Wave Function Collapse

Group 2

<https://jbeckettsweeney.github.io/WebDev/>

1. Objective

The Wave Function Collapse (WFC) algorithm is an extremely neat algorithm that takes some input, groups it into “similar parts,” then generates a new output in the same format as the input but of near limitless size and randomization.

We set out to use this algorithm in an easy to understand web page that allows a user to create their own pixel bitmap and generate larger random images from that input. This was intended to be used primarily for game design, but the applications can be more broad.

2. Summary (How does it (should it) work? Provide high-level overview tools, frameworks, APIs, front/back ends, etc.)

The main things that this project takes advantage of are HTML, CSS, and Javascript. More specifically, the pixel editor uses HTML5 Canvas, which makes drawing and editing pixels very simple.

In theory it should have taken the image file generated from the users bitmap (or uploaded by the user) and separate that into the smaller sections to be randomized and combined together to create the larger finished image. We intended to have a feature where the user could select areas they want to keep and then generate a new image that works around the same section. We also intended to have the website be very user friendly and good looking as compared to the already implemented one on the github.

3. Members (Group members' contributions and work logs (tasks and how they were overcome))

3.1 Sage

I attempted to understand and implement the wave function collapse algorithm. To begin, I looked at the original algorithm [1] but wanted to see a web version. Bailey had found a tutorial [6] which really shows how complex this algorithm can get. It outlines the large number of input parameters for different variations. This tutorial was actually trying to explain the code found on a different site [7], but didn’t explain around 400 lines of code. The unexplained code consisted of the actual algorithm, so I decided to just look at the website that implemented the actual code. At the time, I decided the 1500 or

so lines was a bit much to try and tackle when I was still trying to understand the algorithm itself, so I looked for more tutorials.

The most helpful tutorial I used actually implemented WFC in Python [3], which was perfect for me because that is what I am most familiar with at this point in time. The implementation was a very slimmed down version. In fact, it was slimmed down to the point where it used characters instead of subsets of a bitmap. These characters were stylized during printing with Colorama [5], which I learned. The simplicity helped me understand the algorithm, but the use of characters and the way certain things were implemented made it very difficult to port back to Javascript. To better understand the use and manipulation of the canvas elements I found a tutorial [4] that stepped through every conceivable usage of a canvas.

For implementation, I attempted to create what could be considered an encoder. The purpose was to split the image into its sub-blocks and assign integer values to different sections. This resulted in many problems. Different sections had to be compared pixel-wise to determine if they are the same block. Then, pixel-wise comparisons had to be made on the edges to determine the rules. This is not feasible when things like symmetry are allowed.

Unfortunately, I have not been able to combine everything I have learned from all these tutorials to make WFC work in a browser, thus far.

3.2 Bailey

Along with Sage, I also worked on implementing the algorithm. We used two different approaches as time ran out to attempt to get it done, and I definitely underestimated the work that it would take to successfully make this algorithm run on a webpage.

To help with the process, I utilized a lot of sources including the algorithm github, a tutorial for implementing it in Javascript on a webpage, and the Javascript port of the github page. My idea was to reverse engineer the existing Javascript port website and look at how they made it work with their code and inputs. It was really undocumented and the code was confusing to read through without comments and with me being admittedly unexperienced with Javascript. I had some success in the implementation in the end we continually got caught up on getting a running version within our scope of the project.

I think the biggest hurdle I kept running into along the way was how undocumented everything in the Javascript port was. It made it very hard for me to see what was going on without doing more research into the algorithm itself or just attempting to implement it in another language. To combat this, I did do a lot of reading about WFC on the github, read through the code to try to break it down into sections that I could document and understand, and trial and error test sites. This project was frustrating because it was super hard to test. You either had a fully functioning algorithm that takes an input and creates the correct output, or you had a blank canvas on a webpage, which did not help.

In the future I would really like to keep working with the others on trying to get this implemented in the capacity that we wanted. I think we ran into an issue of scope and the time we were given and if we had more time we could definitely get really close, as Sage and I are already both super close in our separate ways of trying to make it work.

3.3 Owen

My contribution consisted of the CSS for the UI/UX of the site. Initially, I intended to put a bunch of divs together in a fairly unorganized manner. Although we had gone over better alternatives, I've worked with hacking together CSS in the past, so I figured that would be suitable to my experience. However, as I worked through iterations of design, I found the grid pattern to be more and more suitable to the task. Our site is largely modular, making the grid pattern ideal.

Early designs included sidebars which had the buttons in order to run the backend Javascript. By having these in the sidebar a more game-like UI would be created, which suited the material as that is one of the many applications of WFC. However, I eventually decided to scrap this idea, since I couldn't find a way to make this look nice, which was ultimately my task. Instead, I went for a more barebones button appearance, following a similar aesthetic as other similar WFC sites. I did decide to keep the colored sidebars, as I felt they added a unique flair which helps to differentiate our site from the aforementioned similar WFC sites.

A more functional decision made was to allow the screen to extend further downward, which allows for larger image outputs. This could have been done by stretching the screen post image generation, however I decided that this site would be more conducive to being static in appearance, rather than more dynamic. The reason for this is that it should be fairly simple in terms of user interaction, and the design should in turn reflect that.

3.4 Beckett

Most of my contribution was on the Input portion of the project. I created the pixel editor for the site and prepared the user-generated bitmaps for being saved as an image.

I began with the first part of a tutorial for making an HTML5 Canvas pixel editor. Upon finishing this tutorial (which only really went over how to draw in a canvas), I found that a second part was never made. So I thought about how the rest of the editor might look and went on my own from there.

The editor takes in the user-input size that they want for the editor (10 gives a 10x10 image) and uses that size and the pixel size of the canvas itself with the mod function to get the size that each drawn pixel should be. From that point, a simple double for-loop went through and drew each pixel the proper size in the color of the color picker.

A problem that I ran into at this step was that the pixel size of the canvas didn't always mod very nicely, so occasionally the last row and column of the drawn pixels would be left out or misshapen. The easiest solution to this would be finding the least common multiple of all reasonable input pixel values (roughly 1 through 20) which results in a number that is FAR too large to use on a web page. I ended up playing around with certain values and ended up with one that works well for MOST values between 1 and 20, but not all.

Next I needed to make the canvas capable of being edited. I made a listener for the coordinates inside the canvas where a user would click, then used the mod function again to see how far off that click would be from the top left corner of the current drawn pixel. At that point, I drew over that drawn pixel in whatever color was currently selected for the color picker. I thankfully ran into very few issues with this part of my portion.

The last major part that I needed to do was create an actual size version of the pixel editor. I made a second canvas that was the exact size of the user-input size. Then I added a line to the original pixel draw function so that in addition to drawing the select pixel on the editor, it also drew that corresponding single pixel on the actual size canvas. This was so that we could use either image for use with the algorithm depending on what it needed. A simple single line allows either canvas to be saved as a png in a variable so that we could input it into the algorithm.

4. Conclusion (What you learned, what worked, what didn't, what you would do differently or the same.)

At the moment, getting the algorithm to function properly is a nightmare.

For the algorithm I would definitely start with hard-coded rules, when the code is being developed. Rule generation is very difficult, especially if the side lengths of the sub-blocks of the input can very. This would allow the core loop to be developed without concern over other parts of the algorithm.

I would look into a framework such as Easel.js or something else that could simplify the interactions with the canvas elements.

Something that would be cool to implement would be encoding the bitmap values in the URL using a GET request. This would allow the information from the pixel art editor to be sent to the WFC page without having to use storage, caches, or download an image. It would also allow the page to be bookmarked so users could return to a work in progress. Some additional features that could be implemented in the future would be the actual algorithm, locking sections into place to only regenerate portions of the algorithm, keeping track of the last few generated patterns to the user can revert to a previous one, and various parameters in the main algorithm such as seed (to manually seed the randomness), symmetry, periodic (to modify tiled inputs), and ground (to change the rules so absolute position of the original matters).

MDN has good tutorials and documentation. Colorama will be useful in future python projects.

References

1. https://github.com/mxgmn/WaveFunctionCollapse
2. https://medium.com/@arhillis/pixel-art-maker-an-html5-canvas-tutorial-part-1-setting-up-the-canvas-fd71c868cc13
3. https://robertheaton.com/2018/12/17/wavefunction-collapse-algorithm/
4. https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial
5. https://pypi.org/project/colorama/
6. http://www.procjam.com/tutorials/wfc/
7. http://www.kchapelier.com/wfc-example/simple-tiled-model.html