Maze Solver

a web based application

by

Nathan Joyal

CS480 Fall 2021

01 Introduction

At first, when presented with CS480 Senior Project, I was overwhelmed by the open-endedness of the project itself. I consider myself to be a competent and, perhaps more importantly, persistent computer programmer. However, the responsibility for conceiving of a project from nothing, which matched the scope for the course, was difficult for me, for some reason.

I had several false starts. First, I thought of doing a very simple financial calculator that made advisements about buying crypto currency versus the cost of mining. However, this was too simply to meet the scope of the project. Another idea I explored was some sort of game or simulation based on the exploitation and development of land and natural resources by various historical societies. This proved to be too great a reach, and getting even a cohesive demo together became somewhat absurd and ill-defined for me. And so it went, through a handful of either poorly conceived or unattainable ideas.

What finally led me to the project I submitted for review was a role in life that I fill which has nothing to do with my computer programming, directly. I am a father to seven children, five of whom are between the ages of 10 and 15. So, I get a lot of questions about what I spend my time and energy working on all day. At this age group, they have had little to no exposure to actual coding, and only minimal exposure to direct logic concepts. They only understand logic as an indirect thing, and have never really given a lot of thought to how we make decisions from the standpoint of calculation.

Due to my role as a father and educator in my own home, and my aspirations to become an educator professionally, I decided to produce a project that illustrated in a very satisfying and visual way some core computer programming concepts. This tool could then serve as a jumping

off point to talk to people (teenagers being my target audience). Beyond this core concept, I wanted a product that was highly visual and simple to interact with.

I decided that a visual representation of different search methodologies would be a good fit. As a side note, I was also fresh from Dr. Horn's CS470 AI class, where we explored various search algorithms, which had resonated with me. I decided on putting "depth first search" and "breadth first search" side by side trying to solve a visual maze.

The final piece of the 'puzzle', as it were, came from my project advisor, Dr. Appleton. I had initially conceived of doing the project in Java using the JavaFX visual library, simply because this was the delivery option I was most familiar with historically. However, Dr. Appleton really strongly advised that a web or mobile based delivery would be better suited, in order to reach a wider audience and be more readily deployable in various avenues. I didn't have any experience or knowledge of any of these systems, so I chose JavaScript to write the project in, because the tools were the simplest and most accessible – I did most of my coding in Text Pad 8.

And so it was decided – I would be producing a program which solved a user generated maze. The program would be written in JavaScript (and I think maybe HTML? What a fun adventure!) Further this program would serve as an interactive tool to put in the hands of teenagers or people with no coding experience, as a jumping off point to talk about computer programming in general, and more specifically how computers make decisions.

02 Initial Deployment

The initial phase of development went quite well. I had essentially no JavaScript experience – 30 lines of code ever, for one assignment in CS 322 Principles of Programming Languages once here at NMU. I relied heavily on my strong background in object oriented programming – I have always been most comfortable in object land – and based my initial designed on the type of work I have done in various other classes here.

The maze consists of a "map class" that represents the map itself. The map holds a two-dimensional array of a class of objects called "map squares" as well as some internal variables and methods for maintaining itself and interacting with the rest of the program. Each square knows its position in the map array (x/y coordinates), contents, and neighbors. This way, the map as a whole knows where all of the walls, the start, and the exit are located at. This was not unlike a basic data structures problem, which I enjoy.

Next came making the solver objects. The depth first and breath first were not that different to code. The primary difference was how they kept a record of potential other squares to visit, and how and when they accessed that list (from the front or the back, especially). Coding these algorithms was new using the peculiarities of JavaScript, but fundamentally well within my wheel house.

At this point, I had achieved everything I really knew how to do. I had a working 'map' which could describe an orthogonal grid which contained walls, empty spaces, and a start and a finish. I had two different algorithms which, when handed said map, could then solve the maze the map represented and explore squares following their given paradigm until they found the exit, or explored every available square if the exit could not be reached.

03 Technological Hurdles and Achievements

3.1 Web Deployment & Graphics

To deploy my maze and accompanying solvers, I needed to house it in an HTML document for display in a browser window. Beyond this obvious step, I needed to identify and learn a methodology or creating and manipulating graphical objects in this environment. Remember – I had no prior experience with JavaScript or HTML environments prior to this. I did some research and determined that using either what was called HTML Canvas or alternatively HTML CSS were my two relevant options. Being the traditional, studious fellow that I am, I ordered a book on HTML & CSS. This was an extremely valuable resource, and I've included a reference to it in the appendices.

To be completely honest, I reviewed both options and HTML CSS seemed like the more powerful, but also the more complex, option to choose. Due to the fact that this application required only fairly light graphical work (display a square grid and some interface buttons) I decided to go with HTML Canvas. The interface between HTML and JavaScript was an additional layer I was unfamiliar with – however, the way shapes and regions were coded in HTML Canvas was fairly similar to JavaFX. Actually designing the visual layout was a fairly straight forward affair, in terms of describing various rectangles and their location.

A design choice that was made was to use a single HTML Canvas object. This way, the HTML was fairly simple and all of the layout work was done in JavaScript by defining various rectangles and text fields within that Canvas space. The alternative would have been to use HTML to manage the layout, and have each button or map square be an independent Canvas object. The

choice to use one large Canvas object meant that in the JavaScript code, that Canvas object had only a single “click event” listener.

This design choice, meant that part of handling each click event was using math to determine exactly where on the screen the user had clicked. This – while labor intensive – was a fairly straight forward process of testing clicks and mapping out coordinates to the various buttons. We used a half-finished version of the product which, rather than operating, simply reported screen coordinates to accomplish this. The whole thing is then packaged in an HTML file, and viewed through a browser window.

3.2 Background Thread Execution

At one point in the development process, I had a working version which – while it was missing many of the features I had hoped to incorporate – was basically a working proof of concept. I took this “demo version” to my project advisor, to discuss where I was at and to ask for help prioritizing the remaining features. It proved to be a wildly beneficial meeting.

The most glaring feature missing at this time, was real time progress tracking of the solvers. How the JavaScript was programmed, was that each square the solver explored, it colored the grid square, and updated the visual display. However, what actually happened in practice, was that the user clicked solve and the program hung up thinking until it finished, and then colored ALL of the squares the solver explored, and reported the intended statistics about its work. This wait-until-done execution was less than satisfying, especially since two alternative solving scenarios were supposed to be racing to the finish.

I wanted something palpable and exciting for the user to be engaged with, not this boring "reading a book" feel that waiting for the computer to show all of the results at once caused. At my meeting with Dr. Appleton showing off the 'proof of concept demo' he recommended I speak to Dr. Kowalski, who was the most well informed of the faculty on the specific peculiarities of JavaScript. This proved to be an inflection point in my work, where we shifted from basic ground work to the real meat and bones of the thing.

After relating my frustrations with how JavaScript seemed to run in HTML – in that it needed to finish running before it would update the HTML – Dr. Kowalski pointed me toward a website describing what is known as a "WebWorker". This is a single thread of background execution which can update the DOM (Document Object Model, or the HTML part of the program) by passing messages in event handlers back and forth to the main page. This seemed to be the perfect tool for the job, as what I really wanted to do was hand each solving algorithm to a separate thread and race them against each other in real time.

The documentation and examples on the internet for WebWorkers which I looked at (included in the references appendix) all use very simple examples. Learning how they are defined and initialized, and how results were passed through their special event handlers, was perhaps the technical milestone I am most proud of. It was not totally straight forward to adapt them to run in competition and incorporate them into the underlying application and user interface, and getting them to work at all required significant trial and error.

Worth noting, the web worker draws its JavaScript code from a separate file stored on the hosting machine. This means that when one right clicks to view the code in the website, one can see that the website initializes a web worker, but not what code that worker actually executes. This is a fact of their usage which has potentially huge security ramifications.

3.3 Breadth First Disappointment

Another benefit of producing this proof of concept demo, was that it yielded some actionable information regarding my underlying concept. While comparing “depth first search” to “breadth first search” is an interesting and deeply meaningful problem in computer science, it was not very rewarding the way that I was presenting my results. This was true to such an extent that I ultimately decided to change the paradigm of search algorithm I was demonstrating.

The conditions of the context here which caused breadth first to be underwhelming where that we are not looking for an optimized path to find the exit. While this is within the scope of the search algorithms and the CS 480 project, it was less efficient as an inroad to the attention of teenagers. What was efficient was a flashy visual display. So we tracked the progress of the solving algorithms by visually coloring in the squares of the map, to show their progress as they raced each other, and ultimately to demonstrate which areas of the maps the two different algorithms explored on their way to the exit, as a historical record of their journey. The problem when using breadth first under these conditions, are that breadth first just colored in the whole map.

The result of this, was that in the demo version almost every single map you gave it was entirely colored the color of the breadth first search. This didn’t really provide the kind of “race condition” I was hoping for, and worse yet it left the colored map in a state that was definitely not ideal for piquing the curiosity of teenagers. Due to these factors, the decision was made to shift away from a breadth first search algorithm all together, and compare two different paradigms for depth first search, in order to achieve more interesting visual result.

This was how we ultimately came to a “left seeking” and “right seeking” model. I was inspired by the old human idea of picking a wall (left or right) and following that until you had

come to the exit. Of course, the problem with that, is that if there are legitimate “islands” of wall or large open spaces that exist in the maze, sometimes wall following does not reach everywhere. Rather than getting lost in a quagmire of conditional logic, I decide to slightly modify the concept of “wall following” to be “edge seeking”.

Ultimately both of the algorithms are somewhat purpose tailored to the conditions of the maze, as it’s presented here. Since the start is at the top and the exit is at the bottom, both solvers prefer to move “down”, always checking this neighbor first. The thing that makes them different, is what they do after that. The left solver checks left, right, up in that order, where as conversely the right solver checks right, left, up. In this way, they will “fall” toward their respective edge and “fill” spaces in the way that water being pour into the maze might fill cavities. This method of exploring the maze in a rank and file way often mirrors edge following, but can handle more complex spaces like caverns and islands. Additionally, it leads to a very fun competition to watch.

3.4 Saved Maps as Cookies

Another thing that Dr. Appleton keyed me on to is that accessing cookies from JavaScript code embedded in HTML was actually quite straight forward. Like anything else in computer science, a cookie is nothing more than a binary sequence stored in a particular location. Due to the availability of this resource, the ability to actually save a maze the user was working on to their local machine, and retrieve it at a subsequent date, was a reasonable goal to reach for.

The code for creating and retrieving cookies was actually quite straight forward. First, I developed a simple, reversable paradigm for converting a map into a binary sequence and back.

Then I wrote simple methods to pair that string with key values in the cookie buffer (individual cookie information is stored with paired key values in a combined buffer space).

3.5 Physical Web Hosting

The final aspect of deploying the project which is worth discussing from a technical aspect, is the physical hosting of the project. What I found once I started delving into deeper features of the JavaScript => HTML environment was that these features did not work when run off of a local file. Accessing cookies and web workers was things that only an actual-factual, living website could do, in most browsing environments.

This meant that in order to operate, test, and demonstrate my program, I had to have the HTML code and supporting files served live from a webserver which I connect to. Due to the fact that I recently took Dr. Appleton's CS 302 Unix Administration as an elective (I daresay this should be a required course) I was actually in a really good position to do this. While none of it is rocket science, the process I went through to bring the project live is worth describing here.

A dedicated Linux terminal was created by taking an older, decommissioned NMU laptop holding a blank OEM Windows 10 installation and formatting that completely and installing Ubuntu LTS 20.14. We then used Apache 2 to host the website. We kept the Apache setup extremely simple, we index to a file structure inside our website root directory, inside of which is the final version of the project, as well as a few other copies that either demonstrate prior concepts (like breadth first search) or contain diagnostic code.

04 Final Deployment

Ultimately the final product is something I can be proud of. With the notable exception of "dragging" walls onto the map, I have implemented all of the features which I discussed with my advising team during the initial phase of the project. The product works as intended and is, usable by people of various walks of life (including the intended teenagers) on a variety of devices.

Due to the web deployment, any device which can access a visual HTML page can access the program. I was happily surprised with how well it displayed on phones and tablets, as my intention for this deployment was for it to be accessed on a desktop or laptop computer. The solvers actively color the map squares in real time, racing, in competition, filling squares in their search for the exit. In addition to the web workers, I was able to incorporate use of cookies to save and restore maps.

Additional features I included were a button to generate a random map. This proved to be a great feature, as teenagers can quickly hit random map, then hit solve, and see meaningful results. In fact, in my limited testing in a pool of five participants, I found that testing various random maps was an enticing and informative "game" to play. Quickly examining many different maps allowed me to talk with my kids about what features of the map made the left or right solver more successful, and really played well into a discussion of how the computer actually works to solve the maze in the first place.

I included a number of quality of life and polishing features, some of which were recommended by Dr. Horn on my advisement committee. Without enumerating them all, I will mention quickly a few things worth noting. There is an abort button, in case the user wants to try a different maze or the solver takes too long to solve the current maze. This way, we can try a new

maze or restart the current maze without reloading the whole program, when the solvers struggle. Additionally, I found free sounds and incorporated into the JavaScript button response noises and other sounds. These polish and quality of life features, while not technically challenging, really went a long way to improve user experience and engagement, I found.

05 Reflections and Potential Future Work

Overall, I was quite happy with how the project turned out. During the semester I was reminded of John Conway's "Game of Life" and ruefully reflected that this – or something like it – might have been a better hook for actual computer programming concepts. However, I think that is just second guessing myself. Search algorithms are a huge part of computer science and I've had good results using it as a jumping off tool to talk to my kids about my work in a more informed way.

However, placing the walls is somewhat tedious. The random map feature was a stroke of genius which yielded far more beneficial results than I had anticipated when I made it. Outside of random maps, however, interacting the program is definitely tedious, and yields rewards disproportionate to the time investment it takes to actually create a meaningful map. Being able to "drag" walls onto the map was I feature I had set my sites on, but never quite been able to master. I tried using a web worker to draw the walls while the user held the button, but I was never able to get it to work.

I incorporated a frame work for having multiple exits (I thought of multiple starts, but I couldn't conceptualize a way for this to work which I liked). What happened when I had multiple

exits was just that whichever exist happened to be furthest to the left or right tended to determine who one – in other words it just added a higher degree of randomness that was difficult to chart, or to make heads or tails of in terms of a lesson in computer programming.

One thing that did really strike me as a possibility for future work on this particular product, is to port it to mobile operating systems. This semester, along with CS 480, I took CS 345 Android Programming. This was my first exposure to a mobile development environment. I immediately fell in love with the Kotlin programming language for a lot of its convenience and ease-of-life features. Also, the Kotlin-and-XML pairing is not unlike the JavaScript-and-HTML-Canvas pairing. For this reason, recreating this product in the Android environment wouldn't be an insurmountable task, and could produce rewarding results. It could then run as a standalone program, without a dedicated server hosting it. This independent deploy-ability would be a concrete and measurable improvement from the current iteration.

Appendix A: Proposed Grading Rubric

Although it's an artifact of previous iterations of CS480, I did create a 'proposed grading rubric' at the start of this project. While it isn't strictly speaking part of the evaluation of this project, I believe it is still worth including as a measure of what I considered, and proposed to my advisement team, to be a reasonable volume of work and corresponding set of goals to expect for a project of this scope. Again, while it isn't absolutely relevant, I believe it is worth noting that we did quite well versus our expectations at the outset.

Learning Objectives

Basic knowledge of JavaScript syntax:	5
Working knowledge of JavaScript classes:	8
Working knowledge of JavaScript methods:	10
Working knowledge of JavaScript program organization:	6
Working knowledge of JavaScript interface with HTML:	5
Basic knowledge of HTML 5:	8
Basic knowledge of HTML Canvas for visual objects:	8
Learning Objectives Total:	50

Development Objectives

Basic interface displays visually to user:	7
Map Squares can be toggled to create maze:	7
Solve button works to activate solver:	7
Depth First Search solves maze:	8
Breadth First Search solves maze:	8
Information about results output to screen visually:	7
Maze solvers highlight squares after solving maze:	8

Maze solvers highlight squares in real time while solving maze:	8
Mazes can be saved:	8
Mazes can be loaded:	8
Demo Maze for BFS included:	8
Demo Maze for DFS included:	8
Program is embedded in a website via HTML:	8
Development Objectives Total:	100
Overall Total:	150

Proposed Grading Scale

A – 150 - 130

B – 129 - 110

C – 109 - 90

D – 89 - 70

F – < 70

Appendix B: List of References Used

All sounds provided by:
ZapSplat. "Free Sound Effects". Accessed October 20, 2021. https://www.zapsplat.com/

Learn JavaScript Quickly: A Complete Beginner's Guide to Learning JavaScript, Even If You're New to Programming. Drip Digital publishing (November 10, 2020). 174 Pages. ISBN10 1951791479

HTML & CSS QuickStart Guide: The Simplified Beginner's Guide to Developing a Strong Coding Foundation, Building Responsive Websites, and Mastering the Fundamentals of Modern Web Design. Clydebank Media LLC (January 25, 2021). 362 pages. ISBN10 1636100015

Mozilla Developer Network. "Using Web Workers". Accessed October 1, 2021. https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers

Mozilla Developer Network. "Document.cookie". Accessed October 10, 2021. https://developer.mozilla.org/en-US/docs/Web/API/Document/cookie

w3 schools. "JavaScript Cookies". Accessed October 10, 2021. https://www.w3schools.com/js/js_cookies.asp

w3 schools. "HTML Canvas Graphics". Accessed September 5, 2021. https://www.w3schools.com/html/html5_canvas.asp

w3 schools. "HTML Web Workers API". Accessed October 5, 2021. https://www.w3schools.com/html/html5_webworkers.asp