Xiaoran Wang, Yujing Wu
Geospatial Software Design Homework

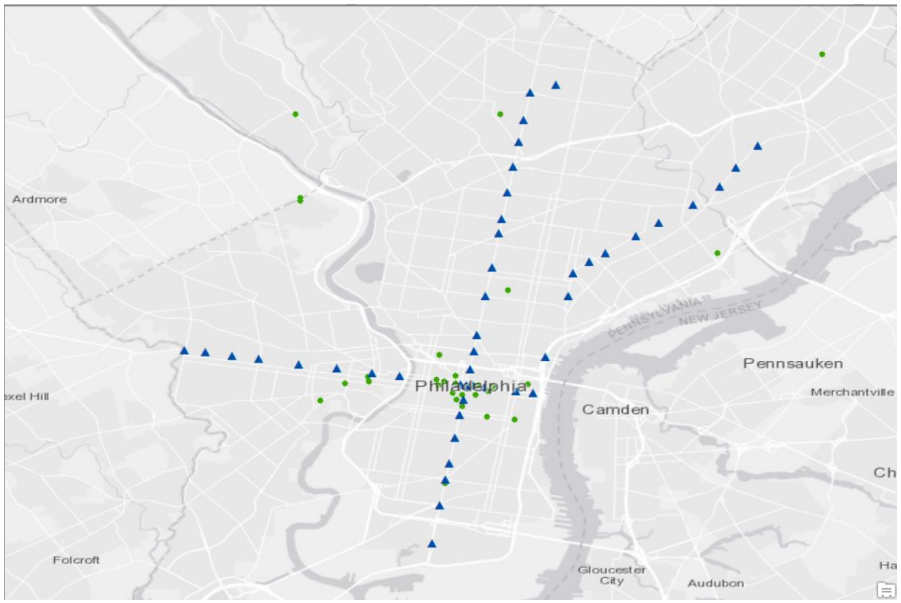
Script tool #1

- Develop a script-based ArcToolbox tool that performs a sequence of geoprocessing functions in order to generate a new grid or shapefile from one or more existing grids and/or shapefiles.

Description

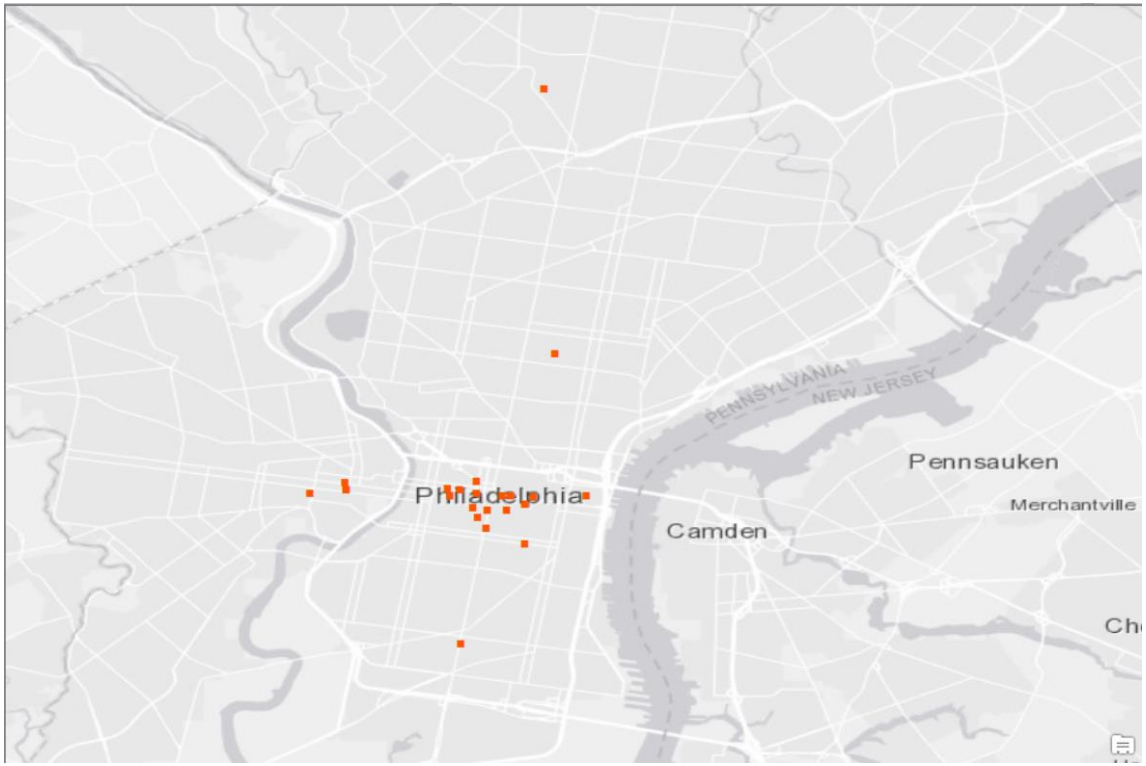
Finishing geoprocessing with ArcGIS Desktop is usually very time-consuming. Users have to click several times and search for the tools to finish the task. Meanwhile, they will export multiple shapefiles before getting their final results. Our script tool aims to simplify the above process to save more storage space for users' computer drive.

In this script tool, users will be asked to provide two point shapefiles and set a radius around the first point shapefile. The script tool will automatically create a buffer around the first shapefile and clip all points of the second shapefile within the buffer radius. The remaining points of the second shapefile will be the only export of this script. As a result, users can easily create a shapefile of points within a user-defined distance from another set of points.



Inputs:

- Blue points: Public Transportation in Philadelphia; Market-Frankford Line & Broad Street Line
- Green points: Starbucks in Philadelphia

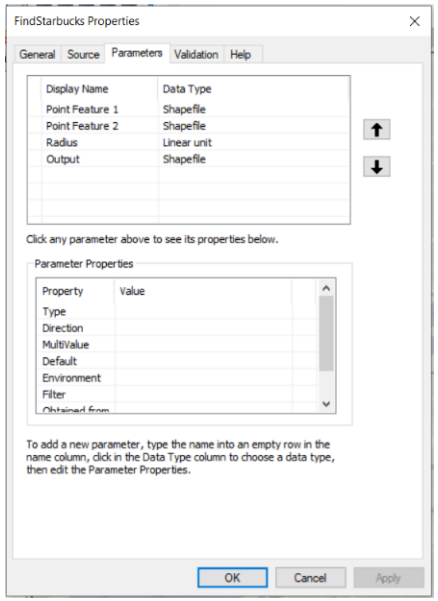


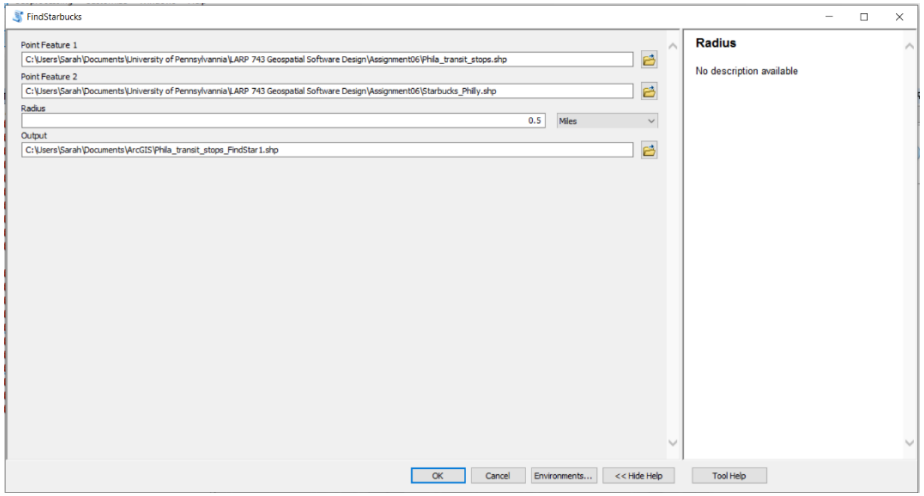
Output: Starbucks within 0.5 miles from the stations

Scripts

```
# Import the necessary modules.
import arcpy, sys, string, os, traceback
try:
# Set the input and output parameters.
    in_features_station = arcpy.GetParameterAsText(0)
    in_features_starbucks = arcpy.GetParameterAsText(1)
    in_radius = arcpy.GetParameterAsText(2)
    out_features_starbucks = arcpy.GetParameterAsText(3)
```

```
# Create the buffer according to the distance input by the user.
    intermediate_buffer = arcpy.Buffer_analysis(in_features_station, out_features_starbucks[:-4] + "_temp"  +
".shp", in_radius, "FULL", "ROUND", "NONE", "")
# Clip the starbuck shapefile according to the buffer.
    arcpy.Clip_analysis(in_features_starbucks, intermediate_buffer, out_features_starbucks, "")
# Delete the buffer created by the previous step.
    arcpy.Delete_management(intermediate_buffer)
# Output the necessary error message in case our script fails.
except Exception as e:
    arcpy.AddError('\n' + "Script failed because: \t\t" + e.message )
    exceptionreport = sys.exc_info()[2]
    fullermessage   = traceback.format_tb(exceptionreport)[0]
    arcpy.AddError("at this location: \n\n" + fullermessage + "\n")
```





Script Tool Interface

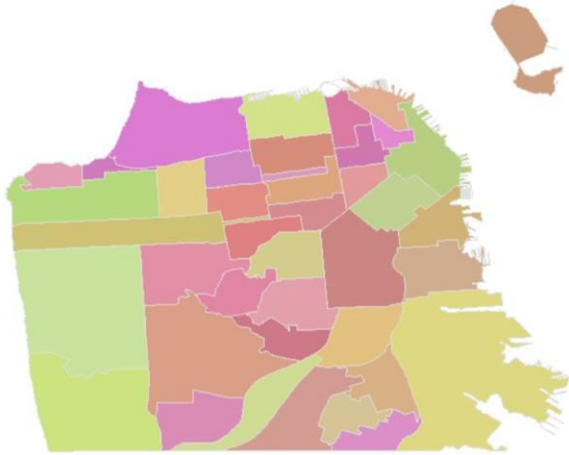
Tool #2

- Develop a script-based ArcToolbox tool that generates values in the attribute table of a new shapefile by processing values in the attribute table of an existing shapefile.

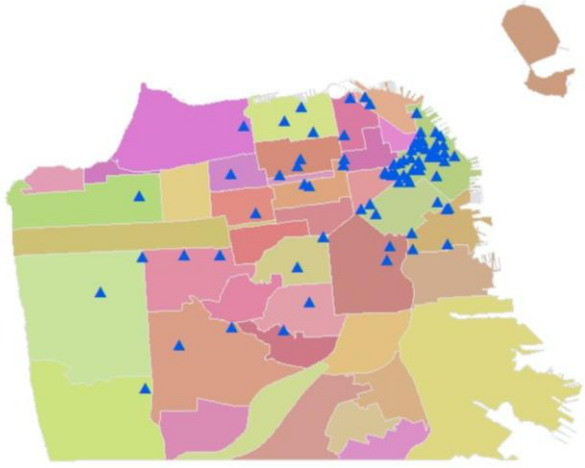
Description

The script tool is designed to calculate the distribution ratio of points in polygons (average area of polygon per point). To keep it simple and convenient, this script tool only needs users to specify two inputs and one output. The script tool is able to count the number of points in each polygon and do the calculation of distribution ratio. The final output creates a duplicate of the input polygon layer with a new column added to the attribute table, which shows the result of the above calculation.

To illustrate the function of our script, we used the locations of Starbucks and chose San Francisco as the study area. We want to use this tool to calculate the average area served by each Starbuck in each San Francisco neighborhood.



Input 1: Neighborhoods in San Francisco



Input 2: Starbucks in San Francisco

FID	Shape *	Join_Count	nhood	Area_st
0	Polygon	0	Bayview Hunters Point	0
1	Polygon	0	Bernal Heights	0
2	Polygon	3	Castro/Upper Market	0.00008
3	Polygon	3	Chinatown	0.00002
4	Polygon	0	Excelsior	0
5	Polygon	28	Financial District/South Beach	0.00001
6	Polygon	1	Glen Park	0.00018
7	Polygon	0	Inner Richmond	0
8	Polygon	0	Golden Gate Park	0
9	Polygon	0	Haight Ashbury	0
10	Polygon	0	Hayes Valley	0
11	Polygon	2	Inner Sunset	0.00019
12	Polygon	0	Japantown	0
13	Polygon	0	McLaren Park	0
14	Polygon	6	Tenderloin	0.00002
15	Polygon	1	Lakeshore	0.00076
16	Polygon	0	Lincoln Park	0
17	Polygon	1	Lone Mountain/USF	0.00015
18	Polygon	3	Marina	0.00009
19	Polygon	2	Russian Hill	0.00007
20	Polygon	2	Mission	0.00025
21	Polygon	5	Mission Bay	0.00004

Output: A new column “Area_st” added to the attribute table of the polygon shapefile

Scripts

```
# Import necessary modules.
import arcpy, sys, string, os, traceback

# Allow output file to overwrite any existing file of the same name
arcpy.env.overwriteOutput = True
```

```
try:
    # specify the input and output
    in_features_neighborhood = arcpy.GetParameterAsText(0)
    in_features_business = arcpy.GetParameterAsText(1)
    OutputShapefile = arcpy.GetParameterAsText(2)

    # specify the target and join features for spatial join
    targetFeatures = in_features_neighborhood
    joinFeatures = in_features_business

    # create a field mapping and add the input features
    fieldmappings = arcpy.FieldMappings()
    fieldmappings.addTable(targetFeatures)
    fieldmappings.addTable(joinFeatures)

    # get the output field's properties as a field object, specify that we want to count the number of join features
per polygon in the target feature
    Num_starbucks = fieldmappings.findFieldMapIndex("Name")
    fieldmap = fieldmappings.getFieldMap(Num_starbucks)
    field = fieldmap.outputField
    field.name = "Num_star"
    field.aliasName = "Num_star"
    fieldmap.outputField = field
    fieldmap.mergeRule = "count"
    fieldmappings.replaceFieldMap(Num_starbucks, fieldmap)

    # spatial join, add new field
    arcpy.SpatialJoin_analysis(targetFeatures, joinFeatures, OutputShapefile, "#", "#", fieldmappings)
```

```
    arcpy.AddField_management(OutputShapefile, 'Area_st', "DOUBLE", 20, 5)    enumerationOfRecords =
    arcpy.UpdateCursor(OutputShapefile)
    # loop over the attribute table
   # change the type of column in case it is not a numeric type
   # calculate area/number of starbucks and input the value into the new field    nameOfShapeField =
   arcpy.Describe(OutputShapefile).shapeFieldName
   for nextRecord in enumerationOfRecords:
       nextShape   = nextRecord.getValue(nameOfShapeField)
       area       = nextShape.area
       number_starbucks = str(nextRecord.getValue('Num_star')).strip()
       if number_starbucks == ' ':
           number_starbucks = 0
     else:
           number_starbucks = float(number_starbucks)
           distribution_ratio    = area/number_starbucks
           nextRecord.setValue('Area_st', distribution_ratio)
           enumerationOfRecords.updateRow(nextRecord)

   del nextRecord
   del enumerationOfRecords
except Exception as e:
   # If unsuccessful, end gracefully by indicating why
   arcpy.AddError('\n' + "Script failed because: \t\t" + e.message )
   # ... and where
   exceptionreport = sys.exc_info()[2]
   fullermessage   = traceback.format_tb(exceptionreport)[0]
   arcpy.AddError("at this location: \n\n" + fullermessage + "\n")
   enumerationOfRecords.updateRow(nextRecord)
```

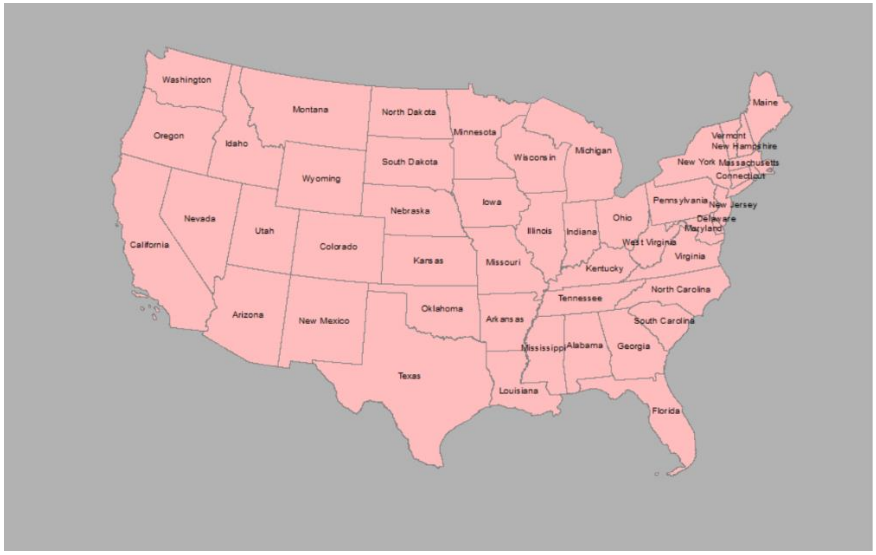
Tool #3

- Develop a tool that generates a shapefile of all-new points, lines, or polygons.

Description

Calculating and changing polygon size is the hardest part of this script tool. There are limitations on using the numeric variable to change the polygon areas. When the numeric data has strong outliers, or the data range is too wide, the new polygon size would be hard to control. Since the data we are using (the number of starbucks in each state) has a very skewed distribution, we use log() to make it easier to represent through size of the polygons. Additionally, the original size of the polygon may affect the new size. For instance, after the size transformation, the large polygons with a low number of Starbucks are likely to have similar size with the small polygons with a large number of Starbucks. The cartogram would be meaningless in this scenario. We used size instead of area to represent the numerical variable since the irregular shapes of the polygons make it hard to transform the polygon to achieve a certain area. A more complicated algorithm would be needed if we want to directly transform the area based on the density and keep the true shape.

Below, the white boundaries represent the resulting shapefile of polygons expanded 4 times based on the centroid of US. The orange polygons represent the resulting shapefile of polygons contracted 4 times based on the centroid of each polygon. Finally, the green polygons represent the final product: the size of the orange polygons times log(number of starbucks) divided by log(current area of the polygon).



Starbucks_bystate

FID	Shape *	Join_Count
0	Polygon	20
1	Polygon	557
2	Polygon	506
3	Polygon	122
4	Polygon	214
5	Polygon	23
6	Polygon	51
7	Polygon	20
8	Polygon	229
9	Polygon	4
10	Polygon	97
11	Polygon	16

Input: Polygon shapefile with a column of numerical variable.

Output: Intermediate and final outputs: US expanded to four times its size (white boundary), each state contracted to ¼ of its size (orange polygons), the final cartogram (the green polygons)

Scripts

```
# THIS SCRIPT CREATES A CARTOGRAM FROM A SHAPEFILE CONTAINING A SET OF POLYGONS.
"""
  DISPLAY NAME        DATA TYPE       PROPERTY>DIRECTION>VALUE
  in_polygon          Shapefile       Input
  in_field            Double          Obtained from Input Shapefile
  out_polygon         Shapefile       Output
"""
```

```
# Import necessary modules
import sys, os, string, math, arcpy, traceback

# Allow output file to overwrite any existing file of the same name
arcpy.env.overwriteOutput = True


try:
    # Read and write names of input and output shapefiles
    nameOfInputFeatureLayer     = arcpy.GetParameterAsText(0)
    nameoffield                 = arcpy.GetParameterAsText(2)
    nameOfOutputShapefile       = arcpy.GetParameterAsText(1)
    arcpy.AddMessage('\n' + "Input shapefile: \t" + nameOfInputFeatureLayer)
    arcpy.AddMessage("Output shapefile: \t" + nameOfOutputShapefile + "\n")

    # Create a duplicate of the input shapefile.
    arcpy.Copy_management(nameOfInputFeatureLayer, nameOfOutputShapefile)

    # Dissolve the input shapefile and find its centroid.
    to_dissolve = nameOfInputFeatureLayer
    dissolved = arcpy.env.scratchGDB + '/dissolved'
    dissolved_output = arcpy.Dissolve_management(to_dissolve, dissolved)
    attributeTable = arcpy.SearchCursor(dissolved_output)
    nameOfShapeField = arcpy.Describe(dissolved_output).shapeFieldName
    for nextRecord in attributeTable:
        # Get the centroid of the dissolved shapefile.
        nextFeature             = nextRecord.getValue(nameOfShapeField)
        pointAtCenter           = nextFeature.centroid
```

```
arcpy.AddMessage("The centroid of the shapefile")
del nextRecord
del attributeTable

# First, we want to enlarge the input shapefile by 4 times.
enlargementFactor = 4

# Initialize an object to temporarily hold each new point as it is created
newPoint = arcpy.Point()

# Initialize a list to hold all of the new features to be created
listOfNewFeatures = []

# Get the input shapefile's attribute table and the name of its shape field
attributeTable = arcpy.UpdateCursor(nameOfOutputShapefile)
nameOfShapeField = arcpy.Describe(nameOfOutputShapefile).shapeFieldName

# Loop through the records of the input shapefile's attribute table, i.e. its features
for nextRecord in attributeTable:
  # Get the next feature of the input shapefile.
    nextFeature            = nextRecord.getValue(nameOfShapeField)

    # Initialize an array to hold the islands (i.e. parts) of a new feature to be created
    arrayOfNewIslands      = arcpy.Array()

    # Cycle through the islands of the current feature
    for nextIsland in nextFeature:
       # Initialize an array to hold the points for a new island to be created
        arrayOfNewPoints    = arcpy.Array()
```

```
                # Cycle through original island's vertices, creating a new point from each
                for nextVertex in nextIsland:
                    if nextVertex:
                      #If the next vertex is non-Null, create a new point and add it to the array of new points
                        newPoint.X = pointAtCenter.X-((pointAtCenter.X - nextVertex.X)*enlargementFactor)
                        newPoint.Y = pointAtCenter.Y-((pointAtCenter.Y - nextVertex.Y )*enlargementFactor)
                        arrayOfNewPoints.add(newPoint)
                    else:
                        arcpy.AddMessage("\t\tHOLE: (beginning with a Null point)")
                      # If the next vertex is Null, insert a new point that is also Null
                        arrayOfNewPoints.append(None)
                # After creating an array of new points for a given island, add it to this feature's array of new
islands
                    arrayOfNewIslands.append(arrayOfNewPoints)

            # After creating an array new islands for a given feature, create a new feature from that array
            newFeature = arcpy.Polygon(arrayOfNewIslands)

            # After creating a new feature, assign it the the record
            nextRecord.setValue(nameOfShapeField, newFeature)
            attributeTable.updateRow(nextRecord)
        arcpy.AddMessage("Enlarge the shapefile by 4 times")

        # Initialize an object to temporarily hold each new point as it is created
        newPoint = arcpy.Point()
        # Initialize a list to hold all of the new features to be created
        listOfNewFeatures = []
```

```
        # Get the input shapefile's attribute table and the name of its shape field
        attributeTable = arcpy.UpdateCursor(nameOfOutputShapefile)
        nameOfShapeField = arcpy.Describe(nameOfOutputShapefile).shapeFieldName

        # Loop through the records of the input shapefile's attribute table, i.e. its features
        for nextRecord in attributeTable:
          # Get the next feature of the input shapefile.
            nextFeature  = nextRecord.getValue(nameOfShapeField)
            pointAtCenter    = nextFeature.centroid
            count_starbucks = nextRecord.getValue(nameoffield)
            current_area = nextFeature.area

            # Initialize an array to hold the islands (i.e. parts) of a new feature to be created
            arrayOfNewIslands      = arcpy.Array()

            # Cycle through the islands of the current feature
            for nextIsland in nextFeature:
                arcpy.AddMessage("\tISLAND:")

                # Initialize an array to hold the points for a new island to be created
                arrayOfNewPoints  = arcpy.Array()

                # Cycle through original island's vertices, creating a new point from each
                for nextVertex in nextIsland:
                    if nextVertex:
                        # If the next vertex is non-Null, create a new point and add it to the array of new points
                        newPoint.X  = pointAtCenter.X  - (((pointAtCenter.X - nextVertex.X ) / enlargementFactor)
*  math.log(count_starbucks + 1) / (math.log(current_area) / 10))
```

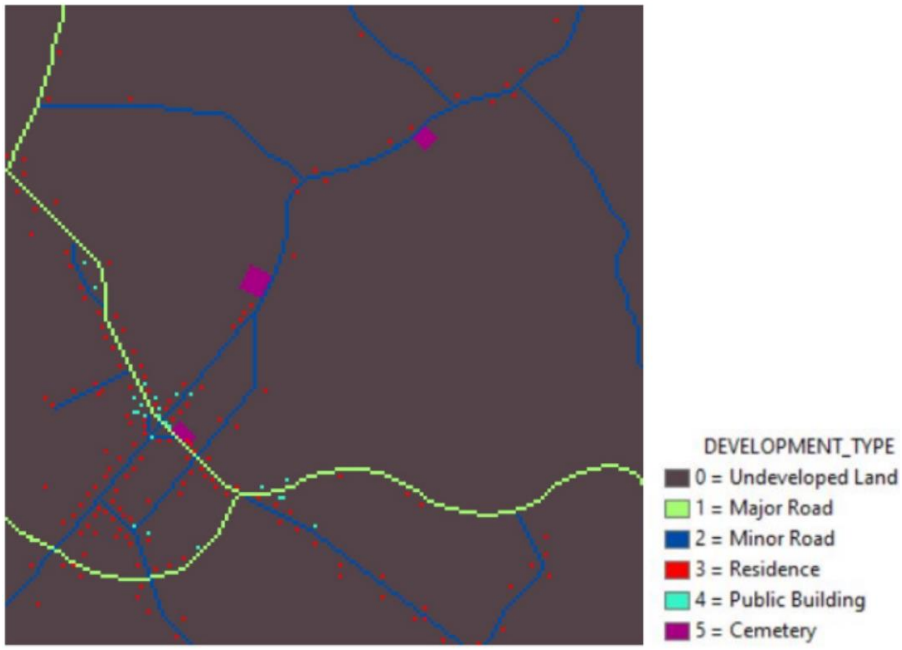
```
                            newPoint.Y  = pointAtCenter.Y  - (((pointAtCenter.Y - nextVertex.Y ) / enlargementFactor) *
math.log(count_starbucks + 1) / (math.log(current_area) / 10))
                            arrayOfNewPoints.add(newPoint)
                        else:
                            # If the next vertex is Null, insert a new point that is also Null
                            arrayOfNewPoints.append(None)
                    # After creating an array of new points for a given island, add it to this feature's array of new
islands
                    arrayOfNewIslands.append(arrayOfNewPoints)

                # After creating an array new islands for a given feature, create a new feature from that array
                newFeature = arcpy.Polygon(arrayOfNewIslands)

                # After creating a new feature, assign it the the record
                nextRecord.setValue(nameOfShapeField, newFeature)
                attributeTable.updateRow(nextRecord)

            # Delete row and update cursor objects to avoid locking attribute table
            del nextRecord
            del attributeTable

        except Exception as e:
            # If unsuccessful, end gracefully by indicating why
            arcpy.AddError('\n' + "Script failed because: \t\t" + e.message )
            # ... and where
            exceptionreport = sys.exc_info()[2]
            fullermessage   = traceback.format_tb(exceptionreport)[0]
            arcpy.AddError("at this location: \n\n" + fullermessage + "\n")
```

Tool #4

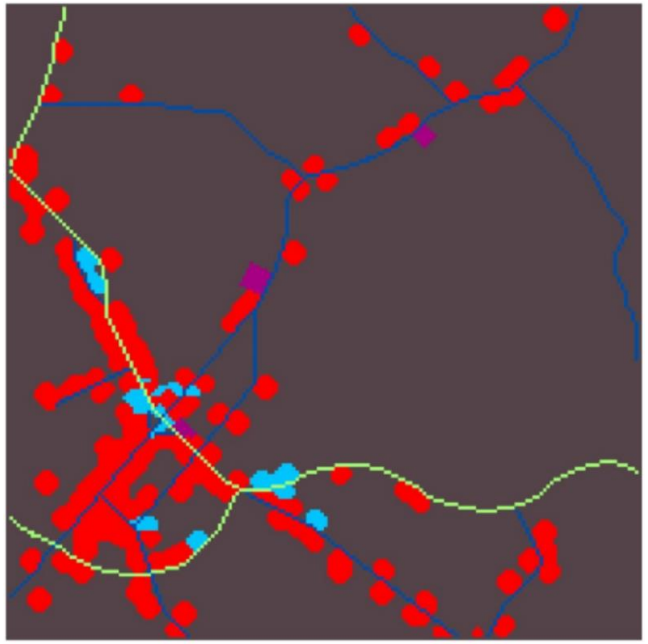
- Develop a tool to generate a new grid in which each pixel's value is calculated from the values of its adjacent neighbors on an existing grid.

Description

We aimed to simulate the process of urbanization through the development dataset. There are five categories of land use in the development dataset: undeveloped land, major roads, minor roads, residences, public buildings, and cemeteries. Among these five different land uses, we only expanded two: residences and public buildings. In addition, the urbanization we simulated only converted undeveloped land to developed land uses. Thus, we examined whether the cell is undeveloped first. Since we want to control the speed of urbanization, we made sure that the undeveloped cell was converted to residence or public building only if the number of neighbors that was residence or public building was higher than the iteration number.



Example Input: The development raster dataset



Example Output: The development raster dataset after 10 iterations

Scripts

```
"""
This script stimulates the process of urbanization.
We want to urbanize the undeveloped area through expanding regions that are already urbanized (residence and public
buildings).

DISPLAY NAME        DATA TYPE       PROPERTY>DIRECTION>VALUE
Input Grid          Raster Layer    Input
Output Grid         Raster Dataset  Output
Iterations          Long            Input
"""

# Import external modules
import sys, os, string, math, arcpy, traceback, numpy, random

# Allow output to overwite any existing grid of the same name
arcpy.env.overwriteOutput = True

# If Spatial Analyst license is available, check it out
if arcpy.CheckExtension("spatial") == "Available":
    arcpy.CheckOutExtension("spatial")

    try:
        # Create a real-valued InputArray from the initial input grid and note its dimensions
        InputGridName       = arcpy.GetParameterAsText(0)
        inputArray          = arcpy.RasterToNumPyArray(InputGridName)
        inputArray          = inputArray.astype(float)
        howManyRows         = inputArray.shape[0]
        howManyColumns      = inputArray.shape[1]
```

```
        # Initialize an OutputArray that is similar to that InutArray but filled with zeroes
        intermediateArray   = numpy.zeros_like(inputArray)      # a real-number array storing each iteration's output
values

        # Get User-specified number of iterations
        howManyIterations   = int(arcpy.GetParameterAsText(2))  # an integer indicating the number of local dispersions to
apply
        if howManyIterations < 0: howManyIterations = 10

        # Start timing
        timeStart           = time.clock()

        # Loop through as many iterations as requested
        for iterationNumber in range(howManyIterations):
            arcpy.AddMessage("\nIteration " + str(iterationNumber))

            # Loop through rows and columns of pixels, skipping those at the edges (recalling that range(A,B) stops just
short of B)
            for thisRow in range(1,howManyRows-1):
                for thisColumn in range(1,howManyColumns-1):
                    if inputArray[thisRow][thisColumn] != 0:
                        intermediateArray[thisRow][thisColumn] = inputArray[thisRow][thisColumn]
                    else:
                        numOfResident = 0
                        numOfPublic = 0
                        # Loop through the immediate neighborhood of each pixel
                        for neighborRow in range(thisRow-1,thisRow+2):
                            for neighborColumn in range(thisColumn-1,thisColumn+2):
```

```
                            if inputArray[neighborRow][neighborColumn] == 3:
                                numOfResident += 1
                            elif inputArray[neighborRow][neighborColumn] == 4:
                                numOfPublic += 1
                        if numOfPublic > numOfResident:
                            if numOfPublic >= iterationNumber + 1:
                                intermediateArray[thisRow][thisColumn] = 4
                        elif numOfResident >= iterationNumber + 1:
                            intermediateArray[thisRow][thisColumn] = 3

            # Once neighborhood means have been computed for all pixels, use them to update inputArray
            inputArray    = numpy.copy(intermediateArray)

        # Stop timing
        timeStop       = time.clock()
        timeTaken      = timeStop - timeStart
        arcpy.AddMessage("\nElapsed time = " + str(timeTaken) + " seconds\n")

        # Create output grid from that new array
        inputGrid      = arcpy.Raster(InputGridName)
        gridExtent     = inputGrid.extent
        lowerleftPoint  = gridExtent.lowerLeft
        gridResolution  = inputGrid.meanCellWidth
        outputGrid     = arcpy.NumPyArrayToRaster(intermediateArray,lowerleftPoint,gridResolution)
        outputGrid.save(arcpy.GetParameterAsText(1))

    except Exception as e:
        # If unsuccessful, end gracefully by indicating why
        arcpy.AddError('\n' + "Script failed because: \t\t" + e.message )
```

```
        # ... and where
        exceptionreport = sys.exc_info()[2]
        fullermessage   = traceback.format_tb(exceptionreport)[0]
        arcpy.AddError("at this location: \n\n" + fullermessage + "\n")

    # Check in Spatial Analyst extension license
    arcpy.CheckInExtension("spatial")
else:
    print "Spatial Analyst license is " + arcpy.CheckExtension("spatial")
```