Final Project

Project Approach and Lessons Learned

ZombieMapper Tool

Parsing KML Files for Shapefile Analysis

Table of Contents

Overview

As outlined in the final project proposal, this tool parses data from a KML file, and writes geometries to a point feature class. An integer value is also part of each point (Magnitude) which is written as an attribute of the geometry point. The integer represents the number of zombies seen at that point.

Using variable buffer analysis, the tool identifies zombie "hot spots". The point feature class can then be used by analysts in conjunction with polygon or other feature classes to create information which can be provided to survivors.

Approach

My approach was centered around a make-believe scenario, from which I could base design decisions.

My first step was to familiarize myself with general XML and KML file structure, as they were fairly new to me. I then I researched how to parse an XML file to obtain the important elements. I discovered there were several module libraries available to me that would be preferable options over parsing 'from scratch'. I chosed the minidom.parse function from xml.dom.minidom module, as it seemed the most straight-forward to me, and did not require an install with my version of PythonWin.

I also wanted to generate my own KML files for this project, rather than obtain one from the internet. I did some iPhone app research, and found a tool called ***iMapIt Pro*** (v1.5) and installed the app on my iPhone4S. This tool allowed me to:

- create a new project
- create surveys within the project
- drop points wherever I wanted (I specified one point per survey), name the point and input a description, which I used for recording zombie magnitude
- export the whole project as a KML file in geographic coordinates, and email to myself

Lessons Learned

New Lessons

There were several new lessons for me in this project, specifically:

- The **syntax of parsing XML** using a new library module, minidom (a minimal implementation of DOM interface) was the principal lesson
- Learning about the structure of XML and KML files
- Successfully defining field values (BUFF_DIST) based on another field using an update cursor (I had not done this successfully in Lesson 3)
- Wrote buffer values as a string using legal distance values, i.e., "500 Meters" of "2600 Feet". See Table 1.

Reinforced Topics

There were also several steps that served as good reinforcement, even though I had achieved them in other projects. These include:

- Adding values to a list
- Inserting an array into a shapefile
- Creating a new shapefile and adding new fields to it
- General buffer experience (I had learned about buffering from the GEOG 482 & 483 classes)
- Obtaining Census files (to create context for my point geometries)

Script & Tool Features

There are some features of the deliverables, including:

- Custom messaging, printing each of the locations as they're parsed
- Error handling using Try / Except
- Documented Tool and commented script
- Get parameter as Text – to keep it simple for the analyst I chose to only have one parameter, the KML file

Implementation Steps

Here are the steps you'll need to implement the script/tool:

1. A sample KML is provided, KML.kml. This is the one parameter for both the script and tool
    a. Note the default workspace is **C:\GEOG485\ProjectFinal**
2. A Python script, **FinalProject.py**, is provided that can be run from PythonWin.
3. A .tbx file, **FinalProject.tbx**, is provided. Run this from your ArcMap toolbox. The script source is C:\GEOG485\ProjectFinal\FinalProject.py

4. (Optional) attached Delaware County zipped shapefiles are there to provide context for area
5. (Optional) a .mdx file, **FinalProject.mdx**, is prepared to see all the shape files together

Table

walkerSightings

	FID	Shape	Id	NAME	MAGNITUDE	BUFF_DIST
▶	0	Point	0	Tired Hands Brewery	23	500 Meters
	1	Point	0	Brookside Park	3	200 Meters
	2	Point	0	476 and Darby Rd	76	2600 Feet
	3	Point	0	Pep Boys	76	2600 Feet
	4	Point	0	Newton Square McDonalds	1	200 Meters
	5	Point	0	Bryn Mawr Ave near 320	9	300 Meters
	6	Point	0	SAP America	15	300 Meters
	7	Point	0	WaWa	4	200 Meters
	8	Point	0	209 Lee Circle	80	2600 Feet
	9	Point	0	476 and Rt 3	4	200 Meters
	10	Point	0	Manoa	11	300 Meters
	11	Point	0	Kohls	54	500 Meters
	12	Point	0	Darby and Eagle	93	2600 Feet
	13	Point	0	Haverford Rd and Ardmore Ave	40	500 Meters
	14	Point	0	Villanova	45	500 Meters
	15	Point	0	Bryn Mawr Train Station	47	500 Meters

1 (0 out of 16 Selected)

walkerSightings

Table 1: walkerSightings table with updated buffer distances, based on Magnitude

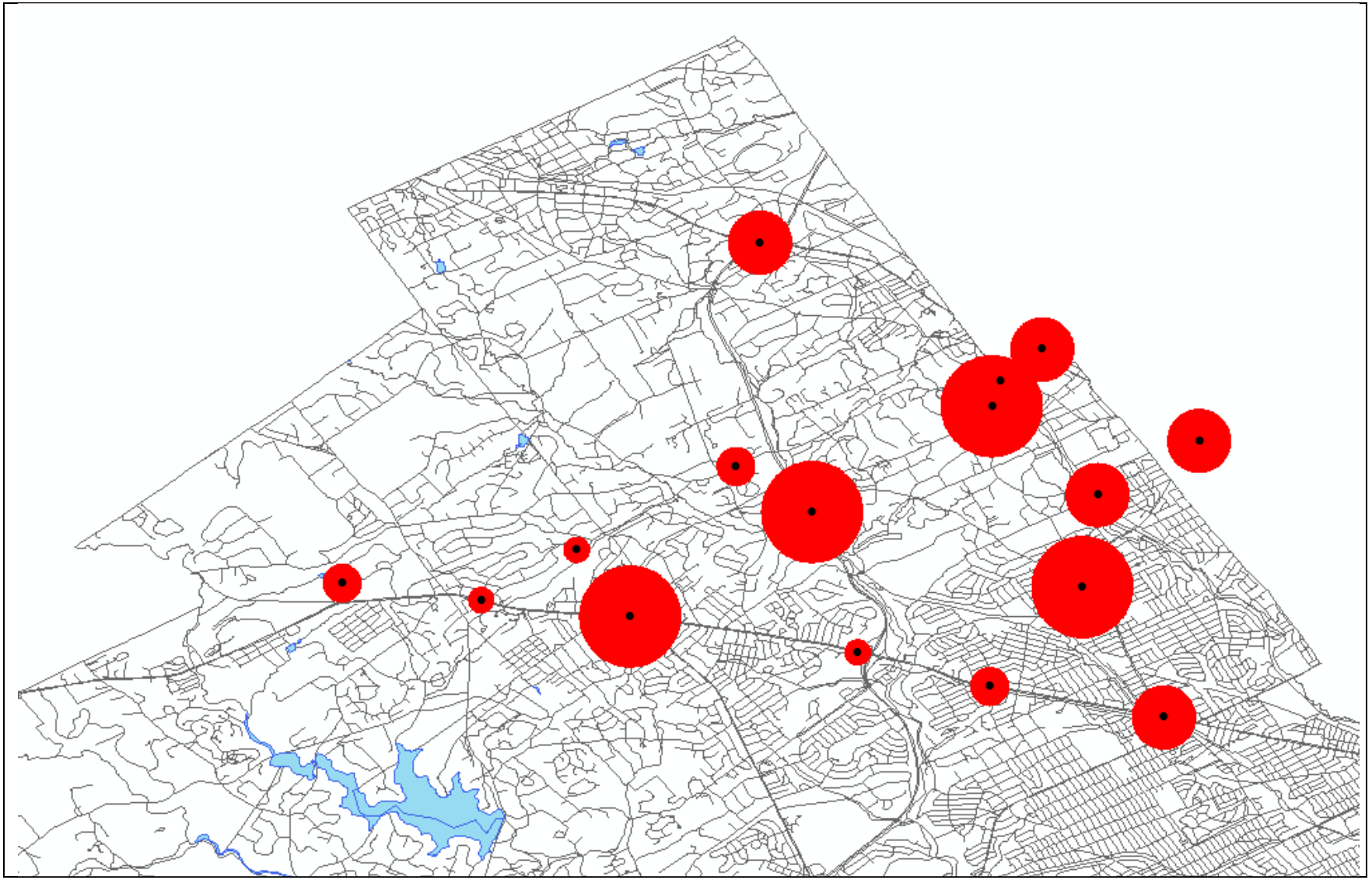


Figure 1: Zombie Hot Spots in upper Delaware County, PA circa March 2013.
Larger circles represent zones of high zombie activity, which should be avoided. Image background is Delaware County, PA line and hydrology 2009 TIGER/Line shapefiles obtained from the US Census.

Source Code

```
# FinalProject.py // parsing KML file --> shapefile
# This script parses a KML file and write geometries and attributes to a point geometry shapefile
# The tool identifies “hot spots” by creating buffers around the points that vary based on an attribute of the point

# Import system modules
import arcpy
from xml.dom import minidom

# Check out the ArcGIS Spatial Analyst extension license
arcpy.CheckOutExtension("Spatial")

# set to overwrite output
arcpy.env.overwriteOutput = True

#Get Parameters
walkerKML = arcpy.GetParameterAsText(0)

arcpy.env.workspace = "C:\\GEOG485\\ProjectFinal"

# Set local variables for default path, shapefile and buffered shapefile
out_path = "C:\\GEOG485\\ProjectFinal"
shapefile = "walkerSightings.shp"
shapefileHotSpot = "C:\\GEOG485\\ProjectFinal\\walkerSightingsBUFF.shp"

# set other variables
buffDistance = "BUFF_DIST"
geometry_type = "POINT"
has_m = "DISABLED"
has_z = "DISABLED"
sideType = "FULL"
endType = "ROUND"
dissolveType = "ALL"

# Obtain a SpatialReference object from a .prj file
spatialRef = arcpy.SpatialReference("C:\\Program Files\\ArcGIS\\Desktop10.0\\Coordinate Systems\\Geographic Coordinate Systems\\World\\WGS 1984.prj")

try:
   # create a new shapefile using CreateFeatureclass, projected in desired spatial reference
   arcpy.CreateFeatureclass_management(out_path, shapefile, geometry_type, "", has_m, has_z, spatialRef)
except:
   print "Could not create shapefile."

try:
   # add a new TEXT field into shapefile to store name of sighting
   arcpy.AddField_management(shapefile, "NAME", "TEXT")
except:
```

```
    print "Could not add NAME text field."

try:
    # add a new SHORT integer field into shapefile to store sighting magnitude
    arcpy.AddField_management(shapefile, "MAGNITUDE", "SHORT")
except:
    print "Could not add MAGNITUDE short integer field."

try:
    # add a new TEXT field into shapefile to store variable buffer distances
    # TEXT field because this will hold legally labled distance values (i.e., '500 Feet')
    arcpy.AddField_management(shapefile, "BUFF_DIST", "TEXT")
except:
    print "Could not add BUFF_DIST long integer field."

# parse the KML file using minidom parse function
kmlFile = minidom.parse(walkerKML)

# get the elements of the kml file within the <kml> tags
kmlElements = kmlFile.getElementsByTagName("kml")[0]

# get the elements within the <Document> tags
documentElements = kmlElements.getElementsByTagName("Document")[0]

# get the elements within the <Placemark> tags
placemarkElements = documentElements.getElementsByTagName("Placemark")

# set up a variable for a new array
pointArray = arcpy.Array()

# set up variables for an insert cursor
cursor = arcpy.InsertCursor(shapefile)
row = cursor.newRow()

try:
    # loop through the Placemark sections of the KML file
    for tag in placemarkElements:
        # set a variable for the coordinates found within the <coordinates> tags
        # .firstChild looks for the first child of that node and .data returns the contents of it
        coordinatePair = tag.getElementsByTagName("coordinates")[0].firstChild.data
        # set a variable list of the coordinates split by a comma
        coordinatePairList = coordinatePair.split(",")
        # set a variable for longitude, in the 0 index position of the list
        longtitude = float(coordinatePairList[0])
        # set a variable for latitude, in the 1 index position of the list
        latitude = float(coordinatePairList[1])
        # set a variable for the name
        nameSighting = tag.getElementsByTagName("name")[0].firstChild.data
        # set a variable for the magnitude (within the discription tag)
        magnitudeSighting = tag.getElementsByTagName("description")[0].firstChild.data
```

```
      # print elements from the KML file
      print nameSighting
      arcpy.AddMessage(nameSighting)

      print "longtidue = " + str(longtitude)
      arcpy.AddMessage("longtidue = " + str(longtitude))

      print "latitude = " + str(latitude)
      arcpy.AddMessage("latitude = " + str(latitude))

      print str(magnitudeSighting) + " walkers found at this location"
      arcpy.AddMessage(str(magnitudeSighting) + " walkers found at this location")

      print ""
      arcpy.AddMessage(" ")

      # Create a point, assigning the X and Y values from your list
      currentPoint = arcpy.Point(longtitude, latitude)
      # Add the newly-created Point to the Array
      pointArray.add(currentPoint)
      # specify the current point as geometry
      row.SHAPE = currentPoint
      # specify nameSighting as the attribute in the NAME field
      row.NAME = nameSighting
      # specify magnitudeSighting as the attributein the MAGNITUDE field
      row.MAGNITUDE = magnitudeSighting
      # insert record into shapefile using the insert cursor
      cursor.insertRow(row)
   else:
      pass
except:
   print "Could not read through KML file"

# Now update the BUFF_DIST field using an update cursor
# Create update cursor to use in next step
cityRows = arcpy.UpdateCursor(shapefile)
city = cityRows.next()

# Define the buffer distance for each row in the BUFF_DIST field, based on the MAGNITUDE Field
# loop through each record and use update cursor to update rows
while city:
   try:
      if city.MAGNITUDE > 60:
         city.BUFF_DIST = '2600 Feet'
      elif city.MAGNITUDE > 20 and city.MAGNITUDE <=60:
         city.BUFF_DIST = '500 Meters'
      elif city.MAGNITUDE > 5 and city.MAGNITUDE <=20:
         city.BUFF_DIST = '300 Meters'
      else:
```

```
          city.BUFF_DIST = '200 Meters'
# call updateRow
       cityRows.updateRow(city)
# go to next row
       city = cityRows.next()
    except:
       print "Could not update buffer distances"

# Create dissolved buffers around sighting locations based on the BUFF_DIST field in the shapefile
try:
    arcpy.Buffer_analysis(shapefile, shapefileHotSpot, buffDistance, sideType, endType, dissolveType)
except:
    print "Could not create hot spots"

# Release locks by deleting cursors
del row
del cursor
del cityRows
del city

# Check in the Spatial Analyst extension now that you're done
arcpy.CheckInExtension("Spatial")
```

Appendix: Project Proposal

ZombieMapper

Vision

When the zombie apocalypse arrives, time management will be critical. Having information on where zombies are congregating will be crucial to survival. Assuming internet networks are still up and running, GIS technology can be used to aid people not-yet-turned in making smarter decisions on-the-go.

See **Figure 1: Functional Requirements Overview** on following page

Project Scope

- An ArcMap Tool which will parse data (a KML file, or an XML or JSON response from a Web service), convert to a common coordinate system as needed, and write geometries to a point feature class.
    - Since zombies tend to congregate in groups, and an observer won't have time to send data about each individual zombie, data on sighting magnitude will also be collected
        - This may be done using an integer value which will also be written to the feature class as an attribute of the geometry point. The integer could represent the number of zombies seen at that point.

- The tool will also identify zombie "hot spots", possibly by aggregating points around a variable radius value

The point feature class can be used by analysts in conjunction with polygon or other feature classes to create information which can be provided to survivors.

Why it's Needed

This tool will automate several steps that would take several minutes for each point. Without a tool, an analyst would have to do the following, for each point:

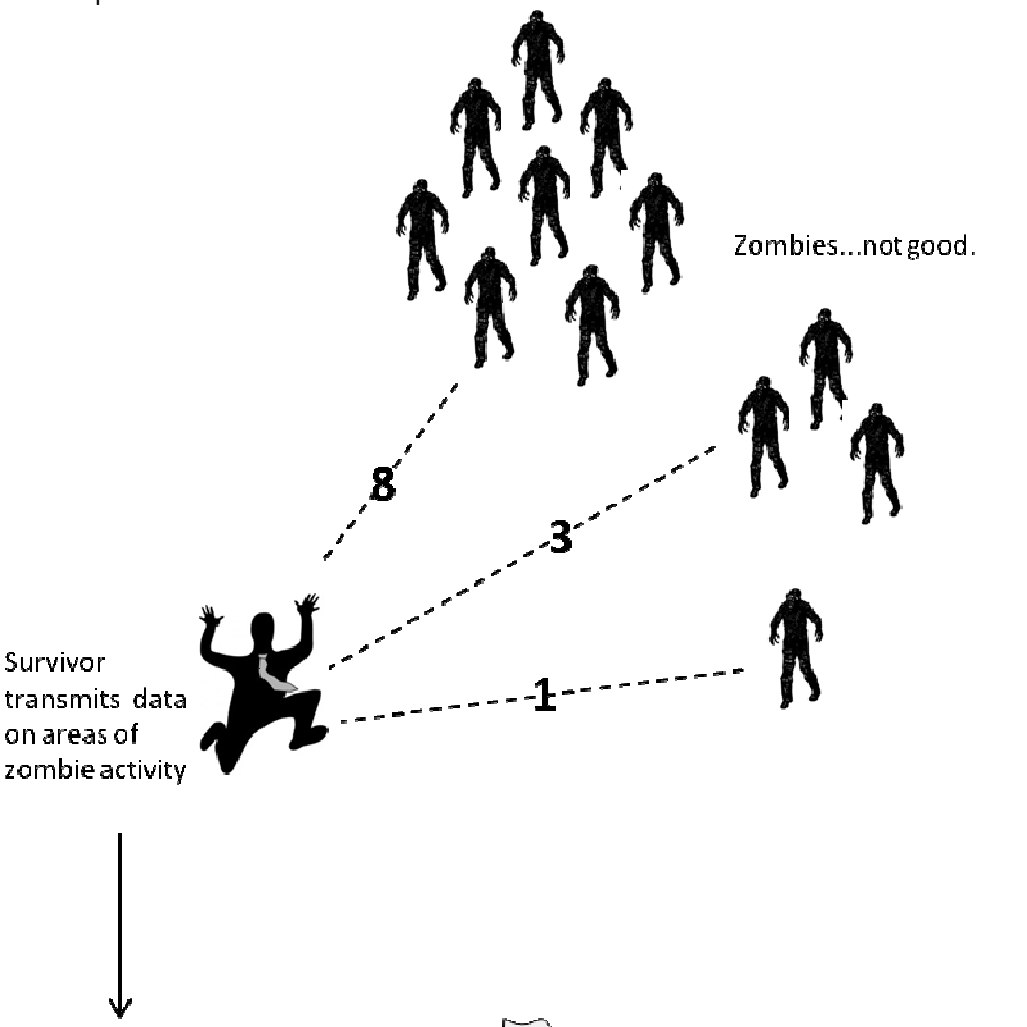
1. Open the file
2. Digitize a new point
3. Possibly re-project
4. Enter attributes into the attribute table for the point
5. Use a tool to aggregate points or do some reclassification

The tool will also be flexible enough so that GIS analyst survivors all over the world could share this tool with minimal modifications. If the zombie apocalypse never happens, this tool could be adapted for other uses, such as recording bird mortality related to West Nile virus, or reporting on potholes by road crews.

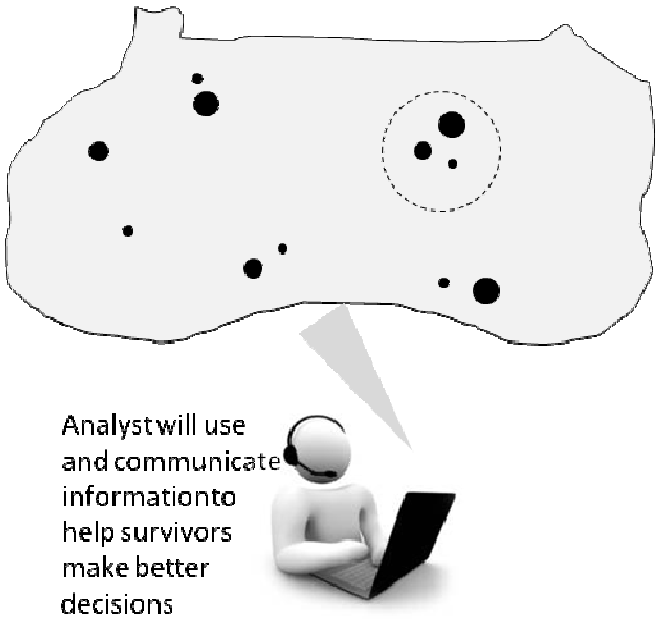
Deliverables

1. A well-documented script tool
2. Sample data
3. Sample Feature Class

Figure 1: Functional Requirements Overview



- The ZombieMapper tool needs to parse zombie location data, convert as needed, and write geometries to a point feature class.
- Data on the magnitude of a sighting also needs to be collected
- The tool needs to identify zombie "hot spots", possibly by aggregating points around a variable radius value.



Sources

KML files for this project were generated using *iMapIt Pro*, Version 1.5 by Marcus Silva on an iPhone4S.

United States Census Bureau. *2009 TIGER/Line® Shapefiles for Delaware, County Pennsylvania*. Retrieved March 2, 2013 from http://www2.census.gov/cgi-bin/shapefiles2009/state-files?state=42.