```java
package edu.seaaddicts.brockbutler.maps;

/**
 * Locate.java
 * Brock Butler
 * portion of Brock Butler.
 * Created by Thomas Nelson 2013-03-10
 * Copyright (c) 2013 Sea Addicts. All rights reserved.
 */

import java.util.List;
import android.content.Context;
import android.net.wifi.ScanResult;
import android.net.wifi.WifiManager;
import android.util.Log;

public class Locate {

  /**
   * Class variables
   */
  private static WifiManager      wifiMgr;
  private static List<ScanResult> scanResults;
  static Context parentContext;
  int[] answer = new int[10];
  /**
   * Wireless information containers
   */
  private static int[] sigStr  = new int[10];
  private static String[] address = new String[10];
  private static double[] addIn = new double[10];
  /**
   * Layers
   */
  private static final int inputs = 2;
  private static final int hidden = 8;
  private static final int output = 5;
  /**
   * Weights
   */
  private static double[][] W = new double[inputs][hidden];
  private static double[][] V = new double[hidden][output];
  private static double[] HB = new double[hidden];
  private static double[] OB = new double[output];
  /**
   * Neurons
   */
  private static double[] hiddenVal = new double[hidden];
  private static double[] outputVal = new double[output];
  private static double[][] inputVal = new double[10][inputs];

  /**
   * Constructor for the Locate class sets context and initializes weights
   * only once.
   * @param pc
   */
  public Locate (Context pc) {
    parentContext = pc;
    initWeights();
  }

  /**
   * Used by the mapping thread to get current user position, returns null if no
   position is found.
   * @return
   */
  public String getUserPosition ( ) {
    try {
      //getWirelessData(); // Use when you are testing on device
      initTestData(); // Use when testing on simulator testData1, testData2, or
      testData3
```

```java
        //initData(); // Use when you are testing on device

        for(int i=0; i<10; i++) {
          calcNetwork(i);
          answer[i] = (int) (outputVal[0]*16 + outputVal[1]*8 + outputVal[2]*4 +
          outputVal[3]*2 + outputVal[4]*1);
        }

        int location = mode(answer);
        Log.i("LOCATE", "Node: " + location);
        switch(location) {
          case 1:
            return "J01";
          case 2:
            return "J02";
          case 3:
            return "J03";
          case 4:
            return "J04";
          case 5:
            return "J05";
          case 6:
            return "J06";
          case 7:
            return "J07";
          case 8:
            return "J08";
          case 9:
            return "J09";
          case 10:
            return "J10";
          case 11:
            return "J11";
          case 12:
            return "J12";
          case 13:
            return "J13";
          case 14:
            return "J14";
          case 15:
            return "J15";
          case 16:
            return "J16";
          case 17:
            return "J17";
          case 18:
            return "J18";
          case 19:
            return "J19";
          case 20:
            return "J20";
          case 21:
            return "J21";
          case 22:
            return "J22";
          case 23:
            return "J23";
          default:
            return "";
        }
      } catch (Exception err) {
        Log.e("LOCATE", err.getMessage());
      }
      return null;
    }

    /**
     * Gathers wireless information from the device for 10 wireless access points
     * currently in range. Gathers MAC address and received signal strength
     */
    @SuppressWarnings("unused")
    private static void getWirelessData() {
      wifiMgr = (WifiManager)parentContext.getSystemService(Context.WIFI_SERVICE);
        int x = 0;

        for(int num=0; num<10; num++) {
          wifiMgr.startScan();
          scanResults = wifiMgr.getScanResults();

          x = 0;
          sigStr  = new int[10];
          address = new String[10];

          for(ScanResult scanRes : scanResults) {
            if(x < 10) {
              address[x] = scanRes.BSSID;
              sigStr[x] = scanRes.level;
              x++;
            }
          }
        }
    }

    /**
     * This Method will return the sigmoid value of an argument
     * for the final node value.
     * @param x
     * @return
     */
    private static double sigmoid(double x) {
      return 1 / (1 + Math.exp(-x));
    }

    /**
     * The beans of this class, uses normalized wireless data to predict
     * user location based on an inputed input pattern.
     * @param pat
     */
    private static void calcNetwork(int pat) {
      for(int h=0; h<hidden; h++) {
        hiddenVal[h] = -HB[h];
        for(int i=0; i<inputs; i++) {
          hiddenVal[h] += (inputVal[pat][i] * W[i][h]);
        }
        hiddenVal[h] = sigmoid(hiddenVal[h]);
      }

      for(int o=0; o<output; o++) {
        outputVal[o] = -OB[o];
        for(int h=0; h<hidden; h++) {
          outputVal[o] += (hiddenVal[h] * V[h][o]);
        }
        outputVal[o] = sigmoid(outputVal[o]);

        if(outputVal[o] >= 0.5)
          outputVal[o] = 1;
          else if(outputVal[o] < 0.5)
            outputVal[o] = 0;
      }
    }

    /**
     * Searches through network output to find the most likely
     * user position.
     * @param a
     * @return
     */
    private static int mode(int a[]) {
      int maxValue=0, maxCount=0;

      for (int i = 0; i < a.length; ++i) {
        int count = 0;
        for (int j = 0; j < a.length; ++j) {
          if (a[j] == a[i]) ++count;
        }
        if (count > maxCount) {
          maxCount = count;
          maxValue = a[i];
        }
      }

      return maxValue;
    }

    private void initTestData() {
      inputVal[0][0] = 1; inputVal[0][1] = -65;
      inputVal[1][0] = 2; inputVal[1][1] = -60;
      inputVal[2][0] = 3; inputVal[2][1] = -64;
      inputVal[3][0] = 4; inputVal[3][1] = -64;
      inputVal[4][0] = 5; inputVal[4][1] = -68;
      inputVal[5][0] = 6; inputVal[5][1] = -69;
      inputVal[6][0] = 7; inputVal[6][1] = -69;
      inputVal[7][0] = 8; inputVal[7][1] = -72;
      inputVal[8][0] = 9; inputVal[8][1] = -74;
      inputVal[9][0] = 10; inputVal[9][1] = -74;
    }

    @SuppressWarnings("unused")
    private void initTestData2() {
      inputVal[0][0] = 1; inputVal[0][1] = -65;
      inputVal[1][0] = 1; inputVal[1][1] = -89;
      inputVal[2][0] = 6; inputVal[2][1] = -64;
      inputVal[3][0] = 8; inputVal[3][1] = -64;
      inputVal[4][0] = 11; inputVal[4][1] = -66;
      inputVal[5][0] = 12; inputVal[5][1] = -66;
      inputVal[6][0] = 13; inputVal[6][1] = -71;
      inputVal[7][0] = 14; inputVal[7][1] = -72;
      inputVal[8][0] = 15; inputVal[8][1] = -72;
      inputVal[9][0] = 16; inputVal[9][1] = -72;
    }

    @SuppressWarnings("unused")
    private void initTestData3() {
      inputVal[0][0] = 6; inputVal[0][1] = -58;
      inputVal[1][0] = 8; inputVal[1][1] = -58;
      inputVal[2][0] = 11; inputVal[2][1] = -72;
      inputVal[3][0] = 12; inputVal[3][1] = -72;
      inputVal[4][0] = 13; inputVal[4][1] = -75;
      inputVal[5][0] = 14; inputVal[5][1] = -66;
      inputVal[6][0] = 15; inputVal[6][1] = -77;
      inputVal[7][0] = 16; inputVal[7][1] = -75;
      inputVal[8][0] = 9; inputVal[8][1] = -69;
      inputVal[9][0] = 16; inputVal[9][1] = -72;
    }

    /**
     * Makes the wireless data usable and sets it up for
     * the network to use by putting the values between 0 and 1
     * with min/max normalization.
     */
    @SuppressWarnings("unused")
    private void initData() {
      for (int x=0; x<10; x++) {
        if (address[x].equalsIgnoreCase("00:0b:86:91:ce:a1"))
          addIn[x] = 1;
        else if (address[x].equalsIgnoreCase("00:0b:86:8a:8c:02"))
          addIn[x] = 2;
        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:af:21"))
          addIn[x] = 3;
        else if (address[x].equalsIgnoreCase("00:0b:86:91:ce:a2"))
          addIn[x] = 4;
        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b0:e2"))
          addIn[x] = 5;
        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b0:e1"))
          addIn[x] = 6;
        else if (address[x].equalsIgnoreCase("00:0b:86:89:f6:e1"))
          addIn[x] = 7;
        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:af:22"))
          addIn[x] = 8;
        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b2:62"))
          addIn[x] = 9;
        else if (address[x].equalsIgnoreCase("00:0b:86:4d:8f:21"))
          addIn[x] = 10;
        else if (address[x].equalsIgnoreCase("00:0b:86:4d:8f:22"))
          addIn[x] = 11;
        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b0:21"))
          addIn[x] = 12;
        else if (address[x].equalsIgnoreCase("00:1a:1e:a7:dc:22"))
          addIn[x] = 13;
        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b0:22"))
          addIn[x] = 14;
        else if (address[x].equalsIgnoreCase("00:1a:1e:a7:dc:21"))
          addIn[x] = 15;
        else if (address[x].equalsIgnoreCase("00:0b:86:8a:8c:01"))
          addIn[x] = 16;
        else if (address[x].equalsIgnoreCase("00:1a:1e:a7:e4:c2"))
          addIn[x] = 17;
        else if (address[x].equalsIgnoreCase("00:1a:1e:a7:e4:c1"))
          addIn[x] = 18;
        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:ac:82"))
          addIn[x] = 19;
        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:ac:81"))
          addIn[x] = 20;
        else if (address[x].equalsIgnoreCase("00:0b:86:91:ce:a0"))
          addIn[x] = 21;
        else if (address[x].equalsIgnoreCase("00:0b:86:8a:8c:00"))
          addIn[x] = 22;
        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:ac:80"))
          addIn[x] = 23;
        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b2:61"))
          addIn[x] = 24;
        else if (address[x].equalsIgnoreCase("00:0b:86:42:de:80"))
          addIn[x] = 25;
        else if (address[x].equalsIgnoreCase("00:0b:86:42:de:82"))
          addIn[x] = 26;
        else if (address[x].equalsIgnoreCase("00:1a:1e:a7:e4:c0"))
          addIn[x] = 27;
        else if (address[x].equalsIgnoreCase("00:1a:1e:fc:b2:60"))
          addIn[x] = 28;
        else
          addIn[x] = 0;
      }

      for (int x=0; x<10; x++) {
        inputVal[x][0] = (addIn[x] - 1) / 28;
        inputVal[x][1] = (sigStr[x] - -97) / 58;
      }
    }

    /**
     * Initializes the network with pre-defined weights, currently will only find a
     * position in JBlock.
     */
    private void initWeights ( ) {
      W[0][0] = -5.191953555370145;
      W[1][0] = 8.311119623052747;
      HB[0] = 13.645070679100112;
      V[0][0] = -1.4188817448241078;
      OB[0] = 8.485931304116875;
      V[0][1] = -0.12644482595532947;
      OB[1] = 7.676266130312892;
      V[0][2] = -0.8742897792429708;
      OB[2] = 7.71163044603109;
      V[0][3] = 8.174930730213324;
```

```
        OB[3] = 5.050964297399435;
        V[0][4] = -1.483533992384484;
        OB[4] = -0.9537318879960314;
        W[0][1] = 1.2747639658533194;
        W[1][1] = 35.85241447388153;
        HB[1] = 22.187068222811735;
        V[1][0] = 0.08863280595533382;
        OB[0] = 8.485931304116875;
        V[1][1] = -0.711381178420192;
        OB[1] = 7.676266130312892;
        V[1][2] = 0.47717648845370575;
        OB[2] = 7.71163044603109;
        V[1][3] = -0.9106351920878477;
        OB[3] = 5.050964297399435;
        V[1][4] = -21.858140121995646;
        OB[4] = -0.9537318879960314;
        W[0][2] = 72.6159523077509;
        W[1][2] = 62.83113577555713;
        HB[2] = 48.81891128647234;
        V[2][0] = -0.2564556186462736;
        OB[0] = 8.485931304116875;
        V[2][1] = -1.3189848572962326;
        OB[1] = 7.676266130312892;
        V[2][2] = -1.132371452477312;
        OB[2] = 7.71163044603109;
        V[2][3] = 6.23102460219137;
        OB[3] = 5.050964297399435;
        V[2][4] = 0.9547440817617039;
        OB[4] = -0.9537318879960314;
        W[0][3] = -12.93760218402065;
        W[1][3] = -27.44465213223537;
        HB[3] = -12.785991014176767;
        V[3][0] = -0.3900489003340711;
        OB[0] = 8.485931304116875;
        V[3][1] = -0.4526342230399839;
        OB[1] = 7.676266130312892;
        V[3][2] = -1.1900195982331039;
        OB[2] = 7.71163044603109;
        V[3][3] = 32.266073798358136;
        OB[3] = 5.050964297399435;
        V[3][4] = 0.8870702490085969;
        OB[4] = -0.9537318879960314;
        W[0][4] = 20.564192571900584;
        W[1][4] = 64.556122443447;
        HB[4] = 51.94209137066473;
        V[4][0] = -0.14542636219949362;
        OB[0] = 8.485931304116875;
        V[4][1] = -0.18725148477033168;
        OB[1] = 7.676266130312892;
        V[4][2] = -1.3082405929431982;
        OB[2] = 7.71163044603109;
        V[4][3] = 7.750490464388548;
        OB[3] = 5.050964297399435;
        V[4][4] = 7.381930098882;
        OB[4] = -0.9537318879960314;
        W[0][5] = 26.203506852804754;
        W[1][5] = 30.430410744252843;
        HB[5] = 30.471677989844203;
        V[5][0] = 0.6785829497339613;
        OB[0] = 8.485931304116875;
        V[5][1] = -0.7290230440401178;
        OB[1] = 7.676266130312892;
        V[5][2] = 0.018637889553507 68;
        OB[2] = 7.71163044603109;
        V[5][3] = 13.290562812876107;
        OB[3] = 5.050964297399435;
        V[5][4] = -3.191226914646271;
        OB[4] = -0.9537318879960314;
        W[0][6] = -4.992559685945157;
        W[1][6] = 88.04743967482369;
        HB[6] = 50.39739242687603;
        V[6][0] = 0.0746839836828444;
        OB[0] = 8.485931304116875;
        V[6][1] = -0.5879731640198912;
        OB[1] = 7.676266130312892;
        V[6][2] = -0.25250658203018556;
        OB[2] = 7.71163044603109;
        V[6][3] = 1.4602581997464505;
        OB[3] = 5.050964297399435;
        V[6][4] = 18.463904409636967;
        OB[4] = -0.9537318879960314;
        W[0][7] = -4.74928985780168;
        W[1][7] = 40.78597102067532;
        HB[7] = 35.79898233956728;
        V[7][0] = -0.09439880889024627;
        OB[0] = 8.485931304116875;
        V[7][1] = -1.1230801185236317;
        OB[1] = 7.676266130312892;
        V[7][2] = -0.2674705623236563;
        OB[2] = 7.71163044603109;
        V[7][3] = 8.441254951069187;
        OB[3] = 5.050964297399435;
        V[7][4] = -20.04591503798929;
        OB[4] = -0.9537318879960314;
      }
    }


```

```java
package edu.seaaddicts.brockbutler.maps;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.Window;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.RelativeLayout;
import android.widget.Toast;
import edu.seaaddicts.brockbutler.R;
import edu.seaaddicts.brockbutler.help.HelpActivity;

public class MapsActivity extends Activity {
  private static final String TAG = "MapsActivity";

  private EditText mSearchEditText;

  private Handler mHandler;
  private MapsTouchImageView mMapImage;
  private MapsHandler mMapsHandler;

  private Position mStartPosition;
  private Position mGoalPosition;
  private Astar school;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    getWindow().requestFeature(Window.FEATURE_ACTION_BAR_OVERLAY);
    setContentView(R.layout.activity_map);

    RelativeLayout layout = (RelativeLayout) findViewById(R.id.maps_layout);
    layout.setSystemUiVisibility(View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN);
    getActionBar().setBackgroundDrawable(
        getResources().getDrawable(R.drawable.actionbar_bg));
    init();

    mHandler = new Handler() {

      // Handles Messages sent from Thomas.
      @Override
      public void handleMessage(Message msg) {
        switch (msg.what) {
        case MapsHandler.MAPS_REQUEST_UPDATE:
          Log.d("MAIN HANDLER", "YAYAAA!!!");
          break;
        case MapsHandler.THREAD_UPDATE_POSITION:
          Log.d("TEST",msg.getData().getString("pos"));
          break;
        default:
          Log.d(TAG,"-----+++++ Got THREAD_UPDATE_POSITION message. +++++-----");
          break;
        }
      }
    };
    mMapsHandler = new MapsHandler(mHandler,this);
        school = new Astar();
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
```

```java
        getMenuInflater().inflate(R.menu.activity_map, menu);
        return true;
    }

    private void init() {
        mMapImage = (MapsTouchImageView) findViewById(R.id.imgv_map);
        // mMapImage.setOnClickListener(new OnClickListener() {
        //
        // public void onClick(View v) {
        // mMapsHandler.sendEmptyMessage(MapsHandler.MAPS_REQUEST_UPDATE);
        // }
        // });
        // start = (Button) findViewById(R.id.btnstart);
        // start.setOnClickListener(new OnClickListener() {
        //
        // public void onClick(View v) {
        // mMapsHandler.sendEmptyMessage(MapsHandler.THREAD_REQUEST_START);
        // }
        // });
        // stop = (Button) findViewById(R.id.btnstop);
        // stop.setOnClickListener(new OnClickListener() {
        //
        // public void onClick(View v) {
        // mMapsHandler.sendEmptyMessage(MapsHandler.THREAD_REQUEST_PAUSE);
        // }
        // });
        //
        // resume = (Button) findViewById(R.id.btnresume);
        // resume.setOnClickListener(new OnClickListener() {
        //
        // public void onClick(View v) {
        // mMapsHandler
        // .sendEmptyMessage(MapsHandler.THREAD_REQUEST_RESUME);
        // }
        // });
    }

    @Override
    public void onBackPressed() {
        mMapsHandler.sendEmptyMessage(MapsHandler.THREAD_REQUEST_PAUSE);
        mMapsHandler = null;
        super.onBackPressed();
    }

    /**
     *
     * @param item
     */
    public void exitMaps(MenuItem item) {
        onBackPressed();
    }

    /**
     * Prompts user to search for existence of a location.
     *
     * @param item
     */
    public void displaySearchDialog(MenuItem item) {
        AlertDialog.Builder editalert = new AlertDialog.Builder(this);

        editalert.setTitle("MChown Location Search");
        editalert.setMessage("Enter block or room name (i.e. B314)");

        mSearchEditText = new EditText(this);
        mSearchEditText.setSingleLine(true);
        LinearLayout.LayoutParams lp = new LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.MATCH_PARENT,
            LinearLayout.LayoutParams.MATCH_PARENT);
        mSearchEditText.setLayoutParams(lp);
        editalert.setView(mSearchEditText);

        editalert.setPositiveButton("Search",
            new DialogInterface.OnClickListener() {
              public void onClick(DialogInterface dialog, int whichButton) {
                try {
                  Position p1, p2=new Position(), p3=new Position(), p4=new Position(),
                  p5=new Position();

                  p1 = school.findPosition(mSearchEditText.getText().toString());

                  p2.xPosition = p1.xPosition - 1;
                  p2.yPosition = p1.yPosition - 1;

                  p3.xPosition = p1.xPosition + 1;
                  p3.yPosition = p1.yPosition + 1;

                  p4.xPosition = p1.xPosition + 1;
                  p4.yPosition = p1.yPosition - 1;

                  p5.xPosition = p1.xPosition - 1;
                  p5.yPosition = p1.yPosition + 1;

                  Position[] pTest = {p5,p3,p4,p2,p5,p4,p3,p2};

                  if(p1 != null && p2 != null)
                    mMapImage.drawPosition(pTest,30);
                  }
                catch (NullPointerException e) {
                  Toast.makeText(getApplicationContext(), "Location not found",Toast.
                  LENGTH_LONG).show();
                }
              }
            });
        editalert.setNegativeButton("Cancel",
            new DialogInterface.OnClickListener() {
              public void onClick(DialogInterface dialog, int whichButton) {
                // do nothing
              }
            });

        editalert.show();
      }

      /**
       * Displays AlertDialog for user to enter destination. First the location is
       * determined to exist, then if true path is drawn on map.
       *
       * @param item
       */
      public void displayGetDirectionsDialog(MenuItem item) {
        school = new Astar();
        AlertDialog.Builder alert = new AlertDialog.Builder(this);

        alert.setTitle("Get Directions");
        alert.setMessage("Enter Start and End Points");

        LinearLayout lila1= new LinearLayout(this);
          lila1.setOrientation(1); //1 is for vertical orientation
          final EditText input = new EditText(this);
          final EditText input1 = new EditText(this);
          lila1.addView(input);
          lila1.addView(input1);
          alert.setView(lila1);

        alert.setPositiveButton("Ok", new DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialog, int whichButton) {
            try {
              mStartPosition = school.findPosition(input.getText().toString());
              mGoalPosition = school.findPosition(input1.getText().toString());

              Position[] p = school.pathGeneration(mStartPosition, mGoalPosition);
              if(mStartPosition != null && mGoalPosition != null)
                mMapImage.drawPosition(p,8);
            }
            catch (NullPointerException e){
              Toast.makeText(getApplicationContext(), "Location not found",Toast.
              LENGTH_LONG).show();
            }
          }
        });

        alert.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialog, int whichButton) {
            // Canceled.
          }
        });

        alert.show();
      }

      public void showHelp(MenuItem item)
      {
        Intent intent = new Intent(MapsActivity.this,HelpActivity.class);
        Bundle bundle = new Bundle();
        bundle.putString("activity", "maps");
        intent.putExtras(bundle);
        startActivity(intent);
      }

      /**
       * Menu item onClick that calls mapping function to manual update user
       * location in case first reported location isnot accurate.
       * @param item
       */
      public void updatePosition(MenuItem item) {
        Toast.makeText(getApplicationContext(), "Thomas' method goes here.",
            Toast.LENGTH_LONG).show();
        // call Thomas' method to update current position.
      }
    }

```

```java
package edu.seaaddicts.brockbutler.maps;

import android.content.Context;
import android.os.Handler;
import android.os.Looper;
import android.os.Message;
import android.util.Log;

public class MapsHandler extends Handler {

  public static final int MAPS_REQUEST_UPDATE = 0x001;
  public static final int MAPS_REQUEST_LOCATION_EXISTS = 0x002;
  public static final int MAPS_REQUEST_DIRECTION = 0x003;

  public static final int MAPS_SEND_POSITION = 0x004;
  public static final int MAPS_SEND_DIRECTIONS = 0x005;

  public static final int MAPS_ERROR_NO_LOCATION = 0x006;
  public static final int MAPS_ERROR_NO_WIFI = 0x007;

  public static final int THREAD_REQUEST_START = 0x008;
  public static final int THREAD_REQUEST_STOP = 0x009;
  public static final int THREAD_REQUEST_PAUSE = 0x010;
  public static final int THREAD_REQUEST_RESUME = 0x011;

  public static final int THREAD_UPDATE_POSITION = 0x012;

  private static final String tag = "MapsHandler";
  private Handler mMainHandler;
  private Thread mMapsThread;

  private Object mPauseLock;
  private boolean mIsPaused;
  private boolean mIsFinished;

  public MapsHandler(Looper main, Context c) {
    super(main);
    Log.d(tag, "-----++++++ Creating Handler from Looper +++++------");
    mMainHandler = new Handler(main);
    mIsPaused = true;
    init();
  }

  public MapsHandler(Handler main, Context c) {
    Log.d(tag, "-----++++++ Creating Handler. +++++------");
    mMainHandler = main;
    mIsPaused = true;
    init();
  }

  private void init() {
    mPauseLock = new Object();

    // Set up Maps Thread
    mMapsThread = new Thread() {

      int count = 20;


      @Override
      public void run() {
        Log.d(tag, "Thread started.");

        while (!mIsFinished) {
          // do your stuff here

          mMainHandler.sendEmptyMessage(count);
          try {
            Thread.sleep(3000);
          } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
          }
          count += 1;
          synchronized (mPauseLock) {
            while (mIsPaused) {
              try {
                mPauseLock.wait();
                Log.d(tag,
                    "-----++++++ Thread paused. +++++------");
              } catch (InterruptedException e) {
                e.printStackTrace();
              }
            }
          }
        }
      }
    };
  }

  @Override
  public void handleMessage(Message msg) {
    switch (msg.what) {

    case MAPS_REQUEST_UPDATE:
      Log.d(tag, "-----+++++ Sending update to MapsActivity. +++++-----");
      mMainHandler.sendEmptyMessage(MAPS_SEND_POSITION);
      break;

    case MAPS_REQUEST_LOCATION_EXISTS:
      Log.d(tag, "-----+++++ Checking for location. +++++-----");

      // Runs Thomas' code to check for existence.

      // if (mDoesExist)
      // send information
      // else
      // mMainHandler.sendEmptyMessage(MAPS_ERROR_NO_LOCATION);
      break;

    case MAPS_REQUEST_DIRECTION:
      Log.d(tag, "-----+++++ Getting directions. +++++-----");
      mMainHandler.sendEmptyMessage(MAPS_SEND_DIRECTIONS);
      break;
    case THREAD_REQUEST_START:
      if (!mMapsThread.isAlive()) {
        Log.d(tag, "-----+++++ Starting thread. +++++-----");
        mMapsThread.start();
        mIsPaused = false;
      }
      break;
    case THREAD_REQUEST_PAUSE:
      if (!mIsPaused) {
        synchronized (mPauseLock) {
          Log.d(tag, "-----+++++ Pausing thread. +++++-----");
          mIsPaused = true;
        }
      }
      break;
    case THREAD_REQUEST_RESUME:
      if (mIsPaused) {
        synchronized (mPauseLock) {
          Log.d(tag, "-----+++++ Resuming thread. +++++-----");
          mIsPaused = false;
          mPauseLock.notifyAll();
        }
      }
      break;
    default:
      break;
    }
  }
```

```java
}

```

```java
/**
 * Android: TouchImageView.java
 * Created by: Mike Ortiz
 * Updated by: Vince Pascuzzi
 * Date: 3/14/2013
 *
 * Allows pinching, zooming, translating, and drawing on an ImageView.
 */

package edu.seaaddicts.brockbutler.maps;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Matrix;
import android.graphics.Paint;
import android.graphics.PointF;
import android.graphics.drawable.Drawable;
import android.util.AttributeSet;
import android.util.Log;
import android.view.MotionEvent;
import android.view.ScaleGestureDetector;
import android.view.View;
import android.widget.ImageView;

public class MapsTouchImageView extends ImageView {
  //private static final String TAG = "MapsTouchImageView";

  @SuppressWarnings("unused")
  private static final int MAP_WIDTH = 2000;
  private static final int MAP_HEIGHT = 1100;
  private static final int CLICK = 3;

  private Matrix mMatrixMap;

  // States of touch.
  private static final int NONE = 0;
  private static final int DRAG = 1;
  private static final int ZOOM = 2;
  private int mode = NONE;
  private int stroke = 8;

  // Zooming variables.
  private PointF last = new PointF();
  private PointF start = new PointF();
  private float minScale = 1f;
  private float maxScale = 8f;
  private float[] m;

  // Ratio of screen resolution to map image resolution
  private double mMapRatio;

  private int viewWidth, viewHeight;
  @SuppressWarnings("unused")
  private int oldMeasuredWidth, oldMeasuredHeight;

  private float scaleFactor = 1f;
  private float origWidth, origHeight;

  private final Paint mPathPaint = new Paint();

  private ScaleGestureDetector mScaleDetector;

  //private Context mContext;
  int actionBarHeight;

  public Position[] mPosition = null;

  public MapsTouchImageView(Context context) {
    super(context);
    sharedConstructing(context);
  }

  public MapsTouchImageView(Context context, AttributeSet attrs) {
    super(context, attrs);
    sharedConstructing(context);
  }

  @Override
  protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    mPathPaint.setColor(Color.CYAN);
    mPathPaint.setStrokeWidth(stroke);
    canvas.setMatrix(mMatrixMap);

    if (mPosition != null) {
      for (int i = 0; i < mPosition.length - 1; i++) {
        Position p = mPosition[i];
        Position q = mPosition[i + 1];
        int x1 = p.xPosition;
        int y1 = p.yPosition;
        float[] f1 = convertDimensions(x1, y1);
        int x2 = q.xPosition;
        int y2 = q.yPosition;
        float[] f2 = convertDimensions(x2, y2);
        canvas.drawLine(f1[0], f1[1], f2[0], f2[1], mPathPaint);
      }
    }
  }

  private void sharedConstructing(Context context) {
    super.setClickable(true);
    //this.mContext = context;
    mScaleDetector = new ScaleGestureDetector(context, new ScaleListener());
    mMatrixMap = new Matrix();
    m = new float[9];
    setImageMatrix(mMatrixMap);
    setScaleType(ScaleType.MATRIX);

    setOnTouchListener(new OnTouchListener() {

      public boolean onTouch(View v, MotionEvent event) {
        mScaleDetector.onTouchEvent(event);
        PointF curr = new PointF(event.getX(), event.getY());

        switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
          last.set(curr);
          start.set(last);
          mode = DRAG;
          break;

        case MotionEvent.ACTION_MOVE:
          float fixTransX;
          float fixTransY;
          if (mode == DRAG) {
            float deltaX = curr.x - last.x;
            float deltaY = curr.y - last.y;
            fixTransX = getFixDragTrans(deltaX, viewWidth,
                origWidth * scaleFactor);
            fixTransY = getFixDragTrans(deltaY, viewHeight,
                origHeight * scaleFactor);
            mMatrixMap.postTranslate(fixTransX, fixTransY);
            fixTrans();
            last.set(curr.x, curr.y);
          }
          break;

        case MotionEvent.ACTION_UP:
          mode = NONE;
          int xDiff = (int) Math.abs(curr.x - start.x);
          int yDiff = (int) Math.abs(curr.y - start.y);
```

```java

            if (xDiff < CLICK && yDiff < CLICK)
              performClick();
            break;

          case MotionEvent.ACTION_POINTER_UP:
            mode = NONE;
            break;
          }

          setImageMatrix(mMatrixMap);
          invalidate();
          return true; // indicate event was handled
        }

      });
    }

    public void setMaxZoom(float x) {
      maxScale = x;
    }

    private class ScaleListener extends
        ScaleGestureDetector.SimpleOnScaleGestureListener {
      @Override
      public boolean onScaleBegin(ScaleGestureDetector detector) {
        mode = ZOOM;
        return true;
      }

      @Override
      public boolean onScale(ScaleGestureDetector detector) {
        float mScaleFactor = detector.getScaleFactor();
        float origScale = scaleFactor;
        scaleFactor *= mScaleFactor;
        if (scaleFactor > maxScale) {
          scaleFactor = maxScale;
          mScaleFactor = maxScale / origScale;
        } else if (scaleFactor < minScale) {
          scaleFactor = minScale;
          mScaleFactor = minScale / origScale;
        }

        if (origWidth * scaleFactor <= viewWidth
            || origHeight * scaleFactor <= viewHeight)
          mMatrixMap.postScale(mScaleFactor, mScaleFactor, viewWidth / 2,
              viewHeight / 2);
        else
          mMatrixMap.postScale(mScaleFactor, mScaleFactor,
              detector.getFocusX(), detector.getFocusY());
        fixTrans();
        return true;
      }
    }

    void fixTrans() {
      mMatrixMap.getValues(m);
      float fixTransX;
      float fixTransY;
      float transX = m[Matrix.MTRANS_X];
      float transY = m[Matrix.MTRANS_Y];

      fixTransX = getFixTrans(transX, viewWidth, origWidth * scaleFactor);
      fixTransY = getFixTrans(transY, viewHeight, origHeight * scaleFactor);

      if (fixTransX != 0 || fixTransY != 0)
        mMatrixMap.postTranslate(fixTransX, fixTransY);
    }

    /*
     * Fixes (when required) the translation matrix.
     */
    float getFixTrans(float trans, float viewSize, float contentSize) {
      float minTrans, maxTrans;

      if (contentSize <= viewSize) {
        minTrans = 0;
        maxTrans = viewSize - contentSize;
      } else {
        minTrans = viewSize - contentSize;
        maxTrans = 0;
      }

      if (trans < minTrans)
        return -trans + minTrans;
      if (trans > maxTrans)
        return -trans + maxTrans;
      return 0;
    }

    /*
     * Adjusts the translation when dragging so that this stays in the correct
     * location on screen.
     */
    float getFixDragTrans(float delta, float viewSize, float contentSize) {
      if (contentSize <= viewSize) {
        return 0;
      }
      return delta;
    }

    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
      super.onMeasure(widthMeasureSpec, heightMeasureSpec);
      viewWidth = MeasureSpec.getSize(widthMeasureSpec);
      viewHeight = MeasureSpec.getSize(heightMeasureSpec);

      // Does image rescaling on rotation. Not necessary since our orientation
      // is fixed in landscape.
      if (oldMeasuredHeight == viewWidth && oldMeasuredHeight == viewHeight
          || viewWidth == 0 || viewHeight == 0)
        return;
      oldMeasuredHeight = viewHeight;
      oldMeasuredWidth = viewWidth;

      if (scaleFactor == 1) {
        // Fit to screen.
        float scale;

        Drawable drawable = getDrawable();
        if (drawable == null || drawable.getIntrinsicWidth() == 0
            || drawable.getIntrinsicHeight() == 0)
          return;
        int bmWidth = drawable.getIntrinsicWidth();
        int bmHeight = drawable.getIntrinsicHeight();

        mMapRatio = (double) (bmHeight) / (double) MAP_HEIGHT;

        Log.d("bmSize", "bmWidth: " + bmWidth + " bmHeight : " + bmHeight
            + "ratio" + mMapRatio);

        float scaleX = (float) viewWidth / (float) bmWidth;
        float scaleY = (float) viewHeight / (float) bmHeight;
        scale = Math.min(scaleX, scaleY);
        mMatrixMap.setScale(scale, scale);

        // Center the image
        float redundantYSpace = (float) viewHeight
            - (scale * (float) bmHeight);
        float redundantXSpace = (float) viewWidth
            - (scale * (float) bmWidth);
        redundantYSpace /= (float) 2;
```

```
            redundantXSpace /= (float) 2;

            mMatrixMap.postTranslate(redundantXSpace, redundantYSpace);

            origWidth = viewWidth - 2 * redundantXSpace;
            origHeight = viewHeight - 2 * redundantYSpace;
            setImageMatrix(mMatrixMap);
        }
        fixTrans();
    }

    float[] convertDimensions(float x, float y) {
        float f[] = new float[2];
        f[0] = (float) mMapRatio * x;
        f[1] = (float) mMapRatio * y;
        return f;
    }

    public void drawPosition(Position[] p, int n) {
        stroke = n;
        mPosition = p;
        invalidate();
    }
}
```

```java
package edu.seaaddicts.brockbutler.maps;

/**
 * Position.java
 * Brock Butler
 * Type for holding Position node
 * portion of Brock Butler.
 * Created by Thomas Nelson 2013-03-05
 * Copyright (c) 2013 Sea Addicts. All rights reserved.
 */

import android.util.Log;

public class Position implements Comparable<Object> {

  /**
   * Class variable for the POSITION class. All are public
   * to avoid using get/set variables to increase performance
   */
  public int      xPosition;
  public int      yPosition;
  public double   fScore;
  public double   gScore;
  public double   hScore;
  public String   nodeNumber;
  public String   nodeName;
  public boolean  visited;
    public Position from;
    public Position accesible[];
    public Position nonaccesible[];

  /**
   * Constructor methods for no arguments
   */
  public Position ( ) {
    xPosition = 0;
    yPosition = 0;

    nodeNumber = "";
    nodeName   = "";

    fScore = Double.MAX_VALUE;
        gScore = Double.MAX_VALUE;
        hScore = -1;

        visited = false;
        from    = null;
  }

  /**
   * Constructor with coordinates set
   * @param inputX
   * @param inputY
   */
  public Position (int inputX, int inputY) {
    xPosition = inputX;
    yPosition = inputY;

    nodeNumber = "";
    nodeName   = "";

    fScore = Double.MAX_VALUE;
    gScore = Double.MAX_VALUE;
    hScore = Double.MAX_VALUE;

    visited = false;
        from    = null;
  }

  /**
   * Constructor with all position information set
```

```java
     * @param inputX
     * @param inputY
     * @param inputName
     * @param inputNumber
     */
    public Position (int inputX, int inputY, String inputName, String inputNumber) {
      xPosition = inputX;
      yPosition = inputY;

      nodeNumber = inputNumber;
      nodeName   = inputName;

      fScore = Double.MAX_VALUE;
      gScore = Double.MAX_VALUE;
      hScore = Double.MAX_VALUE;

      visited = false;
          from    = null;
    }

    /**
     * Set coordinates
     * @param inputX
     * @param inputY
     */
    public void setCoordinates (int inputX, int inputY) {
      xPosition  = inputX;
      yPosition  = inputY;
    }

    /**
     * Set position number
     * @param inputNumber
     */
    public void setNumber (String inputNumber) {
      nodeNumber  = inputNumber;
    }

    /**
     * Set position description
     * @param inputName
     */
    public void setName (String inputName) {
      nodeName  = inputName;
    }

    /**
     * get x coordinate
     * @return
     */
    public int getX ( ) {
      return xPosition;
    }

    /**
     * get y coordinate
     * @return
     */
    public int getY ( ) {
      return yPosition;
    }

    /**
     * get node numner
     * @return
     */
    public String getNumber ( ) {
      return nodeNumber;
    }

    /**
```

```java
     * Get node name
     * @return
     */
    public String getName ( ) {
      return nodeName;
    }

    /**
     * Compares this node to another
     * @param node
     * @return
     */
    public boolean compare (Position node) {
      if(this.xPosition == node.xPosition && this.yPosition == node.yPosition && this.
      nodeNumber.equals(node.nodeNumber) && this.nodeName.equals(node.nodeName))
        return true;
      return false;
    }

    /**
     * Not Used but required???
     */
    public int compareTo (Object node) {
      Position temp = (Position)node;
          return (int)(fScore - temp.fScore);
    }


    /**
     * Testing methods for the POSITION class. These methods are provided
     * for testing and debugging purposes capable of printing variables to the log
     */
    public void printCoordinates ( ) {
      Log.d("POSITION CLASS", "Coordinates: (" + xPosition + "," + yPosition + ")");
    }

    public void printNumber ( ) {
      Log.d("POSITION CLASS", "Node Number: " + nodeNumber);
    }

    public void printName ( ) {
      Log.d("POSITION CLASS", "Node Name: " + nodeName);
    }
  }
```

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

    <item
        android:id="@+id/menu_search"
        android:title="@string/menu_search"
        android:onClick="displaySearchDialog"
        android:titleCondensed="@string/menu_search">
    </item>
    <item
        android:id="@+id/menu_get_directions"
        android:title="@string/menu_get_directions"
        android:onClick="displayGetDirectionsDialog"
        android:titleCondensed="@string/menu_get_directions">
    </item>
  <!--   <item
       android:id="@+id/menu_update_position"
        android:title="@string/menu_update_position"
        android:onClick="updatePosition"
        android:titleCondensed="@string/menu_update_position">
    </item>-->
    <item
        android:id="@+id/menu_exit_maps"
        android:title="@string/menu_exit_maps"
        android:onClick="exitMaps"
        android:titleCondensed="@string/menu_exit_maps">
    </item>
    <item
        android:id="@+id/menu_show_maps_help"
        android:title="@string/menu_show_maps_help"
        android:onClick="showHelp">
    </item>

</menu>
```

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/maps_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MapActivity" >

    <edu.seaaddicts.brockbutler.maps.MapsTouchImageView
        android:id="@+id/imgv_map"
        android:layout_alignParentTop="true"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:src="@drawable/mch_maps"
        android:contentDescription="@string/test_image" />

    <TextView
        android:id="@+id/txtv_count"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/imgv_map"
        android:contentDescription="@string/temp"
        android:textColor="#fff" />


</RelativeLayout>
```