

```
1  /**
2   * Android: TouchImageView.java
3   * Created by: Mike Ortiz
4   * Updated by: Vince Pascuzzi
5   * Date: 3/14/2013
6   *
7   * Allows pinching, zooming, translating, and drawing on an ImageView.
8   */
9
10 package edu.seaaddicts.brockbutler.maps;
11
12 import android.content.Context;
13 import android.graphics.Canvas;
14 import android.graphics.Color;
15 import android.graphics.Matrix;
16 import android.graphics.Paint;
17 import android.graphics.PointF;
18 import android.graphics.drawable.Drawable;
19 import android.util.AttributeSet;
20 import android.util.Log;
21 import android.view.MotionEvent;
22 import android.view.ScaleGestureDetector;
23 import android.view.View;
24 import android.widget.ImageView;
25
26 public class MapsTouchImageView extends ImageView {
27     private static final String TAG = "MapsTouchImageView";
28
29     private static final int MAP_WIDTH = 2000;
30     private static final int MAP_HEIGHT = 1100;
31     private static final int CLICK = 3;
32
33     private Matrix mMatrixMap;
34
35     // States of touch.
36     private static final int NONE = 0;
37     private static final int DRAG = 1;
38     private static final int ZOOM = 2;
39     private int mode = NONE;
40     private int stroke = 8;
41
42     // Zooming variables.
43     private PointF last = new PointF();
44     private PointF start = new PointF();
45     private float minScale = 1f;
46     private float maxScale = 8f;
47     private float[] m;
48
49     // Ratio of screen resolution to map image resolution
50     private double mMapRatio;
51
52     private int viewWidth, viewHeight;
53     private int oldMeasuredWidth, oldMeasuredHeight;
54
55     private float scaleFactor = 1f;
56     private float origWidth, origHeight;
57
58     private final Paint mPathPaint = new Paint();
59
60     private ScaleGestureDetector mScaleDetector;
61
62     private Context mContext;
63     int actionBarHeight;
64
65     public Position[] mPosition = null;
66
67     public MapsTouchImageView(Context context) {
68         super(context);
69         sharedConstructing(context);
70     }
71 }
```

```

72     public MapsTouchImageView(Context context, AttributeSet attrs) {
73         super(context, attrs);
74         sharedConstructing(context);
75     }
76
77     @Override
78     protected void onDraw(Canvas canvas) {
79         super.onDraw(canvas);
80         mPathPaint.setColor(Color.CYAN);
81         mPathPaint.setStrokeWidth(stroke);
82         canvas.setMatrix(mMatrixMap);
83
84         if (mPosition != null) {
85             for (int i = 0; i < mPosition.length - 1; i++) {
86                 Position p = mPosition[i];
87                 Position q = mPosition[i + 1];
88                 int x1 = p.xPosition;
89                 int y1 = p.yPosition;
90                 float[] f1 = convertDimensions(x1, y1);
91                 int x2 = q.xPosition;
92                 int y2 = q.yPosition;
93                 float[] f2 = convertDimensions(x2, y2);
94                 canvas.drawLine(f1[0], f1[1], f2[0], f2[1], mPathPaint);
95             }
96         }
97     }
98
99     private void sharedConstructing(Context context) {
100         super.setClickable(true);
101         this.mContext = context;
102         mScaleDetector = new ScaleGestureDetector(context, new ScaleListener());
103         mMatrixMap = new Matrix();
104         m = new float[9];
105         setImageMatrix(mMatrixMap);
106         setScaleType(ScaleType.MATRIX);
107
108         setOnTouchListener(new OnTouchListener() {
109
110             public boolean onTouch(View v, MotionEvent event) {
111                 mScaleDetector.onTouchEvent(event);
112                 PointF curr = new PointF(event.getX(), event.getY());
113
114                 switch (event.getAction()) {
115                     case MotionEvent.ACTION_DOWN:
116                         last.set(curr);
117                         start.set(last);
118                         mode = DRAG;
119                         break;
120
121                     case MotionEvent.ACTION_MOVE:
122                         float fixTransX;
123                         float fixTransY;
124                         if (mode == DRAG) {
125                             float deltaX = curr.x - last.x;
126                             float deltaY = curr.y - last.y;
127                             fixTransX = getFixDragTrans(deltaX, viewWidth,
128                                 origWidth * scaleFactor);
129                             fixTransY = getFixDragTrans(deltaY, viewHeight,
130                                 origHeight * scaleFactor);
131                             mMatrixMap.postTranslate(fixTransX, fixTransY);
132                             fixTrans();
133                             last.set(curr.x, curr.y);
134                         }
135                         break;
136
137                     case MotionEvent.ACTION_UP:
138                         mode = NONE;
139                         int xDiff = (int) Math.abs(curr.x - start.x);
140                         int yDiff = (int) Math.abs(curr.y - start.y);
141
142                         if (xDiff < CLICK && yDiff < CLICK)

```

```
143         performClick();
144         break;
145
146         case MotionEvent.ACTION_POINTER_UP:
147             mode = NONE;
148             break;
149     }
150
151     setImageMatrix(mMatrixMap);
152     invalidate();
153     return true; // indicate event was handled
154 }
155
156 });
157 }
158
159 public void setMaxZoom(float x) {
160     maxScale = x;
161 }
162
163 private class ScaleListener extends
164     ScaleGestureDetector.SimpleOnScaleGestureListener {
165     @Override
166     public boolean onScaleBegin(ScaleGestureDetector detector) {
167         mode = ZOOM;
168         return true;
169     }
170
171     @Override
172     public boolean onScale(ScaleGestureDetector detector) {
173         float mScaleFactor = detector.getScaleFactor();
174         float origScale = scaleFactor;
175         scaleFactor *= mScaleFactor;
176         if (scaleFactor > maxScale) {
177             scaleFactor = maxScale;
178             mScaleFactor = maxScale / origScale;
179         } else if (scaleFactor < minScale) {
180             scaleFactor = minScale;
181             mScaleFactor = minScale / origScale;
182         }
183
184         if (origWidth * scaleFactor <= viewWidth
185             || origHeight * scaleFactor <= viewHeight)
186             mMatrixMap.postScale(mScaleFactor, mScaleFactor, viewWidth / 2,
187                 viewHeight / 2);
188         else
189             mMatrixMap.postScale(mScaleFactor, mScaleFactor,
190                 detector.getFocusX(), detector.getFocusY());
191         fixTrans();
192         return true;
193     }
194 }
195
196 void fixTrans() {
197     mMatrixMap.getValues(m);
198     float fixTransX;
199     float fixTransY;
200     float transX = m[Matrix.MTRANS_X];
201     float transY = m[Matrix.MTRANS_Y];
202
203     fixTransX = getFixTrans(transX, viewWidth, origWidth * scaleFactor);
204     fixTransY = getFixTrans(transY, viewHeight, origHeight * scaleFactor);
205
206     if (fixTransX != 0 || fixTransY != 0)
207         mMatrixMap.postTranslate(fixTransX, fixTransY);
208 }
209
210 /*
211  * Fixes (when required) the translation matrix.
212  */
213 float getFixTrans(float trans, float viewSize, float contentSize) {
```

```

214     float minTrans, maxTrans;
215
216     if (contentSize <= viewSize) {
217         minTrans = 0;
218         maxTrans = viewSize - contentSize;
219     } else {
220         minTrans = viewSize - contentSize;
221         maxTrans = 0;
222     }
223
224     if (trans < minTrans)
225         return -trans + minTrans;
226     if (trans > maxTrans)
227         return -trans + maxTrans;
228     return 0;
229 }
230
231 /*
232  * Adjusts the translation when dragging so that this stays in the correct
233  * location on screen.
234  */
235 float getFixDragTrans(float delta, float viewSize, float contentSize) {
236     if (contentSize <= viewSize) {
237         return 0;
238     }
239     return delta;
240 }
241
242 @Override
243 protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
244     super.onMeasure(widthMeasureSpec, heightMeasureSpec);
245     viewWidth = MeasureSpec.getSize(widthMeasureSpec);
246     viewHeight = MeasureSpec.getSize(heightMeasureSpec);
247
248     // Does image rescaling on rotation. Not necessary since our orientation
249     // is fixed in landscape.
250     if (oldMeasuredHeight == viewWidth && oldMeasuredHeight == viewHeight
251         || viewWidth == 0 || viewHeight == 0)
252         return;
253     oldMeasuredHeight = viewHeight;
254     oldMeasuredWidth = viewWidth;
255
256     if (scaleFactor == 1) {
257         // Fit to screen.
258         float scale;
259
260         Drawable drawable = getDrawable();
261         if (drawable == null || drawable.getIntrinsicWidth() == 0
262             || drawable.getIntrinsicHeight() == 0)
263             return;
264         int bmWidth = drawable.getIntrinsicWidth();
265         int bmHeight = drawable.getIntrinsicHeight();
266
267         mMapRatio = (double) (bmHeight) / (double) MAP_HEIGHT;
268
269         Log.d("bmSize", "bmWidth: " + bmWidth + " bmHeight : " + bmHeight
270             + "ratio" + mMapRatio);
271
272         float scaleX = (float) viewWidth / (float) bmWidth;
273         float scaleY = (float) viewHeight / (float) bmHeight;
274         scale = Math.min(scaleX, scaleY);
275         mMatrixMap.setScale(scale, scale);
276
277         // Center the image
278         float redundantYSpace = (float) viewHeight
279             - (scale * (float) bmHeight);
280         float redundantXSpace = (float) viewWidth
281             - (scale * (float) bmWidth);
282         redundantYSpace /= (float) 2;
283         redundantXSpace /= (float) 2;
284

```

```
285         mMapMatrix.postTranslate(redundantXSpace, redundantYSpace);
286
287         origWidth = viewWidth - 2 * redundantXSpace;
288         origHeight = viewHeight - 2 * redundantYSpace;
289         setImageMatrix(mMapMatrix);
290     }
291     fixTrans();
292 }
293
294 float[] convertDimensions(float x, float y) {
295     float f[] = new float[2];
296     f[0] = (float) mMapRatio * x;
297     f[1] = (float) mMapRatio * y;
298     return f;
299 }
300
301 public void drawPosition(Position[] p, int n) {
302     stroke = n;
303     mPosition = p;
304     invalidate();
305 }
306 }
```