```java
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 +
                 outputVal[3]*2 + outputVal[4]*1);
 75          }
 76
 77          int location = mode(answer);
 78          Log.i("LOCATE", "Node: " + location);
 79          switch(location) {
 80            case 1:
 81              return "J01";
 82            case 2:
 83              return "J02";
 84            case 3:
 85              return "J03";
 86            case 4:
 87              return "J04";
 88            case 5:
 89              return "J05";
 90            case 6:
 91              return "J06";
 92            case 7:
 93              return "J07";
 94            case 8:
 95              return "J08";
 96            case 9:
 97              return "J09";
 98            case 10:
 99              return "J10";
100            case 11:
101              return "J11";
102            case 12:
103              return "J12";
104            case 13:
105              return "J13";
106            case 14:
107              return "J14";
108            case 15:
109              return "J15";
110            case 16:
111              return "J16";
112            case 17:
113              return "J17";
114            case 18:
115              return "J18";
116            case 19:
117              return "J19";
118            case 20:
119              return "J20";
120            case 21:
121              return "J21";
122            case 22:
123              return "J22";
124            case 23:
125              return "J23";
126            default:
127              return "";
128          }
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      private static void getWirelessData() {
```

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

```
211              if (a[j] == a[i]) ++count;
212           }
213           if (count > maxCount) {
214              maxCount = count;
215              maxValue = a[i];
216           }
217        }
218
219        return maxValue;
220     }
221
222     private void initTestData() {
223        inputVal[0][0] = 1; inputVal[0][1] = -65;
224        inputVal[1][0] = 2; inputVal[1][1] = -60;
225        inputVal[2][0] = 3; inputVal[2][1] = -64;
226        inputVal[3][0] = 4; inputVal[3][1] = -64;
227        inputVal[4][0] = 5; inputVal[4][1] = -68;
228        inputVal[5][0] = 6; inputVal[5][1] = -69;
229        inputVal[6][0] = 7; inputVal[6][1] = -69;
230        inputVal[7][0] = 8; inputVal[7][1] = -72;
231        inputVal[8][0] = 9; inputVal[8][1] = -74;
232        inputVal[9][0] = 10; inputVal[9][1] = -74;
233     }
234
235     @SuppressWarnings("unused")
236     private void initTestData2() {
237        inputVal[0][0] = 1; inputVal[0][1] = -65;
238        inputVal[1][0] = 1; inputVal[1][1] = -89;
239        inputVal[2][0] = 6; inputVal[2][1] = -64;
240        inputVal[3][0] = 8; inputVal[3][1] = -64;
241        inputVal[4][0] = 11; inputVal[4][1] = -66;
242        inputVal[5][0] = 12; inputVal[5][1] = -66;
243        inputVal[6][0] = 13; inputVal[6][1] = -71;
244        inputVal[7][0] = 14; inputVal[7][1] = -72;
245        inputVal[8][0] = 15; inputVal[8][1] = -72;
246        inputVal[9][0] = 16; inputVal[9][1] = -72;
247     }
248
249     @SuppressWarnings("unused")
250     private void initTestData3() {
251        inputVal[0][0] = 6; inputVal[0][1] = -58;
252        inputVal[1][0] = 8; inputVal[1][1] = -58;
253        inputVal[2][0] = 11; inputVal[2][1] = -72;
254        inputVal[3][0] = 12; inputVal[3][1] = -72;
255        inputVal[4][0] = 13; inputVal[4][1] = -75;
256        inputVal[5][0] = 14; inputVal[5][1] = -66;
257        inputVal[6][0] = 15; inputVal[6][1] = -77;
258        inputVal[7][0] = 16; inputVal[7][1] = -75;
259        inputVal[8][0] = 9; inputVal[8][1] = -69;
260        inputVal[9][0] = 16; inputVal[9][1] = -72;
261     }
262
263     /**
264      * Makes the wireless data usable and sets it up for
265      * the network to use by putting the values between 0 and 1
266      * with min/max normalization.
267      */
268     @SuppressWarnings("unused")
269     private void initData() {
270        for (int x=0; x<10; x++) {
271           if (address[x].equalsIgnoreCase("00:0b:86:91:ce:a1"))
272              addIn[x] = 1;
273           else if (address[x].equalsIgnoreCase("00:0b:86:8a:8c:02"))
274              addIn[x] = 2;
275           else if (address[x].equalsIgnoreCase("00:1a:1e:fc:af:21"))
276              addIn[x] = 3;
277           else if (address[x].equalsIgnoreCase("00:0b:86:91:ce:a2"))
278              addIn[x] = 4;
279           else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b0:e2"))
280              addIn[x] = 5;
281           else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b0:e1"))
```

```
282          addIn[x] = 6;
283        else if (address[x].equalsIgnoreCase("00:0b:86:89:f6:e1"))
284          addIn[x] = 7;
285        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:af:22"))
286          addIn[x] = 8;
287        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b2:62"))
288          addIn[x] = 9;
289        else if (address[x].equalsIgnoreCase("00:0b:86:4d:8f:21"))
290          addIn[x] = 10;
291        else if (address[x].equalsIgnoreCase("00:0b:86:4d:8f:22"))
292          addIn[x] = 11;
293        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b0:21"))
294          addIn[x] = 12;
295        else if (address[x].equalsIgnoreCase("00:1a:1e:a7:dc:22"))
296          addIn[x] = 13;
297        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b0:22"))
298          addIn[x] = 14;
299        else if (address[x].equalsIgnoreCase("00:1a:1e:a7:dc:21"))
300          addIn[x] = 15;
301        else if (address[x].equalsIgnoreCase("00:0b:86:8a:8c:01"))
302          addIn[x] = 16;
303        else if (address[x].equalsIgnoreCase("00:1a:1e:a7:e4:c2"))
304          addIn[x] = 17;
305        else if (address[x].equalsIgnoreCase("00:1a:1e:a7:e4:c1"))
306          addIn[x] = 18;
307        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:ac:82"))
308          addIn[x] = 19;
309        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:ac:81"))
310          addIn[x] = 20;
311        else if (address[x].equalsIgnoreCase("00:0b:86:91:ce:a0"))
312          addIn[x] = 21;
313        else if (address[x].equalsIgnoreCase("00:0b:86:8a:8c:00"))
314          addIn[x] = 22;
315        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:ac:80"))
316          addIn[x] = 23;
317        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b2:61"))
318          addIn[x] = 24;
319        else if (address[x].equalsIgnoreCase("00:0b:86:42:de:80"))
320          addIn[x] = 25;
321        else if (address[x].equalsIgnoreCase("00:0b:86:42:de:82"))
322          addIn[x] = 26;
323        else if (address[x].equalsIgnoreCase("00:1a:1e:a7:e4:c0"))
324          addIn[x] = 27;
325        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b2:60"))
326          addIn[x] = 28;
327        else
328          addIn[x] = 0;
329      }
330
331      for (int x=0; x<10; x++) {
332        inputVal[x][0] = (addIn[x] - 1) / 28;
333        inputVal[x][1] = (sigStr[x] - -97) / 58;
334      }
335    }
336
337    /**
338     * Initializes the network with pre-defined weights, currently will only find a
339     * position in JBlock.
340     */
341    private void initWeights ( ) {
342      W[0][0] = -5.191953555370145;
343      W[1][0] = 8.311119623052747;
344      HB[0] = 13.645070679100112;
345      V[0][0] = -1.4188817448241078;
346      OB[0] = 8.485931304116875;
347      V[0][1] = -0.12644482595532947;
348      OB[1] = 7.676266130312892;
349      V[0][2] = -0.8742897792429708;
350      OB[2] = 7.71163044603109;
351      V[0][3] = 8.174930730213324;
352      OB[3] = 5.050964297399435;
```

```
353        V[0][4] = -1.483533992384484;
354        OB[4] = -0.9537318879960314;
355        W[0][1] = 1.2747639658533194;
356        W[1][1] = 35.85241447388153;
357        HB[1] = 22.187068222811735;
358        V[1][0] = 0.08863280595533382;
359        OB[0] = 8.485931304116875;
360        V[1][1] = -0.711381178420192;
361        OB[1] = 7.676266130312892;
362        V[1][2] = 0.47717648845370575;
363        OB[2] = 7.71163044603109;
364        V[1][3] = -0.9106351920878477;
365        OB[3] = 5.050964297399435;
366        V[1][4] = -21.858140121995646;
367        OB[4] = -0.9537318879960314;
368        W[0][2] = 72.6159523077509;
369        W[1][2] = 62.83113577555713;
370        HB[2] = 48.81891128647234;
371        V[2][0] = -0.2564556186462736;
372        OB[0] = 8.485931304116875;
373        V[2][1] = -1.3189848572962326;
374        OB[1] = 7.676266130312892;
375        V[2][2] = -1.132371452477312;
376        OB[2] = 7.71163044603109;
377        V[2][3] = 6.23102460219137;
378        OB[3] = 5.050964297399435;
379        V[2][4] = 0.9547440817617039;
380        OB[4] = -0.9537318879960314;
381        W[0][3] = -12.93760218402065;
382        W[1][3] = -27.44465213223537;
383        HB[3] = -12.785991014176767;
384        V[3][0] = -0.3900489003340711;
385        OB[0] = 8.485931304116875;
386        V[3][1] = -0.4526342230399839;
387        OB[1] = 7.676266130312892;
388        V[3][2] = -1.1900195982331039;
389        OB[2] = 7.71163044603109;
390        V[3][3] = 32.266073798358136;
391        OB[3] = 5.050964297399435;
392        V[3][4] = 0.8870702490085969;
393        OB[4] = -0.9537318879960314;
394        W[0][4] = 20.564192571900584;
395        W[1][4] = 64.556122443447;
396        HB[4] = 51.94209137066473;
397        V[4][0] = -0.14542636219949362;
398        OB[0] = 8.485931304116875;
399        V[4][1] = -0.18725148477033168;
400        OB[1] = 7.676266130312892;
401        V[4][2] = -1.3082405929431982;
402        OB[2] = 7.71163044603109;
403        V[4][3] = 7.750490464388548;
404        OB[3] = 5.050964297399435;
405        V[4][4] = 7.381930098882;
406        OB[4] = -0.9537318879960314;
407        W[0][5] = 26.203506852804754;
408        W[1][5] = 30.430410744252843;
409        HB[5] = 30.471677989844203;
410        V[5][0] = 0.6785829497339613;
411        OB[0] = 8.485931304116875;
412        V[5][1] = -0.7290230440401178;
413        OB[1] = 7.676266130312892;
414        V[5][2] = 0.01863788955350768;
415        OB[2] = 7.71163044603109;
416        V[5][3] = 13.290562812876107;
417        OB[3] = 5.050964297399435;
418        V[5][4] = -3.191226914646271;
419        OB[4] = -0.9537318879960314;
420        W[0][6] = -4.992559685945157;
421        W[1][6] = 88.04743967482369;
422        HB[6] = 50.39739242687603;
423        V[6][0] = 0.0746839836828444;
```

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