Project 3: Aggregating graffiti incidents

You've just been hired as a GIS developer in a large city and are receiving your first Python assignment. Your co-workers in the GIS department are very excited because they've just deployed the city's first "citizen participation" Web map. The map is a community effort to target graffiti: anyone on the Web can go to the map and place a point to report a graffiti incident.

The application has already been wildly successful in its first few months of operation and your department has amassed a large amount of point data showing graffiti incidents. However, the police chief is now interested in seeing an aggregation of this data by patrol zones. The goal is to set a priority on each zone and allot more resources to fighting graffiti in the high priority zones.

The data for this project is contained in a file geodatabase called PoliceData.gdb.

Your task

You have a point feature class of graffiti incidents and a polygon feature class of patrol zones with some empty attributes already created for you. You must write a script that updates the attributes of the patrol zones with:

The number of graffiti incidents falling within the patrol zone. This is an integer that goes in the INCIDENTS field.

The priority ranking for the patrol zone. This is a string that goes in the PRIORITY field. You will derive this string using some simple math that compares the number of incidents in the zone with the area of the zone.

Patrol zone priority rankings

You will calculate a priority ranking for each zone by dividing the number of graffiti incidents in the zone by the area of the zone. Your script should then examine the result and assign the appropriate priority ranking (PRIORITY). These are the priority rankings:

TOP CONCERN—15 or more incidents per square mile

HIGH CONCERN— At least 12 but less than 15 incidents per square mile

SOME CONCERN— At least 6 but less than 12 incidents per square mile

LOW CONCERN—Fewer than 6 incidents per square mile

Deliverables

The deliverables for this project are:

A structured zipped archive that contains:

Your main project folder: Project3_<your_name>

The following subfolders: a Docs folder, a Scripts folder, a Data folder or the file geodatabase (you have some flexibility in naming these folders, but the purpose should be as outlined above)

Your Python script (.py file) that performs the above tasks, stored in the Scripts folder

A screenshot of your patrol zones attribute table after running the script, stored in the Docs folder.

A short writeup (about 300 words) describing what you learned and how you approached the problem. If you included any "over and above" efforts, please describe these here so that your instructor know to look for them. Store this document in your Docs folder.

A toolbox containing a script tool that runs your model. You can expose any parameters that you want, but the primary one that will keep the tool flexible is a parameter for setting the workspace environment. The toolbox should be in the primary project folder, not a subfolder.

A map document with the relative path set to the file geodatabase. The map document should be in the primary project folder, not a subfolder.

Successful delivery of the above requirements is sufficient to earn 90% on the project. The remaining 10% is reserved for efforts that go "over and above" the minimum requirements. This could include (but is not limited to) addition of another column to the attribute table that is updated with some meaningful statistics about the incidents in each zone, the creation of a new table or feature class that you insert new rows into with your primary script tool or a secondary script tool, a map symbolizing the resulting zone priorities, or a list of other real-world problems a similar script could solve. There will not be over and above points for good commenting and error-handling as this is expected now.

Challenges

In this project, you need to manage an update cursor and perform repeated SelectLayerByLocation operations in order to figure out how many incidents each zone contains. You then need to use the number of incidents to calculate the incidents per square mile, and make a decision about which priority to assign.

Approach

One approach you could take for finding the number of incidents per zone could be very similar to what is presented in Lesson 3 Practice Exercise B (see https://www.e-education.psu.edu/geog485/node/158) to find the number of park and rides inside a city boundary. In the PSU example exercise, the goal is to count whether two or more items were found. For Project 3, you will need to get a count on the entire number of incidents selected. This is easily done using the GetCount tool. I encourage you to spend some time studying Practice Exercise B and its accompanying solution. Another important point is to create a feature layer to use with the SelectLayerByLocation tool.

There are many ways to approach the script, and the majority of available credit will be awarded just for solving the problem. More points will be awarded for scripts that solve the problem in the most efficient way possible. For example, if you can get a job done with one loop instead of two, this is more efficient. If you loop through 10 objects instead of 100 and accomplish the same thing, this is more efficient. Look for ways to economize what your script is doing. It is a wonderful feeling to delete unnecessary code.

Hints

Take some time to attempt the PSU practice exercises for Lesson 3. It is worth your time to study their solutions to the point where you understand everything that is happening.

Do not procrastinate. If you are new to programming, plan to spend the majority of the allotted lesson time working on the project.

Before you begin working, manually make a copy of the patrol areas and place it in the same file geodatabase (name it something like PatrolZones_Backup). Your script is going to modify the patrol zones, and as you test, you need a convenient way to restore the data to its original state. After each test run, you can make a new copy of the dataset. If you fail to do this, you'll have to download the data and unzip it again each time you re-run the script.

Not only do the cursors place locks on your data, but the feature layers do as well. You can call arcpy.Delete_management("TheFeatureLayer") to delete the feature layers; however, know that it might not be possible to get rid of all the locks. Be prepared to close PythonWin and/or ArcCatalog or ArcMap if you're having trouble restoring your original zones dataset after a test run. Save often.

Use the debugging toolbar to help you. Set up watches on your variables and observe how your variable values change as you step through each line of your code. This is what full-time programmers do during a vast chunk of their time.

File geodatabases always have an area field called SHAPE_Area. Use this to get the initial area, which for this dataset will be in square meters. Divide (/) the number of square meters by 2589988.11 to get square miles.

If you get stuck or burned out, you may want to take a break and read through the lesson again, front to back. The lesson material will take on a whole new meaning once you have actually tried to write some code using the described techniques, and you may read something that will help you get past your brick wall. Also, the ArcGIS Help topics linked to in the lesson are very helpful for this assignment, especially the ones about making feature layers and selecting data.