Chris Mikus

GWU-Leaflet-Challenge-Responses

**Briefly explain the logic for generating the base map**.

The base map was generated by creating the variable “graymap,” and applying a tileLayer() through Leaflet. The tileLayer() function used a Mapbox API to generate the front-end map visible to the user when visiting the html page. The “graymap” variable included the path to the API, defined map attributes (tile size and zoom settings), the type of map to be generated (in this case, the streets-v11 map), and a Mapbox access token in the form of an API key obtained by the developer (this key was defined as a variable prior to creating the variable for “graymap”).

After creating “graymap,” the variable “map” was created in order to connect the base map (generated in the back-end logic.js file) to the web page (generated in the front-end index.html file). Within the variable “map,” the id “mapid” (found in the index.html file) was stated in order to reference the location within the html where this “map” variable would be placed. Finally, the line “graymap.addTo(map)” adds our mapbox-generated map to the “map” variable and loads it in the “mapid” id on the index.html.

**Describe how the JSON was loaded and how was the data traversed. Explain how the information from the JSON was used to render data on the map.**

JSON data was loaded by reading in the .geojson file from the usgs.gov website via the d3.json() function. The d3.json() function followed the path provided in the logic.js file and pulled the data provided on that path in JSON format. Each item in the JSON file contains all the relevant information for one earthquake, including the time of incident, location (latitude, longitude, and depth), and magnitude. Each of these values have a unique key within the JSON file that can be extracted and examined within the logic.js file. The d3.json() function is designed to run as a loop, pulling out the data and style information requested for the current object within the JSON file, styling the data point as requested within the function, adding the information to the base map established earlier (using magnitude data to determine the size and color of the markers and the lat/lng data to determine the geographic location), and then moving to the next JSON object to extract information on the next earthquake.

**Explain the logic for generating the circles and amending the size of them. What does this communicate?**

The circles were generated with the “radius” key in the styleInfo() function while extracting data from the JSON file. The “radius” key implies that the markers to be generated and rendered on the base map will be circles of the specified radius. This particular key calls on another function: getRadius(), defined later in the d3.json() function. getRadius() extracts the value of “feature.properties.mag” from the JSON file, which in this case is a number (of type “float”) that specifies the magnitude of that particular earthquake. Once the number is obtained, the getRadius() function takes that magnitude and multiplies it by 4 to obtain the final value of the marker radius (unless the value of the magnitude is 0, in which case the radius is automatically set to a value of 1). This method more effectively communicates the locations of the more larger earthquakes, since earthquakes of a larger magnitude will have markers with a larger radius (as well as different colors, which is established with the getColor() function).

**Describe how the layer for the Tectonic plates was generated.**

First, a variable named “tectonicplates” was defined immediately after generating the base map as described before. This variable was given a value of L.LayerGroup(), indicating to Leaflet that it needs to create another layer named “tectonicplates,” which is also defined as an overlay with the name “Tectonic Plates.” After developing all of the markers for the earthquake locations and magnitudes, another d3.json() function is run that pulls tectonic plate coordinate data from another source in JSON form. Similar to how the logic.js file read the earthquake data, the logic.js file reads the data for the tectonic plates, draws the lines representing the plate boundaries by using the coordinate data provided, and adds the lines to the basemap (as a layer) through the line “tectonicplates.addTo(map).”

**What are the components in the layer control? How were they generated?**

The components of the layer control are the three base maps (satellite, grayscale, outdoor) and two tile layers (tectonicplate, earthquake). Although all of the maps and layers are created with variables similar to those described previously, the ability for the user to interact with the map and control layer appearance is in the line “L.control.layers(baseMaps, overlays).addTo(map),” which adds the layers to the map generated by Leaflet so that the user can toggle through the options as they desire.

**Explain the difference between the base map (tile layer) and the data layer(s).**

The main difference between the base map (tile layer) and the data layers is that only one base map can be visible at a time, but the data layers can be toggled on and off independently. For instance, the map contains three base layers: satellite, grayscale, and outdoor. If satellite is selected, only that visualization can be viewed in the window; selecting grayscale would remove the satellite map and replace it with the grayscale. The map also contains two tile layers: tectonic plates and earthquake markers. The user can add or remove either from the visualization at any time without needing to regenerate the entire map (the user can choose to see neither layer, only plates, only markers, or both layers).

**Walk through the logic of how the legend was generated and rendered on the page.**

The legend is generated after the earthquake markers have been created. The variable “legend” is defined in order to render on the bottom right corner of the map. When added, the logic.js file creates another “div” on the index.html file in which it will place and populate the legend. A variable “grades” was created to specify the earthquake magnitude, and the variable “colors” was created to match the colors in the legend to those on the map. A for loop is used to match the magnitude, circle size, and circle color before adding the legend to the map with the line “legend.addTo(map).”