TIGIS Assignment 3 - ArcheoLogic Visualiser - Report

Student 9217953

January 2020

1 Website Address

The project can be accessed at the following address:

`https://www.geos.ed.ac.uk/~s0092179/cgi-bin/visualiser/visualiser.py`

2 Design Brief

To design a website capable of interacting with an Oracle database, dynamically rendering a web map and associated data, and allowing interactivity including querying of the data by the user.

3 Assumptions

- Database contents will remain constant (though some future expansion is factored into the code)
- The physical units of the co-ordinate system are unknown "map units", for simplicity
- The website users will have a basic familiarity with the subject matter of the database
- Finds found on the border of a field are counted as being in that field
- Finds found on the border of two or more fields are counted as being in all of those fields
- Fields can only have one owner
- A field contains only one crop

4 Method

The task was achieved by using a combination of Python3, HTML, CSS and JavaScript. The website is hosted on the University webserver.

The intention was to build the system using the minimum of add-ons and specialist modules and packages. This ensures that the code written is original and it is more obvious what the student can do in terms of the fundamentals of coding. Web design tools such as JQuery, Bootstrap, and specialist python modules for rendering web maps were avoided for this reason, though they would have made certain aspects of the design easier to implement.

Jinja2 was used as it is fundamental to the clean interaction between the python code and the HTML page. *cx_Oracle* was used to provide a python interface to the Oracle database.

4.1 Logic & Backend

The core of the logic is handled in python. The program first connects the database and accesses in turn the four tables/relations relevant to the project - **MY_FINDS, MY_FIELDS, MY_CROPS**, and **MY_CLASS**. It reads each row in each tables, and creates an appropriate object (*Find*, *Field*, *Crop*, or *MyClass*) in python for each row. These objects are packed into lists by type.

A series of functions are then run to add extra attributes to some of the objects, including colours for use in rendering the elements of the webpage, and cross-referencing between objects of different types - effectively rendering the effects of specific table joins as object attributes. For instance, the function "*get_which_finds(fields, finds)*" takes in the lists of fields and finds, and cross-references them via a spatial query, rendering into each field object an attribute containing a list of which finds were found in the field, and vice-versa for each find object an attribute with information of which field(s) it was found in.

The final step before rendering the webpage is the creation of a new *GraphicsArea* object which defines the area on the website which will show the SVG map. Then *Jinja2* is invoked in the *render_html()* function to display the website and make the various python objects and variables available to it.

4.2 Webpage

The page is made up of three main elements - the map (for visual display and user interaction), the forms (for user queries), and the tables (for displaying details of selected items).

4.2.1 Map

The map is rendered dynamically using SVG elements <rect> (for fields), <circle> (for finds), and <pattern> for the grid. Text labels are also rendered using <text> tags.

4.2.2 Forms

Three forms are added to allow a basic matching of owner and crop types with finds and fields, and also to allow some somewhat more interesting basic spatial querying based on rectangular and circular search areas. Dropdown menus are used for simplicity and to make validation much easier - it would be nice to be able to type in whatever input you wanted, but this would require a much greater degree of coding in the background to validate inputs and avoid errors.

4.2.3 Tables

The two tables for finds and fields are rendered row by row at runtime, but are each given a default class of "*hidden*" which means they are not immediately visible. JavaScript code is used to dynamically alter the classes on each table row to show and hide them as necessary.

4.2.4 Other Page Elements

Buttons are provided to clear the map and the tables, or to clear everything and reset all the forms to their default states. There is a basic header and footer with page and author information.

4.3 Interactivity

JavaScript and CSS are used to provide interactivity, mostly through the forms triggered by form submit buttons, or through the use of "*onclick=someJavaScripFunction()*" elements. Showing and hiding of elements is done by getting elements by ID or some other attribute, and altering the class attribute to show or hide them.

5 Problems & Challenges

The python logic of the project was fairly straightforward to implement. The main difficulties arose from working with JavaScript code - I was unable to find out an effective way of passing python objects into JavaScript functions by using *Jinja2*, and so the implementation of the JavaScript for form processing seems unnecessarily complex. It took significant time to make progress on these challenges. The other main challenge related to the use of CSS and HTML to lay out and style page elements properly. The theory seems very straightforward but in practice proves to be quite a difficult thing to do well. I had hopes for implementing a responsive webpage that worked on all devices and browsers, from small phone screen up to the university lab machines, but was unable to do so on time and to a high enough standard. However, a great deal was learned in the process, and the next attempt will be far easier.

6 Potential Improvements

There are many ways in which my implementation could be improved. Here are just a few ideas.

6.1 Functionality

Ideally there would be a way for the user to add details of new finds and fields, and have everything aspect of the page updated to reflect this. The code is already set up to be very flexible and to handle the addition of new items, although as it currently stands the map would not render properly if a new field or find was added that was much outside the bounds of the current graphics area. However, this would be relatively easy to implement if needed.

When spatially searching, I would like to have added functionality for rendering the shape of the search area onto the map. Also the ability to search by clicking and dragging on the map, and having a shape appear, would be a nice addition. The ability to export data from the tables would be a bonus. Zooming and panning of the map would be a nice touch, especially if a much larger number of fields and finds were to be added to the tables in future.

A further simple piece of functionality would be the ability to select more items to match ALL of - at the moment the code only allows the choice to match ALL of only two conditions - owner name and crop type. Ideally the user would be able to match many more items, for instance to be able to answer a more query like "show me all the bronze age metalwork finds found in strawberry fields owned by farmer Green at a depth of less than 1 metre" and so on.

6.2 Layout & Style

The design of the website is adequate but far from ideal. Given more time and expertise I would have made the site fully responsive and better laid out, mainly by using *@media* queries in the CSS. For instance, the tables tend to be off the bottom of the screen, and the user is required (especially on smaller screen) to scroll down to see the selections, which is not ideal. The addition of some graphics and a logo would improve the look and feel of the site.

6.3 Code & Performance

The code works quite well for the current size of database. However, if the number of fields, finds, crops, and classes were to greatly increase, it would no longer make sense to create python objects from every line of the tables. In that case it may be better to find a better way to render only those objects most salient at a given scale or zoom level, perhaps a little like the way in which Google Maps loads "tiles" of data covering the area of the device viewport and a little way outside of the visible screen, in order to make the user experience smooth when zooming and panning. It also does not necessarily make sense to use python functions to populate object properties that may never be accessed, and therefore a more efficient method would be needed to achieve this. Although the current system is fast, I think performance would be appreciably diminished with a large increase in database size.

7 Conclusion

The *ArchaeoLogic Visualiser* works as a basic interactive web mapping tool meeting the basic design specifications set out at the beginning. It is fast and is quite robust. Some interesting functionality has been implemented, and overall the visual design makes the process easy to understand for the uninitiated user. However, a great many improvements to function, design, and performance could be implemented with a little more time and thought.

References

[1] J. David Eisenberg and Amelia Bellamy-Royds. *SVG essentials*. O'Reilly, 2015.

[2] Adrian Holovaty and Jacob Kaplan-Moss. *The Django Template System*, pages 31–58. Apress, Berkeley, CA, 2008.

[3] A. Martelli, A. Ravenscroft, and S. Holden. *Python in a Nutshell: A Desktop Quick Reference*. O'Reilly Media, 2017.

[4] Tyler. Mitchell. *Web Mapping Illustrated*. O'Reilly Media, 2005.

[5] Michael. Worboys. *GIS : a computing perspective, Second edition*. CRC Press, 2004.