Final Project (part 2) Script

```
"""
THIS SCRIPT TAKES IN A SHAPEFILE OF POLYGONS, COMPUTES A MEASURE OF
ANNEXATION-PRIORITY FOR EACH NEIGHBOR PAIR OF POLYGONS,
AND MERGES THE POLYGONS WITH THE LOWEST MEASURE VALUE.  IT DOES THIS
REPEATEDLY UNTIL A DESIRED NUMBER OF POLYGONS ARE PRESENT

To create an ArcToolbox tool with which to execute this script, do the following.
1   In  ArcMap > Catalog > Toolboxes > My Toolboxes, either select an existing toolbox
        or right-click on My Toolboxes and use New > Toolbox to create (then rename) a new
one.
2   Drag (or use ArcToolbox > Add Toolbox to add) this toolbox to ArcToolbox.
3   Right-click on the toolbox in ArcToolbox, and use Add > Script to open a dialog box.
4   In this Add Script dialog box, use Label to name the tool being created, and press Next.
5   In a new dialog box, browse to the .py file to be invoked by this tool, and press Next.
6   In the next dialog box, specify the following inputs (using dropdown menus wherever
possible)
        before pressing OK or Finish.
        DISPLAY NAME                DATA TYPE    PROPERTY>DIRECTION>VALUE
        Input Shapefile             Feature Layer   Input
        Desired Number of Features          Double          Input
        Input Area Field            Field           Input   ("use 'area' as default)
        Input Unique ID Field       Field           Input   ("use 'Id' as default)
        Input Value Field           Field           Input   ("use 'gridcode' as default)
        Output NNs dbf              Table           Output

  To later revise any of this, right-click to the tool's name and select Properties.
"""

# Import necessary modules
import sys, os, string, math, arcpy, traceback, numpy, random, cmath, decimal, operator


# Check to see if Spatial Analyst license is available
if arcpy.CheckExtension("spatial") == "Available":

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

        try:
        nameOfInputFeatureLayer     = arcpy.GetParameterAsText(0)
        desiredNumberOfFeatures     = arcpy.GetParameterAsText(1)
```

```
        nameOfAreaField             = arcpy.GetParameterAsText(2)  #for now note that this
field should be named area.  note that this should not be the automatically generated
"SHAPE_AREA" field -- add new field and calculate a duplicate so that ArcMap won't throw the
obnoxious "can't change required field" error.
        nameOfUniqueIDField         = arcpy.GetParameterAsText(3)
        nameOfValueField            = arcpy.GetParameterAsText(4)  #for now note that this
field should be named gridcode
        nameOfOutputNNs             = arcpy.GetParameterAsText(5)

        # Confirm import of raster file
        arcpy.AddMessage("Input shapefile is named " + (nameOfInputFeatureLayer))
        arcpy.AddMessage("desired Number Of Features is " + (desiredNumberOfFeatures))
        arcpy.AddMessage("Area field is named " + (nameOfAreaField))
        arcpy.AddMessage("Field with Unique ID is " + str(nameOfUniqueIDField))

        """
        1) for each polygon, compile list of polygon with which they are contiguous.  Use the
non-dissolved raster to polygon for this.
        2) for each pair of polygon and its contiguous neighbor:
        difference in value = absolute value of (subtract polygon's value from neighbor's value)
        difference in area = subtract neighbor's area (in cells) from polygon's area (in cells).
                If neighbor's area in cells is greater than polygon's area,
                i.e. difference in area is negative, then assign value .000001 (or null) and skip
        measure = difference in value/difference in area
        3) Pair with lowest measure should be combined, by assigning polygon's value to its
neighbor.
        """

        #create reference objects for the fields of the NNs table
        newSourceIDField = str("src_" + nameOfUniqueIDField)
        newSourceValueField = str("src_" + nameOfValueField)
        newSourceAreaField = str("src_" + nameOfAreaField)
        newNeighborIDField = str("nbr_" + nameOfUniqueIDField)
        newNeighborValueField = str("nbr_" + nameOfValueField)
        newNeighborAreaField = str("nbr_" + nameOfAreaField)
        arcpy.AddMessage("new source ID field is " + str(newSourceIDField))
        arcpy.AddMessage("new source Value field is " + str(newSourceValueField))
        arcpy.AddMessage("new source Area field is " + str(newSourceAreaField))
        arcpy.AddMessage("new neighbor ID field is " + str(newNeighborIDField))
        arcpy.AddMessage("new neighbor Value field is " + str(newNeighborValueField))
        arcpy.AddMessage("new neighbor Area field is " + str(newNeighborAreaField))
        toJoinFieldList = [nameOfValueField,nameOfAreaField]
```

```
        #Define several feature layers to use
        intermediateFeatureLayer = nameOfInputFeatureLayer
        arcpy.AddMessage("intermediateFeatureLayer is " + str(intermediateFeatureLayer))

        count_of_features_not_a_number =
arcpy.GetCount_management(nameOfInputFeatureLayer)
        count_of_features = int(count_of_features_not_a_number.getOutput(0))

        arcpy.AddMessage("initial count of features is " + str(count_of_features))
        arcpy.AddMessage("describe initial count of features " + str(type(count_of_features)))

        desired_feature_count = int(desiredNumberOfFeatures)
        arcpy.AddMessage("desired count of features is " + str(desired_feature_count))
        arcpy.AddMessage("describe desired count of features " +
str(type(desired_feature_count)))

        while count_of_features > desired_feature_count:
        arcpy.AddMessage("start count of features is " + str(count_of_features))

        #create list of nearest neighbors
        outputNNsTable = arcpy.PolygonNeighbors_analysis(intermediateFeatureLayer,
                        nameOfOutputNNs,nameOfUniqueIDField)
        #delete those pairs which are not queen contiguous with each other
        where_condition_queen = '"LENGTH" = 0'
        arcpy.AddMessage("where_condition_queen is " + str(where_condition_queen))
        queen_cursor = arcpy.da.UpdateCursor(outputNNsTable,"*",where_condition_queen)
        with queen_cursor as cursor:
                for row in cursor:
                cursor.deleteRow()
        #Join in source cells' and neighbor cells' value and area
        arcpy.JoinField_management(outputNNsTable, newSourceIDField,
intermediateFeatureLayer, nameOfUniqueIDField, toJoinFieldList)
        arcpy.AlterField_management(outputNNsTable, nameOfValueField,
newSourceValueField)
        arcpy.AlterField_management(outputNNsTable, nameOfAreaField,
newSourceAreaField)
        arcpy.JoinField_management(outputNNsTable, newNeighborIDField,
intermediateFeatureLayer, nameOfUniqueIDField, toJoinFieldList)
        arcpy.AlterField_management(outputNNsTable, nameOfValueField,
newNeighborValueField)
        arcpy.AlterField_management(outputNNsTable, nameOfAreaField,
newNeighborAreaField)
        #generate parameters to calculate the differences and measures
```

```
        arcpy.AddField_management(outputNNsTable, "VAL_DIF", "DOUBLE")
        arcpy.AddField_management(outputNNsTable, "AREA_DIF", "DOUBLE")
        arcpy.AddField_management(outputNNsTable, "MEASURE", "DOUBLE")
        valDifExpression = "abs(!src_gridcode! - !nbr_gridcode!) + 1"
        areaDifExpression = "areaDif(!src_area!, !nbr_area!)" #"areaDif(!src_area!, !nbr_area!)"
"!src_area! - !nbr_area!"
        areaDifCodeblock = """def areaDif(areaValSrc, areaValNbr):
                if (areaValSrc - areaValNbr) < 0:
                return .000001
                else:
                return (1 + areaValSrc - areaValNbr)"""
        measureExpression = "(!VAL_DIF!*!VAL_DIF!)/!AREA_DIF!"

        #calculate the differences and measures
        #need to iron out this measure -- also need to revisit polygon neighbors and see if
there's a way to confine only queen contiguous pairs to the list.
        arcpy.CalculateField_management(outputNNsTable,
"VAL_DIF",valDifExpression,"PYTHON")
        arcpy.AddMessage("VAL_DIF calculated")
        arcpy.CalculateField_management(outputNNsTable,
"AREA_DIF",areaDifExpression,"PYTHON",areaDifCodeblock) #,areaDifCodeblock
        arcpy.AddMessage("AREA_DIF calculated")
        #delete those where src's area is smaller than neighbor's area
        where_condition_area = """"src_area" < "nbr_area""""
        arcpy.AddMessage("where_condition_area is " + str(where_condition_area))
        area_cursor = arcpy.da.UpdateCursor(outputNNsTable,"*",where_condition_area)
        with area_cursor as cursor:
                for row in cursor:
                arcpy.AddMessage(u'{0}, {1}, {2}'.format(row[0], row[1], row[2]))
                cursor.deleteRow()

        arcpy.CalculateField_management(outputNNsTable,
"MEASURE",measureExpression,"PYTHON")
        arcpy.AddMessage("MEASURE calculated")

        #find lowest value
        min_measure_value = arcpy.da.SearchCursor(outputNNsTable,
["MEASURE","src_Id","nbr_Id"], "MEASURE IS NOT NULL", sql_clause = (None, "ORDER BY
MEASURE ASC")).next()[0]
        min_measure_src_id = arcpy.da.SearchCursor(outputNNsTable,
["MEASURE","src_Id","nbr_Id"], "MEASURE IS NOT NULL", sql_clause = (None, "ORDER BY
MEASURE ASC")).next()[1]
```

```
        min_measure_src_area = arcpy.da.SearchCursor(outputNNsTable,
["MEASURE","src_area","nbr_area"], "MEASURE IS NOT NULL", sql_clause = (None, "ORDER
BY MEASURE ASC")).next()[1]
        min_measure_nbr_id = arcpy.da.SearchCursor(outputNNsTable,
["MEASURE","src_Id","nbr_Id"], "MEASURE IS NOT NULL", sql_clause = (None, "ORDER BY
MEASURE ASC")).next()[2]
        min_measure_nbr_area = arcpy.da.SearchCursor(outputNNsTable,
["MEASURE","src_area","nbr_area"], "MEASURE IS NOT NULL", sql_clause = (None, "ORDER
BY MEASURE ASC")).next()[2]
        max_measure_value = arcpy.da.SearchCursor(outputNNsTable, "MEASURE",
"MEASURE IS NOT NULL", sql_clause = (None, "ORDER BY MEASURE DESC")).next()[0]
        arcpy.AddMessage("min_measure_value is " + str(min_measure_value))
        arcpy.AddMessage("min_measure_src_id is " + str(min_measure_src_id))
        arcpy.AddMessage("min_measure_src_area is " + str(min_measure_src_area))
        arcpy.AddMessage("min_measure_nbr_id is " + str(min_measure_nbr_id))
        arcpy.AddMessage("min_measure_nbr_area is " + str(min_measure_nbr_area))
        arcpy.AddMessage("max_measure_value is " + str(max_measure_value))


#_____
_____

        ## merge the source feature and neighbor feature with lowest measure score, giving
them all attributes of source feature

        #create where conditions for cursor functions, field object for nonspatial variables of
interest, and cursors
        where_condition_src = str('"' + str(nameOfUniqueIDField) + '" = ' +
str(min_measure_src_id))
        where_condition_nbr = str('"' + str(nameOfUniqueIDField) + '" = ' +
str(min_measure_nbr_id))
        where_condition_both = str('"' + str(nameOfUniqueIDField) + '" = ' +
str(min_measure_src_id) + " or " + '"' + str(nameOfUniqueIDField) + '" = ' +
str(min_measure_nbr_id))
        arcpy.AddMessage("where_condition_src is " + where_condition_src)
        arcpy.AddMessage("where_condition_both is " + where_condition_both)
        fields =
[str(nameOfUniqueIDField),str(nameOfValueField),str(nameOfAreaField),str('SHAPE@')]
        arcpy.AddMessage("fields is " + str(fields))
        src_cursor =
arcpy.da.UpdateCursor(intermediateFeatureLayer,fields,where_condition_src)
        nbr_cursor =
arcpy.da.UpdateCursor(intermediateFeatureLayer,fields,where_condition_nbr)
```

```
        both_cursor =
arcpy.da.UpdateCursor(intermediateFeatureLayer,fields,where_condition_both)
        arcpy.AddMessage("src_cursor is " + str(src_cursor))


#_____________________________________________________________________________
_____________________________________
        #for each cursor, iterate over its rows to accumulate its spatial and nonspatial data to
two list objects
        src_nonspatial_fields = []
        src_spatial_fields = []
        nbr_nonspatial_fields = []
        nbr_spatial_fields = []
        with src_cursor as cursor:
                for row in cursor:
                arcpy.AddMessage(u'{0}, {1}, {2}, {3}'.format(row[0], row[1], row[2], row[3]))
                src_nonspatial_fields.append(row[0])
                src_nonspatial_fields.append(row[1])
                src_nonspatial_fields.append(row[2])
                src_spatial_fields.append(row[3])
                row[1] = row[1]
                #cursor.updateRow(row)
        with nbr_cursor as cursor:
                for row in cursor:
                arcpy.AddMessage(u'{0}, {1}, {2}'.format(row[0], row[1], row[2]))
                nbr_nonspatial_fields.append(row[0])
                nbr_nonspatial_fields.append(row[1])
                nbr_nonspatial_fields.append(row[2])
                nbr_spatial_fields.append(row[3])
                row[1] = row[1]
                #cursor.updateRow(row)

#_____________________________________________________________________________
_____________________________________
        #get both features' spatial field value and combine them into one new geometry
        arcpy.AddMessage("count of features started with: " + str(arcpy.GetCount_management
(intermediateFeatureLayer)))

        poly_geom = src_spatial_fields[0].union(nbr_spatial_fields[0])
        arcpy.AddMessage("poly_geom is " + str(poly_geom))

        src_cursor =
arcpy.da.UpdateCursor(intermediateFeatureLayer,fields,where_condition_src)
```

```
        #update the geometry on the source row and delete the neighboring row
        src_nonspatial_fields = []
        nbr_nonspatial_fields = []
        arcpy.AddMessage("count of features started with: " +
str(arcpy.GetCount_management(intermediateFeatureLayer)))
        with src_cursor as cursor:
                for row in cursor:
                arcpy.AddMessage(u'{0}, {1}, {2}, {3}'.format(row[0], row[1], row[2], row[3]))
                row[3] = poly_geom
                row[2] = min_measure_src_area + min_measure_nbr_area
                arcpy.AddMessage(u'{0}, {1}, {2}, {3}'.format(row[0], row[1], row[2], row[3]))
                cursor.updateRow(row)
        with nbr_cursor as cursor:
                for row in cursor:
                arcpy.AddMessage(u'{0}, {1}, {2}'.format(row[0], row[1], row[2]))
                cursor.deleteRow()
        count_of_features_not_a_number =
arcpy.GetCount_management(intermediateFeatureLayer)
        count_of_features = int(count_of_features_not_a_number.getOutput(0))
        arcpy.AddMessage("count of features returned: " + str(count_of_features))

        nameOfOutputShapefile = intermediateFeatureLayer

        arcpy.AddMessage("end count of features is " + str(count_of_features))

        if count_of_features <= desired_feature_count:
                arcpy.AddMessage("ending because count of features is " +
str(count_of_features) + "which is equal to desired feature count of " +
str(desired_feature_count))
                break
        else:
                arcpy.AddMessage("continuing because count of features is " +
str(count_of_features) + "which is greater than desired feature count of " +
str(desired_feature_count))
                continue

        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")

else:
        # Report error message if Spatial Analyst license is unavailable
        arcpy.AddMessage ("Spatial Analyst license is unavailable")
```