Agile in the Workplace

Cape Town Spin #89

Running Android Projects With Agile

By

Ewald Horn

Estimated Duration 2h30m

The Picture On The Left

- Has nothing to do with the talk.
- It's a template, just deal with it.
- I normally don't do presentations.
- We won't be doing Agile today.
- Really.
- We won't.
- Sorry to disappoint you.
- The next slide is all about me.

All About Me

- I work for a company called WyseTalk.
- Doing Android programming.
- No game development at the moment.
- We work hard (Promise).
- We need an iOS developer at the moment.
- That's enough about me.

wysetalk™
collaborate | innovate

This is the Presentation

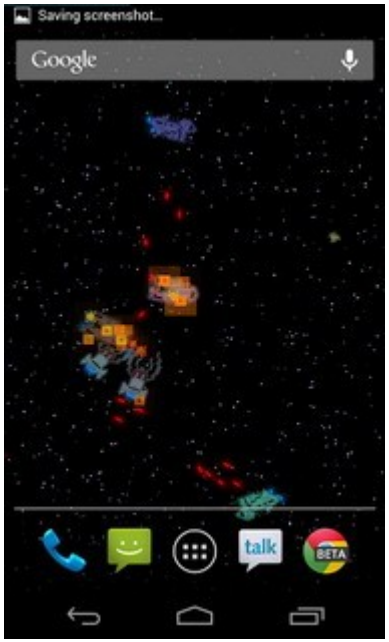
(there might be more slides)

Today is all about Android Live Wallpapers

- Live Wallpapers Are
  - Special Android applications for 2.1 or newer
  - Great battery drainers
  - Fun and easy to develop
  - A good start to get into game development
  - Really better served without sound
  - A great way to build brand awareness
  - A great marketing tool

Some Examples

- Visit http://www.livewallpapers.org/ for great live wallpaper examples



If you think some of these *look* like games, you are getting the idea.
They are popular because they entertain, that's why people love them.

Overview of a Live Wallpaper

- A Live Wallpaper is a Service, so
  - It runs all the time
  - Can access system resources
  - Consumes CPU cycles and Memory
  - Has access to the Android platform
  - Does not require the NDK
  - Needs to handle many events

Getting Started

- Very much like an Android application
- No Activity, obviously. (There are exceptions)
- You need android.permission.BIND_WALLPAPER
- You need some config files
- You need some config settings in the manifest file

(I will be showing code examples as we go along)

There's a great tutorial on http://www.vogella.com/articles/AndroidLiveWallpaper/article.html for those that are interested in rolling their own wallpapers.

Config File

- A file to describe your wallpaper

```
<?xml version="1.0" encoding="UTF-8"?>
<wallpaper
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:thumbnail="@drawable/ic_launcher"
  android:description="@string/app_description"
  android:settingsActivity="za.co.nofuss.livewallpaper.WallPaperPreferencesActivity"/>
```

This is a standard XML file used to describe your wallpaper and is referenced from your manifest file. What you are doing here is to describe your wallpaper to Android, and provide a launch graphic.

AndroidManifest.xml

Just like in an application, you need a manifest file.

The important bits are:

```
<application
    android:allowBackup="false"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name" >
    <service
        android:name="NoFussWallpaperService"
        android:enabled="true"
        android:label="NoFuss Live Wallpaper"
        android:permission="android.permission.BIND_WALLPAPER" >
        <intent-filter>
            <action android:name="android.service.wallpaper.WallpaperService" >
            </action>
        </intent-filter>

        <meta-data
            android:name="android.service.wallpaper"
            android:resource="@xml/livewallpaper" >
        </meta-data>
    </service>
```

Building Blocks

- The Live Wallpaper Service
  - Renders the wallpaper
  - Handles user interaction and events
- The Settings Activity
  - Handles user settings, where applicable

Settings First

Live Wallpapers make use of standard Android Settings, so you can use your application development knowledge.

All you need is a preferences.xml file.

Android can handle the rest for you.

This is the least amount of work, moving on.

This is optional, if you don't have settings, don't bother.

The Service

- Android provides WallpaperService, which we can extend.

```
public class NoFussWallpaperService extends WallpaperService
{
    @Override
    public Engine onCreateEngine()
    {
        return new NoFussWallpaperEngine(NoFussWallpaperService.this);
    }
```

A Live Wallpaper is a Service that has an Engine. We create an Engine, once again, extending a class Android provides for us.

```
private class NoFussWallpaperEngine extends Engine
```

This way we can have a lot of boiler-plate code in place already, something you do not get easily with the NDK.

Our Service

Our Wallpaper Service is only responsible for creating and instance of the Wallpaper Engine. Nothing else, the real code can be found in the Engine itself.

To create the Engine, we extend, conventiently, a class called Engine.

Wallpaper Engine

- Handles rendering
  - Drawing to the canvas
- Handles wallpaper lifecycle
  - The wallpaper can be loading
  - Can become invisible
  - Can be closed
- Handles user interactions
  - We want double-taps to mean something

Rendering

- Old-school style Canvas operations, nothing new or challenging here.
- You can go OpenGL if you need to.
- The only trick is to make sure your surface is available and visible.

Wallpaper Lifecycle

- Wallpapers are services, so they are always running.
- The surface might not be available, track that to prevent NPE's.
- Track visibility changes.
- Track service closing.

Interactions

- Optional to interact with the user.
- Can register a listener for touch events.
- Has access to gestures etc.
- GestureDetector makes it easier.

```
@Override
    public boolean onDoubleTap(MotionEvent e) {
        return true;
    }
```