Geography 575
Lab #2: Interaction Challenge

Lab Objectives:
- Implement the *retrieve* and *sequence* operators and explore additional operators
- Introduce you to skinning and styling UI components using images
- Introduce you to jQuery and jQueryUI

Evaluation:
This lab is worth **40 points** toward the Lab Assignments evaluation item. A grading rubric is provided at the end of the lab to inform your work.

Schedule of Deliverables:
- **February 21st:** Lab #2 Assigned //client contract begins
- **February 28th:** Work Period //input & feedback from client
- **March 7th:** Work Period //input & feedback from client
- **March 14th:** Lab #2 Due //contract deadline

Challenge Description

A reporter from the BBC stumbled upon your contribution to the Geography 140 course when researching a story related to your mapped topic and requested use of the spatiotemporal visualization as part of the online version of her article; happy to gain the experience (and fill out your résumé), you have agreed to work with her on the project. The reporter believes that your animated map will add an interesting presentation element to her article, but thinks that the addition of more interactivity will improve the user experience by (1) allowing users to customize their experience with the map, (2) increasing the amount of detail provided in the display upon request (i.e., overcoming the cartographic problematic), and (3) empowering users with control over the animation (altogether the *why?* of *Interactive Cartography*). The reporter therefore requests that you: (a) add a dynamic information window for extracting specific information from the map (i.e., the *retrieve* operator), (b) add standard controls for manipulating the animation (i.e., the *sequence* operator), and (c) improve the overall website design to match that of the best interactives published in online newspapers (e.g., NYTimes, WaPo, BBC). Being a novice in Interactive Cartography herself, she also requested that you (d) add one additional interesting interaction operator that you think would improve the user experience with the map.

Editor's Notes from the BBC
You need not worry about specifics of the story with which the map will be paired, but are welcomed to redesign your Lab #1 application around a hypothetical story if you desire (and include it in your website). You must add one new interaction operator, not additional flexibility or freedom for an existing operator (other than that required for the *sequence* operator).

1. The *Retrieve* Operator

a. Implementing the *Retrieve* Operator

The first requirement of the Lab #2 challenge is support of the ***retrieve*** operator, allowing users to request specific values from the proportional symbols. The *retrieve* operator is a primary way in which users interactively overcome the cartographic problematic (in the context of Interactive Cartography) and complete Shneiderman's information seeking mantra of "overview first, zoom and filter, then details-on-demand" (in the context of Geographic Visualization). As stated in class, there are three common solutions for supporting the *retrieve* operator that vary in the amount of details presented to the user on demand: (1) a ***dynamic label***, providing a minimal amount of information when probing a feature, (2) an ***information window***, producing a container atop the map when a feature is clicked, and (3) an ***information panel***, populating an HTML element away from the map with associated multimedia content. For Lab #2, you are required to implement the dynamic label solution, described as a ***popup*** in the Leaflet library. Before implementing the *retrieve* operator, review L.Popup in the Leaflet API Reference for additional information about the properties, methods, and events available for creating dynamic labels in Leaflet.

b. Dynamic Labels (Popups) in Leaflet

Leaflet supports popup menus through ***binding***, or a function that establishes a relationship between JavaScript objects and interactions / interaction parameters such that an update to the object also updates the bound elements. In the case of Leaflet popups, the content of the dynamic label is bound to the Leaflet layer, which again represents a single proportional symbol of type `circleMarker`. Thus, the content of the information changes across proportional symbols using the spatiotemporal information stored in the associated JavaScript object. **Code Bank 1** provides the logic needed to implement the dynamic label using the Leaflet popup; existing code from Lab #1 is shaded in light gray.

Like most elements in a webpage, Leaflet popups contain content that is organized using markup language (HTML). First, create a string variable named `popupHTML` to hold the HTML content used in the popup (**CB1: 12-16**); add this variable within the `onEachFeature()` function in *main.js*, after the logic for rescaling the proportional symbol. The implementation in **Code Bank 1** uses three variables within the HTML string in order to customize the `popupHTML` content to a given proportional symbol: (1) `layer.feature.properties[timestamp]`, which holds the attribute value at the current `timestamp` that is used to scale the proportional symbol (**CB1: 13**), (2) `layer.feature.properties.name`, which holds the geographic name (space) of the proportional symbol (**CB1: 15**), and (3) `timestamp`, which holds the current time value of the animation (**CB1: Lines 16**).

You are encouraged to add additional information into your *csvData.csv* spatiotemporal information set for inclusion in the dynamic label; this may include additional place-based information, short textual descriptions (remember this is a dynamic label, not an information window), thumbnail images, or hyperlinks to external references. Your particular implementation of *retrieve* must support the *identify* objective in space, time, and attribute (i.e., the three operands from the TRIAD framework). Points are reserved for restyling your HTML content to conform to the look and feel of your webpage; improve your knowledge of HTML5 using the following tutorials:

- [Lynda Tutorials](#) (free when logging in as a UW student)
- [Codecademy: Web Track](#) (free when logging in as a UW student)
- [DoIT STS Training](#)
- [Mozilla Developer Network: HTML5 Page](#)
- [w3schools](#)

```
1       function onEachFeature(layer) {
2
3               //calculate the area based on the data for that timestamp
4               var area = layer.feature.properties[timestamp] * scaleFactor;
5
6               //calculate the radius
7               var radius = Math.sqrt(area/Math.PI);
8
9               //set the symbol radius
10              layer.setRadius(radius);
11
12              //create and style the HTML in the information popup
13              var popupHTML = "<b>" + layer.feature.properties[timestamp] +
14                              " units</b><br>" +
15                              "<i> " + layer.feature.properties.name +
16                              "</i> in <i>" + timestamp + "</i>";
17
18              //bind the popup to the feature
19              layer.bindPopup(popupHTML, {
20                      offset: new L.Point(0,-radius)
21              });
22
23              //information popup on hover
24              layer.on({
25                      mouseover: function(){
26                              layer.openPopup();
27                              this.setStyle({radius: radius, color: 'yellow'});
28                      },
29                      mouseout: function(){
30                              layer.closePopup();
31                              this.setStyle({color: 'blue'});
32                      }
33              });
34      }
```

Code Bank 1: Binding Information and Styling to the Information Popup (in: *main.js*).

After formatting the `popupHTML` variable, bind it to dynamic label using the `bindPopup()` function included in the Leaflet reference (**CB1: 18-21**). The `bindPopup()` function takes two parameters: (1) the `popupHTML` variable defining the popup content and (2) the location on the map where the popup should be centered, relative to the center (0,0) of the `circleMarker` layer. The latter parameter should be horizontally centered at `0`, but vertically centered at the `radius` of the proportional symbol such that the window does not partially cover the proportional symbol.

Finally, indicate the user event that activates and deactivates the dynamic label using the `on()` function in Leaflet (**CB1: 23-33**). Because this is a dynamic label, and not an information window or information panel, user events related to probing are used: `mouseover` and `mouseout`. Keep in mind that such events work only for indirect pointing devices (i.e., they do not work on tablets

relying on direct pointing); consider how to apply the dynamic label functionality to a click+hold to support such devices (hint: make use of the JavaScript `timer` object).

The `mouseover` event evokes two behaviors: (1) the dynamic label is activated using Leaflet's `openPopup()` function (the content for which is already bound to the popup; **CB1: 26**) and (2) the styling of the proportional symbol is changed to provide additional visual feedback (highlighting) to the user (**CB 1: 27**). Again, you are encouraged to revise the default styling (a yellow stroke) used for highlighting in the **Code Bank 1** solution. Similarly, the `mouseout` event evokes two behaviors: (1) the dynamic label is deactivated using Leaflet's `closePopup()` function (**CB1: 30**) and (2) the styling of the proportional symbol is reverted back to its default state, again providing visual feedback to the user (**CB 1: 31**).

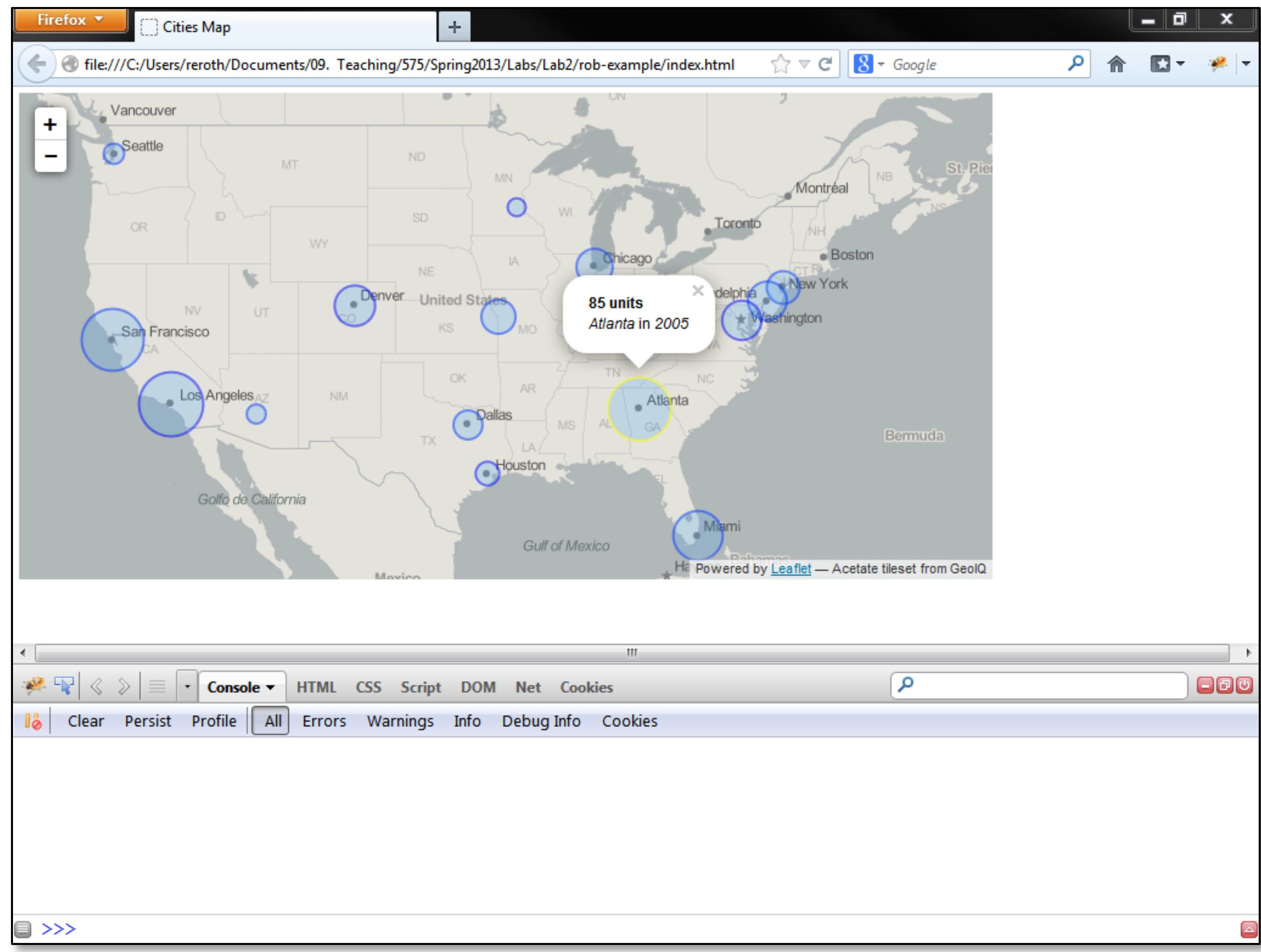

Figure 1: The *Retrieve* Operator with Highlighting.

Open Firefox and load your *index.html* page; you now should have a functional *retrieve* operator with highlighting (**Figure 1**)! For bonus points (2pts per solution), consider how you may be able to implement multiple *retrieve* solutions (i.e., *retrieve* freedom). Increasing *retrieve* freedom includes implementing an expanded, persistent information window on a `click` event (this also makes use of Leaflet popups, but styled differently) or a linked informational panel with additional

spatiotemporal information on a `click` event. Because the dynamic label relies on `mouseover` and `mouseout` events, it should not be difficult to support retrieval of additional information on `click`.

2. *Sequence*: VCR Controls

a. Implementing the *Sequence* Operator with jQuery

The second requirement of the challenge is support of the ***sequence*** operator, defined as cartographic interactions that generate and progress through an ordered set of related cartographic representations. *Sequence* is different from animation (which you implemented in Lab #1), as the former is a user-driven event (thus falling within cartographic interaction) while the latter is a system-driven event (thus falling within cartographic representation). Multiple empirical studies have shown that users grow frustrated of cartographic animations when not provided interactive control for manipulating the animation. To circumvent this problem, and empower the user with control over the animation, you are required to implement seven user interface (UI) controls supporting flexible *sequencing*: (1) play, (2) pause, (3) step, (4) step-full, (5) back, (6) back-full, and (7) a temporal slider, which doubles as a temporal legend. The first six of these features (play, pause, step, back, step-full, and back-full) still commonly are referred to as ***VCR controls***, despite the transition away from the video cassette medium; the temporal slider is treated separately in the next section.

You will use jQuery to coordinate interactions across this array of VCR controls. ***jQuery*** (http://jquery.com) is a JavaScript plug-in that simplifies access to DOM elements for both representation and interaction. Additionally, jQuery handles many of the browser dependency issues that otherwise require custom solutions. Because jQuery is the most widely-used JavaScript library, you are likely to find more interactive mapping implementations on the web using the condensed jQuery syntax than those using pure JavaScript. Before getting started with jQuery, review the jQuery API Documentation as well as the following tutorials:

- Lynda Tutorials (Module #11 on JavaScript Libraries)
- Codecademy: jQueryTrack (free when logging in as a UW student)
- DoIT STS Training (JavaScript II)
- w3schools: jQuery Tutorial

Download the jQuery library at http://jquery.com/download/. Because it is a *.js* file, you will need to right-click on the "Download the compressed, production jQuery 1.9.1" link and save the *jquery-1.9.1.min.js* file to your *js* folder (**Figure 2**). You may download the uncompressed, human readable version to review the set of behaviors provided by jQuery, but it is unlikely that you will extend the jQuery for Lab #2; thus, do not upload the uncompressed version to your public website. Once downloaded, add the script to the `<body>` element of your *index.html* page (**Code Bank 2**); the new `<script>` element should be added after the *leaflet-src.js* script, but before the prototype scripts provided in Lab #1 used to parse *csvData.csv*. You now can make use of jQuery to implement the *sequence* operator!

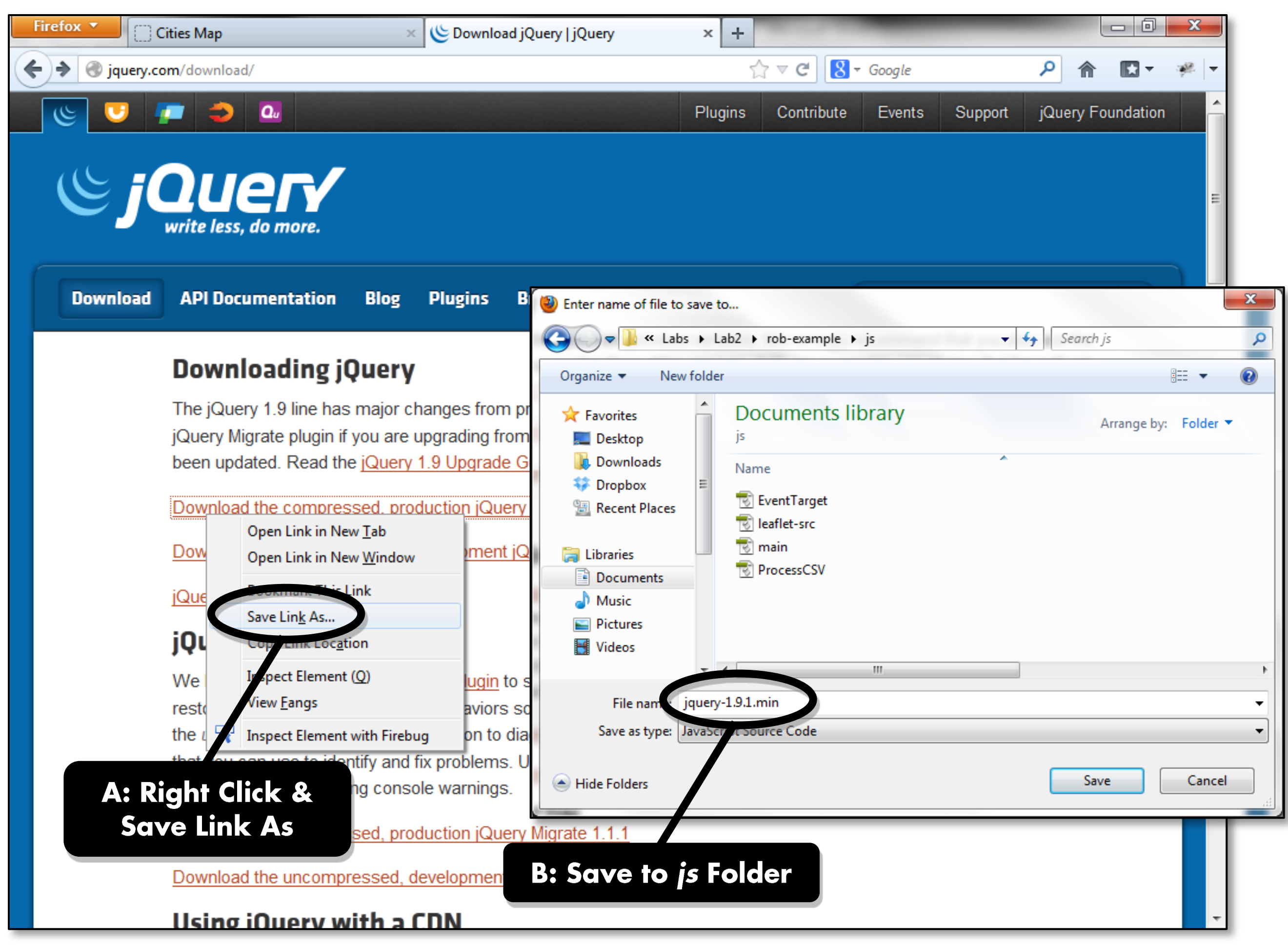

Figure 2: Acquiring jQuery (http://jquery.com/download/).

```
1     <body>
2             <!--div for the map-->
3             <div id="map"></div>
4
5             <!--libraries-->
6             <script src="js/leaflet-src.js"></script>
7             <script src="js/jquery-1.9.1.min.js"></script>
8
9             <!--link to CSV prototype functions-->
10            <script src="js/EventTarget.js"></script>
11            <script src="js/ProcessCSV.js"></script>
12
13            <!--link to main javascript file-->
14            <script src="js/main.js"></script>
15    </body>
```

Code Bank 2: Adding the *jquery-1.9.1.min.js* Script (in: *index.html*).

b. Skinning the VCR Controls

Support of the *sequence* operator begins by designing the graphics constituting the VCR controls. For Lab #2, you will make use of *.png* images for all UI components; Lab #3 describes an alternative use of *.svg* files for UI components. The use of *.png* images for UI components is common across web design contexts, interactive mapping including, with the UI images positioned and styled in the webpage like other HTML elements. A collection of default images for the VCR controls are available at Learn@UW in the *vcr-controls.zip* compressed folder. Download *vcr-controls.zip*, save it to your *img* directory, and extract the files into the *img* folder (*Right-Click -> Extract to here…*).

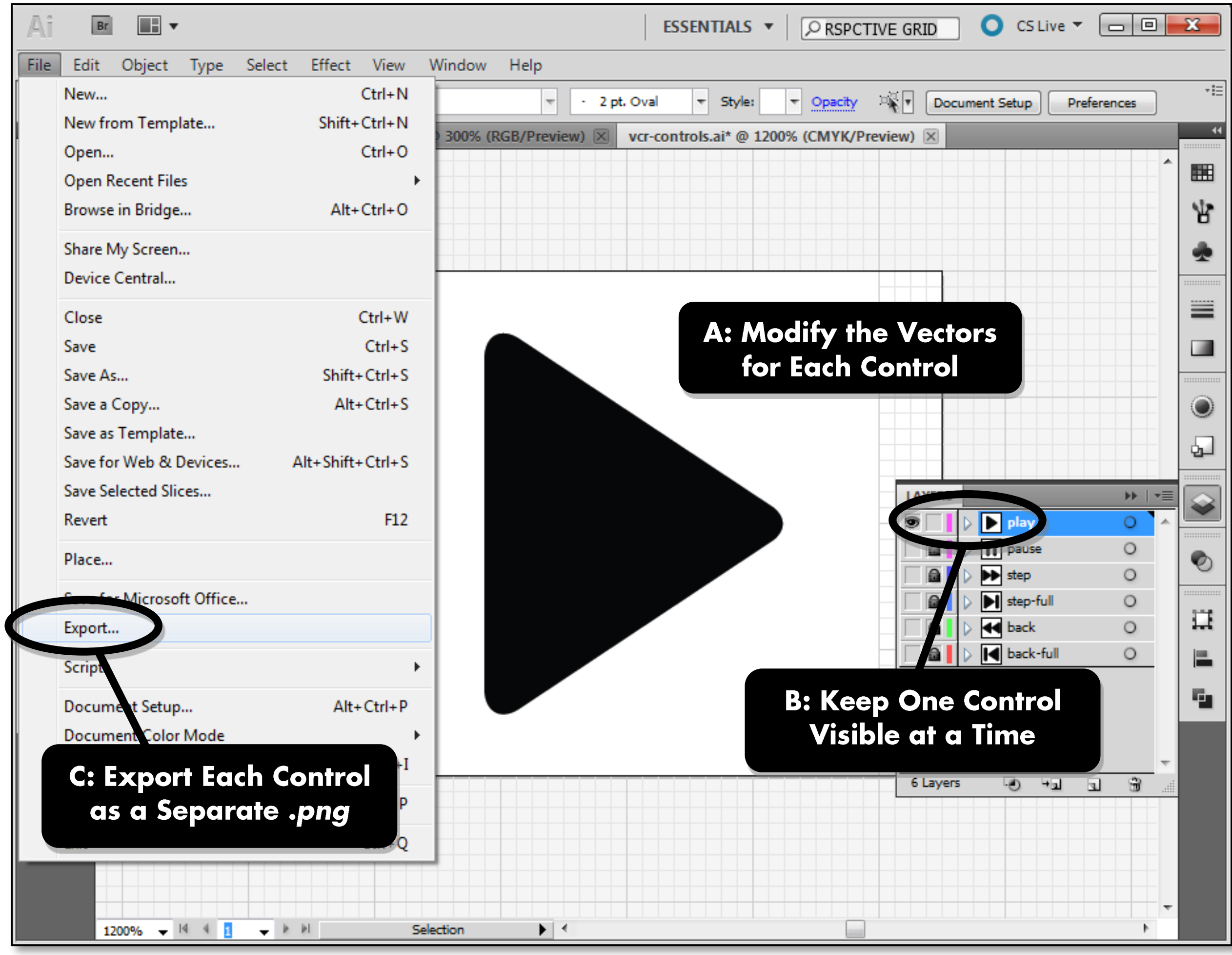

Figure 3: Skinning the VCR Controls.

The *vcr-controls.zip* compressed folder contains seven files: one *.png* image for each of the six VCR controls and an *.ai* file named *vcr-controls.ai* containing the original artwork in vector format. Open the *vcr-controls.ai* file and inspect its contents (Figure 3). The vector graphics associated with each VCR control are organized as separate layers in Illustrator. These layers overlap, as each VCR control was designed using the same bilateral grid (height=32px, width=48px). While the shape of VCR controls is standardized, you are encouraged to experiment with the size and styling of the

controls; adjusting the default size and styling of UI images often is described as ***skinning***. Be sure that any change to a single VCR control also is applied to all others for consistency. Once you update the VCR controls to match the look and feel of your website, re-export them from Illustrator as *.png* files (*File -> Export…*), keeping only one layer visible at a time. The exported *.pngs* should overwrite the existing images in your *img* folder, meaning that you can continue to modify them through development of your website with minimal update to the code.

c. Laying Out and Styling the VCR Controls

Once you have processed the images for your VCR controls, you then can add them to your *index.html* page using the `<img>` element (**Code Bank 4**). It is convention to arrange the VCR controls horizontally in the following order: (1) *back-full.png*, (2) *back.png*, (3) *play.png*, (4) *pause.png*, (5) *step.png*, and (6) *step-full.png*. While you can experiment with the position and spacing of these buttons, be sure that the *play.png* and *pause.png* are placed adjacent to one another; JavaScript is used in the next subsection to make one of these two controls invisible at all times, since play and pause buttons perform inverse functions. Be sure to provide alternative (`alt`) text for each image for screen reader support. The six images are organized within a `<div>` element named `vcr-controls` to provide consistent padding along the outside of the VCR controls (**CB4: 6**).

```
1       <body>
2              <!--div for the map-->
3              <div id="map"></div>
4
5              <!--vcr controls-->
6              <div id="vcr-controls">
7                     <a class="vcr back-full"><img src="img/back-full.png" alt="back
                             full"/></a>
8                     <a class="vcr back"><img src="img/back.png" alt="back"/></a>
9                     <a class="vcr play"><img src="img/play.png" alt="play"/></a>
10                    <a class="vcr pause"><img src="img/pause.png" alt="pause"/></a>
11                    <a class="vcr step"><img src="img/step.png" alt="step"/></a>
12                    <a class="vcr step-full"><img src="img/step-full.png" alt="step
                             full"/></a>
13             </div>
14
15             <!--libraries-->
16             <script src="js/leaflet-src.js"></script>
17             <script src="js/jquery-1.9.1.js"></script>
18
19             <!--link to CSV prototype functions-->
20             <script src="js/EventTarget.js"></script>
21             <script src="js/ProcessCSV.js"></script>
22
23             <!--link to main javascript file-->
24             <script src="js/main.js"></script>
25
26      </body>
```

Code Bank 3: Laying Out the VCR Controls (in: *index.html*).

Each `<img>` element in **Code Bank 3** is wrapped by an `<a>` (anchor) element in order to give each image an `id` in the DOM for JavaScript referencing (`back-full`, `back`, `play`, `pause`, `step`, and `step-full`, respectively) and to assign each image to broader style class called `vcr` for consistent styling in CSS (**Code Bank 4**). Before moving back to *main.as*, return to *style.css* and add three styles to the new created VCR controls: (1) set the `margin` and `padding` for the `vcr-controls` wrapper (**CB4: 1-4**), (2) set the `float`, `margin`, `width`, and `height` of each image instantiating the `vcr` class (**CB4: 5-10**), and (3) set the `opacity` of images instantiating the `vcr` class to `.4` on `hover` to provide a visual affordance to the user that the VCR controls are interactive (**CB4: 11-13**).

```
1	#vcr-controls {
2		margin: 8x 0 0 5px;
3		padding: 3px;
4	}
5	.vcr {
6		float: left;
7		margin-right: 3px;
8		width: 48px;
9		height: 32px;
10	}
11	.vcr:hover {
12		opacity: .4;
13	}
```

Code Bank 4: Styling the VCR Controls (in: *style.css*).

Return to Firefox and refresh the *index.html* page. You now should see your six VCR controls aligned horizontally beneath the map (**Figure 4**). Probing the controls also should result in a style change, making them appear light gray due to the change in transparency. You are encouraged to revise the code in **Code Banks 3** and **4** to place the VCR controls in a position of your liking; use this as an opportunity to improve your knowledge of CSS using the following tutorials:

- Lynda Tutorials
- Codecademy: Web Track (free when logging in as a UW student)
- DoIT STS Training
- Mozilla Developer Network: CSS Page
- w3schools

c. Play and Pause

You now are ready to implement the *sequence* behaviors using jQuery and the processed VCR controls. In Lab #1, you programmed your spatiotemporal visualization to animate when the page is loaded and did not provide functionality to stop the animation. For Lab #2, the animation should only start upon user interaction, with the animation paused upon loading. Before adding behaviors to any of the *sequence* components, remove the function call to `animateMap()` at the very end of the `createMarkers()` function; after doing so, reload *index.html* in Firefox to ensure that the animation does not start automatically upon page load.

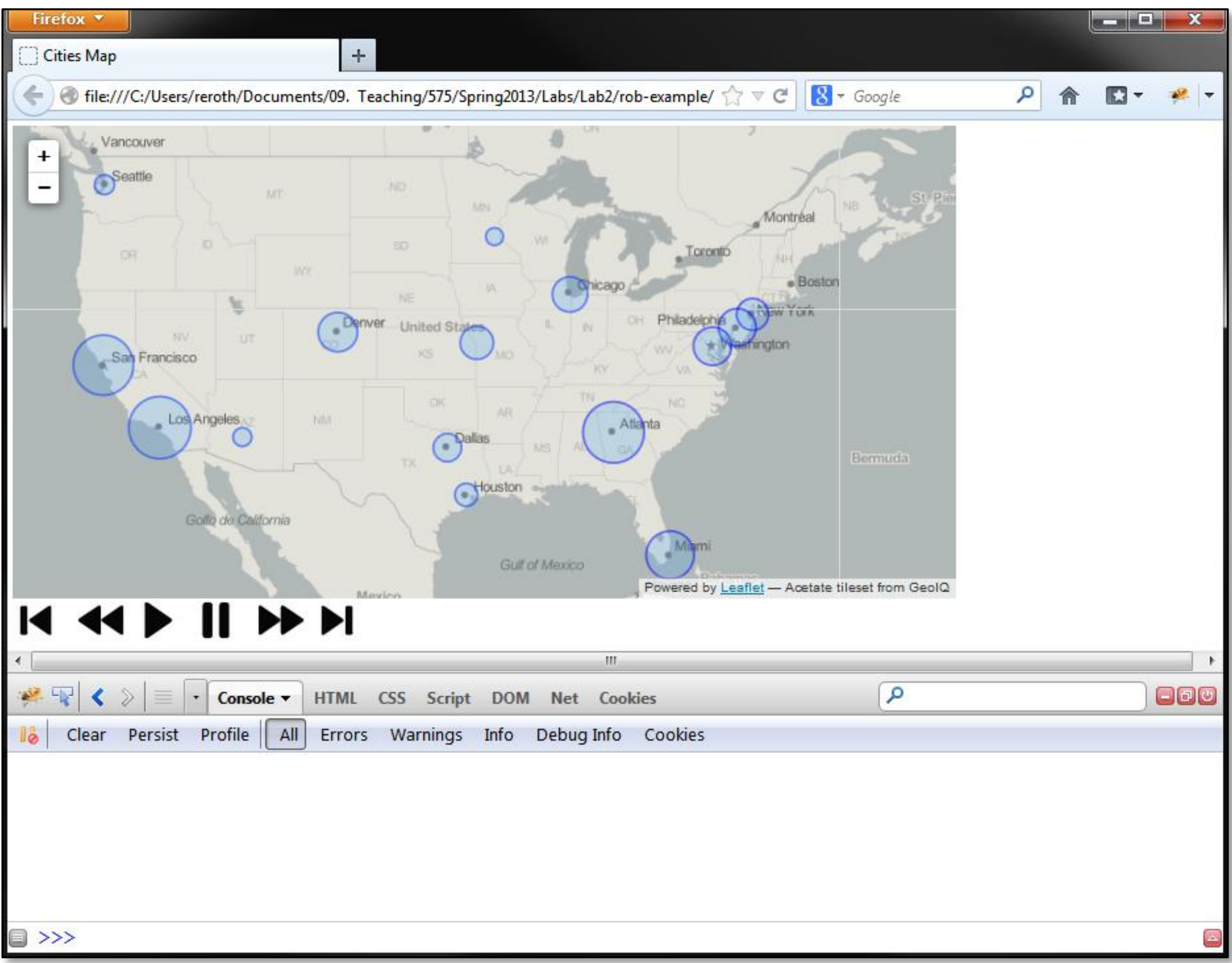

Figure 4: Laying Out and Styling the VCR Controls.

The animation instead will be controlled interactively using the *sequence* operator. Add the appropriate *sequence* behaviors to the `play` and `pause` controls by defining a new function named `sequenceInteractions()` (**Code Bank 5**). The `sequenceInteractions()` function must be called at the end of the `setMap()` function so that the UI events are configured after the window is loaded (not shown in **Code Bank 5**). The `play` and `pause` controls are accessed in the DOM using jQuery syntax; the `$` symbol replaces a call to the lengthier `getElementByID()` function in JavaScript.

Within the `sequenceInteractions()` function, first make the `pause` control invisible using the `hide()` function in jQuery (**CB5: 3**); the `pause` control is hidden—and not the `play` control—because the animation is stopped upon loading. Next, define the behavior for the `play` control (**CB5: 5-10**); when the user clicks the `play` control, the visibility of the `play` and `pause` controls is toggled using the jQuery `show()` and `hide()` functions (**CB5: 7-8**) and the `animateMap()` function is then called (**CB5: 9**). Then, define the behavior for the `pause` control (**CB5: 12-16**), which inversely toggles the visibility of the `play` and `pause` controls (**CB5: 13-14**) and subsequently calls a new `stopMap()` function (**CB5: 15**).

```
1       function sequenceInteractions(){
2
3               $(".pause").hide();
4
5               //play behavior
6               $(".play").click(function(){
7                       $(".pause").show();
8                       $(".play").hide();
9                       animateMap();
10              });
11
12              //pause behavior
13              $(".pause").click(function(){
13                      $(".pause").hide();
14                      $(".play").show();
15                      stopMap();
16              });
17      }
```

Code Bank 5: Adding *Sequence* Behaviors to `play` and `pause` (in: *main.js*).

The final step in implementing the `play` and `pause` controls is definition of the new `stopMap()` function (**Code Bank 6**). The `stopMap()` function is the conceptual opposite of the `animateMap()` function, calling the `clearInterval()` function in JavaScript to stop the `timer` object initializd in `animateMap()`, which itself calls `step()` at a regular interval to play through the animation (return to Lab #1 for details). Once adding `stopMap()` to *main.js*, refresh the *index.html* page in Firefox; you now should see only five controls, with the `pause` control hidden when the page loads. You now have implemented the *sequence* operator!

```
1       function stopMap() {
2               clearInterval(timer);
3       }
```

Code Bank 6: Stopping the Animation (in: *main.js*).

d. Step and Step-Full: *Sequence* Flexibility

After implementing the `play` and `pause` controls, improve *sequence* flexibility by implementing a second pair of controls for progressing the animation forward in time: (1) `step` (advance one timestamp in the animation) and (2) `step-full` (advance to the last timestamp in the animation). As with the `play` and `pause` controls, behaviors for the `step` and `step-full` controls again are defined in the `sequenceInteractions()` function (**Code Bank 7**).

Behavior for the `step` and `step-full` controls is simpler than that for `play` and `pause`, as the `step` and `step-full` buttons remain on the webpage at all times. When a user clicks the step control, the existing `step()` function is called (**CB7: 18-21**). Because `step()` is called directly—rather than from the `animateMap()` function (using the `timer` object)—the code within `step()` will be executed only once, advancing the animation only one timestamp.

```
1       function sequenceInteractions(){
2
3               $(".pause").hide();
4
5               //play behavior
6               $(".play").click(function(){
7                       $(".pause").show();
8                       $(".play").hide();
9                       animateMap();
10              });
11
12              //pause behavior
13              $(".pause").click(function(){
13                      $(".pause").hide();
14                      $(".play").show();
15                      stopMap();
16              });
17
18              //step behavior
19              $(".step").click(function(){
20                      step();
21              });
22
23              //step-full behavior
24              $(".step-full").click(function(){
25                      jump(2011); //update parameter value with last timestamp
26              });
27      }
```

Code Bank 7: Adding *Sequence* Behaviors to `step` and `step-full` (in: *main.js*).

When a user clicks the step-forward control, a new `jump()` function is called, passing the value of the last timestamp as the parameter (**CB7: 23-26**). The `jump()` function, as compared to the `step()` function, is an excellent example of increased ***interface freedom***, or the ability to complete a given interaction with more precision. While `step()` supports the advancement of a single timestamp, `jump()` allows the user to advance forwards or backwards to any timestamp (**Code Bank 8**). The `jump()` function first sets the global `timestamp` variable to the parameter passed in the function call (the final timestamp, in the case of `step-full` behavior). Like `step()`, `jump()` then calls `onEachFeature()` for each `layer` in the in the global `markersLayer`, updating the proportional symbols and bound dynamic labels according to the new value of `timestamp`.

```
1       function jump(t){
2
3               //set the timestamp to the value passed in the parameter
4               timestamp = t;
5
6               //upon changing the timestamp, call onEachFeature to update the display
7               markersLayer.eachLayer(function(layer) {
8                       onEachFeature(layer);
9               });
10      }
```

Code Bank 8: Adding *Sequence* Freedom through `jump()` (in: *main.js*).

e. Back and Back-Full: Transitioning What You Have Learned

The instructions and code banks in Sections #2 provide you with the details needed to implement four VCR controls: `play`,.;/ `pause`, `step`, and `step-full`. Take what you have learned about implementing these UI components to implement the final pair of VCR controls: `back` and `back-full`. The `back` and `back-full` controls are the conceptual opposite of the `step` and `step-full` controls, rewinding the animation rather than advancing it. Think carefully about how to implement `back` and `back-full`:

- How do you reference these images in the DOM using jQuery?
- How do you listen for user events, such as `click`?
- Where should these event listeners be placed in the overall flow of execution?
- What functions should be called from the event handlers?
- How must you modify `step()` to move backwards rather than forwards?
- Can you make use of `jump()` for the `back-full control`?

Solutions for `back` and `back-full` will be discussed during the second work week for Lab #2.

3. *Sequence*: Temporal Slider

a. Implementing a *Sequence* Slider with jQueryUI

The final step in supporting *sequence* flexibility is implementation of a temporal slider. A ***slider*** is a UI widget that allows users to set the value of an ordinal or, more commonly, numerical variable; ***checkboxes*** (allowing compound selection of multiple values) or ***radio buttons*** (constraining selection to a single value in a set) are used for categorical variables. A ***temporal slider*** thus allows the user to set the current timestamp with complete freedom, updating to map to any desired representation in the *sequence*; a slider widget works best for depictions of linear time rather than cyclical time, following a timeline metaphor rather than a clock metaphor. Slider widgets also are common for the *filter* operator when applied to a numerical variable, although they typically include a pair of slider controls (i.e., ***thumbs***) for manipulating the minimum and maximum parameters of the *filter* operation. The temporal slider for *sequence* makes use of a single thumb.

You will be using jQueryUI to implement the temporal slider. ***jQueryUI*** (http://jqueryui.com/) is a plugin library for jQuery that supports a range of common, indirect UI widgets. The jQueryUI plugin includes both default graphics needed for the interface widgets as well as associated events and effects for implementing these widgets. Before getting started with jQueryUI, review the jQueryUI API Documentation and the jQueryUI Demos pages; example implementations of and associated source code for the jQueryUI Slider widget are provided at http://jqueryui.com/slider/.

Download the jQueryUI plugin at http://jqueryui.com/download/. After navigating to the download page, choose the "1.10.1" option under "Version" (**Figure 5a**) and check the "Toggle All" option under "Components" (**Figure 5b**). Unlike jQuery itself—which is extremely pervasive across interactive web design—the jQueryUI plugin may not be advantageous in all interaction design situations, as it is both quite large (it will increase the load time of your webpage) and less stable than jQuery. However, jQueryUI does allow for "custom" downloads that include only portions of

the overall library with significantly reduced file sizes. For now, download the complete jQueryUI library in order to explore its contents; consider replacing the complete download with a reduced download before posting your spatiotemporal visualization to the web in order to minimize the file size.

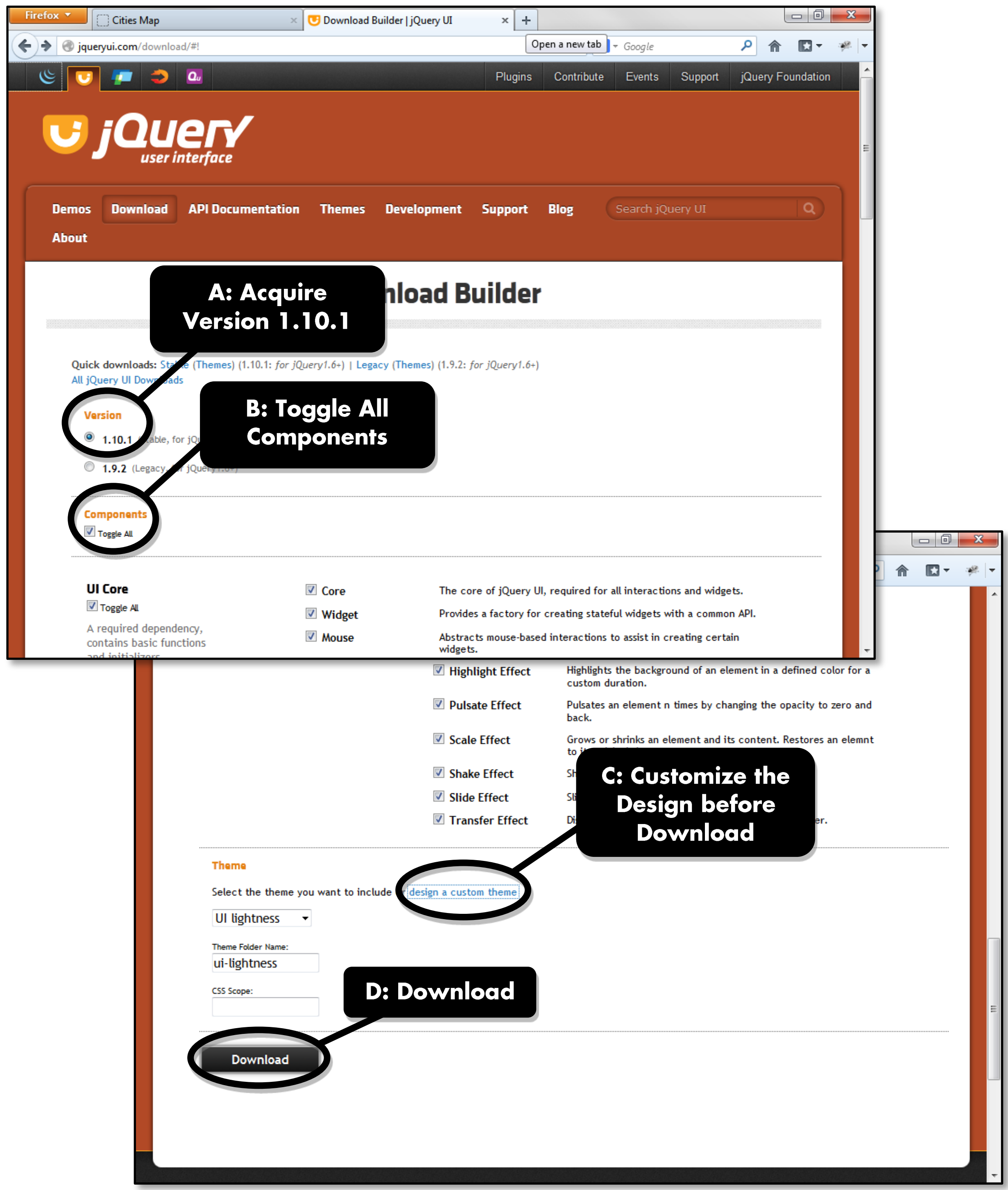

Figure 5: Acquiring jQueryUI (http://jqueryui.com/download/).

Before downloading the complete jQueryUI plugin, use the jQueryUI ThemeRoller to create a look and feel for the UI widgets that match your overall interface design (**Figure 5c**). Although the button icons and several other components of the jQueryUI make use of flat image files, much of the styling of the UI widgets is controlled using CSS. The jQueryUI ThemeRoller allows you to adjust the default CSS rules used to style the UI widgets, ultimately allowing you to blend custom UI interface widgets (i.e., the VCR controls) with jQueryUI widgets. The resulting style rules are included in a stylesheet called *jquery-ui-1.10.1.custom.css* in the overall jQueryUI download. While you can continue to adjust the CSS rules within the *jquery-ui-1.10.1.custom.css* file, it is helpful to work with the jQueryUI ThemeRoller as much as possible prior to download to ensure design consistency across widgets. You also are encouraged to explore the jQueryUI ThemeRoller to continue to improve your understanding of CSS style rules and their impact on interface designs.

After you have customized the styling, download the jQueryUI plugin (**Figure 5d**); the plugin is provided as a compressed library (*.zip*) named *jquery-ui-1.10.1.custom.zip*. It is conventional to store complex libraries like the jQueryUI plugin in a separate project folder named *lib*, as reorganization of the library into the existing *css*, *data*, *img*, and *js* folders often breaks local links within the library; such an approach also helps separate your own, custom code from the code libraries drawn from other sources. Return to the root project directory and add a fifth folder named *lib*. Then, copy the *jquery-ui-1.10.1.custom.zip* file into the *lib* folder and extract its contents into the *lib* folder (*Right-Click -> Extract to here…*).

```
1       <head>
2               <meta charset="utf-8">
3               <title>Cities Map</title>
4
5               <!--main stylesheet-->
6               <link rel="stylesheet" href="css/style.css" />
7
8               <!--leaflet stylesheet-->
9               <link rel="stylesheet" href="css/leaflet.css" />
10              <!--[if lte IE 8]>
11                      <link rel="stylesheet" href="css/leaflet.ie.css" />
12              <![endif]-->
13
14               <!--jQueryUI stylesheet-->
15              <link rel="stylesheet" href="lib/jquery-ui-1.10.1.custom/css/custom-
                        theme/jquery-ui-1.10.1.custom.css" />
16      </head>
```

Code Bank 9: Adding the *jquery-ui-1.10.1.custom.css* Stylesheet (in: *index.html*).

The final step in installing the jQueryUI plugin is to reference the library in your *index.html* page. First, add a `<link>` element to reference the *jquery-ui-1.10.1.custom.css* stylesheet you created through the jQueryUI ThemeRoller (**CB 9: 14-15**); the `<link>` element must be added to the `<head>` element after other stylesheets are referenced. Next, add a `<script>` element to reference the *jquery-ui-1.10.1.custom.min.js* script containing the logic for manipulating and styling the jQueryUI widgets (**CB 10: 21**); the `<script>` element must be added to the `<body>` element after the *jquery-1.9.1.min.js* script is referenced, as it relies on this file.

Note how the paths for the jQueryUI file references are much longer than those for the other stylesheets and scripts added to the *index.html* file. This is because of the decision to maintain the jQueryUI plugin in the *lib* folder, rather than reorganize it according to the pre-existing directory structure. You are encouraged to explore the contents of the jQueryUI folder as a way of understanding how the assorted files are related to one another. You now can make use of jQueryUI to implement a timeline slider, as well as other operators!

```
1       <body>
2               <!--div for the map-->
3               <div id="map"></div>
4
5               <!--vcr controls-->
6               <div id="vcr-controls">
7                       <a class="vcr back-full"><img src="img/back-full.png" alt="back
                                full"/></a>
8                       <a class="vcr back"><img src="img/back.png" alt="back"/></a>
9                       <a class="vcr play"><img src="img/play.png" alt="play"/></a>
10                      <a class="vcr pause"><img src="img/pause.png" alt="pause"/></a>
11                      <a class="vcr step"><img src="img/step.png" alt="step"/></a>
12                      <a class="vcr step-full"><img src="img/step-full.png" alt="step
                                full"/></a>
13              </div>
14
15              <!--div for the timestamp slider-->
16              <div id="temporalSlider"></div>
17
18              <!--libraries-->
19              <script src="js/leaflet-src.js"></script>
20              <script src="js/jquery-1.9.1.min.js"></script>
21              <script src="lib/jquery-ui-1.10.1.custom/js/
                        jquery-ui-1.10.1.custom.min.js"></script>
22
23              <!--link to CSV prototype functions-->
24              <script src="js/EventTarget.js"></script>
25              <script src="js/ProcessCSV.js"></script>
26
27              <!--link to main javascript file-->
28              <script src="js/main.js"></script>
29      </body>
```

Code Bank 10: Adding the *jquery-ui-1.10.1.custom.js* Script (in: *index.html*).

b. Manipulating and Updating the Temporal Slider

To implement the temporal slider, first add a `<div>` element to your *index.html* page to hold the jQueryUI Slider widget (**CB 10: 15-16**); the `id` attribute of the `<div>` element must be `temporalSlider` for the remaining code banks to work properly. Note that in **Code Bank 10**, the `temporalSlider` is placed after the `vcr-controls`. This placement is arbitrary; you are encouraged to experiment with the layout of these *sequence* controls.

Next, add rules for laying out the `temporalSlider` in *style.css*. As stated above, the *jquery-ui-1.10.1.custom.css* stylesheet included in the jQueryUI download contains many of the rules for styling the temporalSlider, including visual affordances and feedback. However, you must add rules

for placing the `temporalSlider` within your webpage (**Code Bank 11**). This includes setting the `margin` around the `temporalSlider` (**CB 11: 2**), which impacts how it will align with the `vcr-controls` `<div>`, as well as setting the `width` (i.e., length) of the `temporalSlider` (**CB 11: 3**). Again, experiment with the placement of the `temporalSlider` to get a layout of your liking. Once you have laid out the `temporalSlider`, return to Firefox and refresh your webpage; you now should have a jQueryUI Slider widget placed on the page, although it will not yet update the map in any way (**Figure 6**)!

```
1       #temporalSlider {
2               margin: 10px 20px 0 270px;
3               width: 500px;
4       }
```

Code Bank 11: Laying Out the `temporalSlider` (in *style.css*)

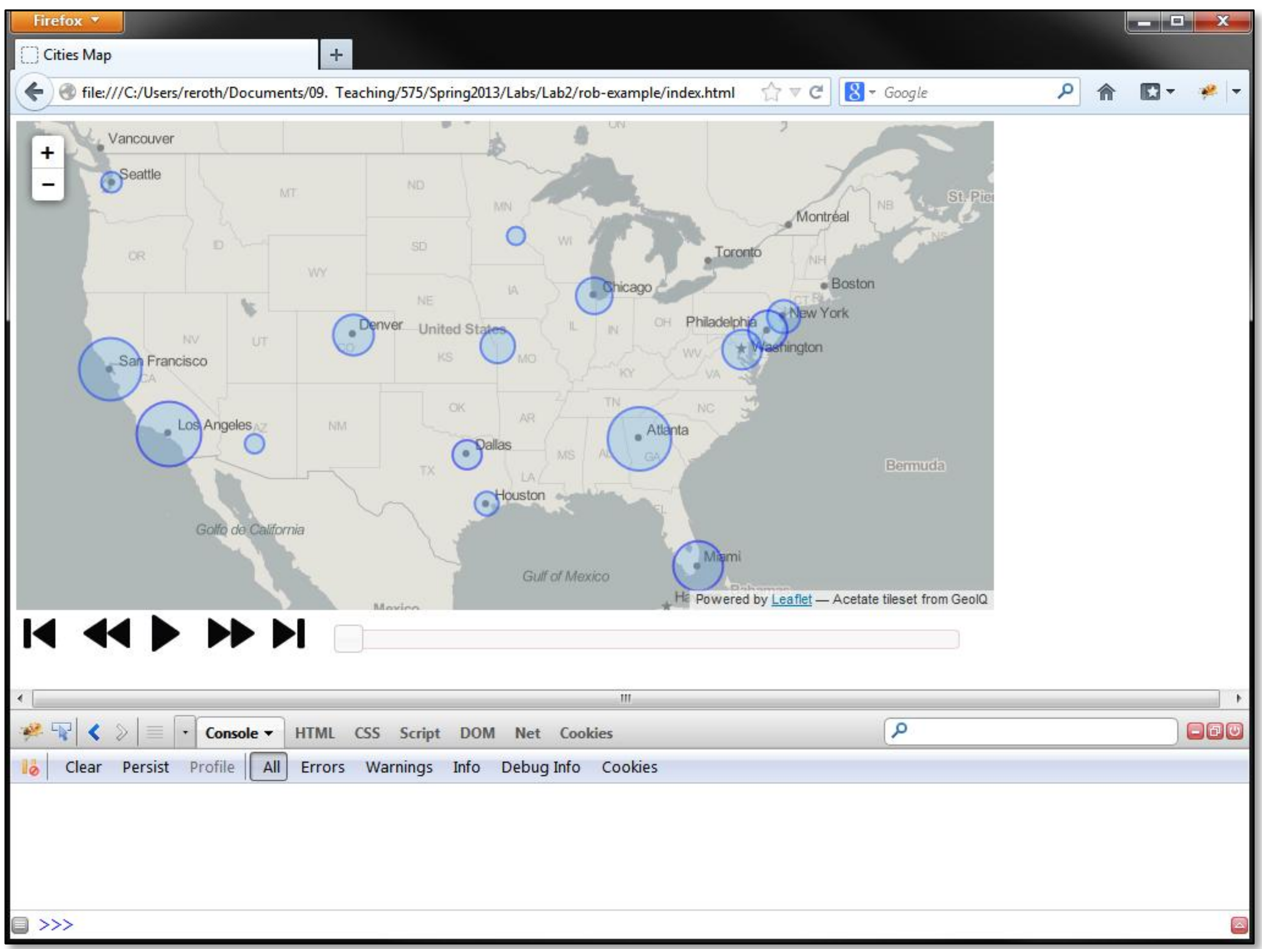

Figure 6: Adding a Slider widget for Free and Flexible *Sequence*

```
1      function sequenceInteractions(){
2
3             $(".pause").hide();
4
5             //play behavior
6             $(".play").click(function(){
7                    $(".pause").show();
8                    $(".play").hide();
9                    animateMap();
10            });
11
12            //pause behavior
13            $(".pause").click(function(){
14                   $(".pause").hide();
15                   $(".play").show();
16                   stopMap();
17            });
18
19            //step behavior
20            $(".step").click(function(){
21                   step();
22            });
23
24            //step-full behavior
25            $(".step-full").click(function(){
26                   jump(2011); //update with last timestamp
27            });
28
29            //back behavior
30            $(".back").click(function(){
31                   back();
32            });
33
34            //back-full behavior
35            $(".back-full").click(function(){
36                   jump(2005); //update with first timestamp
37            });
38
39            $("#temporalSlider").slider({
40                   min: 2005,
41                   max: 2011,
42                   step: 1,
43                   animate: "fast",
44                   slide: function(e, ui){
45                          stopMap();
46                          timestamp = ui.value;
47                          markersLayer.eachLayer(function(layer) {
48                                 onEachFeature(layer);
49                          })
50                   }
51            });
52     }
```

Code Bank 12: Adding *Sequence* Behaviors to `temporalSlider` (in: *main.js*). Note that a solution for `back` and `back-full` has been added.

After placing the `temporalSlider` in your webpage, return to *main.js* to implement the free *sequence* behavior that occurs when the user interacts with the slider. Like the other *sequence*

controls, this behavior is defined within the `sequenceInteractions()` function (**Code Bank 12**); again, this function is called upon initialization of the webpage from the `setMap()` function. The Slider widget is instantiated by calling the jQueryUI `slider()` prototype function on the `temporalSlider` `<div>`; note the continued use of jQuery to reference the `<div>`.

The `slider()` function takes several parameters: (1) `min`, or the first timestamp in the sequence (**CB 12: 40**), (2) `max`, or the last timestamp in the sequence (**CB 12: 41**), (3) `step`, or the interaction freedom provided for moving between timestamps (**CB 12: 42**), (4) `animate`, or the effect given to the slider when the thumb position is updated (**CB 12: 43**), and (5) `slide`, providing a function definition for the behavior that occurs upon a `slide` event (**CB 12: 44-50**). Return to the [jQueryUI Slider API Documentation]() for additional details about the Slider widget.

The `slide` event handler performs three actions. First, the `stopMap()` function is called to stop the animation, if currently playing (**CB 12: 45**). Because the `temporalSlider` provides free *sequence* interaction, it is likely that users will interact with it to set a particular timestamp, rather than rewind or fast-forward the animation (but keep the animation playing). Second, the `timestamp` variable is updated according to the `value` attribute of the `temporalSlider` (**CB 12: 46**). If you properly set the `min` and `max` properties of the `temporalSlider`, you should not need to do any additional math to calculate the `timestamp` variable. Finally, the Leaflet `eachLayer()` function is called on the `markersLayer` in order to call the `onEachFeature()` function, which again updates the radius and popup content for each proportional symbol (**CB 12: 47-49**).

Return to Firefox and refresh you webpage. You now have a functional temporal slider that *sequences* through your spatiotemporal information! However, your work is not yet complete. When implementing flexible interaction, it is important that interaction with one implementation of an operator updates the parameters and visual affordances of the other implementations. Currently, interaction with any of the VCR controls does not update the thumb position of the `temporalSlider`, producing an error related to the gulf of evaluation in which the user may think their interaction did not work because the `temporalSlider` thumb was not advanced.

Add a function named `updateSlider()` at the bottom of the *main.js* file to update the `temporalSlider` when the `timestamp` variable is changed using a different implementation of *sequence*; this function <u>should not</u> be added within the `sequenceInteractions()` function (**Code Bank 13**). The `updateSlider()` function changes the `value` property of the `temporalSlider` (**CB 13: 5**). As a result, the `temporalSlider` thumb will move to the appropriate position whenever the `updateSlider()` function is called (using the `fast` transition effect, as set in **Code Bank 12**). Finally, add a call to the `updateSlider()` function within three existing functions: (1) `step()`, (2) `back()`, and (3) `jump()`. The temporal slider now will update correctly when *sequence* is performed using any of the VCR controls.

```
1       function updateSlider(){
2
3               //move the slider to the appropriate value
5               $("#temporalSlider").slider("value",sliderval);
6       }
```

Code Bank 13: Updating the `temporalSlider` for *Sequence* Flexibility (in: *main.js*).

c. Temporal Legend: Transitioning What You Have Learned

The final step in implementing the temporal slider for *sequence* is modification of the temporal slider such that it doubles as a temporal legend, indicating the current timestamp. This includes two interface design components: (1) visual affordances indicating all possible timestamps to which the user may jump (i.e., adding tic marks along the temporal slider) and (2) visual feedback indicating the current timestamp selected by the user, whether by the temporal slider or other *sequence* controls (you already may have implemented this for extra credit in Lab #1). Think about how you may implement these visual affordances and feedback in your application. Do you need to add these affordances through JavaScript, or if you can add these details through HTML5 and CSS alone; what are the relative advantages and limitations of these different approaches? Consider the various Slider widget implementations provided at the jQueryUI Demos page; can any of these be modified to fit your needs? Finally, be sure that your final solution matches the look and feel of your overall webpage design as well as properly fits the design scenario.

4. On Your Own: Improving the User Experience

a. Operator Primitives: Transitioning What You Have Learned

Now that you have implemented both the *retrieve* and *sequence* operators using the provided Code Bank references, it is time to try your hand at a third operator on your own. You must add at least one additional operator discussed in class, but are encouraged to provide additional operators to improve the user experience. Adding addition freedom or flexibility for the *sequence* or *retrieve* operators does not count towards your third operator, although again you are encouraged to add such additional freedom or flexibility. Think back to lecture discussion of the *how?* question when choosing an appropriate third operator; individual operators may be more or less appropriate in the Lab #2 content of Interactive Cartography. As always, be sure to keep the design scenario and anticipated uses of your interactive map in mind.

Evaluation Rubric: Interaction Challenge (40pts)

Delivery: You are required to publish a version of your map to your webspace AND upload a *.zip* of your entire directory to the Learn@UW Lab #2 Dropbox at least one hour before your lab on **March 14th, 2013**. While there are many opportunities for bonus points, you cannot exceed 40 points overall on this assignment.

Retrieve Operator (4)

(2) Popup Content (space+time+attribute)
(2) Popup HTML Design
(+2) Retrieve Freedom

Sequence Operator (18)

(2) Play
(2) Pause
(2) Back-Step
(2) Back-Full
(2) Forward-Step
(2) Forward-Full
(2) Slider
(4) Sequence Interface Design (Skinning)

Additional Operator (10)

(8) Interaction Functionality
(2) Interaction Design
(+2) Additional Operators

Representation (4; Updated from Lab #1 Feedback)

(1) Basemap: Projection Centering
(1) Proportional Symbols: Scaling and Styling
(1) Proportional Symbol Legend (required for Lab #2)
(1) Temporal Legend (required for Lab #2)

Design for Scenario (4)

(2) Consideration of Additional Operator for Scenario
(2) Surrounding Webpage Design (improvements from Lab #1 + consideration of scenario)