Samuel Levin
GISC7363
14 December 2018
Dr. Qiu

Final Project: Global Terrorism Events Analysis – Custom JavaScript Analysis Widget

GOALS

This web application expands upon the goals of my midterm project, which developed an analysis dashboard for the Global Terrorism Database (GTD). While ArcGIS Online has many tools pre-configured exploring these data, it does not have a tool available for analyzing the path of event occurrences. In the dataset, each event is recorded with a date, terrorist group name, and related events. These fields store critical information allowing the pathways and trajectories of event distributions to be visualized and analyzed. Thus, I developed a tool to draw the path of attacks by a single terrorist group based on a user defined query and provide summary statistics of these results. This was accomplished using a combination of Python, JavaScript, and HTML. In doing so, the spatiotemporal characteristics of the dataset are emphasized.

DATA SOURCES

This web application provides an interface for exploring, visualizing, and analyzing trends in global terrorism events since 1970. It utilizes a subset of the Global Terrorism Database (GTD). This dataset is curated by the National Consortium for the Study of Terrorism and Response to Terrorism (START), led by the University of Maryland.

Source:
National Consortium for the Study of Terrorism and Responses to Terrorism (START). (2018). Global Terrorism Database [Data file]. Retrieved from https://www.start.umd.edu/gtd

This dataset follows a formalized and standardized methodology for encoding descriptive variables related to each terrorism incident. For each row (event) in the dataset, over 130 unique variables are captured (though some are NULL in the earliest events before START standardized the encoding methodology or when historical incidents were poorly documented). A comprehensive guide to these variables, including description, data collection methods, and encoding strategy is found below.

https://www.start.umd.edu/gtd/downloads/Codebook.pdf

Each data point includes latitude and longitude coordinates, which allowed the original CSV format data file to be converted to a hosted feature layer. A subset of this data was used in the final analysis. The tool's functionality derives from being able to track the attacks committed by known groups their related attacks. Thus, the original CSV data was cleaned to include only the top 10 most frequently occurring terrorist groups. This was accomplished using the Pandas library in Python:

```
import pandas as pd

inCSV = r"C:\Users\~\globalterrorismdb_0718dist.csv"
allEvents = pd.read_csv(inCSV, encoding='ANSI')

gKnownEvents = allEvents[allEvents['gname'] != 'Unknown']

grouped = gKnownEvents.groupby(['gname'])

top10=grouped['eventid'].count().sort_values(ascending=False).head(10)

gname
Taliban                                                 7478
Islamic State of Iraq and the Levant (ISIL)             5613
Shining Path (SL)                                       4555
Farabundo Marti National Liberation Front (FMLN)        3351
Al-Shabaab                                              3288
New People's Army (NPA)                                 2772
Irish Republican Army (IRA)                             2671
Revolutionary Armed Forces of Colombia (FARC)           2487
Boko Haram                                              2418
Kurdistan Workers' Party (PKK)                          2310

top10names = pd.DataFrame(top10).index.values.tolist()

top10groupsEvents = allEvents[allEvents['gname'].isin(top10names)]

top10groupsEvents.to_csv(r"C:\Users\~\globalterrorismdb_0718dist_top10groups.csv")
```

The final dataset contains 36943 unique events from the top 10 most common terrorist groups in the dataset. By trimming the data to include only these groups, the most active terrorist organizations became the focus of the dataset, without the noise from attacks by unknown groups and unrelated events. This CSV was uploaded as a feature layer, with each group assigned a distinct symbol color. This was the only configuration achieved using the ArcGIS online GUI interface. This feature layer acts as the primary operational layer for all analysis and functionality in the final web application.

https://services2.arcgis.com/VNo0ht0YPXJoI4oE/arcgis/rest/services/globalterrorismdb_0718dist_top10groups/FeatureServer

INTENDED AUDIENCE AND USAGE

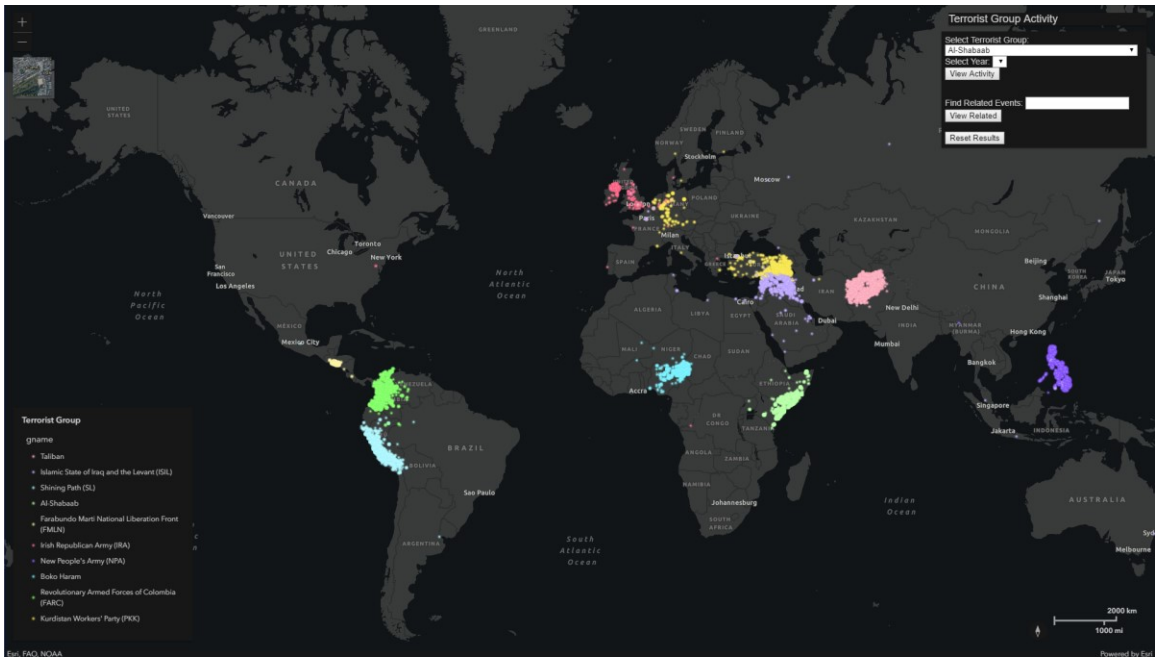
This application was built to suit the needs of experienced researchers, GEOINT professionals, and lay-audiences alike. It is meant to be used in conjunction with the GTD Dashboard developed for my midterm project. Using the dashboard, a particular group, year, or event might be identified where the researcher wishes to visualize the path of event's occurrence. For example, they may be interested to see if an attack in Paris is followed by an attack in Brussels, or whether the next attack is in a totally different geographic area. This kind of path trajectory is not available for point data through the available analysis widgets. Using queries to define the group/year/event of interest, all related events are returned, with a path connecting temporally adjacent attacks. The researcher can use this visualization to assess whether there is a spatio-temporal pattern to the attacks. This type of information could be critical to predicting when and where future attacks might occur and planning counterterrorist efforts accordingly. The interface is

simple, with only three fields to provide the necessary query information. However, the analysis potential is robust, allowing the user to rapidly assess the questions of interest and the dispersion of terrorism events.

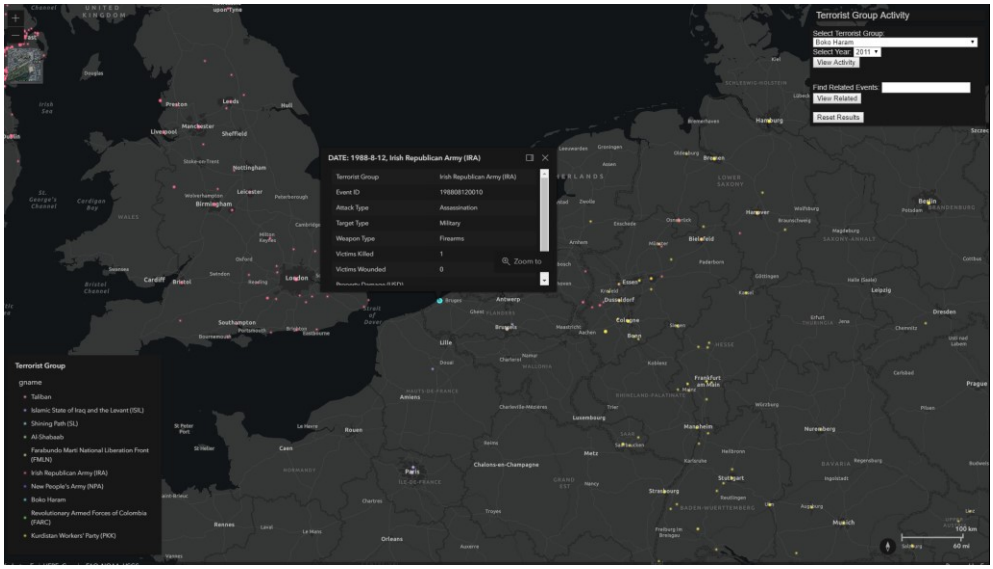
WEB APP FUNCTIONALITY

ALL FUNCTIONALITY CONFIGURED IN JAVASCRIPT AND HTML (+CSS). SEE SOURE CODE (APPENDIX A) WITH COMMENTS FOR DETAILED CODE EXPLORATION

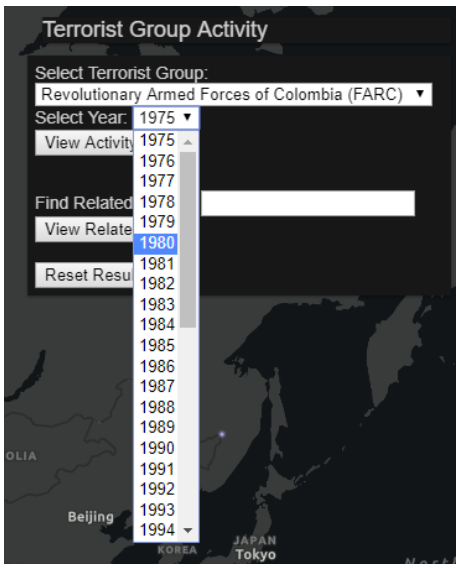
When the web app loads, the user is presented with a global map with one operational layer. This layer shows terrorism events committed by the 10 most active terrorist groups in the dataset. Events committed by each group are symbolized with a unique color, which can be easily referenced in the legend.

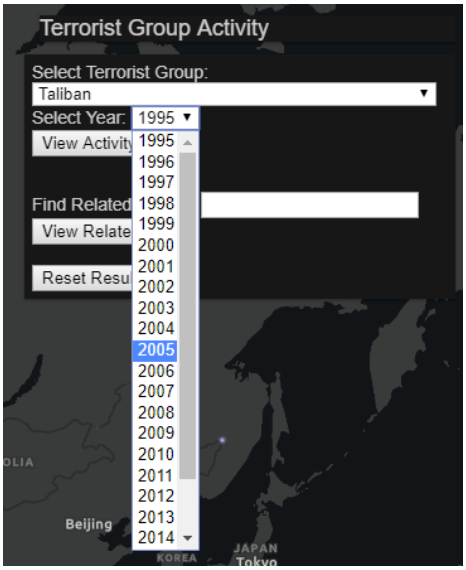


The user is free to explore the map in this state, examining the natural clustering of events by various terrorist groups. A PopUp is configured so that attributes of an event can be examined. In the upper left corner, a basemap toggle widget is included to allow the user to switch to aerial imagery basemap. In the lower left, a scale bar and compass are included for reference and to reset the heading of the map if necessary.

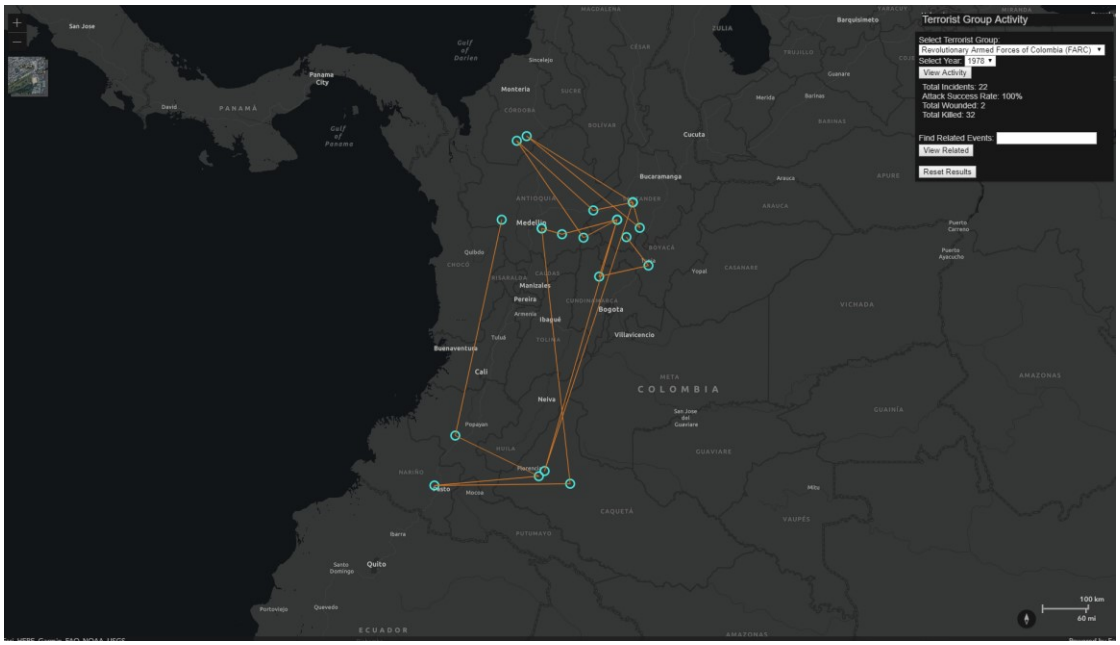


However, the real functionality of the web app is accessed through the fields and buttons in the upper-right corner. These dropdowns and fields allow the user to query the feature layer based on a specific terrorist group and year, or to search for attacks related to a specific eventID. The selection options for these fields are generated dynamically based on data in the feature layer. The "esri/renderers/smartMapping/statistics/uniqueValues" module was used to accomplish this task, providing a more streamlined selection interface. Moreover, this reduces the probability of submitting a query with no results returned, as the dates available for each group are only generated over their active years range.



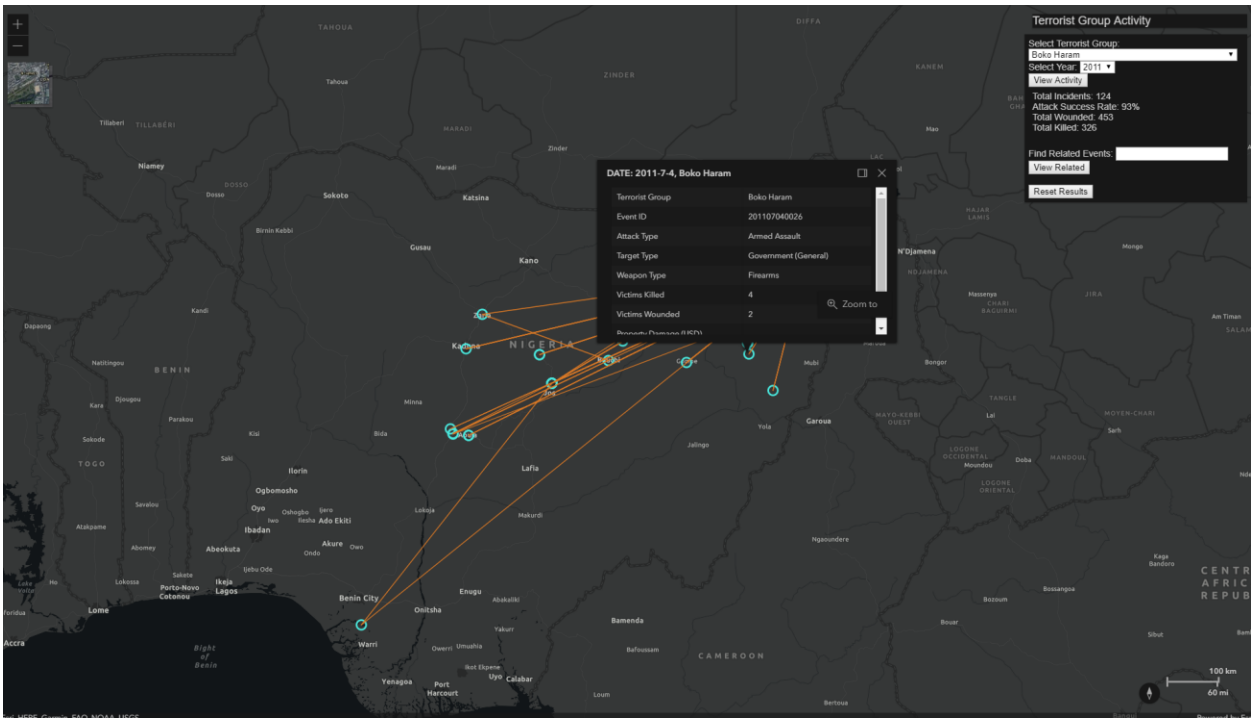


Once a selection is made, the query can be submitted using the “View Activity” button. The original feature layer is hidden from view once the query is submitted, reducing the amount of visual noise on the results display. Additionally, the legend is hidden from view, since the viewer has already specified which group they are interested in. After submitting the query, the map zooms directly to the extent of the returned features. These returned filters are displayed in graphics layers containing points and lines. The points are displayed as hollow cyan circles, with an orange line connecting temporally adjacent events.

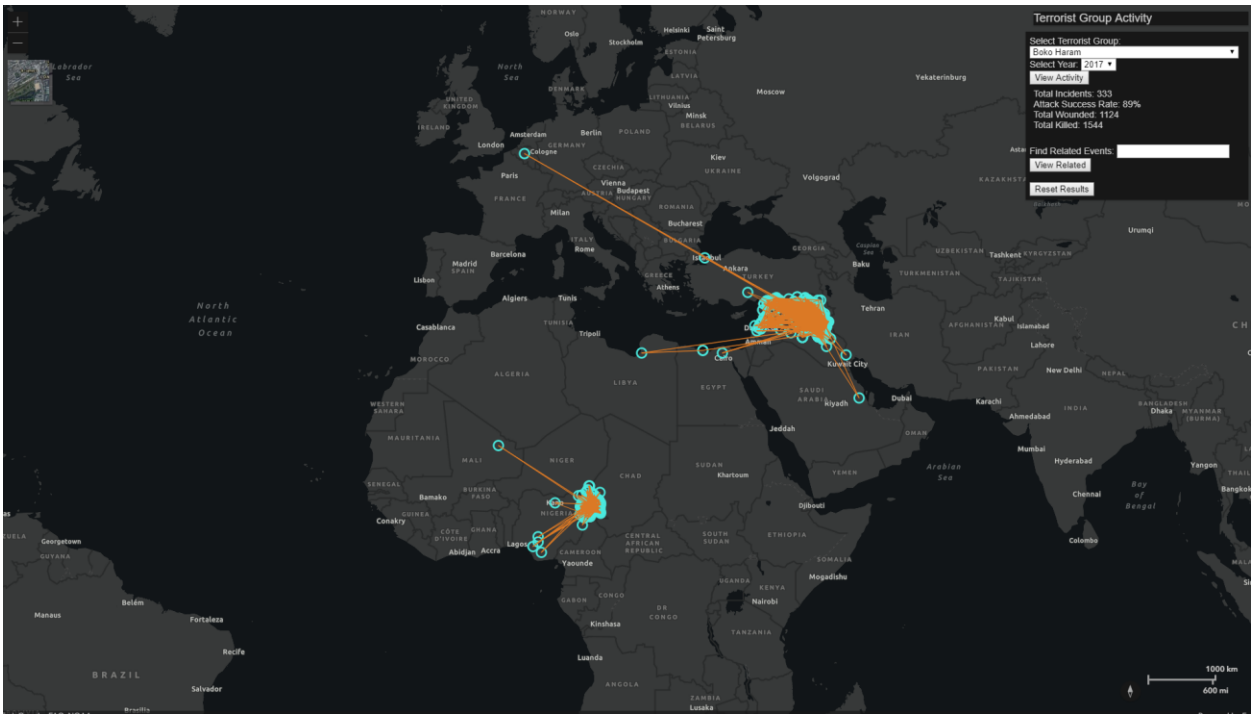


The results layer is returned as a FeatureSet object of point features. These are returned in order of eventID, which is itself temporally chronological. While drawing the points as a graphic is a relatively straightforward task, constructing a graphic line form these points requires some manual manipulation of the returned geometries. Using a for loop, the individual coordinate geometries of each point are appended to a new array. These coordinates define nodes of the final line graphic which constructs the event path between temporally adjacent events.

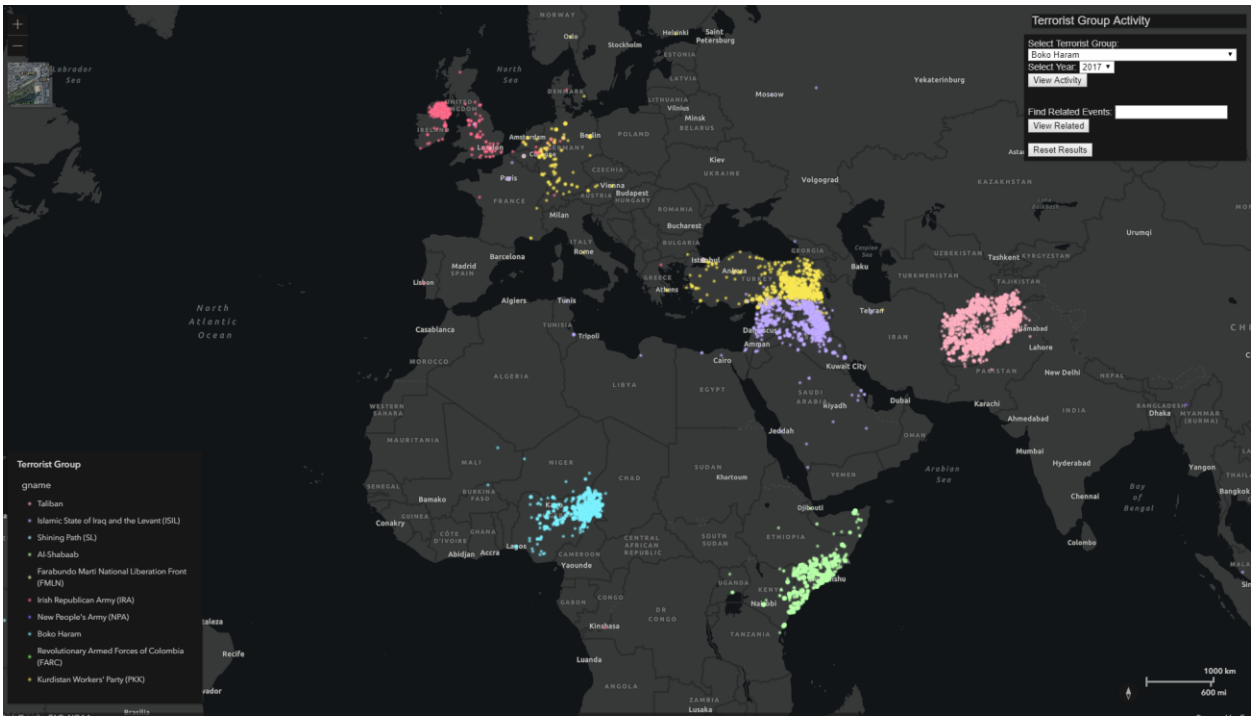
Although the original feature layer is hidden from view, the resulting graphics layer still retains the same PopUp functionality, allowing individual events to be inspected. In the parameter selection box, a collection of basic summary statistics are returned, including: Total Incidents, Attack Success Rate, Total Wounded, Total Killed. Text in this these fields is configured using CSS to match the ArcGIS online styling.



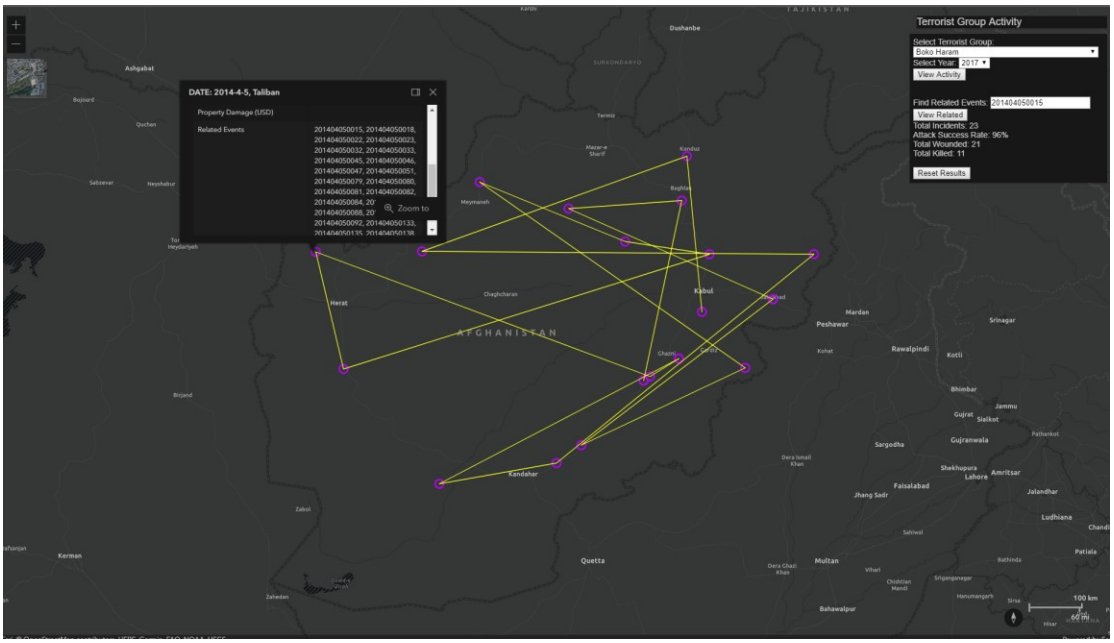
The user is free to submit a second query using the same interface. The view extent will immediately zoom to the new returned features' extent, and the summary statistics will be updated. However, the graphic result of the original query is retained so that the user can visualize two different queries simultaneously.



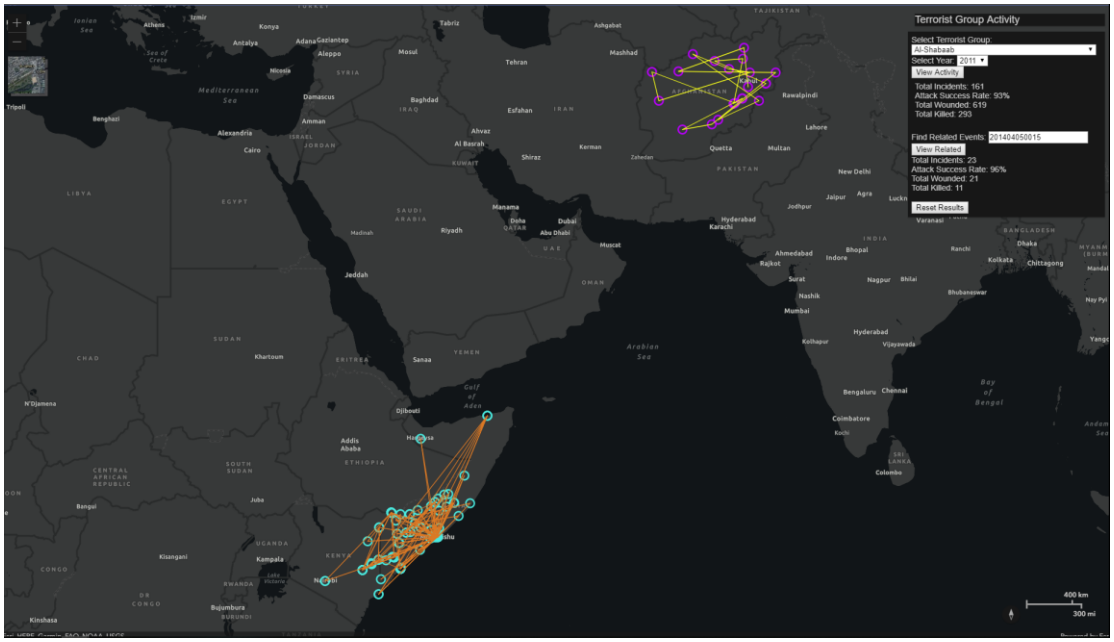
When the user is done with the query, they can click the “Reset Results” button. This button removes all graphics from the display, removes all summary statistics, and restores visibility of the original feature layer, and restores the legend to the viewer.



In addition to querying events by a group name and year, the user can also query by related event IDs. The ‘Related Events’ attribute (visible in feature PopUps) stores a list of any eventIDs that are directly related with the selected event. While not all events have related event IDs, those that do may be queried to identify the events directly associated with it. For example, the event ID 201404050015 is associated with 22 additional terrorism events. These events are likely perpetrated by the same actors or were conducted in coordination. Results from this query are displayed using the same query and graphics display methodology, but assigned unique symbology so that they can be differentiated from the group+year queries. They also retain the PopUp functionality and return their own summary statistics. Again, the user is free to enter a second query, which will update the display extent and summary statistics, but retain the graphics from the previous query. All results can be cleared using the same ‘Reset Results’ button.



The user may also display a group+year based query and eventID query simultaneously. Both graphics will be visible on screen, each with their own summary statistics.



In future improvements, this web app might be improved by refining the symbology for the returned graphics results. Using a gradient for the line hue, the user would be able to better distinguish the intersecting lines in the path and determine the start/end nodes. Transparency of the graphic line might

also improve its readability. Furthermore arrows indicating the direction of event occurrences would improve the output's readability. Similarly these principles might be applied to the point graphics so that events of different types are symbolized differently.

APPENDIX A: SOURCE CODE (also submitted as .html file)

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1,maximum-scale=1,user-scalable=no">
    <title>Samuel Levin, GISC7363 Final</title>
    <!-- BY SAMUEL LEVIN, GISC7363, 14 DECEMBER 2018-->

    <link rel="stylesheet" href="https://js.arcgis.com/4.10/esri/themes/dark/main.css">
    <script src="https://js.arcgis.com/4.9/"></script>

    <style>
    html,
    body,
    #viewDiv {
      height: 100%;
      width: 100%;
      margin: 0;
      padding: 0;
    }

    #selectionDiv {
      background-color: rgb(36,36,36);
      color: rgb(209,209,209);
      padding: 6px;
      width: 330px;
      font-family: sans-serif;
    }

    #resultStatsDiv {
        background-color: rgb(36,36,36);
        color: rgb(209,209,209);
        padding: 6px;
        font-family: sans-serif;
    }

    #titleDiv {
        background-color: rgb(36,36,36);
        color: rgb(209,209,209);
        padding: 6;
        width: 330px;
        font-size: large;
        font-family: sans-serif;
    }
    </style>

    <script>
    require([
        "esri/Map",
        "esri/views/MapView",
        "esri/layers/FeatureLayer",
        "esri/PopupTemplate",
        "esri/widgets/ScaleBar",
        "esri/widgets/Legend",
        "esri/widgets/Compass",
        "esri/tasks/support/Query",
        "esri/Graphic",
        "esri/layers/GraphicsLayer",
```

```
        "esri/renderers/smartMapping/statistics/uniqueValues",
        "esri/widgets/BasemapToggle",
        ],
        function(
          Map, MapView, FeatureLayer, PopupTemplate, ScaleBar, Legend, Compass, Query, Graphic,
GraphicsLayer, uniqueValues, BasemapToggle
        )
        {

        // Create variables from HTML items and variables chosen by user.
        var groupSelect = document.getElementById("group-name");
        var year = document.getElementById("yr");
        var trackActivity = document.getElementById("trackActivity");
        var resetResults = document.getElementById("reset");
        var relatedSelect = document.getElementById("related");
        var trackRelated = document.getElementById("trackRelated");

        // Create map and starting view extent
        var map = new Map({
          basemap: "dark-gray-vector"
        });

        var view = new MapView({
          container: "viewDiv",
          map: map,
          zoom: 2,
          center: [0, 20] // longitude, latitude
        });

        // Add mapDiv and variable selectionDiv to view
        view.ui.add("titleDiv", "top-right");
        view.ui.add("selectionDiv", "top-right");

        // Configure PopUp template for feature layer
        var template = {
            title: "DATE: {iyear}-{imonth}-{iday}, {gname}",
            content: [{
                type: "fields",
                fieldInfos: [{
                    fieldName: "gname",
                    label: "Terrorist Group",
                    visible: true
                }, {
                    fieldName: "eventID",
                    label: "Event ID",
                    visible: true
                }, {
                    fieldName: "attacktype1_txt",
                    label: "Attack Type",
                    visible: true
                }, {
                    fieldName: "targtype1_txt",
                    label: "Target Type",
                    visible: true
                }, {
                    fieldName: "weaptype1_txt",
                    label: "Weapon Type",
                    visible: true
                }, {
                    fieldName: "nkill",
                    label: "Victims Killed",
                    visible: true
                }, {
                    fieldName: "nwound",
                    label: "Victims Wounded",
                    visible: true
                }, {
                    fieldName: "propvalue",
                    label: "Property Damage (USD)",
```

```
                visible: true
            }, {
                fieldName: "related",
                label: "Related Events",
                visible: true
            }]
        }]
    };

    // Reference feature layer and configure with PopUp template
    var featureLayer = new FeatureLayer({
        url:
"https://services2.arcgis.com/VNo0ht0YPXJoI4oE/arcgis/rest/services/globalterrorismdb_0718dist_top10groups
/FeatureServer/0",
        popupTemplate: template
    });
    // Add feature layer to map
    map.add(featureLayer);

    // Add ScaleBar widget
    var scaleBar = new ScaleBar({
        view: view,
        unit: 'dual',
        style: 'line'
    });

    view.ui.add(scaleBar, {
        position: "bottom-right",
        index: 0
    });

    // Add Compass widget
    var compass = new Compass({
        view: view
    });

    view.ui.add(compass, {
        position: "bottom-right",
        index: 1
    });

    // Add Legend widget
    var legend = new Legend({
        view: view,
        layerInfos: [{
            layer: featureLayer,
            title: "Terrorist Group"
        }]
    });

    view.ui.add(legend, {
        position: "bottom-left",
        type: "card",
        layout: "auto"
    });

    // Add BasemapToggle widget
    var basemapToggle = new BasemapToggle({
        view: view,
        nextBasemap: "hybrid"
    });

    view.ui.add(basemapToggle, {
        position: "top-left",
        index: 1
    });

    // Generate unique values found in 'gname' field
    // Add unique terrorist groups to selection dropdown
```

```
uniqueValues({
    layer: featureLayer,
    field: "gname"
}).then(function(response){
    var infos = response.uniqueValueInfos;
    infos.forEach(function(info){
    var option = document.createElement("option");
    option.text = info.value;
    groupSelect.add(option);

    //console.log("GROUP NAME: ", info.value, " # OF ATTACKS: ", info.count);
    });
});

// Based on group selection, populate year dropdown with appropriate event year range
groupSelect.addEventListener("change", function(){
    console.log('listened!');
    document.getElementById('yr').options.length = 0;
    group = groupSelect.value;
    if (group == 'Al-Shabaab'){
        var i = 2007
        while (i <= 2017){
            var yrOption = document.createElement("option");
            yrOption.text = i;
            year.add(yrOption);
            i++;
        }
    } else if (group == 'Boko Haram'){
        var i = 2009
        while (i <= 2017){
            var yrOption = document.createElement("option");
            yrOption.text = i;
            year.add(yrOption);
            i++;
        }
    } else if (group == 'Farabundo Marti National Liberation Front (FMLN)'){
        var i = 1978
        while (i <= 1994){
            var yrOption = document.createElement("option");
            yrOption.text = i;
            year.add(yrOption);
            i++;
        }
    } else if (group == 'Irish Republican Army (IRA)'){
        var i = 1970
        while (i <= 2011){
            var yrOption = document.createElement("option");
            yrOption.text = i;
            year.add(yrOption);
            i++;
        }
    } else if (group == 'Islamic State of Iraq and the Levant (ISIL)'){
        var i = 2013
        while (i <= 2017){
            var yrOption = document.createElement("option");
            yrOption.text = i;
            year.add(yrOption);
            i++;
        }
    } else if (group == "Kurdistan Workers' Party (PKK)"){
        var i = 1984
        while (i <= 2017){
            var yrOption = document.createElement("option");
            yrOption.text = i;
            year.add(yrOption);
            i++;
        }
    } else if (group == "New People's Army (NPA)"){
        var i = 1970
```

```
        while (i <= 2017){
            var yrOption = document.createElement("option");
            yrOption.text = i;
            year.add(yrOption);
            i++;
        }
    } else if (group == 'Revolutionary Armed Forces of Colombia (FARC)'){
        var i = 1975
        while (i <= 2016){
            var yrOption = document.createElement("option");
            yrOption.text = i;
            year.add(yrOption);
            i++;
        }
    } else if (group == 'Shining Path (SL)'){
        var i = 1978
        while (i <= 2017){
            var yrOption = document.createElement("option");
            yrOption.text = i;
            year.add(yrOption);
            i++;
        }
    } else if (group == 'Taliban'){
        var i = 1995
        while (i <= 2017){
            var yrOption = document.createElement("option");
            yrOption.text = i;
            year.add(yrOption);
            i++;
        }
    }
});

// Execute analysis based on a Group+Year query selection
// Passed group name and year arguments
function executeGroupYear(groupName, year){
    // Hide original feature layer and legend
    featureLayer.visible = false;
    view.ui.remove(legend);

    // create SQL query from arguments
    where = "gname LIKE '" + groupName.split(" ")[0] + "%' AND iyear = " + String(parseInt(year));

    // Create query object
    var geomQuery = featureLayer.createQuery();
    geomQuery.where = where;
    console.log(geomQuery.where);
    geomQuery.outFields = ["*"];
    geomQuery.returnGeometry = true;

    // Query feature layer
    featureLayer.queryFeatures(geomQuery)
        .then(function(results){
            // results is a returned FeatureSet object
            //console.log(results);

            // Create array to store event points
            var locationArray = [];

            // Create new graphics layer for output
            var eventPointsGraphicLayer = new GraphicsLayer();
            map.add(eventPointsGraphicLayer);

            // Configure result points graphic and add to graphic layer
            var eventPoints = results.features.map(function(graphic){
                graphic.symbol = {
                    type: "simple-marker",
                        color: [0,0,0,0],
                        size: 11,
```

```
            outline: {
                color: "#25eae0",
                width: 2
            }
        };
        return graphic;
});

eventPointsGraphicLayer.addMany(eventPoints);

// Add each event point to an array (used to construct line graphic)
results.features.forEach(function(item){
    //console.log(item.geometry);
    console.log('geometry!');
    var eventLoc = [item.geometry.longitude, item.geometry.latitude];
    locationArray.push(eventLoc);
});

console.log('array!');
console.log(locationArray);

// Configure path geometry type and symbology
var polyline = {
    type: "polyline",
    paths: locationArray,
};

var polylineSymbol = {
    type: "simple-line",
    color: [226, 119, 40],
    width: 1
};

// Create line graphic object and add to map
var eventLineGraphicLayer = new GraphicsLayer();
map.add(eventLineGraphicLayer);


var eventPathGraphic = new Graphic({
    geometry: polyline,
    symbol: polylineSymbol
});

eventLineGraphicLayer.add(eventPathGraphic);

// Zoom to ouput
view.goTo({
    target: eventPathGraphic
});

// Create new statistics query
var statsQuery = featureLayer.createQuery();
statsQuery.where = where
console.log(statsQuery.where);
statsQuery.outFields = ["*"];
statsQuery.returnGeometry = false;

// Define summary statistics
var countEvent = {
    onStatisticField: "eventid",
    outStatisticFieldName: "Count_eventid",
    statisticType: "count"
};
var sumSuccess = {
    onStatisticField: "success",
    outStatisticFieldName: "Success_sum",
    statisticType: "sum"
};
var sumWound = {
```

```
                        onStatisticField: "nwound",
                        outStatisticFieldName: "Wound_sum",
                        statisticType: "sum"
                    };
                    var sumKill = {
                        onStatisticField: "nkill",
                        outStatisticFieldName: "Kill_sum",
                        statisticType: "sum"
                    };

                    statsQuery.outStatistics = [countEvent, sumSuccess, sumWound, sumKill];

                    // Query statistics and add to variable selection window
                    featureLayer.queryFeatures(statsQuery)
                        .then(function(results){
                            var stats = results.features[0].attributes;

                            var eventCount = stats.Count_eventid;
                            var successRate = ((stats.Success_sum)/(eventCount))*100;
                            var woundSum = stats.Wound_sum;
                            var killSum = stats.Kill_sum;
                            var propSum = stats.Property_sum;

                            document.getElementById("resultStatsDiv").innerHTML = "Total Incidents: " +
eventCount +
                                                                                  "<br>Attack Success
Rate: " + Math.round(successRate) + "%" +
                                                                                  "<br>Total Wounded: " +
woundSum +
                                                                                  "<br>Total Killed: " +
killSum;
                        });

                    // Reset results -- remove all graphics layers, reset layer visibility, restore
legend, remove output statistics
                    resetResults.addEventListener("click", function(){
                        console.log('resetting...');
                        eventPointsGraphicLayer.graphics.removeAll();
                        eventLineGraphicLayer.graphics.removeAll();
                        featureLayer.visible = true;
                        document.getElementById("resultStatsDiv").innerHTML = null;
                        document.getElementById("relatedStatsDiv").innerHTML = null;
                        view.ui.add(legend, {
                            position: "bottom-left",
                            type: "card",
                            layout: "auto"
                        });
                    });
                });
        }

        // Execute analysis based on a related EventID query selection
        // Passed EventID as argument
        function executeRelated(eventId){
            // Hide original feature layer and legend
            featureLayer.visible = false;
            view.ui.remove(legend);

            // Create SQL query from argument
            var where = "related LIKE '%" + String(eventId) + "%'";

            // Create Query objetct
            var relatedQuery = featureLayer.createQuery();
            relatedQuery.where = where;
            console.log(relatedQuery.where);
            relatedQuery.outFields = ["*"];
            relatedQuery.returnGeometry = true;
            console.log('pre related query');
```

```
// Query feature layer
featureLayer.queryFeatures(relatedQuery)
    .then(function(results){

        console.log(results);

        // Create array to store event points
        var relatedLocationArray = [];

        // Create new graphics layer for output
        var relatedPointsGraphicLayer = new GraphicsLayer();
            map.add(relatedPointsGraphicLayer);


        var relatedPoints = results.features.map(function(graphic){
            graphic.symbol = {
                type: "simple-marker",
                    color: [0,0,0,0],
                    size: 11,
                    outline: {
                        color: "#b419fc",
                        width: 2
                    }
            };
            return graphic;
        });

        relatedPointsGraphicLayer.addMany(relatedPoints);

        // Add each event point to an array (used to construct line graphic)
        results.features.forEach(function(item){
                //console.log(item.geometry);
                console.log('geometry!');
                var eventLoc = [item.geometry.longitude, item.geometry.latitude];
                relatedLocationArray.push(eventLoc);
        });

        console.log('array!');
        console.log(relatedLocationArray);

        // Create line graphic object and add to map
        var polyline = {
            type: "polyline",
            paths: relatedLocationArray,
        };

        var polylineSymbol = {
            type: "simple-line",
            color: [255, 255, 26],
            width: 1
        };

        var relatedLineGraphicLayer = new GraphicsLayer();
        map.add(relatedLineGraphicLayer);


        var relatedPathGraphic = new Graphic({
            geometry: polyline,
            symbol: polylineSymbol
        });

        relatedLineGraphicLayer.add(relatedPathGraphic);

        // Zoom to output
        view.goTo({
            target: relatedPathGraphic
        });

        // Create new statistics query
```

```
                    var statsQuery = featureLayer.createQuery();
                    statsQuery.where = where
                    console.log(statsQuery.where);
                    statsQuery.outFields = ["*"];
                    statsQuery.returnGeometry = false;

                    // Define summary statistics
                    var countEvent = {
                        onStatisticField: "eventid",
                        outStatisticFieldName: "Count_eventid",
                        statisticType: "count"
                    };
                    var sumSuccess = {
                        onStatisticField: "success",
                        outStatisticFieldName: "Success_sum",
                        statisticType: "sum"
                    };
                    var sumWound = {
                        onStatisticField: "nwound",
                        outStatisticFieldName: "Wound_sum",
                        statisticType: "sum"
                    };
                    var sumKill = {
                        onStatisticField: "nkill",
                        outStatisticFieldName: "Kill_sum",
                        statisticType: "sum"
                    };

                    statsQuery.outStatistics = [countEvent, sumSuccess, sumWound, sumKill];

                    // Query statistics and add to variable selection window
                    featureLayer.queryFeatures(statsQuery)
                        .then(function(results){
                            var stats = results.features[0].attributes;

                            var eventCount = stats.Count_eventid;
                            var successRate = ((stats.Success_sum)/(eventCount))*100;
                            var woundSum = stats.Wound_sum;
                            var killSum = stats.Kill_sum;
                            var propSum = stats.Property_sum;

                            document.getElementById("relatedStatsDiv").innerHTML = "Total Incidents: " +
eventCount +
                                                                                "<br>Attack Success
Rate: " + Math.round(successRate) + "%" +
                                                                                "<br>Total Wounded: " +
woundSum +
                                                                                "<br>Total Killed: " +
killSum;
                        });

                    // Reset results -- remove all graphics layers, reset layer visibility, restore
legend, remove output statistics
                    resetResults.addEventListener("click", function(){
                        console.log('resetting...');
                        relatedPointsGraphicLayer.graphics.removeAll();
                        relatedLineGraphicLayer.graphics.removeAll();
                        featureLayer.visible = true;
                        document.getElementById("resultStatsDiv").innerHTML = null;
                        document.getElementById("relatedStatsDiv").innerHTML = null;
                        view.ui.add(legend, {
                            position: "bottom-left",
                            type: "card",
                            layout: "auto"
                        });
                    });
                });

            }
```

```
        // execute analysis based on Group Name + Year query
        trackActivity.addEventListener("click", function() {
            console.log(groupSelect.value);
            console.log(year.value);
            executeGroupYear(groupSelect.value, year.value);
        });
        // execute analysis based on Related EventID
        trackRelated.addEventListener("click", function() {
            executeRelated(relatedSelect.value);
        });
    });

    </script>
</head>

<body>
    <div id="viewDiv"></div>
    <div id="titleDiv">Terrorist Group Activity</div>
    <div id="selectionDiv">
        Select Terrorist Group:
        <select id="group-name"></select>
        <br>
        Select Year:
        <select id="yr"></select>
        <br>
        <button id="trackActivity">View Activity</button>
        <br>
        <div id="resultStatsDiv"></div>
        <br>
        Find Related Events:
        <input id="related">
        <button id="trackRelated">View Related</button>
        <br>
        <div id="relatedStatsDiv"></div>
        <br>
        <button id="reset">Reset Results</button>
    </div>
</body>

</html>
```