

```

1  package edu.seaaddicts.brockbutler.maps;
2
3  /**
4   * Locate.java
5   * Brock Butler
6   * portion of Brock Butler.
7   * Created by Thomas Nelson 2013-03-10
8   * Copyright (c) 2013 Sea Addicts. All rights reserved.
9   */
10
11  import java.util.List;
12  import android.content.Context;
13  import android.net.wifi.ScanResult;
14  import android.net.wifi.WifiManager;
15  import android.util.Log;
16
17  public class Locate {
18
19      /**
20       * Class variables
21       */
22      private static WifiManager      wifiMgr;
23      private static List<ScanResult> scanResults;
24      static Context parentContext;
25      int[] answer = new int[10];
26      /**
27       * Wireless information containers
28       */
29      private static int[] sigStr  = new int[10];
30      private static String[] address = new String[10];
31      private static double[] addIn = new double[10];
32      /**
33       * Layers
34       */
35      private static final int inputs = 2;
36      private static final int hidden = 8;
37      private static final int output = 5;
38      /**
39       * Weights
40       */
41      private static double[][] W = new double[inputs][hidden];
42      private static double[][] V = new double[hidden][output];
43      private static double[] HB = new double[hidden];
44      private static double[] OB = new double[output];
45      /**
46       * Neurons
47       */
48      private static double[] hiddenVal = new double[hidden];
49      private static double[] outputVal = new double[output];
50      private static double[][] inputVal = new double[10][inputs];
51
52      /**
53       * Constructor for the Locate class sets context and initializes weights
54       * only once.
55       * @param pc
56       */
57      public Locate (Context pc) {
58          parentContext = pc;
59          initWeights();
60      }
61
62      /**
63       * Used by the mapping thread to get current user position, returns null if no
64       * position is found.
65       * @return
66       */
67      public String getUserPosition ( ) {
68          try {
69              //getWirelessData(); // Use when you are testing on device
70              initTestData(); // Use when testing on simulator testData1, testData2, or
71                          testData3

```

```
70      //initData(); // Use when you are testing on device
71
72      for(int i=0; i<10; i++) {
73          calcNetwork(i);
74          answer[i] = (int) (outputVal[0]*16 + outputVal[1]*8 + outputVal[2]*4 +
75              outputVal[3]*2 + outputVal[4]*1);
76      }
77
78      int location = mode(answer);
79      Log.i("LOCATE", "Node: " + location);
80      switch(location) {
81          case 1:
82              return "J01";
83          case 2:
84              return "J02";
85          case 3:
86              return "J03";
87          case 4:
88              return "J04";
89          case 5:
90              return "J05";
91          case 6:
92              return "J06";
93          case 7:
94              return "J07";
95          case 8:
96              return "J08";
97          case 9:
98              return "J09";
99          case 10:
100              return "J10";
101          case 11:
102              return "J11";
103          case 12:
104              return "J12";
105          case 13:
106              return "J13";
107          case 14:
108              return "J14";
109          case 15:
110              return "J15";
111          case 16:
112              return "J16";
113          case 17:
114              return "J17";
115          case 18:
116              return "J18";
117          case 19:
118              return "J19";
119          case 20:
120              return "J20";
121          case 21:
122              return "J21";
123          case 22:
124              return "J22";
125          case 23:
126              return "J23";
127          default:
128              return "";
129      } catch (Exception err) {
130          Log.e("LOCATE", err.getMessage());
131      }
132      return null;
133  }
134
135  /**
136   * Gathers wireless information from the device for 10 wireless access points
137   * currently in range. Gathers MAC address and received signal strength
138   */
139  @SuppressWarnings("unused")
```

```

140 private static void getWirelessData() {
141     wifiMgr = (WifiManager)parentContext.getSystemService(Context.WIFI_SERVICE);
142     int x = 0;
143
144     for(int num=0; num<10; num++) {
145         wifiMgr.startScan();
146         scanResults = wifiMgr.getScanResults();
147
148         x = 0;
149         sigStr = new int[10];
150         address = new String[10];
151
152         for(ScanResult scanRes : scanResults) {
153             if(x < 10) {
154                 address[x] = scanRes.BSSID;
155                 sigStr[x] = scanRes.level;
156                 x++;
157             }
158         }
159     }
160 }
161
162 /**
163  * This Method will return the sigmoid value of an argument
164  * for the final node value.
165  * @param x
166  * @return
167  */
168 private static double sigmoid(double x) {
169     return 1 / (1 + Math.exp(-x));
170 }
171
172 /**
173  * The beans of this class, uses normalized wireless data to predict
174  * user location based on an inputed input pattern.
175  * @param pat
176  */
177 private static void calcNetwork(int pat) {
178     for(int h=0; h<hidden; h++) {
179         hiddenVal[h] = -HB[h];
180         for(int i=0; i<inputs; i++) {
181             hiddenVal[h] += (inputVal[pat][i] * W[i][h]);
182         }
183         hiddenVal[h] = sigmoid(hiddenVal[h]);
184     }
185
186     for(int o=0; o<output; o++) {
187         outputVal[o] = -OB[o];
188         for(int h=0; h<hidden; h++) {
189             outputVal[o] += (hiddenVal[h] * V[h][o]);
190         }
191         outputVal[o] = sigmoid(outputVal[o]);
192
193         if(outputVal[o] >= 0.5)
194             outputVal[o] = 1;
195         else if(outputVal[o] < 0.5)
196             outputVal[o] = 0;
197     }
198 }
199
200 /**
201  * Searches through network output to find the most likely
202  * user position.
203  * @param a
204  * @return
205  */
206 private static int mode(int a[]) {
207     int maxValue=0, maxCount=0;
208
209     for (int i = 0; i < a.length; ++i) {
210         int count = 0;

```

```

211     for (int j = 0; j < a.length; ++j) {
212         if (a[j] == a[i]) ++count;
213     }
214     if (count > maxCount) {
215         maxCount = count;
216         maxValue = a[i];
217     }
218 }
219
220 return maxValue;
221 }
222
223 private void initTestData() {
224     inputVal[0][0] = 1; inputVal[0][1] = -65;
225     inputVal[1][0] = 2; inputVal[1][1] = -60;
226     inputVal[2][0] = 3; inputVal[2][1] = -64;
227     inputVal[3][0] = 4; inputVal[3][1] = -64;
228     inputVal[4][0] = 5; inputVal[4][1] = -68;
229     inputVal[5][0] = 6; inputVal[5][1] = -69;
230     inputVal[6][0] = 7; inputVal[6][1] = -69;
231     inputVal[7][0] = 8; inputVal[7][1] = -72;
232     inputVal[8][0] = 9; inputVal[8][1] = -74;
233     inputVal[9][0] = 10; inputVal[9][1] = -74;
234 }
235
236 @SuppressWarnings("unused")
237 private void initTestData2() {
238     inputVal[0][0] = 1; inputVal[0][1] = -65;
239     inputVal[1][0] = 1; inputVal[1][1] = -89;
240     inputVal[2][0] = 6; inputVal[2][1] = -64;
241     inputVal[3][0] = 8; inputVal[3][1] = -64;
242     inputVal[4][0] = 11; inputVal[4][1] = -66;
243     inputVal[5][0] = 12; inputVal[5][1] = -66;
244     inputVal[6][0] = 13; inputVal[6][1] = -71;
245     inputVal[7][0] = 14; inputVal[7][1] = -72;
246     inputVal[8][0] = 15; inputVal[8][1] = -72;
247     inputVal[9][0] = 16; inputVal[9][1] = -72;
248 }
249
250 @SuppressWarnings("unused")
251 private void initTestData3() {
252     inputVal[0][0] = 6; inputVal[0][1] = -58;
253     inputVal[1][0] = 8; inputVal[1][1] = -58;
254     inputVal[2][0] = 11; inputVal[2][1] = -72;
255     inputVal[3][0] = 12; inputVal[3][1] = -72;
256     inputVal[4][0] = 13; inputVal[4][1] = -75;
257     inputVal[5][0] = 14; inputVal[5][1] = -66;
258     inputVal[6][0] = 15; inputVal[6][1] = -77;
259     inputVal[7][0] = 16; inputVal[7][1] = -75;
260     inputVal[8][0] = 9; inputVal[8][1] = -69;
261     inputVal[9][0] = 16; inputVal[9][1] = -72;
262 }
263
264 /**
265  * Makes the wireless data usable and sets it up for
266  * the network to use by putting the values between 0 and 1
267  * with min/max normalization.
268  */
269 @SuppressWarnings("unused")
270 private void initData() {
271     for (int x=0; x<10; x++) {
272         if (address[x].equalsIgnoreCase("00:0b:86:91:ce:a1"))
273             addIn[x] = 1;
274         else if (address[x].equalsIgnoreCase("00:0b:86:8a:8c:02"))
275             addIn[x] = 2;
276         else if (address[x].equalsIgnoreCase("00:1a:1e:fc:af:21"))
277             addIn[x] = 3;
278         else if (address[x].equalsIgnoreCase("00:0b:86:91:ce:a2"))
279             addIn[x] = 4;
280         else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b0:e2"))
281             addIn[x] = 5;

```

```

282     else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b0:e1"))
283         addIn[x] = 6;
284     else if (address[x].equalsIgnoreCase("00:0b:86:89:f6:e1"))
285         addIn[x] = 7;
286     else if (address[x].equalsIgnoreCase("00:1a:1e:fc:af:22"))
287         addIn[x] = 8;
288     else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b2:62"))
289         addIn[x] = 9;
290     else if (address[x].equalsIgnoreCase("00:0b:86:4d:8f:21"))
291         addIn[x] = 10;
292     else if (address[x].equalsIgnoreCase("00:0b:86:4d:8f:22"))
293         addIn[x] = 11;
294     else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b0:21"))
295         addIn[x] = 12;
296     else if (address[x].equalsIgnoreCase("00:1a:1e:a7:dc:22"))
297         addIn[x] = 13;
298     else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b0:22"))
299         addIn[x] = 14;
300     else if (address[x].equalsIgnoreCase("00:1a:1e:a7:dc:21"))
301         addIn[x] = 15;
302     else if (address[x].equalsIgnoreCase("00:0b:86:8a:8c:01"))
303         addIn[x] = 16;
304     else if (address[x].equalsIgnoreCase("00:1a:1e:a7:e4:c2"))
305         addIn[x] = 17;
306     else if (address[x].equalsIgnoreCase("00:1a:1e:a7:e4:c1"))
307         addIn[x] = 18;
308     else if (address[x].equalsIgnoreCase("00:1a:1e:fc:ac:82"))
309         addIn[x] = 19;
310     else if (address[x].equalsIgnoreCase("00:1a:1e:fc:ac:81"))
311         addIn[x] = 20;
312     else if (address[x].equalsIgnoreCase("00:0b:86:91:ce:a0"))
313         addIn[x] = 21;
314     else if (address[x].equalsIgnoreCase("00:0b:86:8a:8c:00"))
315         addIn[x] = 22;
316     else if (address[x].equalsIgnoreCase("00:1a:1e:fc:ac:80"))
317         addIn[x] = 23;
318     else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b2:61"))
319         addIn[x] = 24;
320     else if (address[x].equalsIgnoreCase("00:0b:86:42:de:80"))
321         addIn[x] = 25;
322     else if (address[x].equalsIgnoreCase("00:0b:86:42:de:82"))
323         addIn[x] = 26;
324     else if (address[x].equalsIgnoreCase("00:1a:1e:a7:e4:c0"))
325         addIn[x] = 27;
326     else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b2:60"))
327         addIn[x] = 28;
328     else
329         addIn[x] = 0;
330 }
331
332 for (int x=0; x<10; x++) {
333     inputVal[x][0] = (addIn[x] - 1) / 28;
334     inputVal[x][1] = (sigStr[x] - -97) / 58;
335 }
336 }
337
338 /**
339  * Initializes the network with pre-defined weights, currently will only find a
340  * position in JBlock.
341  */
342 private void initWeights ( ) {
343     W[0][0] = -5.191953555370145;
344     W[1][0] = 8.311119623052747;
345     HB[0] = 13.645070679100112;
346     V[0][0] = -1.4188817448241078;
347     OB[0] = 8.485931304116875;
348     V[0][1] = -0.12644482595532947;
349     OB[1] = 7.676266130312892;
350     V[0][2] = -0.8742897792429708;
351     OB[2] = 7.71163044603109;
352     V[0][3] = 8.174930730213324;

```

```
353     OB[3] = 5.050964297399435;
354     V[0][4] = -1.483533992384484;
355     OB[4] = -0.9537318879960314;
356     W[0][1] = 1.2747639658533194;
357     W[1][1] = 35.85241447388153;
358     HB[1] = 22.187068222811735;
359     V[1][0] = 0.08863280595533382;
360     OB[0] = 8.485931304116875;
361     V[1][1] = -0.711381178420192;
362     OB[1] = 7.676266130312892;
363     V[1][2] = 0.47717648845370575;
364     OB[2] = 7.71163044603109;
365     V[1][3] = -0.9106351920878477;
366     OB[3] = 5.050964297399435;
367     V[1][4] = -21.858140121995646;
368     OB[4] = -0.9537318879960314;
369     W[0][2] = 72.6159523077509;
370     W[1][2] = 62.83113577555713;
371     HB[2] = 48.81891128647234;
372     V[2][0] = -0.2564556186462736;
373     OB[0] = 8.485931304116875;
374     V[2][1] = -1.3189848572962326;
375     OB[1] = 7.676266130312892;
376     V[2][2] = -1.132371452477312;
377     OB[2] = 7.71163044603109;
378     V[2][3] = 6.23102460219137;
379     OB[3] = 5.050964297399435;
380     V[2][4] = 0.9547440817617039;
381     OB[4] = -0.9537318879960314;
382     W[0][3] = -12.93760218402065;
383     W[1][3] = -27.44465213223537;
384     HB[3] = -12.785991014176767;
385     V[3][0] = -0.3900489003340711;
386     OB[0] = 8.485931304116875;
387     V[3][1] = -0.4526342230399839;
388     OB[1] = 7.676266130312892;
389     V[3][2] = -1.1900195982331039;
390     OB[2] = 7.71163044603109;
391     V[3][3] = 32.266073798358136;
392     OB[3] = 5.050964297399435;
393     V[3][4] = 0.8870702490085969;
394     OB[4] = -0.9537318879960314;
395     W[0][4] = 20.564192571900584;
396     W[1][4] = 64.556122443447;
397     HB[4] = 51.94209137066473;
398     V[4][0] = -0.14542636219949362;
399     OB[0] = 8.485931304116875;
400     V[4][1] = -0.18725148477033168;
401     OB[1] = 7.676266130312892;
402     V[4][2] = -1.3082405929431982;
403     OB[2] = 7.71163044603109;
404     V[4][3] = 7.750490464388548;
405     OB[3] = 5.050964297399435;
406     V[4][4] = 7.381930098882;
407     OB[4] = -0.9537318879960314;
408     W[0][5] = 26.203506852804754;
409     W[1][5] = 30.430410744252843;
410     HB[5] = 30.471677989844203;
411     V[5][0] = 0.6785829497339613;
412     OB[0] = 8.485931304116875;
413     V[5][1] = -0.7290230440401178;
414     OB[1] = 7.676266130312892;
415     V[5][2] = 0.01863788955350768;
416     OB[2] = 7.71163044603109;
417     V[5][3] = 13.290562812876107;
418     OB[3] = 5.050964297399435;
419     V[5][4] = -3.191226914646271;
420     OB[4] = -0.9537318879960314;
421     W[0][6] = -4.992559685945157;
422     W[1][6] = 88.04743967482369;
423     HB[6] = 50.39739242687603;
```

```
424     V[6][0] = 0.0746839836828444;  
425     OB[0] = 8.485931304116875;  
426     V[6][1] = -0.5879731640198912;  
427     OB[1] = 7.676266130312892;  
428     V[6][2] = -0.25250658203018556;  
429     OB[2] = 7.71163044603109;  
430     V[6][3] = 1.4602581997464505;  
431     OB[3] = 5.050964297399435;  
432     V[6][4] = 18.463904409636967;  
433     OB[4] = -0.9537318879960314;  
434     W[0][7] = -4.74928985780168;  
435     W[1][7] = 40.78597102067532;  
436     HB[7] = 35.79898233956728;  
437     V[7][0] = -0.09439880889024627;  
438     OB[0] = 8.485931304116875;  
439     V[7][1] = -1.1230801185236317;  
440     OB[1] = 7.676266130312892;  
441     V[7][2] = -0.2674705623236563;  
442     OB[2] = 7.71163044603109;  
443     V[7][3] = 8.441254951069187;  
444     OB[3] = 5.050964297399435;  
445     V[7][4] = -20.04591503798929;  
446     OB[4] = -0.9537318879960314;  
447 }  
448 }  
449  
450
```

```

1  package edu.seaaddicts.brockbutler.maps;
2
3  import android.app.Activity;
4  import android.app.AlertDialog;
5  import android.content.DialogInterface;
6  import android.content.Intent;
7  import android.os.Bundle;
8  import android.os.Handler;
9  import android.os.Message;
10 import android.util.Log;
11 import android.view.Menu;
12 import android.view.MenuItem;
13 import android.view.View;
14 import android.view.Window;
15 import android.widget.EditText;
16 import android.widget.LinearLayout;
17 import android.widget.RelativeLayout;
18 import android.widget.Toast;
19 import edu.seaaddicts.brockbutler.R;
20 import edu.seaaddicts.brockbutler.help.HelpActivity;
21
22 public class MapsActivity extends Activity {
23     private static final String TAG = "MapsActivity";
24
25     private EditText mSearchEditText;
26
27     private Handler mHandler;
28     private MapsTouchImageView mMapImage;
29     private MapsHandler mMapHandler;
30
31     private Position mStartPosition;
32     private Position mGoalPosition;
33     private Astar school;
34
35     @Override
36     protected void onCreate(Bundle savedInstanceState) {
37         super.onCreate(savedInstanceState);
38         getWindow().requestFeature(Window.FEATURE_ACTION_BAR_OVERLAY);
39         setContentView(R.layout.activity_map);
40
41         RelativeLayout layout = (RelativeLayout) findViewById(R.id.maps_layout);
42         layout.setSystemUiVisibility(View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN);
43         getActionBar().setBackgroundDrawable(
44             getResources().getDrawable(R.drawable.actionbar_bg));
45         init();
46
47         mHandler = new Handler() {
48
49             // Handles Messages sent from Thomas.
50             @Override
51             public void handleMessage(Message msg) {
52                 switch (msg.what) {
53                     case MapsHandler.MAPS_REQUEST_UPDATE:
54                         Log.d("MAIN HANDLER", "YAYAAA!!!");
55                         break;
56                     case MapsHandler.THREAD_UPDATE_POSITION:
57                         Log.d("TEST", msg.getData().getString("pos"));
58                         break;
59                     default:
60                         Log.d(TAG, "-----+++++ Got THREAD_UPDATE_POSITION message. +++++-----");
61                         break;
62                 }
63             }
64         };
65         mMapHandler = new MapsHandler(mHandler, this);
66         school = new Astar();
67     }
68
69     @Override
70     public boolean onCreateOptionsMenu(Menu menu) {
71         // Inflate the menu; this adds items to the action bar if it is present.

```



```

72     getMenuInflater().inflate(R.menu.activity_map, menu);
73     return true;
74 }
75
76 private void init() {
77     mMapImage = (MapsTouchImageView) findViewById(R.id.imgv_map);
78     // mMapImage.setOnClickListener(new OnClickListener() {
79     //
80     //     public void onClick(View v) {
81     //         mMapsHandler.sendMessage(MapsHandler.MAPS_REQUEST_UPDATE);
82     //     }
83     // });
84     // start = (Button) findViewById(R.id.btnstart);
85     // start.setOnClickListener(new OnClickListener() {
86     //
87     //     public void onClick(View v) {
88     //         mMapsHandler.sendMessage(MapsHandler.THREAD_REQUEST_START);
89     //     }
90     // });
91     // stop = (Button) findViewById(R.id.btnstop);
92     // stop.setOnClickListener(new OnClickListener() {
93     //
94     //     public void onClick(View v) {
95     //         mMapsHandler.sendMessage(MapsHandler.THREAD_REQUEST_PAUSE);
96     //     }
97     // });
98     //
99     // resume = (Button) findViewById(R.id.btnresume);
100    // resume.setOnClickListener(new OnClickListener() {
101    //
102    //     public void onClick(View v) {
103    //         mMapsHandler
104    //             .sendMessage(MapsHandler.THREAD_REQUEST_RESUME);
105    //     }
106    // });
107 }
108
109 @Override
110 public void onBackPressed() {
111     mMapsHandler.sendMessage(MapsHandler.THREAD_REQUEST_PAUSE);
112     mMapsHandler = null;
113     super.onBackPressed();
114 }
115
116 /**
117  *
118  * @param item
119  */
120 public void exitMaps(MenuItem item) {
121     onBackPressed();
122 }
123
124 /**
125  * Prompts user to search for existence of a location.
126  *
127  * @param item
128  */
129 public void displaySearchDialog(MenuItem item) {
130     AlertDialog.Builder editalert = new AlertDialog.Builder(this);
131
132     editalert.setTitle("MChown Location Search");
133     editalert.setMessage("Enter block or room name (i.e. B314)");
134
135     mSearchEditText = new EditText(this);
136     mSearchEditText.setSingleLine(true);
137     LinearLayout.LayoutParams lp = new LinearLayout.LayoutParams(
138         LinearLayout.LayoutParams.MATCH_PARENT,
139         LinearLayout.LayoutParams.MATCH_PARENT);
140     mSearchEditText.setLayoutParams(lp);
141     editalert.setView(mSearchEditText);
142

```

```

143     editalert.setPositiveButton("Search",
144         new DialogInterface.OnClickListener() {
145             public void onClick(DialogInterface dialog, int whichButton) {
146                 try {
147                     Position p1, p2=new Position(), p3=new Position(), p4=new Position(),
148                         p5=new Position();
149
150                     p1 = school.findPosition(mSearchEditText.getText().toString());
151
152                     p2.xPosition = p1.xPosition - 1;
153                     p2.yPosition = p1.yPosition - 1;
154
155                     p3.xPosition = p1.xPosition + 1;
156                     p3.yPosition = p1.yPosition + 1;
157
158                     p4.xPosition = p1.xPosition + 1;
159                     p4.yPosition = p1.yPosition - 1;
160
161                     p5.xPosition = p1.xPosition - 1;
162                     p5.yPosition = p1.yPosition + 1;
163
164                     Position[] pTest = {p5,p3,p4,p2,p5,p4,p3,p2};
165
166                     if(p1 != null && p2 != null)
167                         mMapImage.drawPosition(pTest,30);
168                 }
169                 catch (NullPointerException e) {
170                     Toast.makeText(getApplicationContext(), "Location not found",Toast.
171                         LENGTH_LONG).show();
172                 }
173             }
174         });
175     editalert.setNegativeButton("Cancel",
176         new DialogInterface.OnClickListener() {
177             public void onClick(DialogInterface dialog, int whichButton) {
178                 // do nothing
179             }
180         });
181     editalert.show();
182 }
183 /**
184  * Displays AlertDialog for user to enter destination. First the location is
185  * determined to exist, then if true path is drawn on map.
186  *
187  * @param item
188  */
189 public void displayGetDirectionsDialog(MenuItem item) {
190     school = new Astar();
191     AlertDialog.Builder alert = new AlertDialog.Builder(this);
192
193     alert.setTitle("Get Directions");
194     alert.setMessage("Enter Start and End Points");
195
196     LinearLayout lilal= new LinearLayout(this);
197     lilal.setOrientation(1); //1 is for vertical orientation
198     final EditText input = new EditText(this);
199     final EditText input1 = new EditText(this);
200     lilal.addView(input);
201     lilal.addView(input1);
202     alert.setView(lilal);
203
204     alert.setPositiveButton("Ok", new DialogInterface.OnClickListener() {
205         public void onClick(DialogInterface dialog, int whichButton) {
206             try {
207                 mStartPosition = school.findPosition(input.getText().toString());
208                 mGoalPosition = school.findPosition(input1.getText().toString());
209
210                 Position[] p = school.pathGeneration(mStartPosition, mGoalPosition);
211                 if(mStartPosition != null && mGoalPosition != null)

```

```
212         mMapImage.drawPosition(p,8);
213     }
214     catch (NullPointerException e){
215         Toast.makeText(getApplicationContext(), "Location not found",Toast.
216             LENGTH_LONG).show();
217     }
218 }));
219
220 alert.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
221     public void onClick(DialogInterface dialog, int whichButton) {
222         // Canceled.
223     }
224 });
225
226 alert.show();
227 }
228
229 public void showHelp(MenuItem item)
230 {
231     Intent intent = new Intent(MapsActivity.this,HelpActivity.class);
232     Bundle bundle = new Bundle();
233     bundle.putString("activity", "maps");
234     intent.putExtras(bundle);
235     startActivity(intent);
236 }
237
238 /**
239  * Menu item onClick that calls mapping function to manual update user
240  * location in case first reported location isnot accurate.
241  * @param item
242  */
243 public void updatePosition(MenuItem item) {
244     Toast.makeText(getApplicationContext(), "Thomas' method goes here.",
245         Toast.LENGTH_LONG).show();
246     // call Thomas' method to update current position.
247 }
248 }
249
```

```
1  package edu.seaaddicts.brockbutler.maps;
2
3  import android.content.Context;
4  import android.os.Handler;
5  import android.os.Looper;
6  import android.os.Message;
7  import android.util.Log;
8
9  public class MapsHandler extends Handler {
10
11     public static final int MAPS_REQUEST_UPDATE = 0x001;
12     public static final int MAPS_REQUEST_LOCATION_EXISTS = 0x002;
13     public static final int MAPS_REQUEST_DIRECTION = 0x003;
14
15     public static final int MAPS_SEND_POSITION = 0x004;
16     public static final int MAPS_SEND_DIRECTIONS = 0x005;
17
18     public static final int MAPS_ERROR_NO_LOCATION = 0x006;
19     public static final int MAPS_ERROR_NO_WIFI = 0x007;
20
21     public static final int THREAD_REQUEST_START = 0x008;
22     public static final int THREAD_REQUEST_STOP = 0x009;
23     public static final int THREAD_REQUEST_PAUSE = 0x010;
24     public static final int THREAD_REQUEST_RESUME = 0x011;
25
26     public static final int THREAD_UPDATE_POSITION = 0x012;
27
28     private static final String tag = "MapsHandler";
29     private Handler mMainHandler;
30     private Thread mMapsThread;
31
32     private Object mPauseLock;
33     private boolean mIsPaused;
34     private boolean mIsFinished;
35
36     public MapsHandler(Looper main, Context c) {
37         super(main);
38         Log.d(tag, "-----+++++++ Creating Handler from Looper ++++++-----");
39         mMainHandler = new Handler(main);
40         mIsPaused = true;
41         init();
42     }
43
44     public MapsHandler(Handler main, Context c) {
45         Log.d(tag, "-----+++++++ Creating Handler. ++++++-----");
46         mMainHandler = main;
47         mIsPaused = true;
48         init();
49     }
50
51     private void init() {
52         mPauseLock = new Object();
53
54         // Set up Maps Thread
55         mMapsThread = new Thread() {
56
57             int count = 20;
58
59
60             @Override
61             public void run() {
62                 Log.d(tag, "Thread started.");
63
64                 while (!mIsFinished) {
65                     // do your stuff here
66
67                     mMainHandler.sendMessage(count);
68                     try {
69                         Thread.sleep(3000);
70                     } catch (InterruptedException e) {
71                         // TODO Auto-generated catch block
```

```

72         e.printStackTrace();
73     }
74     count += 1;
75     synchronized (mPauseLock) {
76         while (mIsPaused) {
77             try {
78                 mPauseLock.wait();
79                 Log.d(tag,
80                     "-----++++++ Thread paused. ++++++-----");
81             } catch (InterruptedException e) {
82                 e.printStackTrace();
83             }
84         }
85     }
86 }
87 }
88 };
89 }
90
91 @Override
92 public void handleMessage(Message msg) {
93     switch (msg.what) {
94
95     case MAPS_REQUEST_UPDATE:
96         Log.d(tag, "-----++++++ Sending update to MapsActivity. ++++++-----");
97         mMainHandler.sendEmptyMessage(MAPS_SEND_POSITION);
98         break;
99
100    case MAPS_REQUEST_LOCATION_EXISTS:
101        Log.d(tag, "-----++++++ Checking for location. ++++++-----");
102
103        // Runs Thomas' code to check for existence.
104
105        // if (mDoesExist)
106        // send information
107        // else
108        // mMainHandler.sendEmptyMessage(MAPS_ERROR_NO_LOCATION);
109        break;
110
111    case MAPS_REQUEST_DIRECTION:
112        Log.d(tag, "-----++++++ Getting directions. ++++++-----");
113        mMainHandler.sendEmptyMessage(MAPS_SEND_DIRECTIONS);
114        break;
115    case THREAD_REQUEST_START:
116        if (!mMapsThread.isAlive()) {
117            Log.d(tag, "-----++++++ Starting thread. ++++++-----");
118            mMapsThread.start();
119            mIsPaused = false;
120        }
121        break;
122    case THREAD_REQUEST_PAUSE:
123        if (!mIsPaused) {
124            synchronized (mPauseLock) {
125                Log.d(tag, "-----++++++ Pausing thread. ++++++-----");
126                mIsPaused = true;
127            }
128        }
129        break;
130    case THREAD_REQUEST_RESUME:
131        if (mIsPaused) {
132            synchronized (mPauseLock) {
133                Log.d(tag, "-----++++++ Resuming thread. ++++++-----");
134                mIsPaused = false;
135                mPauseLock.notifyAll();
136            }
137        }
138        break;
139    default:
140        break;
141    }
142 }

```

```
143     }  
144
```

```
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     @SuppressWarnings("unused")
30     private static final int MAP_WIDTH = 2000;
31     private static final int MAP_HEIGHT = 1100;
32     private static final int CLICK = 3;
33
34     private Matrix mMatrixMap;
35
36     // States of touch.
37     private static final int NONE = 0;
38     private static final int DRAG = 1;
39     private static final int ZOOM = 2;
40     private int mode = NONE;
41     private int stroke = 8;
42
43     // Zooming variables.
44     private PointF last = new PointF();
45     private PointF start = new PointF();
46     private float minScale = 1f;
47     private float maxScale = 8f;
48     private float[] m;
49
50     // Ratio of screen resolution to map image resolution
51     private double mMapRatio;
52
53     private int viewWidth, viewHeight;
54     @SuppressWarnings("unused")
55     private int oldMeasuredWidth, oldMeasuredHeight;
56
57     private float scaleFactor = 1f;
58     private float origWidth, origHeight;
59
60     private final Paint mPathPaint = new Paint();
61
62     private ScaleGestureDetector mScaleDetector;
63
64     //private Context mContext;
65     int actionBarHeight;
66
67     public Position[] mPosition = null;
68
69     public MapsTouchImageView(Context context) {
70         super(context);
71         sharedConstructing(context);
```

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



```

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

```

```

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

```

```
285         redundantXSpace /= (float) 2;
286
287         mMapMatrix.postTranslate(redundantXSpace, redundantYSpace);
288
289         origWidth = viewWidth - 2 * redundantXSpace;
290         origHeight = viewHeight - 2 * redundantYSpace;
291         setImageMatrix(mMapMatrix);
292     }
293     fixTrans();
294 }
295
296 float[] convertDimensions(float x, float y) {
297     float f[] = new float[2];
298     f[0] = (float) mMapRatio * x;
299     f[1] = (float) mMapRatio * y;
300     return f;
301 }
302
303 public void drawPosition(Position[] p, int n) {
304     stroke = n;
305     mPosition = p;
306     invalidate();
307 }
308 }
```

```
1  package edu.seaaddicts.brockbutler.maps;
2
3  /**
4   * Position.java
5   * Brock Butler
6   * Type for holding Position node
7   * portion of Brock Butler.
8   * Created by Thomas Nelson 2013-03-05
9   * Copyright (c) 2013 Sea Addicts. All rights reserved.
10  */
11
12  import android.util.Log;
13
14  public class Position implements Comparable<Object> {
15
16      /**
17       * Class variable for the POSITION class. All are public
18       * to avoid using get/set variables to increase performance
19       */
20      public int      xPosition;
21      public int      yPosition;
22      public double   fScore;
23      public double   gScore;
24      public double   hScore;
25      public String   nodeNumber;
26      public String   nodeName;
27      public boolean   visited;
28      public Position from;
29      public Position accesible[];
30      public Position nonaccesible[];
31
32      /**
33       * Constructor methods for no arguments
34       */
35      public Position ( ) {
36          xPosition = 0;
37          yPosition = 0;
38
39          nodeNumber = "";
40          nodeName   = "";
41
42          fScore = Double.MAX_VALUE;
43          gScore = Double.MAX_VALUE;
44          hScore = -1;
45
46          visited = false;
47          from    = null;
48      }
49
50      /**
51       * Constructor with coordinates set
52       * @param inputX
53       * @param inputY
54       */
55      public Position (int inputX, int inputY) {
56          xPosition = inputX;
57          yPosition = inputY;
58
59          nodeNumber = "";
60          nodeName   = "";
61
62          fScore = Double.MAX_VALUE;
63          gScore = Double.MAX_VALUE;
64          hScore = Double.MAX_VALUE;
65
66          visited = false;
67          from    = null;
68      }
69
70      /**
71       * Constructor with all position information set
```

```
72     * @param inputX
73     * @param inputY
74     * @param inputName
75     * @param inputNumber
76     */
77     public Position (int inputX, int inputY, String inputName, String inputNumber) {
78         xPosition = inputX;
79         yPosition = inputY;
80
81         nodeNumber = inputNumber;
82         nodeName   = inputName;
83
84         fScore = Double.MAX_VALUE;
85         gScore = Double.MAX_VALUE;
86         hScore = Double.MAX_VALUE;
87
88         visited = false;
89         from    = null;
90     }
91
92     /**
93     * Set coordinates
94     * @param inputX
95     * @param inputY
96     */
97     public void setCoordinates (int inputX, int inputY) {
98         xPosition = inputX;
99         yPosition = inputY;
100    }
101
102    /**
103    * Set position number
104    * @param inputNumber
105    */
106    public void setNumber (String inputNumber) {
107        nodeNumber = inputNumber;
108    }
109
110    /**
111    * Set position description
112    * @param inputName
113    */
114    public void setName (String inputName) {
115        nodeName = inputName;
116    }
117
118    /**
119    * get x coordinate
120    * @return
121    */
122    public int getX ( ) {
123        return xPosition;
124    }
125
126    /**
127    * get y coordinate
128    * @return
129    */
130    public int getY ( ) {
131        return yPosition;
132    }
133
134    /**
135    * get node numner
136    * @return
137    */
138    public String getNumber ( ) {
139        return nodeNumber;
140    }
141
142    /**
```

```
143     * Get node name
144     * @return
145     */
146     public String getName ( ) {
147         return nodeName;
148     }
149
150     /**
151     * Compares this node to another
152     * @param node
153     * @return
154     */
155     public boolean compare (Position node) {
156         if(this.xPosition == node.xPosition && this.yPosition == node.yPosition && this.
            nodeNumber.equals(node.nodeNumber) && this.nodeName.equals(node.nodeName))
157             return true;
158         return false;
159     }
160
161     /**
162     * Not Used but required???
163     */
164     public int compareTo (Object node) {
165         Position temp = (Position)node;
166         return (int)(fScore - temp.fScore);
167     }
168
169
170     /**
171     * Testing methods for the POSITION class. These methods are provided
172     * for testing and debugging purposes capable of printing variables to the log
173     */
174     public void printCoordinates ( ) {
175         Log.d("POSITION CLASS", "Coordinates: (" + xPosition + "," + yPosition + ")");
176     }
177
178     public void printNumber ( ) {
179         Log.d("POSITION CLASS", "Node Number: " + nodeNumber);
180     }
181
182     public void printName ( ) {
183         Log.d("POSITION CLASS", "Node Name: " + nodeName);
184     }
185 }
```

```
1  <menu xmlns:android="http://schemas.android.com/apk/res/android" >
2
3      <item
4          android:id="@+id/menu_search"
5          android:title="@string/menu_search"
6          android:onClick="displaySearchDialog"
7          android:titleCondensed="@string/menu_search">
8      </item>
9      <item
10         android:id="@+id/menu_get_directions"
11         android:title="@string/menu_get_directions"
12         android:onClick="displayGetDirectionsDialog"
13         android:titleCondensed="@string/menu_get_directions">
14     </item>
15     <!-- <item
16         android:id="@+id/menu_update_position"
17         android:title="@string/menu_update_position"
18         android:onClick="updatePosition"
19         android:titleCondensed="@string/menu_update_position">
20     </item>-->
21     <item
22         android:id="@+id/menu_exit_maps"
23         android:title="@string/menu_exit_maps"
24         android:onClick="exitMaps"
25         android:titleCondensed="@string/menu_exit_maps">
26     </item>
27     <item
28         android:id="@+id/menu_show_maps_help"
29         android:title="@string/menu_show_maps_help"
30         android:onClick="showHelp">
31     </item>
32
33 </menu>
```

```
1 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2   xmlns:tools="http://schemas.android.com/tools"
3   android:id="@+id/maps_layout"
4   android:layout_width="match_parent"
5   android:layout_height="match_parent"
6   tools:context=".MainActivity" >
7
8   <edu.seaaddicts.brockbutler.maps.MapsTouchImageView
9       android:id="@+id/imgv_map"
10      android:layout_alignParentTop="true"
11      android:layout_width="fill_parent"
12      android:layout_height="fill_parent"
13      android:src="@drawable/mch_maps"
14      android:contentDescription="@string/test_image" />
15
16   <TextView
17       android:id="@+id/txtv_count"
18       android:layout_width="wrap_content"
19       android:layout_height="wrap_content"
20       android:layout_below="@id/imgv_map"
21       android:contentDescription="@string/temp"
22       android:textColor="#fff" />
23
24
25 </RelativeLayout>
```