Engineering Design w/Embedded Systems

Lecture 22: Lab 4; Android Graphics

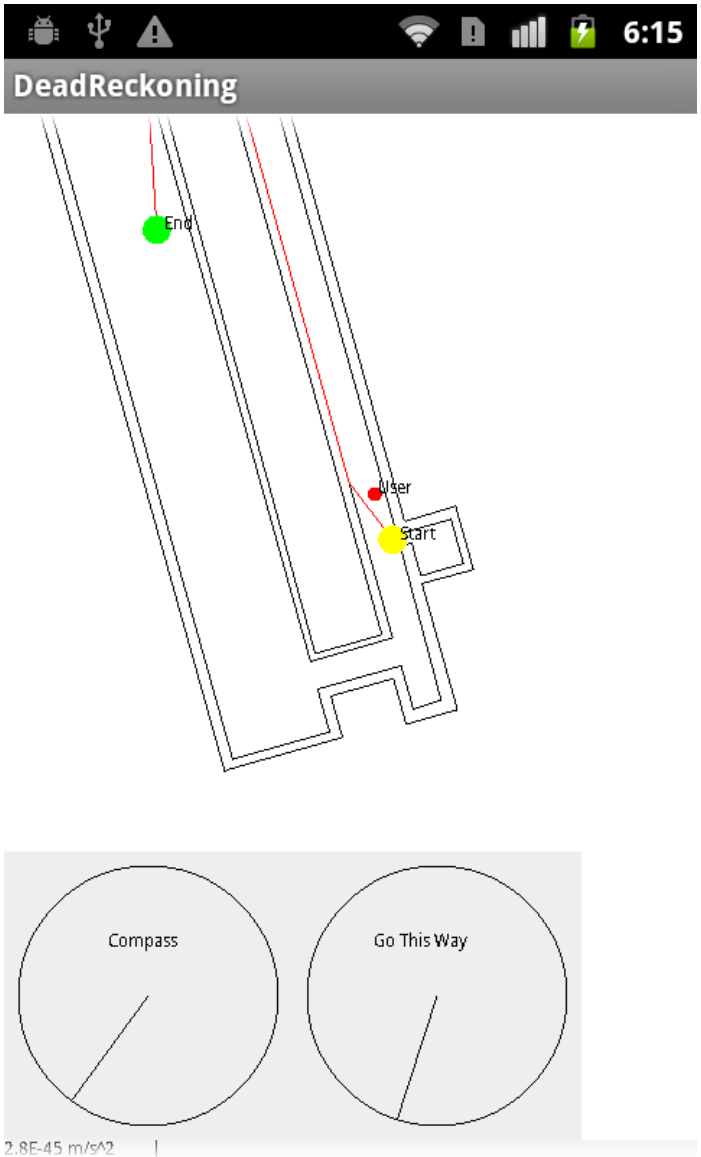
Patrick Lam
University of Waterloo

March 8, 2013

Part I

Lab 4 Discussion

Lab 4: screenshot



Lab 4

Goal: direct the user to a destination.

Lab 4: Logistics

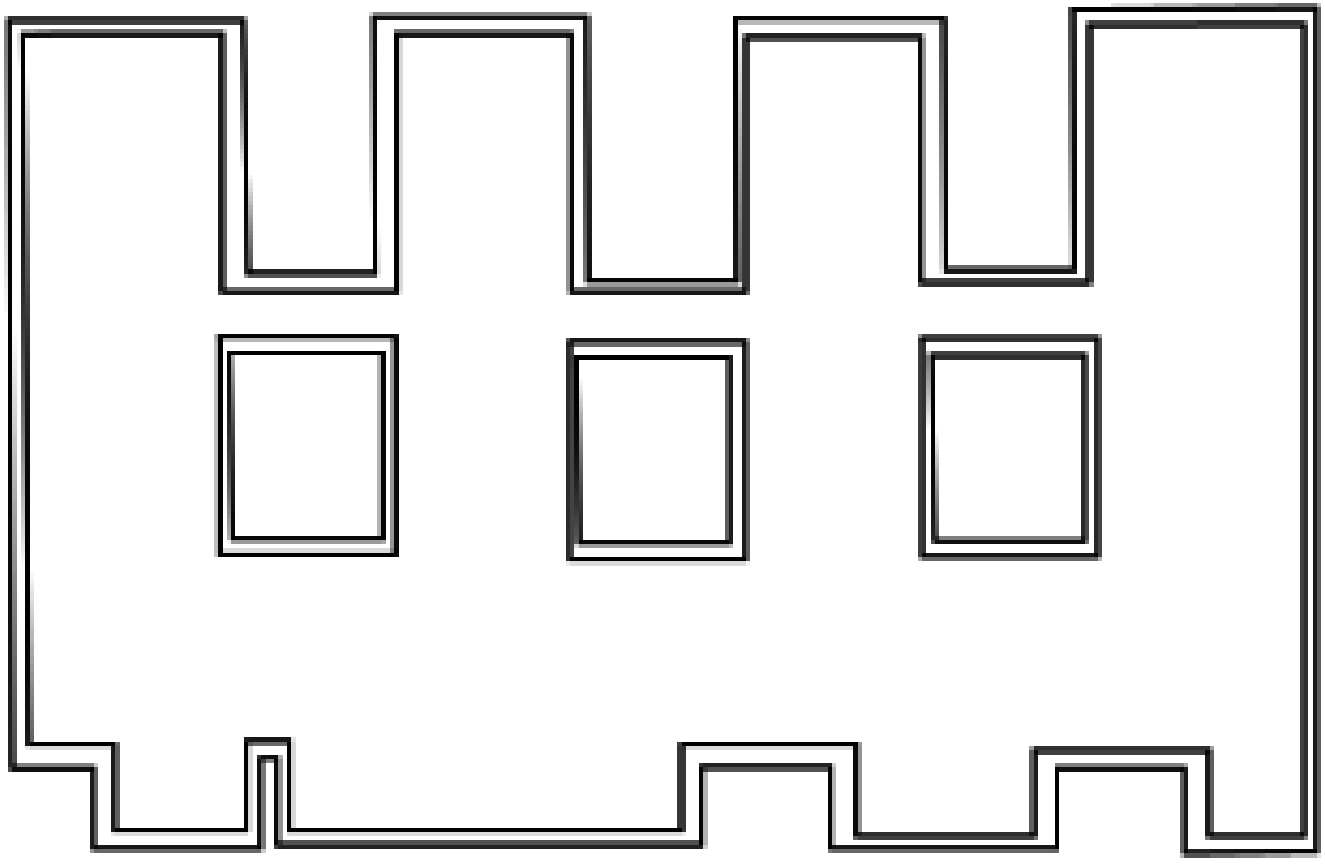
Due date: week of March 25.

MW labs: midway through your scheduled lab session.
F lab: by 5pm Thursday.

Lab 4: Inputs

- a map of the area (E2-2364);
- user-selected start and destination points;
- compass and pedometer readings

Lab 4: map



Lab 4: useful classes that we provide

`MapView`: like `LineGraphView`, but displays a map.

`PedometerMap`: contains the lines describing the room.

`MapLoader`: loads an SVG map into the `PedometerMap`.

`PositionListener`: handles callbacks from the `MapView` to your code.

Lab 4: approach (1)

First, track the user location:

- respond to the `locationChanged` callback, store the location, and call `setUserLocation`.
- update the location when the user makes a step (use Lab 3 code here).

You'll need to do a unit conversion from steps into meters here. We'll provide TA step lengths.

Lab 4: approach (2)

Main challenge of this lab:

- find a route from current point to destination.

Lab 4: route-finding

The map is continuous; want to find paths in the map that don't go through walls.

We will provide a graph class to discretize the map. You don't have to use it.

Using the graph, you can do a search for the destination point (more later).

Part II

Graphics on Android

XML versus Programmatic Construction

Use the right tool for the job!

XML = more safety:

- Select and place items ahead of time.
- Don't need the emulator to see how things will look.
- More error checking.

Java Code = more flexibility:

- Can choose widgets based on user input or computations.
- Can use loops, etc to generate related items.
- Less error checking.

What “inflate” means

These lines keep on showing up in our code:

```
// Inflate the menu; this adds items to the action bar if it is present.
getMenuInflater().inflate(R.menu.activity_main, menu);
```

“Inflate” = taking an XML and
creating `View` objects, based on the description in the XML.

Putting Graphics on the Screen

Two choices:

- use a `View` (easier; infrequent updates); or
- paint to a `Canvas` (more complicated, many updates).

Main class: `Drawable`

Represents “something that can be drawn”, e.g.

- BitmapDrawable
- ShapeDrawable
- PictureDrawable
- etc.

Drawing to a View

As always with Android, either:

- through XML; or
- programmatically.

Bitmaps through ImageView

Easiest way[1]:

- put a picture (`PNG`, `JPG` or `GIF`) in `res/drawables`.
- use an `ImageView` to include it on the screen.

from: `http://developer.android.com/guide/topics/graphics/2d-graphics.html`

```
// Instantiate an ImageView and define its properties programmatically
ImageView i = new ImageView(this);
i.setImageResource(R.drawable.my_image);
 // set the ImageView bounds to match the Drawable's dimensions
i.setAdjustViewBounds(true);
i.setLayoutParams(new Gallery.LayoutParams
    (LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));

// Add the ImageView to the layout and
//  set the layout as the content view
mLinearLayout.addView(i);
```

[1] Thanks to `http://www.cs.umd.edu/class/fall2010/CMSC498G/CMSC498G/Slides_files/Graphics.pptx`

Bitmaps through ImageView XML

Again, you need the appropriate drawable in the `res/drawables` directory.

Note: harder to go wrong here.

```
<ImageView
    android:id="@+id/imageView1"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:src="@drawable/myImage" />
```

Drawing on a ShapeDrawable

Primitive shapes:

- PathShape—lines;
- RectShape—rectangles;
- OvalShape—ovals and rings;

Once again, we put these into an `ImageView`.

Shapes from XML

Again, you need the appropriate drawable in the `res/drawables` directory.

Note: harder to go wrong here.
In the Layout XML:

```
<ImageView android:id="@+id/imageView2"
   android:src="@drawable/cyan_shape" ... />
```

Next, we create an XML for the drawable itself:

```
<shape android:shape="oval" ... >
  <size android:width="160px" android:height="160px" />
  <solid android:color="#7f00ffff" />
</shape>
```

Shapes, programmatically

Everything you can do in XML, you can do in code.

```
private class MyDrawableView extends ImageView {
    private ShapeDrawable mDrawable;
    public MyDrawableView(Context context, int color) {
        ...
        mDrawable = new ShapeDrawable(new OvalShape());
        mDrawable.getPaint().setColor(color);
        mDrawable.setBounds(0, 0, size, size);
        mDrawable.setAlpha(alpha);
    }
    protected void onDraw(Canvas canvas) {
        mDrawable.draw(canvas);
    }
}
```

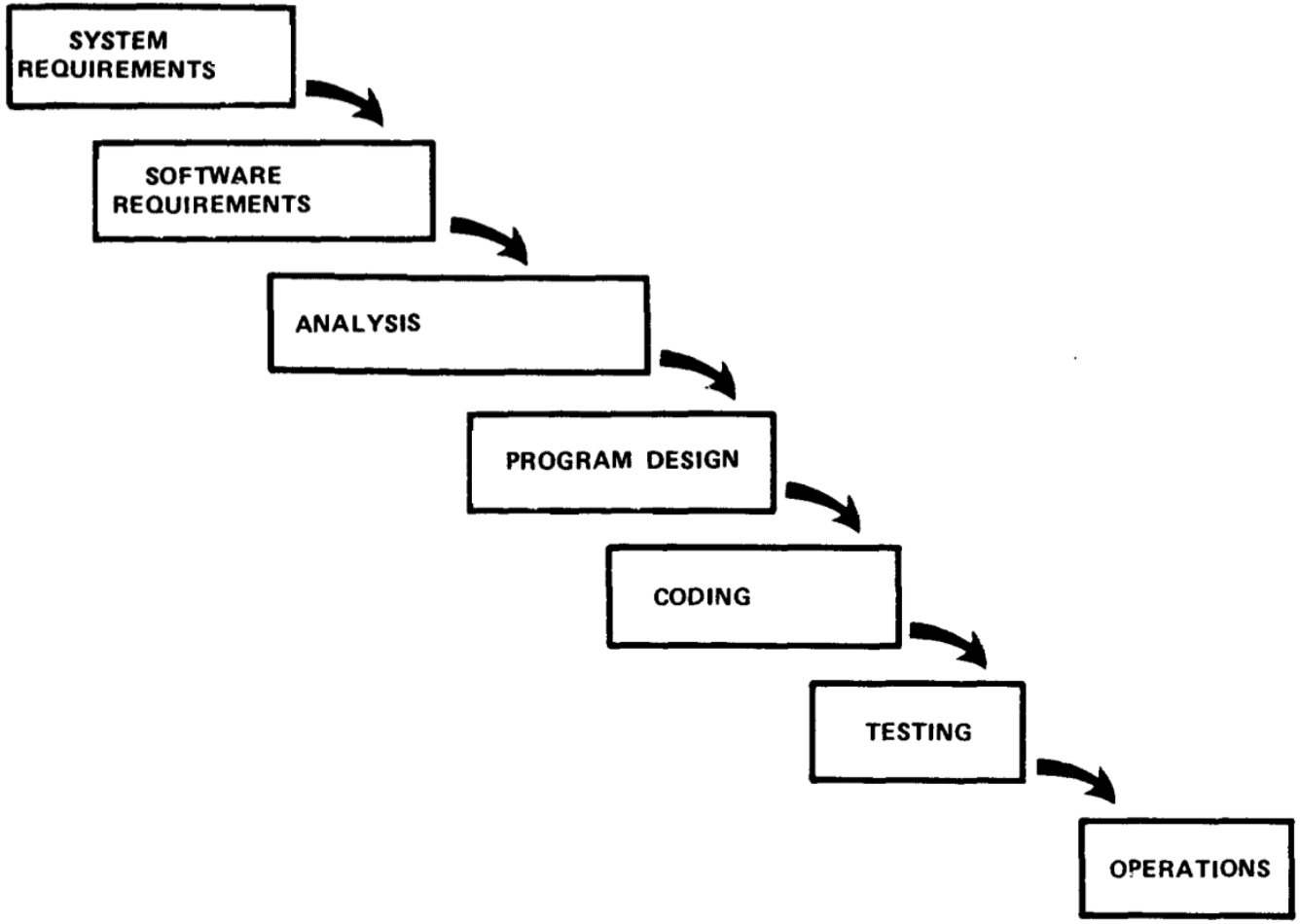
Shapes, programmatically

In the Activity's `onCreate()`:

```
MyDrawableView magentaView =
  new MyDrawableView(this, Color.MAGENTA);
magentaView.setLayoutParams
  (new LinearLayout.LayoutParams(160, 160));
addView(magentaView);
```

9-patches

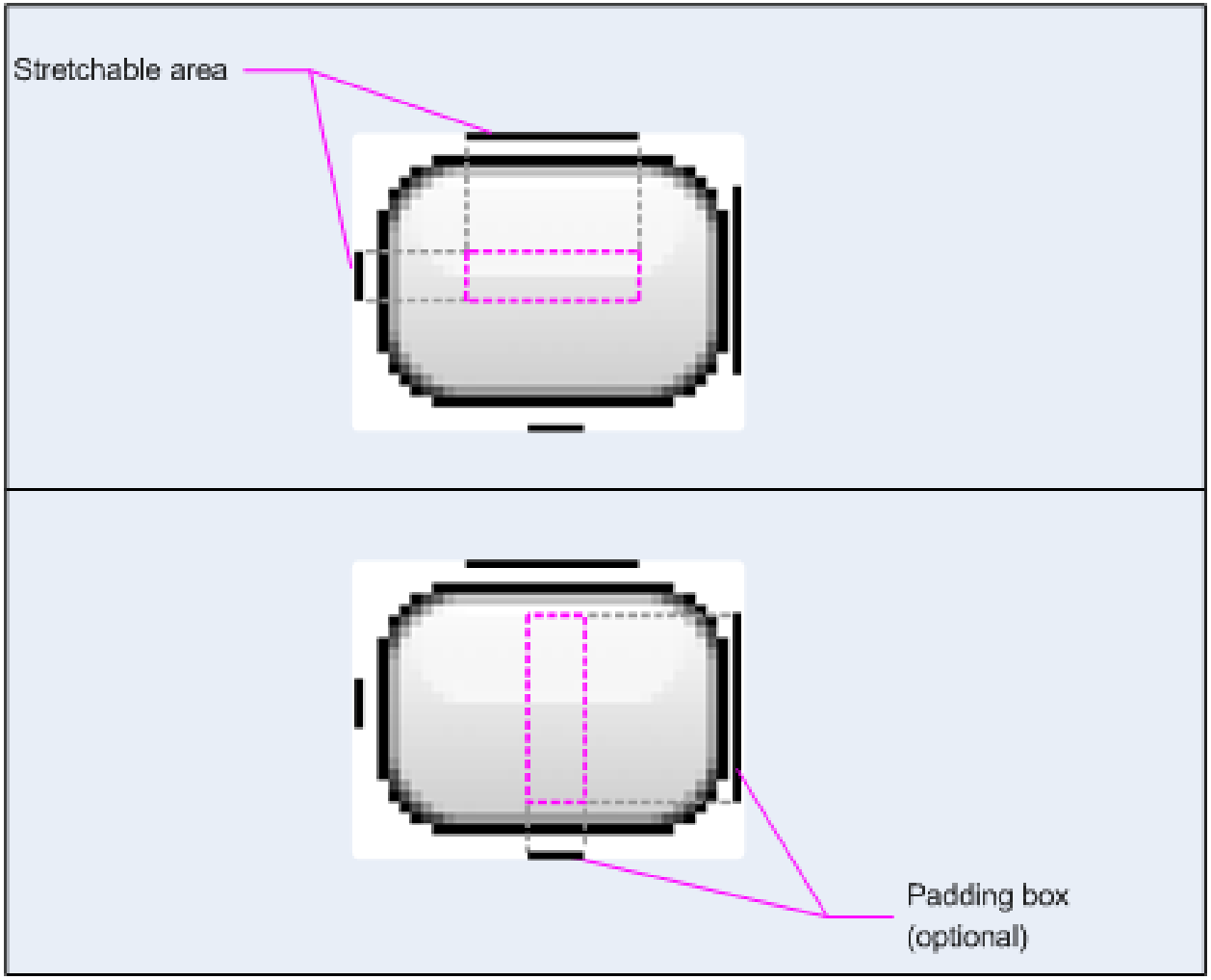
Remember this picture from yesterday?



I manually stretched the boxes using a graphics editor.

9-patches: automatic stretching

`NinePatchDrawable` can stretch your images automatically!



Just use a `.9.png` file. Edit using `tools/draw9patch`.