
<td>57+0</td>
<td>116.045</td>
</tr>
<tr>
<td>4 android/view/ViewRoot.deliverPointerEvent (Android/view/MotionEvent;)V</td>
<td>34.0%</td>
<td>4704.943</td>
<td>0.1%</td>
<td>17.659</td>
<td>50+0</td>
<td>94.099</td>
</tr>
<tr>
<td>5 com/android/internal/policy/impl/PhoneWindow$DecorView.dispatchEvent (Landroid/</td>
<td>33.7%</td>
<td>4671.461</td>
<td>0.1%</td>
<td>7.921</td>
<td>50+0</td>
<td>93.429</td>
</tr>
<tr>
<td>6 com/android/internal/policy/impl/PhoneWindow.superDispatchTouchEvent (Landroid/</td>
<td>33.5%</td>
<td>4644.881</td>
<td>0.0%</td>
<td>4.016</td>
<td>50+0</td>
<td>92.898</td>
</tr>
<tr>
<td>7 com/android/internal/policy/impl/PhoneWindow$DecorView.superDispatchTouchEvent</td>
<td>33.5%</td>
<td>4640.865</td>
<td>0.0%</td>
<td>4.134</td>
<td>50+0</td>
<td>92.817</td>
</tr>
<tr>
<td>8 android/view/ViewGroup.dispatchTouchEvent (Android/view/MotionEvent;Z)</td>
<td>33.5%</td>
<td>4636.731</td>
<td>0.7%</td>
<td>95.170</td>
<td>50+102</td>
<td>30.505</td>
</tr>
<tr>
<td>9 android/app/Activity.dispatchEvent (Android/view/MotionEvent;Z)</td>
<td>32.3%</td>
<td>4478.393</td>
<td>0.1%</td>
<td>10.099</td>
<td>40+0</td>
<td>111.960</td>
</tr>
<tr>
<td>10 android/view/View.dispatchEvent (Android/view/MotionEvent;Z)</td>
<td>32.2%</td>
<td>4457.374</td>
<td>0.1%</td>
<td>16.348</td>
<td>122+0</td>
<td>36.536</td>
</tr>
<tr>
<td>11 com/wiley/fordummies/androidsdk/tictactoe/Board.onTouchEvent (Android/view/M</td>
<td>30.7%</td>
<td>4252.427</td>
<td>0.1%</td>
<td>12.378</td>
<td>18+0</td>
<td>236.246</td>
</tr>
<tr>
<td>12 com/wiley/fordummies/androidsdk/tictactoe/GameSession.humanTakesATurn (II)V</td>
<td>29.8%</td>
<td>4126.459</td>
<td>0.1%</td>
<td>12.882</td>
<td>18+0</td>
<td>229.248</td>
</tr>
<tr>
<td>13 android/view/View.measure (II)V</td>
<td>23.3%</td>
<td>3231.154</td>
<td>0.4%</td>
<td>59.217</td>
<td>16+574</td>
<td>5.477</td>
</tr>
<tr>
<td>14 android/widget/FrameLayout.onMeasure (II)V</td>
<td>23.3%</td>
<td>3229.583</td>
<td>0.4%</td>
<td>55.793</td>
<td>16+58</td>
<td>43.643</td>
</tr>
<tr>
<td>15 android/view/ViewGroup.measureChildWithMargins (Android/view/View;III)V</td>
<td>23.1%</td>
<td>3196.594</td>
<td>0.7%</td>
<td>92.537</td>
<td>16+392</td>
<td>7.835</td>
</tr>
</tbody>
</table>
Outline

• Non-Functional Requirements
• Optimize Performance with Profiler
• Maximize Battery Life
• Optimize for Responsiveness
• Improve App Security
• Testing
Maximize Battery Life

• Reducing computation (same techniques as for performance)
• Reducing network usage
  – Minimizing data services
  – Minimizing location services
• Managing display brightness
Minimize Network Use

• Check for network availability
  
  ```java
  private boolean hasNetworkConnection()
  {
    ConnectivityManager connectivityManager =
        (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo =
        connectivityManager.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
    boolean isConnected = true;
    boolean isWifiAvailable = networkInfo.isAvailable();
    boolean isWifiConnected = networkInfo.isConnected();
    networkInfo =
        connectivityManager.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
    boolean isMobileAvailable = networkInfo.isAvailable();
    boolean isMobileConnected = networkInfo.isConnected();
    isConnected = (isMobileAvailable && isMobileConnected) ||
        (isWifiAvailable && isWifiConnected);
    return isConnected;
  }
  ```

• Use compact data formats (JSON)
Minimize Location Services: Use Last Known Location

public Location getBestCurrentLocation()
{
    Location myLocation = null;
    myLocation = manager.getLastKnownLocation(bestProvider);
    if (myLocation == null) {
        myLocation = manager.getLastKnownLocation("network");
    }
    if (myLocation != null) {
        System.out.println("GeoLocation is">" +
        myLocation.toString() + ">");
        thisLocation = myLocation;
    }
    return thisLocation;
}
Minimize Location Services: Use Cheapest Provider

```java
public GeoLocation(Context theContext) {
    Criteria criteria = new Criteria();
    criteria.setAccuracy(Criteria.ACCURACY_COARSE);
    criteria.setPowerRequirement(Criteria.POWER_LOW);
    LocationManager manager =
        (LocationManager) getSystemService(thisContext.LOCATION_SERVICE);
    String cheapestProvider = myLocationManager.getBestProvider(c, true);
    registerForLocationUpdates();
}
```
Outline

• Non-Functional Requirements
• Optimize Performance with Profiler
• Maximize Battery Life
• Optimize for Responsiveness
• Improve App Security
• Testing
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork();
            // user written method
            // do something with the image ...
        }
    }).start();
}
public void onCreate(Bundle savedInstanceState) {
    ...  
    // Launch a new thread for the splash screen
    Thread splashThread = new Thread() { 
        @Override public void run() { 
            try {
                int elapsedTime = 0;
                while(active && (elapsedTime < splashTime)) {
                    sleep(sleepTime);
                    if(active) elapsedTime = elapsedTime + timeIncrement;
                }
            } catch(InterruptedException e) { 
                // do nothing
            } finally {
                finish();
                startActivity(new Intent(“com.wiley.fordummies.androidsdk.tictactoe.Login’’));
            }
        }
    };
    splashThread.start();
}
Threading: Ex. (2): SplashScreen (2)

```java
public boolean onTouchEvent(MotionEvent event) {
    if (event.getAction() == MotionEvent.ACTION_DOWN) {
        active = false;
    }
    return true;
}
```
private void scheduleAndroidsTurn()
{
    System.out.println("Thread ID in scheduleAndroidsTurn: "+
                    Thread.currentThread().getId());
    board.disableInput(); // prevents user input while machine is thinking
    if (!testMode)
    {
        Random randomNumber = new Random();
        Handler handler = new Handler();
        handler.postDelayed( //Posts callback with a random delay.
            new Runnable()
            {
                public void run()
                {
                    androidTakesATurn();
                }
            }, ANDROIDTIMEOUTBASE + randomNumber.nextInt(ANDROIDTIMEOUTSEED));
    } else {
        androidTakesATurn();
    }
}
public class HelpWithWebView extends Activity implements OnClickListener {

  protected void onCreate(Bundle savedInstanceState) {
    String URL=null;
    super.onCreate(savedInstanceState);
    setContentView(R.layout.helpwithwebview);
    WebView helpInWebView=null;
    helpInWebView = (WebView) findViewById(R.id.helpwithwebview);
    View buttonExit = findViewById(R.id.button_exit);
    buttonExit.setOnClickListener(this);
    Bundle extras = getIntent().getExtras();
    if(extras !=null){URL = extras.getString("URL"); } 
    helpInWebView.loadUrl(URL); // runs in a separate thread
  }

...
The Android Thread Model

- Main thread usually the UI thread (but sometimes not – see below)
- SDK NOT thread-safe
  - Other threads should NOT manipulate UI
  - Only do computation, then offer result to UI thread
- API to access UI thread:
  - Activity.runOnUiThread(Runnable myRunnable) runs the specified runnable object on the UI thread. (See GameSessionTest.java)
  - View.post(Runnable myRunnable) causes the Runnable to be added to the message queue to be run by the UI thread as it processes all its messages.
  - View.postDelayed(Runnable, long) adds Runnable to message queue after specified period.
  - The Handler class lets you perform the preceding post(...) and postDelayed(...) operations when you don’t have access to an active View. (see GameSession.java)
- UI thread subordinated to unit test thread (see section on Testing)
Outline

• Non-Functional Requirements
• Optimize Performance with Profiler
• Maximize Battery Life
• Optimize for Responsiveness
• Improve App Security
• Testing
Security Considerations for Mobile Devices

• Devices store valuable personal information
• Larger security “footprint” – more attack surfaces ⇒ more vulnerabilities
  – Existing threats are magnified (e.g. poorly secured browsers and mobile web sites)
  – Installed apps are sources of insecurity.
    • More apps. Unknown trustworthiness of authors in an open marketplace.
  – Sharing between apps.
  – Private data left behind on file system
• Device is inherently less secure
  – Portable
  – No user login
  – Typically weaker passwords used (due to difficulty of data entry)
  – Less readable screen and environmental distractions means security indications could be ignored
• Lesson: App developers have to share responsibility for security
Systematic Steps to App Security

• Do not: randomly implement security “stuff”
• Instead, brainstorm and define a threat model
  – What are the assets (data)? What is their value?
  – What are the attacks (theft, denial of service)? Where can the attacks originate (network, other apps, browser)
• Identify security tactics:
  – Detection: Determining that an attack is in progress, or loss has taken place.
  – Resistance: Making the loss more difficult to occur.
  – Mitigation: Limiting the degree of loss or breach.
  – Recovery: Restoration from loss.
• Implement tactics using security techniques:
  – Authentication (e.g. two-factor, certificates)
  – Access control (e.g. file ownership, encryption, certificates)
  – Audit trail (e.g. logs)
  – Data integrity (e.g. checksums, encryption)
  – Non-repudiation (e.g. logs, certificates)
Android-Specific Security Considerations

- **Good**: “Privilege-supported” OS
  - Processes “sandboxed” in user space
  - User files and databases are removed on uninstallation
  - Apps must request and be granted permissions (on installation): to system resources, content providers, resources of other apps
  - Apps must be “signed” by developer (however, self-signing allowed!)
  - Google verifies new apps installed in Android 4.3+

- **Bad**: No security through obscurity: Linux is open-source, APK file can be freely inspected, Java can be decompiled
  - Limited vetting process on Google Play (tests apps via QEMU full-system emulator*)
  - Privileges are enforced by installer (runtimes of hacked phones may not enforce privileges)

- **Things to look out for**:  
  - Leaving private data in files on the device and its SD card
  - Database hacking techniques – SQL injection
  - Your app being the Trojan horse
  - Secret literals left in code (e.g. special passwords)
  - Use of reversible security algorithms

* [https://jon.oberheide.org/files/summercon12-bouncer.pdf](https://jon.oberheide.org/files/summercon12-bouncer.pdf)
Examples of Permission Requests

- `<uses-permission
  android:name="android.permission.READ_CONTACTS"/>
- `<uses-permission android:name="android.permissionINTERNET"/>
- `<uses-permission
  android:name="android.permission.ACCESS_NETWORK_STATE"/>
- `<uses-permission
  android:name="android.permission.ACCESS_COARSE_LOCATION"/>
- `<uses-permission
  android:name="android.permission.ACCESS_FINE_LOCATION"/>
- `<uses-permission
  android:name="com.wiley.fordummies.androidsdk.tictactoe.LAUNCHACTIVITY"/>

– Example of a custom permission

Note: Permission elements must be outside the `<application>` block and inside the `<manifest>` block of the AndroidManifest.xml
Custom Permissions – Definition and Placement

Permission must be declared:

```xml
<permission
    android:name =
    "com.wiley.fordummies.androidsdk.tictactoe.LAUNCHACTIVITY"
    android:label="Launch Tic-Tac-Toe Activity"
    android:description="@string/permission_launch_activity"
    android:protectionLevel="normal"
/>
```

Then placed:

In the `AndroidManifest.xml` file outside the `<application>` block and inside the `<manifest>` block (same as `<uses-permission>` elements).
Custom Permission – Declaration of Need

- Declare need using `android:permission` attribute in activity definition in manifest file:

```xml
<activity
    android:name=".Login"
    android:label="@string/app_name"
    android:launchMode="standard"
    android:screenOrientation="portrait"
    android:permission="com.wiley.fordummies.androidsdk.tictactoe.LAUNCHACTIVITY">
...
</activity>
```
Custom Permission – Request

• `<uses-permission
   android:name="com.wiley.fordummies.androidsdk.tictactoe.LAUNCHACTIVITY"/>`

• Requested in any separate package

• But also in containing package!
Permission Checking in Android

• When a call is made to a system function: To prevent an unauthorized invocation
• When starting an Activity: To prevent an unauthorized application from launching the Activity of other applications
• When sending or receiving Broadcasts: To determine who can receive a Broadcast or send it to you
• When accessing, and operating on, a Content Provider: To prevent an unauthorized app from accessing the data in the Content Provider
• When binding to, or starting, a Service: To prevent an unauthorized application from using the Service.
Example logcat Entries During Permission Failures

02-28 12:48:00.864: ERROR/AndroidRuntime(378):
  java.lang.SecurityException: Permission Denial: starting Intent {
    act=com.wiley.fordummies.androidsdk.tictactoe.Login
    cmp=com.wiley.fordummies.androidsdk.tictactoe/.Login
  }
  from ProcessRecord{407740c0
    378:com.wiley.fordummies.androidsdk.tictactoe/10033} (pid=378, uid=10033)
  requires
    com.wiley.fordummies.androidsdk.tictactoe.permission.LAUNCHACTIVITY

02-28 21:04:39.758: ERROR/AndroidRuntime(914): at
  com.wiley.fordummies.androidsdk.tictactoe.SplashScreen$1.run
  (SplashScreen.java:36)
Example logcat Entries for Permission Definition or Placement Errors

02-28 16:53:09.838: DEBUG/PackageManager(77): Permissions: com.wiley.fordummies.androidsdk.tictactoe.LAUNCHACTIVITY

02-28 17:04:18.888: WARN/PackageParser(77): Unknown element under <application>:

  permission at /data/app/vmdl1654102309.tmp Binary XML file line #11

02-28 17:04:20.438: WARN/PackageManager(77): Unknown permission
  com.wiley.fordummies.androidsdk.tictactoe.LAUNCHACTIVITY in package
  com.wiley.fordummies.androidsdk.tictactoe
SQLite Security – SQL Injection

- Entry field: Name: <Enter Name>
- Intended query:
  - SELECT e-mail FROM user_information WHERE NAME='Bob'
- Attacker enters string:
  - ‘Bob’; SELECT table_names FROM user_tables
- Query becomes:
  - SELECT e-mail FROM user_information WHERE name='Bob'; SELECT table_names FROM user_tables
- Attacker knows all the tables. Augh!
private static final String TABLE_NAME = "Accounts";
...
private static final String INSERT = "insert into " +
    TABLE_NAME + "(name, password) values (?, ?)" ;
...
public DatabaseHelper(Context context) {
    ...
    this.insertStmt = this.db.compileStatement(INSERT);
    ...
}
...
public long insert(String name, String password) {
    this.insertStmt.bindString(1, name);
    this.insertStmt.bindString(2, password);
    return this.insertStmt.executeInsert();
}
General Rule – Minimize Vulnerable Spots in App

• Don’t hardwire “secrets” in code.
• Mask sensitive data entry (e.g. passwords)
• Encrypt files
• Don’t write unnecessary temporary files
• Use bind variables
• Ask for the least permissions
• Create checkpoints of application data
• Log data (of course, encrypt your logs)
• Keep Intent Filters specific so Activities don’t respond to generic Intents
• Don’t rely only on permissions when sensitive data is accessed – prompt user.
Outline

• Non-Functional Requirements
• Optimize Performance with Profiler
• Maximize Battery Life
• Optimize for Responsiveness
• Improve App Security
• Testing
Creating Unit Tests (1)

- In Android Studio, right-click the project name, select app, click on Dependencies tab, click “+” icon, select “Library Dependency”, then type “junit” into the dialog
- Create test classes under `<project-name>/app/src/androidTest/java/<package-name>`
- Set up test run configuration (of type Android Test)
Creating Unit Tests (2)
Creating Unit Tests (3)
Passed Unit Test
Failed Unit Test
Unit Test Class

public class GameSessionTest extends ActivityInstrumentationTestCase2
<GameSession> { // Template Class
    private GameSession gameSessionActivity; // Activity to be tested
    private Board board; // Member variable of activity
    private Instrumentation gameSessionActivityInstrumentation = null;
    // Data for the tests – touch coordinates
    final float x[]={(float)56.0, (float)143.0, (float)227.0};
    final float y[]={(float)56.0, (float)143.0, (float)227.0};
    int i = 0;
    public GameSessionTest() {...} // Constructor
    protected void setUp() throws Exception {...} // Setup – gets member variables
    public void testPreconditions() {...} // Test 1
    public void testUI() {...} // Test 2
    @UiThreadTest // Annotation to force the test to run in the UI thread
    public void testUIThreadTest() {...} // Test 3
    protected void tearDown() throws Exception {...} // Cleanup
}
public GameSessionTest() {
    super("com.wiley.fordummies.androidsdk.tictactoe.GameSession",
            GameSession.class);
}

Constructor
protected void setUp() throws Exception {
    // access to member variables
    super.setUp();
    setActivityInitialTouchMode(false);
    gameSessionActivityInstrumentation = getInstrumentation();
    gameSessionActivity = getActivity();
    board = (Board) gameSessionActivity.findViewById(R.id.board);
}

...
public void testPreconditions() {
    assertNotNull(gameSessionActivity);
    assertNotNull(board);
}
Test 2 – Test User Interface

```java
public void testUI() {
    System.out.println("Thread ID in testUI:" + Thread.currentThread().getId());
    getActivity().runOnUiThread(new Runnable() {
        public void run() {
            System.out.println("Thread ID in TestUI.run:" +
                                Thread.currentThread().getId());
            board.requestFocus();
            // Simulates touch event
            // Hint: Instrumented the onTouchEvent(MotionEvent event) to get good pixel
            values for touch. Why not call onTouchEvent of Board directly?
            MotionEvent newMotionEvent =
                MotionEvent.obtain((long)1, (long)1, MotionEvent.ACTION_DOWN,
                                    (float) 53.0, (float) 53.0, 0);
            board.dispatchTouchEvent(newMotionEvent); // Dispatches touch event
        }
    });
    assertEquals(gameSessionActivity.getPlayCount(), 2); // Assert 2 moves
}
```
Test 3 – Series of Moves

```java
final float x[] = {(float)56.0, (float)143.0, (float)227.0};
final float y[] = {(float)56.0, (float)143.0, (float)227.0};
int i = 0;
...

@UiThreadTest
public void testUiThreadTest() {
    System.out.println("Thread ID in testUI:" + Thread.currentThread().getId());
    board.requestFocus();
    for (i=0; i<3; i++){
        MotionEvent newMotionEvent = // Factory Method
            MotionEvent.obtain((long)1, (long)1,
            MotionEvent.ACTION_DOWN,
            (float) x[i], (float) y[i], 0);
        board.dispatchTouchEvent(newMotionEvent);
    }
    assertEquals(gameSessionActivity.getPlayCount(), 2);
}
```
Tests and Threading

- Must explicitly run certain tests on UI thread
  - Via Annotations
  - Via explicit command
- Main UI thread subordinated to unit test thread
- Main UI thread terminated when tests run
- Tasks queued for main UI thread may not launch!
private void scheduleAndroidsTurn() {
    System.out.println("Thread ID in scheduleAndroidsTurn:" +
            Thread.currentThread().getId());
    board.disableInput();
    if (!testMode) {
        Random randomNumber = new Random();
        Handler handler = new Handler();
        handler.postDelayed(new Runnable() {
            public void run() {
                androidTakesATurn();
            }
        },
            500 + randomNumber.nextInt(2000)
        );
    } else {
        androidTakesATurn();
    }
}
Useful Links for Testing

• See: [http://developer.android.com/reference/android/view/MotionEvent.html](http://developer.android.com/reference/android/view/MotionEvent.html) for details of MotionEvent class

• See: [http://developer.android.com/reference/android/view/View.html](http://developer.android.com/reference/android/view/View.html) for how to send an event to a View (Board is a subclass of View)

References

• Chapter 8: Making Your Application Fast and Responsive, from *Android SDK 3 Programming for Dummies*
  

• Jon Bentley, *Writing Efficient Programs*, [www.crowl.org/lawrence/programming/Bentley82.html](http://www.crowl.org/lawrence/programming/Bentley82.html)


• Chapter 12: Effectively Using Your Integration Development Environment
  
  – Covers Eclipse features, logging and unit testing (in Eclipse)