An elaborate pun involving automata

Nate Wessel - University of Cincinnati Department of Geography – December 2013

Abstract

'Sitemaps' are hierarchical online website page listings used as aids to robotic web indexers. The goal of this project is to develop a sitemap that takes 'map' literally, translating the page hierarchy and organization into an imagined transportation hierarchy, and placing it in an imaginary spatial context derived from site metadata. Using principles from cellular automata models, I built a plugin for WordPress that, on each new page request, uses available post metadata to produce a unique, stochastic map-of-the-website. Written in PHP with some client-side animation help from JavaScript, the program outputs the map in Scalable Vector Graphics(SVG) in the body of a page. Human users as much as bots then can explore the map and make sense of the way posts and pages are connected in the logical structure of the site.

Introduction & Background

A 'sitemap' is a tool used by some websites, particularly very large or complex ones, to make sure that indexing search engines have accessed and indexed all of their (important) pages. Generally a search engine's web-crawling bot will try to follow links from any part of the web recursively until it's found and recorded the contents of every website that it's indexing. The sitemap then is a sort of shortcut, and insurance against the formation of content islands in a website, where pages or clusters of pages might exist and even be important in their own right, but might not be linked to from other parts of the web and thus not indexed. The sitemap also intends to relay hierarchical information more clearly and thus to place more fundamental pages higher in search rankings all else being equal; generally, you want your 'www.example.com' to show up before your 'www.example.com/puppies/kittens/tarzan-wearing-a-powdered-wig/toothpaste.jpg'

Sitemaps then are basically meant only to be seen by robots. But what if they could be made fit and even useful for ingestion and indexing by people too? My basic problem was this: I have a website, CincyMap.org, which is really built around maps and particularly designed around a transit theme. I put a lot of work into some of the content, but some of the best content gets buried after a few posts in a blog structure that favors recent posts over older ones. Search engines are keeping track just fine since everything is linked to internally and sometimes from other parts of the web, but I find that readers simply don't know what if anything they're missing. It's also not clear what the structure of the site is beyond the linear temporal structure of the blog format.

My goal then is to secondarily, let's just say it, riff on a engaging graphic theme and make a pun, but primarily to engage readers by making for them an engaging and understandable and thus *indexable* display of what kind of stuff is in my website and how it's interrelated.

A final complicating factor is that, like all good things, the website which the sitemap should attempt to describe will always be changing in unpredictable ways, and so the output

or form of the map must adapt well to an infinite range of inputs. Properties that emerge from complex interactions will set the tone of each fleeting reinvention of the sitemap.

Methods

The sitemap was developed on a WordPress 3.7.1 blog with 85 posts in 26 categories having ~500 tags, ~250 of them unique. Currently, it's implemented as part of a custom 'theme', though my goal is to isolate it in a 'plugin' that may eventually be released to the public. Themes in WordPress are a set of files that dictate the form and presentation of a site, sort of like a die through which content, stored in the same format for each site, is extruded. A plugin, operates similarly but generally focuses not on presentation but on providing some particular functionality like visitor tracking or an inline content item like a survey form.

WordPress and it's themes and plugins are written in PHP and so logically was this project. PHP runs on the server side, and is great for outputting web-pages or other text-based things because code can be interspersed with text elements like any HTML or SVG tags. Rather than having to 'echo' or 'print' every line of output, much of it can simply stay the same, with PHP tags printing just the dynamic content.

The map's essential idea is taken from cellular automata models and there are two basics types of agents that interact in a cellular space: categories and tags. Categories are broad buckets into which posts can fall, and tags are generally more specific descriptive words. Every time the sitemap is requested, ie every time a the webpage with the sitemap is requested from a server by someone visiting the page, the script will be run and a wholly new map result will be output and sent. Two people requesting the page at the same time will get totally different results. Here's how it works:

1. First, basic metadata is gathered from WordPress's APIs. Each unique category is identified with name, id, link and other information, and that information is passed into an object for each that will be the category's agent. The same is done for tags and posts, though posts don't get agency but are rather subservient to the category to which they belong.
2. Next a matrix of cells is created. In the test case, things worked best at 35 x 35 = 1,225 cells, though I intend to implement some trivial code that will allow the matrix to grow dynamically according to the scale of the content.
3. For each post, each of that post's tags are assigned to a random cell in the matrix. Many posts have redundant tags but that only makes those tags more important or relevant to the site. What actually happens programmatically is that cell objects are linked by reference to a unique tag object and some cells just link back to the same tag.
4. Once all the tags are randomly located, the matrix, which was built from left to right is shuffled or randomized. We then walk through the matrix cell by cell for twenty iterations checking on the happiness of each tag in its current location, and if it's unsatisfied, telling it to move to a cell that it finds more to it's liking, looking randomly until it finds it. 'Happiness' here is a combined measure of simple *number of neighbors* and similarity of neighbors. Very similar neighbors are weighted such that two identical tags can be perfectly happy off in the middle of an empty space, but other tags will tend to cluster strongly in a pattern familiar from Schelling segregation

models. Other measures of similarity are length of tag name and assigned color but these are weakly weighted.

5. Once the tag/landuse/landform CA model has been run, some distinct continents or islands or urbanized areas or however you might like to interpret them will have formed in the matrix. It's now time to connect them with transit lines!
6. Each category is a transit line, and each transit line is built one at a time, starting with those on the top level of the category hierarchy(there are subcategories). Each starts at a random point in one of the landmasses and lays down it's first post as a stop on the line. A trajectory is then decided by, if the line is near an edge, steering somewhat toward the center, or if it's not, by favouring a direction that has 'land' or tags. After the initial trajectory is chosen, the line moves forward, still planting a post/stop at each step, by assessing the qualities of the three cells ahead of it that it could potentially occupy next. Transit doesn't (or shouldn't) take sharp turns and neither do categories turn by more than 45 degrees at once. Forward cells are assessed by their distance from the edge of the matrix(no falling off the edge!), by the presence of tags(favored) and by the presence of posts(disfavored). It's thus intended that lines will generally follow the contours of the continent they find themselves on while avoiding overlap and occasionally jumping off and finding footing on solid tag-ground. Each subcategory follows the same pathfinding rules, with the exception that it begins from a position somewhere on the line formed by it's parent, and derives it's color from it's parent's color.
7. Once the categories have run their course, the agent generated part is done and the results are output(sent to the requesting computer), each element getting a descriptive title and a link to the actual post, tag or category.

Results

Results are mixed. The landuse/tag model seems to have been more immediately successful than the transit/category model, almost certainly because the latter is much more complex and thus prone to bugs and my own errors in conception. I think the basic concept has proven itself to be a very interesting one and the project will be further developed for actual, prominent deployment on my site.

Specifically, the tag/landuse model demonstrated strong emergent qualities and looked plausibly like an actual map of some geographic arrangement. It did not however, as I had hoped it would, demonstrate a strong clustering of similar or repeated tags. Tags tended to be content once they were in the middle of a very large cluster, or once they had found an identical partner. This meant that some tags, like the word 'transit', that were repeated as many as 30 times did not cluster among themselves but rather scattered pretty randomly with quite a few pairs of two or three. The transit/category model also demonstrated some emergent qualities but only occasionally presented a plausible transit system that interacted well with the tags and more often produced erratic errors such as a line hopping randomly to the other side of the matrix. This shattered the illusion and weakened the pun.

Some non-critical bugs still need to be ironed out, but for the category agents in particular, the direction selection mechanism may need to be fully redesigned before live deployment will be non-embarrassing.

Discussion

While the results at the moment are less than totally intriguing, I think this is a very interesting problem and that I've taken a good basic approach to addressing it. There are as I've said, a number of particular problems with the implementation of the idea in code, but the approach itself is solid.

The basic problem was to create an image that replicated an artifact that modeled a complex, organic system(a city infrastructure). The emergent qualities of the city or of a landscape are some of the first things that let us identify a map as a map. A map for example of a place with perfectly rectilinear and evenly spaced streets, no center, and a perfectly square coastline, would not at first strike us as a map, even if the visual tropes of a north arrow and scale bar were right on the page. What makes a map a map at least as much as the style, the line-weights, the earth-tone colors is the subject matter and it's recurring organic quality which we as life ourselves are so quick to recognize and appreciate as beautiful. It's this quality that required an agent-based approach.

It's the familiarity we have with this same quality that I hope will eventually make this map interesting to readers, and worth exploring and playing with. Fantasy maps, particularly of urban rail systems, have always seemed to have some inexplicable intrigue. Most of these maps, and there really are a good many of them, are drawn by hand, most of them of real places transformed by an imagined transit system. A brave or naive few do attempt to imagine a wholly new city infrastructure, but I think these attempts are less successful because we can't conceive of plausible cities ourselves for the same reason that socialism couldn't. Cities thrive where central planning and singular conception fail. They result from the actions of agents with conflicting self-interests and they demonstrate qualities and characters that we'd be more likely to locate in a god or demon than in ourselves.

Perhaps I'm suggesting a new subgenre of the fantasy rail map: that created by the same method as a real map but from imagined *initial conditions*. My initial conditions here are a WordPress blog; yours may vary.

The current most-stable version of the sitemap is to be found at
http://www.cincymap.org/blog/sitemap/