Arizona State University
Barrett, the Honors College
School of Computing, Informatics, and Decision Systems Engineering

Mazes of Waverly Place: Interactive Algorithmic Art Generator

Waverly Roeger

APPROVED:

Dr. Andrea Richa
Director, School of Computing X _____

Angela Ellsworth
Second Committee Member, School of Art X _____

ACCEPTED:

Dean, Barrett, the Honors College X _____

Abstract

This paper is a supplement to our interactive algorithmic art generator project which can be found at ***weiverlyroe.github.io/waverlyplace****.*

For this thesis, we demonstrate how with certain input we can algorithmically generate art, specifically a playable random maze with exactly one solution. We explore interactive and algorithmic art and how our mazes are a form of both. Through examining several maze generation algorithms, we show that an ideal representation of a single-solution maze, called a *perfect* maze, is a spanning tree of a planar graph. The final algorithm is a re-imagining of Kruskal's *Minimum Spanning Tree Algorithm* with these adjustments: (1) potential edges are ordered randomly rather than sorted and (2) certain edges are forced in the maze in order for the wall structure to display the player's text input. Lastly, we discuss improvements which could be made and features which we plan to add to the project in the future.

"...because you are a sweet-smelling diamond architecture
that does not know why it grows."

The Poem That Passed the Turing Test
by Zackary Scholl's poetry generator

Contents

Chapter 1

Introduction

If I am painting a mural and I invite you to add your painted hand print to the wall, will you too be an artist of the mural? Or maybe you are part of the artwork? If I write a program to generate a poem, who is the poet? The program? Or perhaps one may argue that I am the poet and the program is the poem? Is the generated text even poetry? In this thesis, we explore these concepts of interactive and algorithmic art and demonstrate how our project of a customizable maze generator is an example of both. Maze generation itself is an age-old concept and many algorithms have already been created. We will examine several of these existing algorithms and share how we implemented our own adaptation of one of them through our final project to display our unique maze style.

Motivation and Objectives

There is something truly fascinating about organized chaos, anything which at first glance seems anarchic and hectic but has some underlying order. This is precisely what draws me to mazes, with their seemingly random walls and diverging paths, but with one clear goal and core structure. I am a maze artist who has been designing mazes for 9 years. It was in the 7th grade that I realized I could channel my enthrallment onto paper and create mazes of my own at which to marvel and solve, specifically *perfect* mazes, or mazes with precisely one solution.

Through drawing dozens of perfect mazes in my lifetime, I have come to develop different styles, discover basic maze theory, and establish my own method for maze design.

Most of my mazes are first sketched out by hand with pencil on paper, then typically translated pixel by pixel or line by line into a digital format on our chosen image editor. The digital translation process is very intricate for both major types of mazes: (1) tessellated mazes which require even proportions and therefore pixel by pixel translation, and (2) free-form mazes which even after being scanned, need to be meticulously traced to ensure that each line is the same width and shade throughout.

Even without these translation methods, designing a maze from start to finish requires a lot of time and laborious line drawing. As I took more and more programming courses, it grew more and more feasible and desirable to write a program to generate mazes for me, and in the Spring 2014 semester that is precisely what I did. For CSE 205 *Object-Oriented Programming and Data Structures*, I invented an algorithm to generate a pseudo-random perfect maze. My knowledge of data structures at the time was limited and so therefore was my algorithm. The mazes that my algorithm produced tended to have vertical-biased walls and were a subset of mazes with its particular structure (i.e. there exist mazes we could draw within the parameters that my algorithm could never produce). As exciting as it was to invent and implement a working maze generation algorithm, I was not satisfied and I pledged to discover a better one.

The objectives of our project were to make good on that promise to myself, to find and implement the best maze generation algorithm with our artistic style, and to make this implementation interesting and available for people to easily interact with. This thesis demonstrates our journey to accomplishing these objectives.

Originality

We are not the first to write a maze generator and put it up on the Internet. *mazegenerator.net*, for example, has been generating mazes very similar to our vision since 2006. It offers many of the customizations that we offer, as well as many more. What makes our mazes unique is

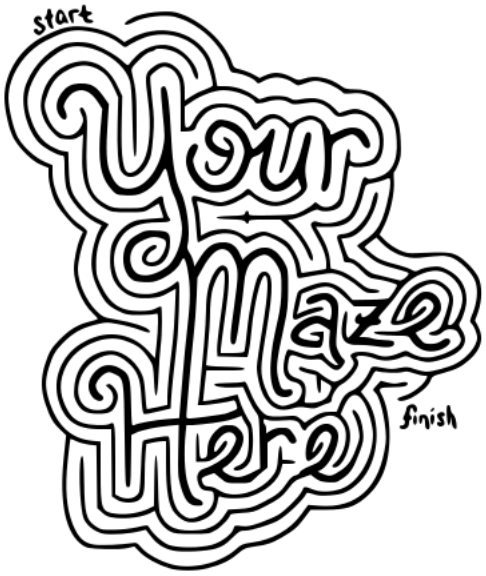

Figure 1.1: *Your Maze Here* by Waverly Roeger (2014)

the very special text field that allows the player to input a string of (almost) any characters to literally shape the walls of the maze. *Figure 1.1* shows one of my hand-drawn mazes from 2014 which is similar to what we envision our ultimate maze generator to produce if the player enters "Your Maze Here". The current version of our project is limited in acceptable input but does succeed in producing mazes shaped by that input, something we believe which has not been done before.

Chapter 2

Interactive and Algorithmic Art

While art has a rather broad definition, at its most basic level, "art is form and content". Form involves the elements of art (line, shape, form, space, texture, value, color), the principles of design (balance, contrast, emphasis, proportion), and materials. Content involves the artist's intent, the resulting composition, and the individual's reactions to these[7]. Mazes, like all games and puzzles, must be art under this definition. Mazes are a "network of paths and hedges designed as a puzzle through which one has to find a way"[10]; the creator utilizes lines, shape, space, contrast, and proportion through a chosen medium (in our case, a web page) and intends to perplex the player who presumably attempts to find their way from the start to the end.

Interactive Art

A work of art becomes *interactive art*, "when audience participation is an integral part of the artwork." Art which is interactive is distinguished by its reliance on the behavior of the audience. How is the audience considered in relation to the physical artwork? We look to Jack Burnham, an influential writer concerned with a systems theory perspective on art, who stated in 1969 that everything "which processes art data... are components of the work of art," so correspondingly the participating audience members are not considered artists, but part of the

artwork itself. As such, the artist "will typically see the physical artwork and the participating audience together as a unified [art] system." The physical artwork and the human audience must come together in order for the piece to be complete[6].

Figure 2.1: *Whose Line Is It Anyway?*, a popular TV show featuring improv (2013)[14]

One prevalent form of interactive art is a performance art known as *improv* (an abbreviation of *improvisation*). Improv is a "form of live theatre in which the plot, characters and dialogue of a game, scene or story are made up in the moment," often using "a suggestion from the audience"[17]. The improv performances dynamically change based on an audience's contributions. The performances depend on and therefore include the audience themselves as part of the improvisations.

Games and puzzles are a form of interactive art because the audience, whom we call the *player*, attempts to win these games and solve these puzzles, satisfying the intent of the pieces' creators. Mazes are a form of interactive art for the same reason, and our project furthermore exemplifies interactivity through asking for the player's input on the actual design of the maze before it is created. The maze product relies on the player's choice of number of rows, number of columns, and text input to be displayed in the shape of the maze walls.

Algorithmic Art

On the other hand, *algorithmic art* does not require anything from the audience, but rather specifies the process of creating the final piece. Art that is algorithmic is generated "using algorithms, i.e. precise defined procedure that... executes step by step," usually by a computer, but humans can carry out simple algorithms as well[5]. Any piece of "art" being generated by something other than a human artist brings up several questions including the most significant: "Can a machine originate anything?" (i.e. Who is the artist?)[11] and "Is this art?". One way to consider algorithms is as the art (or *meta-art*) themselves. To some art collectors, "the notion of collecting code or algorithms from an institutional standpoint was likened to collecting conceptual art and performance pieces"[13]. While the algorist is the artist of the algorithm, their intended result is what the algorithm produces, so by the definition of art stated above[7], the resulting piece as perceived by the audience (who usually cannot access the algorithm) is the art made by the algorist.

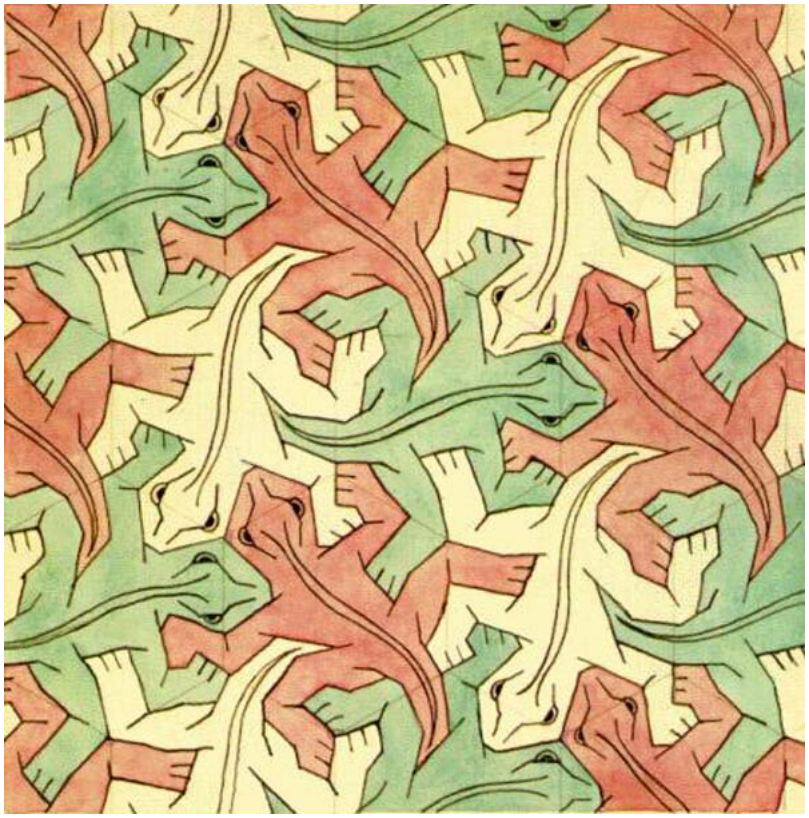

Figure 2.2: *Lizard (No. 25)* by M. C. Escher (1939)[3]

One well-known type of algorithmic art is artistic tiling, like that of popular contemporary tile

artist M. C. Escher. While he was "lacking in formal mathematical training, [and]... [w]ithout the use of computers he invented and applied what can only be called algorithms in the service of art." Escher used tiling, which is "nothing less than the application of abstract systems to decorate specific surfaces."[8] Without the use of mathematics and algorithms, it would be impossible to produce such perfectly tessellating shapes or any kind of tile art we might see around us.

Since our project uses an algorithm to generate the mazes seen by the player, it is most certainly a form of algorithmic art, one which takes in the player's input which can be anything within the set constraints, adds an element of randomness in placing the paths, and generates a visual representation of the maze for the player to subsequently interact with.

Chapter 3

The Theory of Mazes

While we mentioned the definition of maze in the last chapter, we should specify what type of mazes we are actually generating in our project. Mazes can be classified in many ways[16]including:

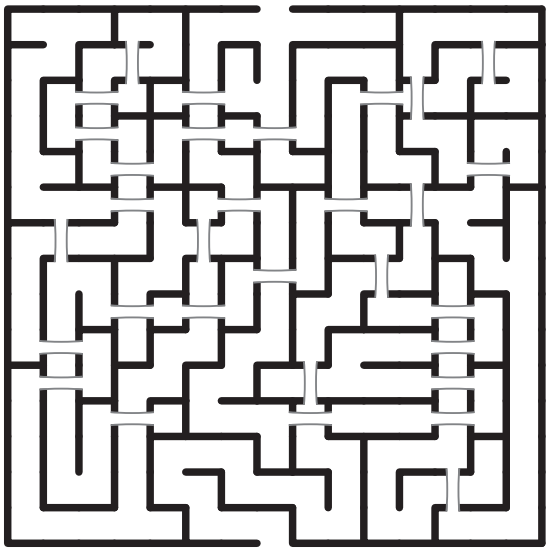

Figure 3.1: Weave maze by Dr Gareth Moore (2013)[12]

1. *Dimension*: how many dimensions the maze environment covers
    - 2D
    - 3D
    - higher dimensions
    - weave (2.5D, or 2D with bridges)

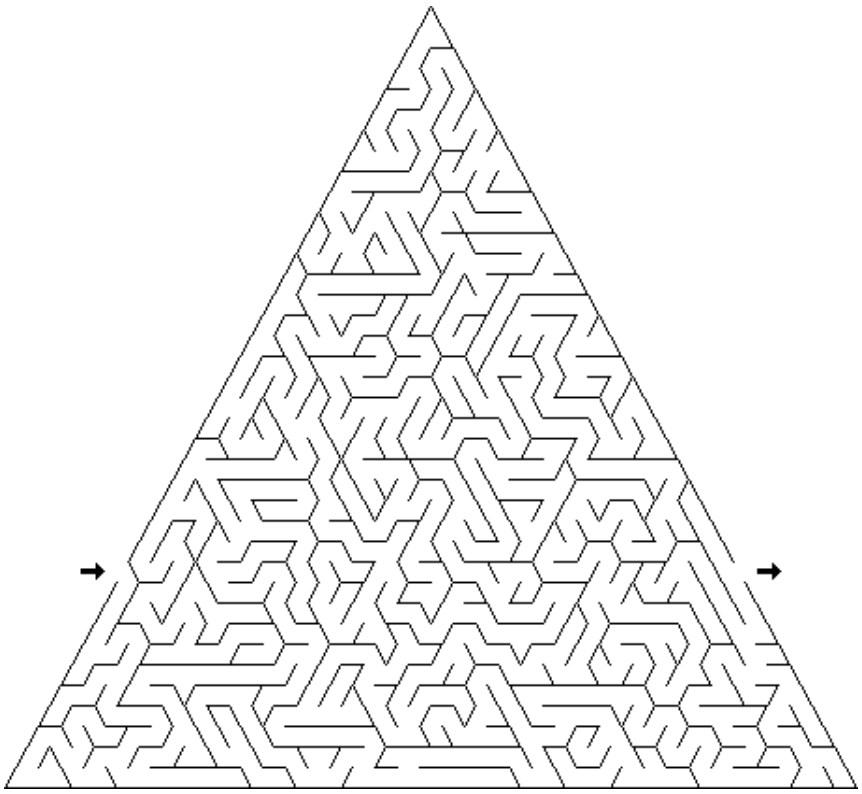

Figure 3.2: Triangular maze by Walter D. Pullen (2015)[15]

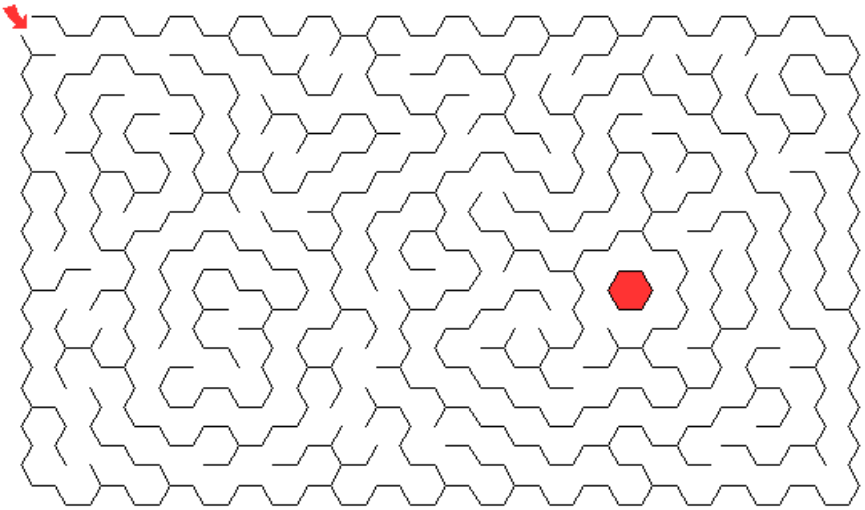

Figure 3.3: Hexagonal maze by David King (2001)[9]

2. *Tessellation*: the shape of the individual cells which compose the maze
    - orthogonal (a standard rectangular grid)
    - triangular
    - hexagonal
    - amorphous (no consistent tessellation)

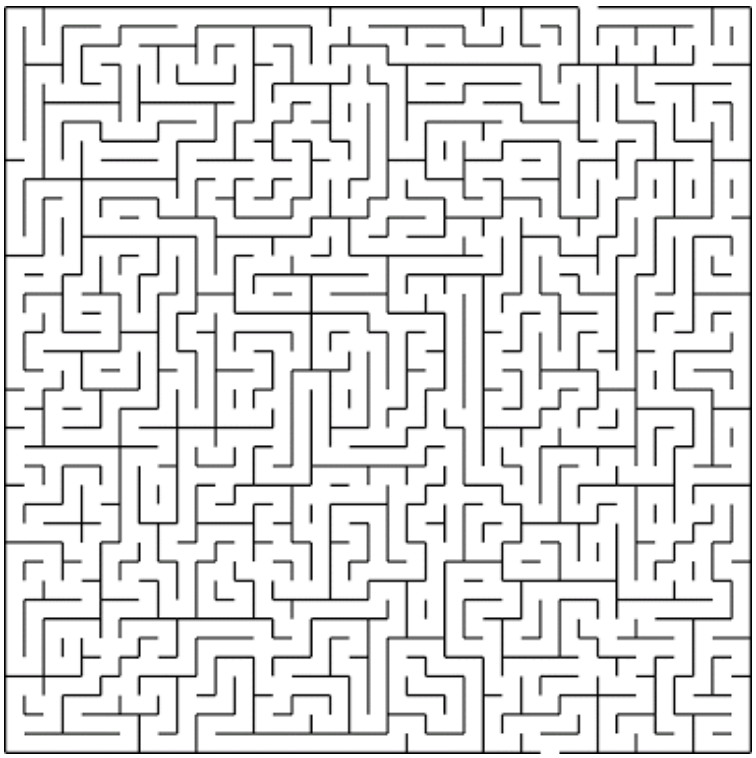

Figure 3.4: Braid maze by Wayne Walter Berry (2009)[4]

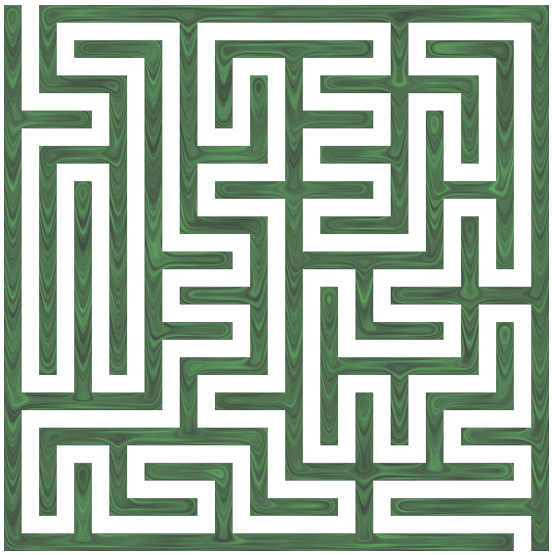

Figure 3.5: Unicursal maze by AlanStanislav (2016)[1]

3. *Routing*: the types of passages throughout the maze

   - perfect (no loops, closed circuits, or inaccessible areas)
   - braid (no dead ends, the complexity coming from "passages that coil around and run back into each other")
   - unicursal (no junctions, also called a *labyrinth*)

Our mazes are strictly two-dimensional and currently only orthogonal, although our algorithm will translate well to any tessellation or amorphous maze once implemented. Our mazes are perfect, so "[f]rom each point, there is exactly one path to any other point," and "[t]he Maze has exactly one solution"[16].

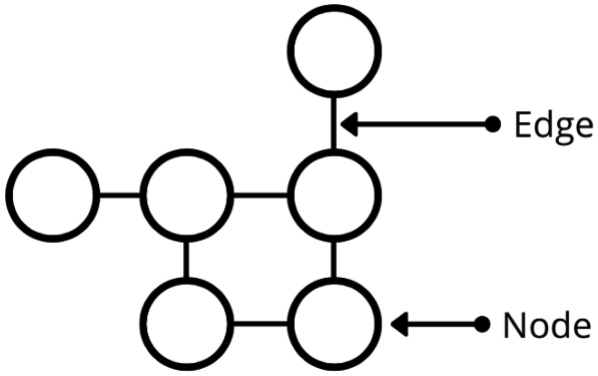

Figure 3.6: A graph with 6 nodes and 6 edges[2]

We can represent mazes using *graphs*, which are made up of individual *nodes* and *edges* between those nodes. If the maze is 2D, the graph should be planar, representable on a 2D surface without edges crossing. The path of a perfect maze is essentially a *spanning tree* of the graph containing all possible path connections (edges) between each path cell (node). In a spanning tree, (1) the nodes are all connected to each other (all cells in the maze are accessible) and (2) no two nodes are connected more than once (there is only one way to access any cell in the maze, guaranteeing only one solution). Algorithms to generate perfect mazes choose a random spanning tree of a graph, and algorithms to solve perfect mazes find a path from one node in a tree to another.

Maze Solving Algorithms

While our project is not concerned with solving the mazes it generates, the ultimate purpose of a maze is to be solved, so we will briefly explore a few maze solving algorithms[16] concerning perfect mazes:

1. *Wall Follower*: As we traverse the maze, we always keep a wall to our right (or left).
2. *Chain Algorithm*: Given that we know the exact end point, we draw a straight line from the start to the end and attempt to follow that line, using *Wall Follower* when the line is blocked by a wall. We essentially treat the maze as a chain of small mazes.
3. *Recursive Backtracker*: We attempt to move in every direction, backtracking if we have already visited that cell or we hit a dead-end. This is essentially a depth-first search of a graph.
4. *Dead-End Filler*: We scan the maze and fill in any dead-end cells and any cells that subsequently become new dead-end cells until only the solution is left.
5. *Shortest Path Finder*: We flood the maze, so that all distances from the start are filled in at the same time, remembering the flow. When the solution is reached, we trace back the flow. This is essentially a breadth-first search of a graph, and called "*Shortest Path Finder*" because if there were multiple solutions, the solution found would be a shortest one.

Since the player can attempt to solve the mazes they generate, these algorithms could be useful in the future should we decide to add a Hints or Solution feature.

Maze Generation Algorithms

Mazes can be described by their *texture*[16] as well as their classification:

1. *Bias*: A high horizontal/vertical bias indicates that horizontal/vertical passages tend to be much longer than opposite vertical/horizontal passages.
2. *Run*: The run factor is how long a straightaway tends be before reaching a turn.
3. *Elitism*: The elitism factor is the length of the solution with respect to the size of the maze.

4. *River*: More river indicates fewer but longer dead-ends, while less river indicates more short dead-ends.

As there are many ways to solve a maze, there are many ways[16] to generate a maze in the first place, each producing mazes with various textures:

1. *Recursive Backtracker*: Rather like with the solver of the same name, starting from the start node, we always travel along and choose an edge to a neighboring unconnected node if one exists, or backtrack until the current node has an unconnected neighbor whose edge to then travel along and choose. The mazes generated have a very high river factor.
2. *Prim's Algorithm*: Like with *Prim's Minimum Spanning Tree Algorithm*, but with equal weights for each edge, we start from any node and choose a random edge that connects the maze to a node not yet connected. The mazes generated have a low river factor.
3. *Kruskal's Algorithm*: Similar to *Kruskal's Minimum Spanning Tree Algorithm*, we consider each edge in a random order. We choose the edge if it connects two nodes not already connected, and discard it otherwise. The mazes generated have a low river factor, although not as low as with Prim's Algorithm.
4. *Aldous-Broder Algorithm*: Starting from any node, we travel along a random neighboring edge. If the new node is not already connected to the maze, we choose that edge. This algorithm is actually never guaranteed to terminate if we randomly never travel to a node, however it is useful in that it generates all possible mazes with equal probability and does not require extra computational storage.
5. *Growing Tree Algorithm*: We choose any node to add to the maze, then select any edge that connects a new node to the maze. Depending on how we select the edges, this could run identically to *Recursive Backtracker* or similarly to *Prim's Algorithm* or have various other textures.

This is not a comprehensive list by far; my 2014 algorithm, for example, produces a perfect maze by zigzagging row by row. Whenever I draw mazes by hand I use a combination of *Kruskal's Algorithm* and the *Growing Tree Algorithm* called the *Growing Forest Algorithm* where many distinct trees are growing at once and can only connect if they are not already connected. The algorithm chosen for any maze design depends on what textures the designer wants, how fast they want the maze to generate, and how much computational space they can use.

Chapter 4

The Final Project

Our final project of a web page with a maze game generator can be found here: *weiverly-roe.github.io/waverlyplace*.

Figure 4.1: A screenshot of our web page with no player input

User Experience

We chose to implement our maze generator on a web page because we wanted it to be easily accessible to an audience and easily sharable for anyone interested.

Figure 4.2: A partial screenshot of our web page after a player inputs 7 rows and 14 columns, then hits Make Maze

When a player first accesses the page, the default number of rows and columns is 4 and 5 respectively, and no maze is shown. 4 and 5 are arbitrary numbers that are relatively small and make a maze with aesthetically-pleasing proportions (according to our personal preference). The number of rows and columns can be adjusted by buttons labeled + and − on either side of the displayed size. Rather than text fields, buttons that increment and decrement the numbers by 1 ensure players can only enter in proper input and also dissuades players from inputting a

large number for either of these which would slow our program down slightly, interrupting the experience. When a minimum or maximum row size or column size is reached, the corresponding button disappears, indicating the limit to the player.

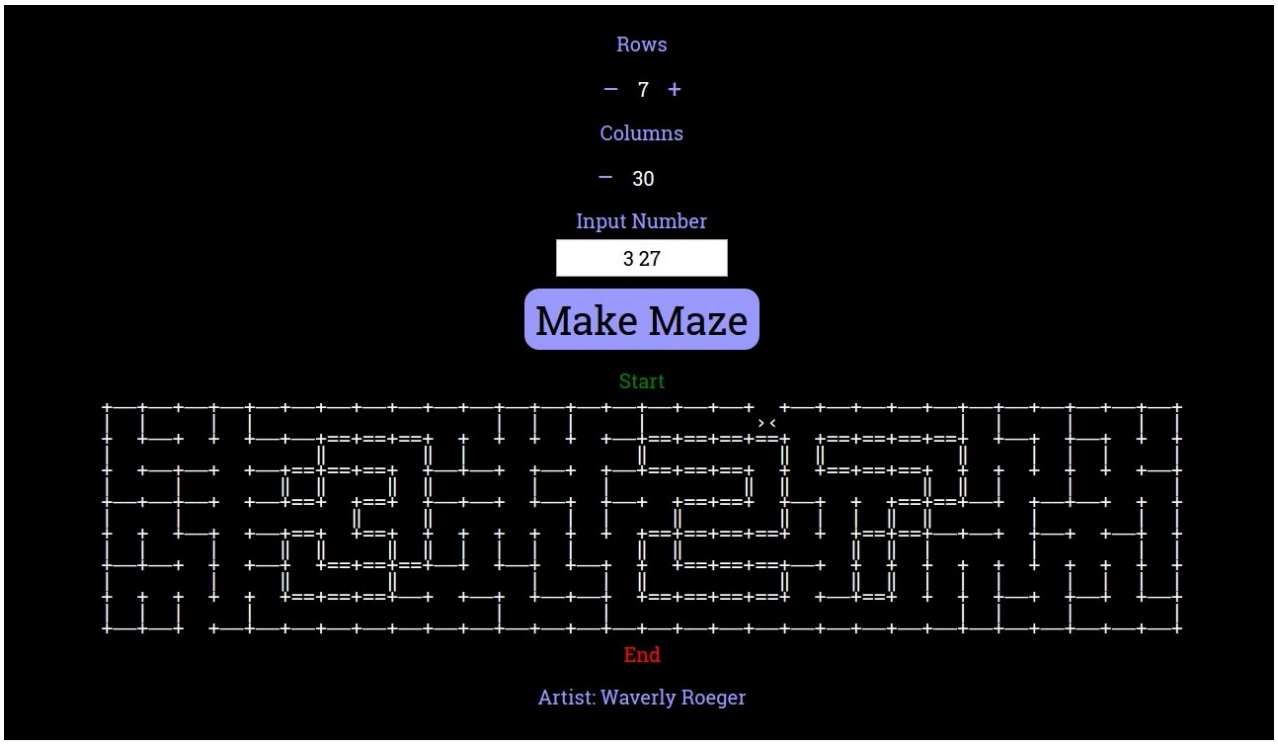

Figure 4.3: A partial screenshot of our web page after a player inputs 3 rows, 30 columns, "3 27" then hits Make Maze

Input Number is a special field where we invite any input from the player, with certain restrictions. The input will be displayed character by character in the wall structure of the maze. Currently only numerals and spaces are accepted because we have to design each character manually which takes time, but we plan to implement the capital and lowercase alphabet as well as other common symbols. A maximum of 5 characters (including spaces) is allowed because that is the maximum that fit on one line in the maze. Line breaks for longer input seem intuitive, but they are complex to implement and not a priority, so we are leaving them for future work. If the row size or column size is too small to hold the given *Input Number*, it will automatically change to the minimum row size or column size needed and display as such (see *Figure 4.3*).

The *Make Maze* button is the instigator of the whole maze generating process and the most important part of our project besides the maze itself, so it is bigger and more emphasized than

most elements on the web page. Whenever the player hits this button, a new maze with the given input is generated and displayed.

The *Start* and *End* of the maze are at the top and bottom, respectively. Going from the top to the bottom of a maze seems natural, even more so than going from left to right, especially on a web page where vertical scrolling is more prevalent than horizontal scrolling. The player icon >< always starts in the path cell right below the maze entrance.

Figure 4.4: A partial screenshot of our web page after a player successfully navigates their icon to the *End*

The player can navigate their icon using the arrow keys or WASD. Whenever they reach the *End*, the text "You Won!" is displayed and the icon disappears. Hitting *Make Maze* will generate a new maze if the player wishes to play another one.

Implementation

Our web page is on GitHub Pages for the simple reason that it was where our code was located already, and it was as easy as flipping a switch to make a public web page from our code. We do plan on registering our own domain for a website in the future.

We used HTML for the base web page, CSS for styling, and JavaScript for implementing algorithms and all interactivity. We chose to use these languages because we had never done web development before this year, we needed something simple to learn which fit our needs. In the future, we will consider more powerful tools for developing web applications with more complex features and interactions.

We chose to implement a variation of *Kruskal's Minimum Spanning Tree Algorithm* similar to *Kruskal's Algorithm* mentioned last chapter. *Kruskal's Minimum Spanning Tree Algorithm* was familiar, its implementation made sense to us, and it was easy to manipulate to suit our maze style.

The number of rows (`rows`), the number of columns (`cols`), and *Input Number* (`inputText`) are variable as they depend on player input. Depending on the length of `inputText`, we adjust `rows` and `cols` if needed.

We use two 2D-arrays of integers to represent the walls: `HWallArray` for horizontal walls, (`rows+1`) by (`cols`), and `VWallArray` for vertical walls, (`rows`) by (`cols+1`). Their integer values indicate:

- `0`: Discard this wall.
- `1`: Keep this wall.
- `2`: Keep and bold this wall, for displaying `inputText`

We first iterate through each character in `inputText` to decide which walls need to be bolded (set to `2`) to display the `inputText`. All other walls are initialized to `1`.

The outer walls of the left and right sides are always set to `1`. The walls of the top and bottom sides are also set to `1`, except for a random wall set to `0` for the Start and End openings, respectively. The inner walls, or path connections, are separately enumerated from `1` to (`(rows-1)*cols + rows*(cols-1)`), then shuffled using the *Fisher-Yates Shuffle Algorithm*, which iterates through each index and swaps it with a random index. We chose this shuffle algorithm because it generates each possible ordering (including the original ordering) with

equal probability, contributing to generating each possible maze with similar, although not truly equal, probability.

We use a 2D-array of `Point`s, called `PointsArray`, (`rows`) by (`cols`), to represent path cells. Each path cell belongs to a set of path cells to which it is connected, so each `Point` contains the parent of the set to which it belongs. Initially, each `Point`'s parent is itself as none of them are connected at the beginning. Iterating through each inner wall in its shuffled order, we need to determine whether the `Point`s on either side of it belong to the same set. `findParent(p)` returns the parent of the set of a given `Point`, and if both parents are the same, we keep the inner wall set to `1`; otherwise, we discard the inner wall by setting it to `0`, and call `joinSets(a,b)` to join the sets of each `Point`.

When the maze is generated, we display it in ASCII (text) using monospace font to line up the walls. We chose this implementation for its simplicity of display. We plan on transitioning to generating an image file in the future, so that we can be more flexible with our maze tessellations and so that players could potentially download their mazes. We store the maze row by row, so that while the player is moving their icon, only the rows which have been changed need to be updated, enabling the display time to be much quicker while the player is navigating the maze.

Chapter 5

Conclusion

Regarding our objectives for this project, we certainly succeeded in finding a better algorithm to produce all possible mazes, rather than simply a subset (like my 2014 algorithm). We succeeded in making our maze generator accessible to anyone with a computer device, Internet access, and the project link. The mazes our generator produces definitely peak interest from a general crowd (based on the reactions we have gotten from friends and passersby) even if we personally still want the mazes to be more interesting.

Through our research we were able to explore the study of art and learn a lot about how different types of art came about and are viewed, and studying Computer Science we do not get to delve into this often. This thesis has inspired us to keep consuming and creating art, even if we are surrounded as we usually are by people who do not identify as artists. We also got the opportunity to more thoroughly examine the algorithmic side of our maze passion and discover different algorithms and maze designs we could not have invented on our own, and we are re-inspired to continue to design different forms of maze puzzles.

These past several years studying Computer Science, I felt like web development was among the most basic of skills that I did not possess, and I am breathing a sigh of relief that I can now mentally check that box of things I believe I should be able to do. It is not the most exciting of processes, but it certainly is useful and I will do more work with websites in the future.

Future Work

As mentioned many times in this thesis, we have plans to better our project in the future:

- Add more acceptable symbols. This is the easiest feature to add because we simply need to design the symbols and program them in the same way as the numbers. The only thing holding us back is that manually programming the designs of each symbol is quite time-consuming.
- Add sensible line breaks if the input is too long rather than rejecting the input.
- Display the maze as an image file rather than as ASCII. Although this can certainly be done, it requires further research on our part. Accomplishing this will allow much more flexibility with the tessellations of mazes we produce and allow for the amorphous mazes I can hand-draw.
- Obtain our own domain address and research other web development languages and tools.
- Add more customization options:
  - Tessellation options:
    - Triangular
    - Hexagonal
    - Snowflake (a combination of hexagons, squares, and triangles)
    - Amorphous
  - Difficulty scale (based on the elitism, run, and river factors)
  - Colors
  - Wall thickness and path thickness
- Add a timer to watch the maze actually generate.
- Add a *Hints* button and a *Solution* button.
- Add download and sharing options, so a player can download their maze as a PDF file and share their maze through social media.

This project is certainly one which we will keep improving as our designs of mazes and algorithms and understanding of art improve.

Bibliography

[1] AlanStanislav (2016). Unicursal maze. *https://openclipart.org/detail/246740/unicursal-maze.*

[2] Amador, J. (2017). Supercharge Your Influencer Marketing Efforts with Social Graph Theory. *The Search Agency.*

[3] Art of Quotation (2015). #11 "We adore chaos because we love to...". *https://artofquotation.wordpress.com/2015/12/27/we-adore-chaos-because-we-love-to/.*

[4] Berry, W. W. (2009). A MAZE A DAY. *http://amazeaday.blogspot.com/.*

[5] Ceric, V. (2008). Algorithmic Art: Technology, Mathematics and Art. In *Information Technology Interfaces, 2008. ITI 2008. 30th International Conference on*, pages 75–82. IEEE.

[6] Edmonds, E. (2011). Interactive Art. In *Art, Research and the Creative Practitioner*, pages 18–32. Libri Publishing.

[7] Esaak, S. (2017). What is the Definition of Art? *https://www.thoughtco.com/what-is-the-definition-of-art-182707.*

[8] Galanter, P. (2003). What is Generative Art? Complexity Theory as a Context for Art Theory. In *In GA2003–6th Generative Art Conference*. Citeseer.

[9] King, D. (2001). Hexagon Mazes. *http://www.drking.org.uk/hexagons/mazes/index.html.*

[10] Maze. (2017). English Oxford Living Dictionaries. *https://en.oxforddictionaries.com/definition/maze.*

[11] McCormack, J., Bown, O., Dorin, A., McCabe, J., Monro, G., and Whitelaw, M. (2014). Ten Questions Concerning Generative Computer Art. *Leonardo*, 47(2):135–141.

[12] Moore, Dr G. (2017). Bridge maze. *http://www.anypuzzle.com/puzzles/picture/Mazes/Bridge%20maze.pdf.*

[13] O'Brien, N. (2015). Are Algorithms Conceptual Art's Next Frontier? *https://www.artsy.net/article/nicholas-o-brien-are-algorithms-conceptual-art-s-next-frontier.*

[14] Porter, R. (2013). TV ratings: 'Whose Line Is It Anyway?' starts strong, MLB All-Star Game up. *http://screenertv.com/news-features/tv-ratings-whose-line-is-it-anyway-starts-strong-mlb-all-star-game-up/.*

[15] Pullen, W. D. (2015a). Technical maze terms. *http://www.astrolog.org/labyrnth/glossary.htm.*

[16] Pullen, W. D. (2015b). Think Labyrinth: Maze Algorithms. *http://www.astrolog.org/labyrnth/algrithm.htm.*

[17] The Hideout Theatre (2017). What is Improv? *http://www.hideouttheatre.com/about/what-is-improv.*