GEOG 8990, Spring 2018, Tobias Fimpel

Summary/Reflection Write-up 4

The objective of developing an Editing Add-In was to understand how ArcGIS Pro’s robust editing tools can be extended with the ArcGIS Pro SDK. I started out by working through two small tutorials and then read more in depth information on associated topics (links listed below). I enjoyed seeing how the Geometry Engine’s Relate method can implement operations using a Dimensionally Extended Nine-Intersection Model. Last semester (CSCI5715) I learned about the Dimensionally Extended Nine-Intersection Model and worked with it using SQL operations. Finding out that I can quite easily implement operations that test for any particular spatial relationship was interesting, and one of the Editing tools I created this week makes use of this.

The Add-In I created is made up of three custom editing tools: a point feature creation tool, a line feature creation tool, and a polygon feature creation tool. I did read up on what the ArcGIS Pro SDK makes available for Annotation feature editing and raster editing (and visualization/processing) but did not come across anything that interested me. I believe Annotation editing and raster processing are areas of the SDK that will be built out more in future releases.

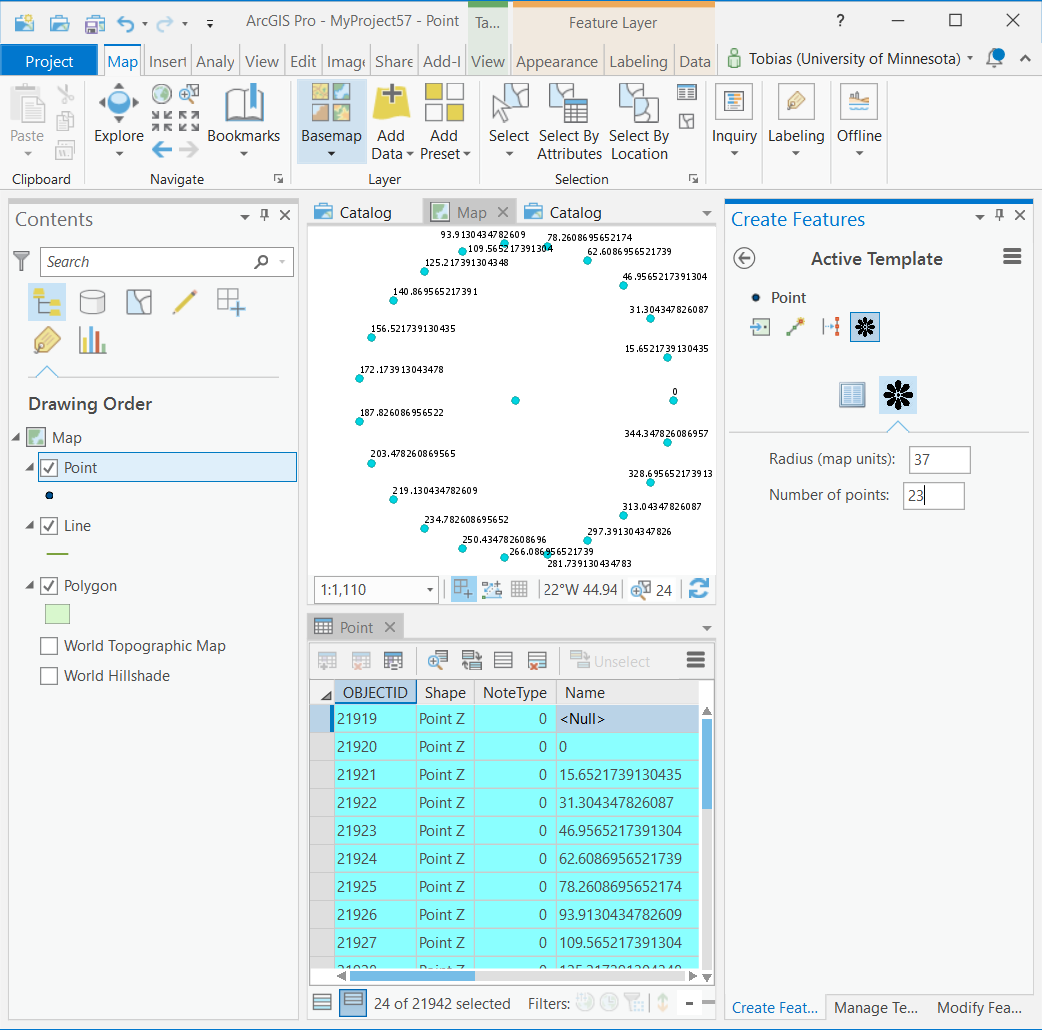
What I did not want to do while developing this Add-In was to get hung up on making something that is necessarily useful for a practical purpose and complete with exception handling and helpful error messages. I enjoyed creating this add-ins because of the interactive and visual nature of how the Editing tools work.

\*Tutorials and References (not an exhaustive list):

* https://developers.arcgis.com/labs/pro/build-a-feature-construction-tool/
* https://github.com/esri/arcgis-pro-sdk/wiki/ProGuide-Construction-Tools-with-Options
* https://github.com/esri/arcgis-pro-sdk/wiki/ProConcepts-Geometry
* https://github.com/esri/arcgis-pro-sdk/wiki/ProGuide-Custom-Relational-Operations
* https://github.com/Esri/arcgis-pro-sdk/wiki/ProGuide-Editing-tool

The following pages show screenshots and key pieces of code. The complete Add-In code is available at <https://github.com/TFimpel/ArcGIS-Pro-March-Sandbox-Editing-AddIn> .

1. Point Feature Creation Tool



The point feature creation tool I built takes two numeric input parameters: “Radius (map units)” and “Number of points”. When the user clicks on the map area to place a new point the tool places additionally x-number of points in a circle around that map point at the specified radius. The attribute field “Name” holds the value of the angle that each point is placed at (starting with an angle of 0 for the point to the right of the center point).

A lot of the feature creation logic is contained in the below block of code:

protected override Task<bool> OnSketchCompleteAsync(Geometry geometry)

{

if (CurrentTemplate == null || geometry == null)

return Task.FromResult(false);

// Create an edit operation

var createOperation = new EditOperation();

createOperation.Name = string.Format("Create {0}", CurrentTemplate.Layer.Name);

createOperation.SelectNewFeatures = true;

// Queue feature creation

createOperation.Create(CurrentTemplate, geometry); //this is the centerpoint

var anglebetweenpoints\_degrees = 360 / CircelNumberOfPoints;

var anglebetweenpoints\_radians = Math.PI \* anglebetweenpoints\_degrees / 180.0;

var radius = Circle;

for (int i = 0; i < CircelNumberOfPoints; i++)

{

var xoffset = radius \* Math.Cos( (i\*anglebetweenpoints\_radians));

var yoffset = radius \* Math.Sin( (i\*anglebetweenpoints\_radians));

Geometry geom = GeometryEngine.Instance.Move(geometry, xoffset, yoffset);

var attributes = new Dictionary<string, object>();

attributes.Add("SHAPE", geom);

attributes.Add("Name", (i \* anglebetweenpoints\_degrees).ToString());

createOperation.Create(CurrentTemplate.Layer, attributes);

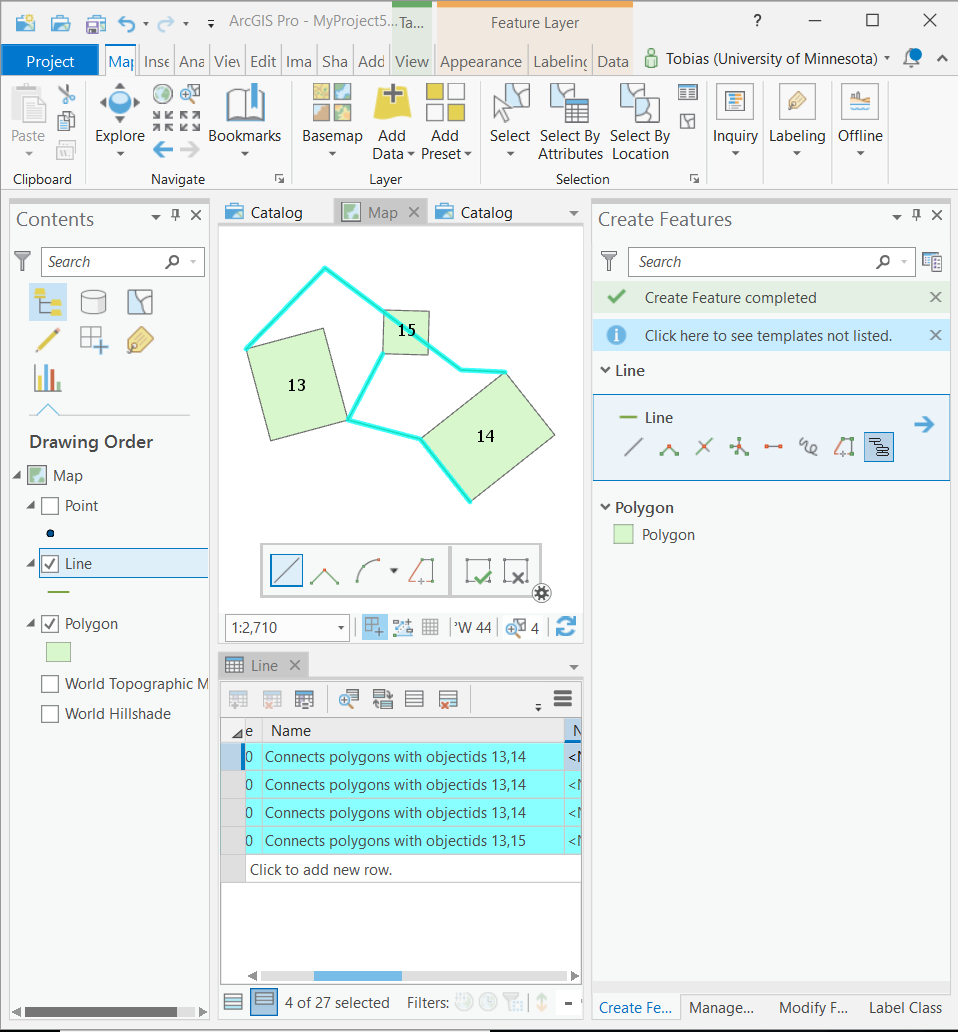
}

// Execute the operation

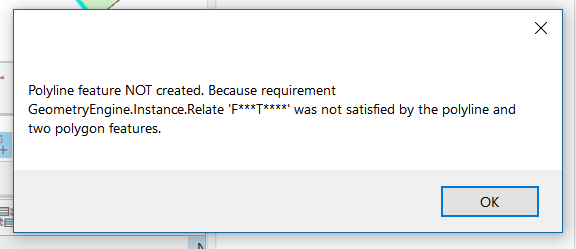
return createOperation.ExecuteAsync();

}

1. Polyline Feature Creation Tool



This polyline feature creation tool tests whether the newly created line feature satisfies a particular spatial relationship (“F\*\*\*T\*\*\*\*”) with two polygon features belonging to polygon layers present in the map. If the relationship is satisfied, the feature creation is carried out and the value of the “Name” attribute field of the newly created line feature describes which two polygon features are connected by the endpoints of the polyline. If the spatial relationship is not satisfied (for example if one of the endpoints of the line does not intersect a polygon boundary) then the below message box is presented to the user and the polyline feature is not created. The below code snipped shows the section of the code that tests for the spatial relationship.



private Task<List<long>> GetRelateObjectIDs(Geometry geometry)

{

return QueuedTask.Run(() =>

{

var polygonLayers = ActiveMapView.Map.GetLayersAsFlattenedList().OfType<FeatureLayer>().Where(lyr => lyr.ShapeType == esriGeometryType.esriGeometryPolygon);

var relateObjectIDList = new List<long>();

foreach (FeatureLayer polygonLayer in polygonLayers)

{

using (RowCursor searchCursor = polygonLayer.Search())

{

while (searchCursor.MoveNext())

{

using (Feature feature = (Feature)searchCursor.Current)

{

//the sketch geometry needs to be projected to the polygon layer spatial reference

var sr = polygonLayer.GetSpatialReference();

Geometry geometry\_prj = (Polyline)GeometryEngine.Instance.Project(geometry, sr);

// Process the feature.

if (GeometryEngine.Instance.Relate(geometry\_prj, feature.GetShape(), "F\*\*\*T\*\*\*\*"))

{

var oid = feature.GetObjectID();

//Debug.WriteLine(feature.GetObjectID().ToString() + "passes test F\*\*\*T\*\*\*\*");

relateObjectIDList.Add(oid);

}

else

{

var oid = feature.GetObjectID();

//Debug.WriteLine(feature.GetObjectID().ToString() + "does NOT pass test F\*\*\*T\*\*\*\*");

}

}

}

//Debug.WriteLine("list of features that pass F\*\*\*T\*\*\*\* test:");

//Debug.WriteLine(string.Join(",", relateObjectIDList.ToArray()));

}

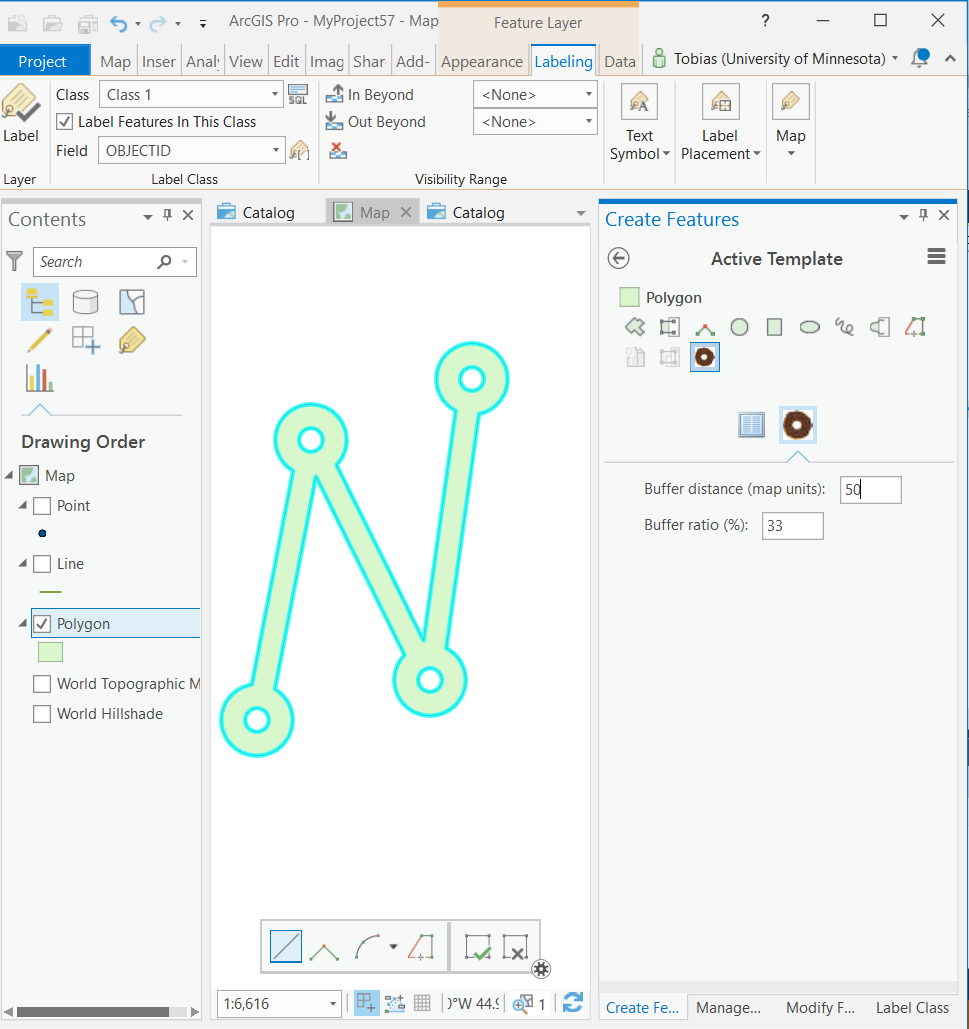
}

return relateObjectIDList; //calling function can test that these are two distinct polygon object ID's

});

}

1. Polygon Feature Creation Tool



The add-in’s polygon feature creation tool takes two numerical input parameters (Buffer distance, Buffer ratio), uses a polyline sketch tool, and does a series of geometry operations to the sketch geometry once it is completed: a larger buffer, a smaller buffer (the difference is controlled by the second input parameter), a union, a symmetrical difference. Below is the code section where these geometry operations are carried out.

private Task<Geometry> ConstructBuffers(Polyline SketchPolyline, Double BufferDistance, Double BufferRatio)

{

return QueuedTask.Run( () =>

{

List<MapPoint> pts = SketchPolyline.Points.ToList();

double brpercent = BufferRatio / 100; //turn input buffer ratio number into percent

//large circle at each vertex

Geometry bufferedGeometry1a = GeometryEngine.Instance.Buffer(pts, BufferDistance);

//smaller circle at each vertex

Geometry bufferedGeometry1b = GeometryEngine.Instance.Buffer(pts, BufferDistance\* brpercent);

//buffered line

Geometry bufferedGeometry2 = GeometryEngine.Instance.Buffer(SketchPolyline, BufferDistance\* brpercent);

//union large circle and buffred line...

Geometry bufferedGeometry1 = GeometryEngine.Instance.Union(bufferedGeometry1a, bufferedGeometry2);

//... and erase small circle from result

Geometry bufferedGeometry = GeometryEngine.Instance.SymmetricDifference(bufferedGeometry1, bufferedGeometry1b);

return bufferedGeometry;

});

}