Mini Project 4: Interactive Programming

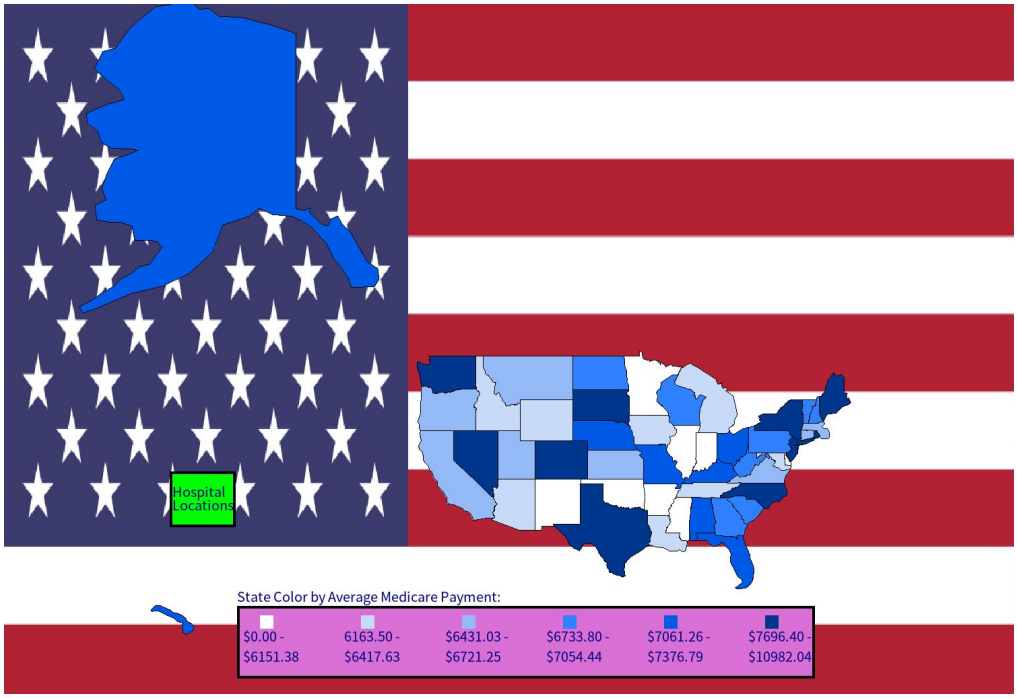
By Daniel Connolly and Jillian MacGregor

Project Overview

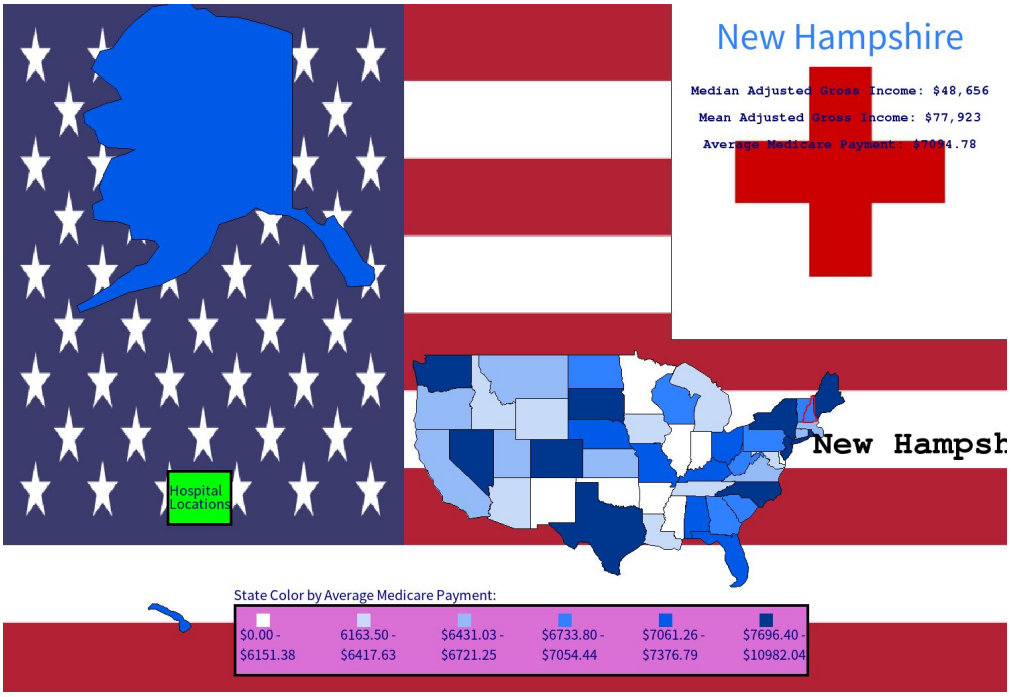
We created an interactive healthcare data visualization of the United States that displays state-specific average medicare payments, income levels, and hospital locations. The map exhibits a country-wide spread of this data, allowing users to view and compare it states in each regard.

Results

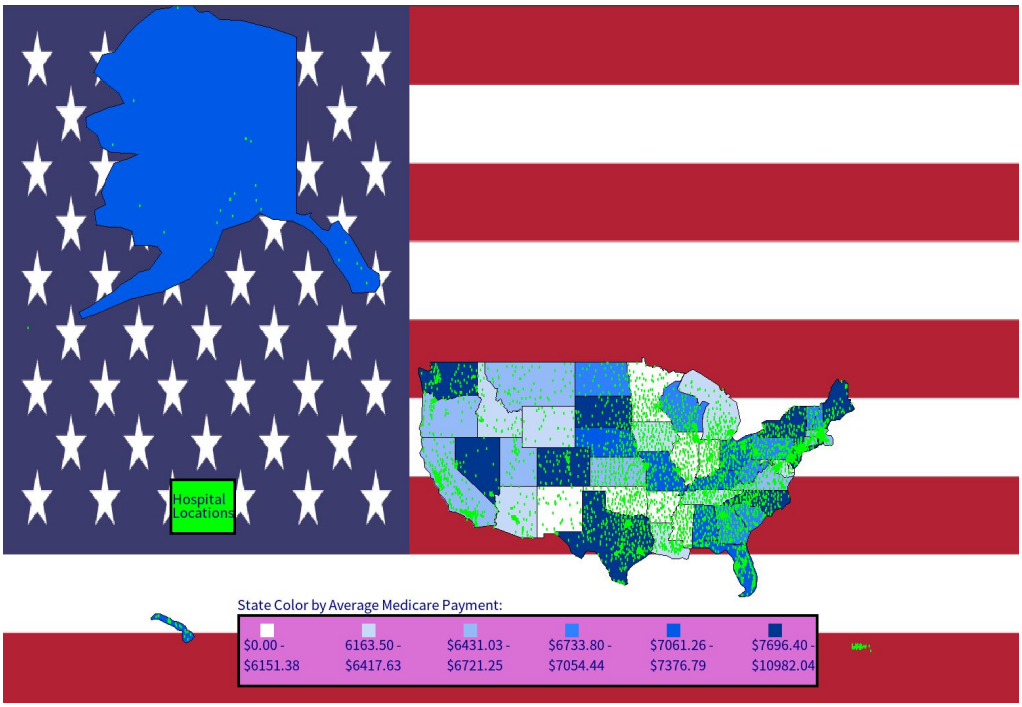
When you run the program, it displays a choropleth-style map of the United States, projected from actual coordinates onto a flat surface using a Mercator Projection. The color of the states, which varies between shades of blue, shows the average Medicare payment in each state. According to the legend at the bottom, the darker states show a higher average Medicare payment statewide, and the lighter/white states show a lower average Medicare payment. This color visualization gives the user the ability to view how expensive payments are by state in relation to others around the country.



In addition to this, the user has the option to click on a particular state, which will then present a pop-up of information about that state; it shows the median adjusted gross income, the mean adjusted gross income, and the average medicare payment. As a result, the user can easily compare the average taxable income versus the average medicare payment per state, so that the user has more of an idea of what the payment means for individuals in that state.



Finally, the user can click the green “Hospital Locations” button. As shown below, selecting it displays the locations of all the hospitals in the country, which one can utilize to compare the frequency of hospitals in each part of the country.



Implementation

Our project is effectively a two component system; data collection and the display of that data. To gather the data necessary for our map, we mainly employed the json and csv modules to

parse information from files found on the data.gov and census.gov websites. We then utilized pygame in order to create an interactive display of the data we had amassed, including drawing individual polygons to represent each of the fifty states, the coordinates for which we sourced from an online json file. Once we had made these polygons, we used a Mercator projection (which explains the vast size of Alaska on our map) to convert their coordinates to the pixel coordinates on our display. One of the more difficult challenges we faced in this regard included finding a way to determine the state in which the mouse was clicked, as pygame provides no easy method for determining an interaction with shapes that are not rectangular. In addition, pygame limited our ability way to effectively show a state-by-state choropleth map in which a state's color corresponds to its average medicare payment.

Because we chose to construct a data visualization rather than a game, we decided not to use the Model-View-Controller class structure in order to implement our map, although not without trying first. We managed to create two working classes, a State class and a Point class, before moving onto the more challenging task of finding a way to create classes for the model, the view, and the controller. As we understand it, in the MVC structure, the controller interacts only with the model, which in turn updates the view for the user. Unfortunately, we ran into problems and found that we needed the controller to be able to interact with the view, as we were unable to design a simple method in the model class to update the view based on an interaction with the controller. In our final attempt at an implementation of this structure, we managed to allow the controller to interact with what we believe to have been a model that was transformed from the position of the map that the view displayed; we were unsuccessful at determining the root cause of this. For this reason, as well as the fact that our structure was not strictly Model-View-Controller, we decided to pivot back to a structure devoid of classes. Based on our experience with this project, it seems that we should now be able to design a better class structure for data visualization than MVC in future projects.

In the end, our structure mainly revolved around several small functions designed to tackle important problems such as converting from longitude and latitude to pixel coordinates as well as creating an efficient way to create a choropleth map. Besides these functions, we include the information necessary for reading in and storing data from our data files as well as a loop that continually runs and updates the display. Though less structured than an implementation involving classes, this design is effective and works well for data visualization.

Reflection

In terms of this project, the data visualization path that we decided upon made our implementation a bit different from that of other groups and from the Model-View-Controller design that was suggested for the project. However, we believe that we appropriately scoped our project given the time limitations that come with a mini-project. We successfully created a minimum viable product early in the process, which allowed us to attempt several implementation styles and methods of finding and parsing data gathered from research. Though pygame proved limiting in terms of data visualization, it enabled us to gradually build on top of a

solid foundation. If we were to have to restart the project, it would be extremely helpful to know the applicability of suggestions regarding code structure to our particular project. For example, the classes, and specifically the Model-View-Controller classes that were suggested to us failed to effectively break the code down into small, easily readable chunks. Rather, they created problems and made the code far more syntactically confusing. Therefore, if we had more time, we would most certainly work to further organize and condense our code.

In terms of teaming, we never solidified a plan regarding how to break up the work, as it was somewhat unclear how best to implement our ideas. instead, we decided to break off and research the best way to display the data. Though we soon converged to one basic model of displaying the data, we still worked separately on a great deal of the implementation and research. Towards the end of the project, we began to meet more often to compare and parse through one another's code and our teaming style became more like pair programming. As it was the first teaming experience in the class, we began the project unsure of our styles in terms of working with others on coding projects, but were able to learn as we went and improve as a team. Because we chose different ways to implement the visualization, we often found it necessary to explain our code to one another when we met. This proved to be helpful, as it allowed us to better understand it ourselves and ultimately led to a more collaborative approach towards the latter end of the project.