Nerdle Final Project Report

Devin Luque, Eunseo Jo, and Murphy Warner

May 4, 2022

Abstract

At the beginning of the 2022 year, guessing games like Wordle became prevalent throughout western society. People began competing and boasting how fast they could guess a word or sound comparatively against the population. This spawned a deep yearn for knowledge about how to formulate the best guessing strategy to solve these puzzles in the shortest amount of iterations. Each of these puzzles are different in there own constructs and heuristics surrounding the "best" guess. One puzzle in particular, Nerdle, stood out to our group because it was a single character based game with little to know research about its potential answers nor a proper heuristic function to solve the puzzle efficiently. Knowing that AI and pathfinding algorithms would lend itself to developing a quick solution for this, our group sought to learn and create an algorithm that would excel at this game. After learning more about the rules of the game and the data behind Nerdle, our group was able to do just that. With our simple algorithm, we were able to boast guaranteed abilities to solve any possible answer. Moreover, we see that our algorithm performs better than the average human at this game with a majority of the answers revealed by the fourth guess. This report explains in detail about the background of Nerdle, our research, approach, design, testing, and results of our work to create a sound program that is able to perform at these margins.

1 Introduction - Eunseo

1.1 Description

The Nerdle game is similar to Wordle, a word guessing game. It is the application of mathematical concepts to Wordle. The player guesses 8 tiles including 0-9 and arithmetic operators and tries within 6 times to get the correct answer. The initial state of this problem is an unnumbered tile with no provided information. The goal of the problem is to find the given equation in 6 trials. The Nerdle game provides hints for the goal equation with three tiles in green, purple, and black after each guess. Green means the number is in the solution in the right place. The purple represents the number in the solution, but it is in a different location, and the black is not in the solution. Since the initial state

has no information, we guess the equation using “9*8-7=65” and “0+12/3=4”. These two equations contain many non-overlapping numbers and arithmetic operations so they are appropriate for obtaining many clues. Based on the input of the returned tile color as 0,1,2, the AI makes the next guess and heads toward the goal. 0 means a variable that does not exist in the equation like black, 1 means a variable that is included in the correct answer expression but is not in an appropriate location like purple, and 2 means an exact location like green. AI predicts the goal equation by considering eight tiles as an index and checking whether each index has an appropriate value. The equation must be a commutative answer and must contain a single ’=’. Moreover, in the Nerdle game, order is an important part. The equations ’10+20=30’ and ’20+10=30’ have the same mathematically meaning, but not in the Nerdle.

1.2 Why it is interesting

The Nerdle is a challenging and interesting game with simple mathematical rules. Unlike the Wordle, a player doesn’t need to know many English words and can be enjoyed by a variety of people regardless of language. In the Nerdle game, there are a large number of cases towards guessing the goal equation, since the same number of operators can be used as long as the basic laws of arithmetic are sufficient. Except for the constraint, each digit has 15 cases and a total of 15^8 cases. Even with the excluded expressions that are not suitable for equations such as continuation operators, there are still many cases. Therefore, someone may not be able to arrive at the correct answer if they do not guess the right initial number. In addition, even if you get some green tiles from the feedback, it is still not resolved when you don’t have enough clues. We use data mining in our project to extract valuable information from a large amount of data. We take out the data suitable for our purpose from numerous cases and apply them to our AI. Since our problem has limited attempts, the path-finding algorithm is appropriate for the process of finding the target answer. Artificial intelligence with a good heuristic function can derive an answer in a short time with a few attempts. AI finds guessable equations based on feedback from the game among the abundant data, and tries them to move closer to the target answer.

2 Background - Eunseo

2.1 Pathfinding

The problem of the Nerdle is a path-finding problem. The solution must store both the guess list and the possible answer list with the incentive of narrowing down the answer list by feedback until one remains. Heuristic functions minimize the number of guesses for future answers. This Heuristic

reduces the guess list and the possible answer list at every attempt. We develop the Nerdle solver using path-finding and Data Mining.

Representative path-finding algorithms include greedy search, Dijkstra algorithm, and A* algorithm. The greedy Search algorithm expands nodes that appear close to the goal without considering the path so far. On the other hand, Dijkstra's algorithm considers the distance from the starting to the current node. Combining these two, the A* algorithm compares both the distance from the node and the remaining distance. The tip from game creator Richard Mann is that on the first try, you should get clues with lots of numbers and symbols instead of duplicates. For example, "9+8*7=65". After that, keep the number on the green tile, move the number on the purple tile, and do not reuse the number on the black tile [Mar22]. The A* algorithm is best suited for our Nerdle games since we need to move towards the target state according to the returned color of the tile-based on the numbers and symbols that are first guessed. In addition, according to Patel, "If you want to find paths to one location, or the closest of several goals, use Greedy Best First Search or A*. Prefer A* in most cases." [Pat14]. To find the solution of the Nerdle moves toward one goal with 6 opportunities, so we have to find the nearest path. The A* algorithm is applied to the Nerdle guessing engine of the Nerdle solver. If the game's feedback is 2 (green), the number is fixed and other numbers remove all equations at that index. In the case of 1 (purple), the number excludes from that index but appends to the in_play list and keeps tracking. Otherwise, in the case of 0 (black) store the number in the invalid list and remove all equations containing it. Applying A* to each guess attempt, the solver minimizes the following guess list. The Nerdle solver with A* tracks all nodes.

As mentioned by Rebecca, the 2 player game approach applied to Wordle [Reb22]. Like this, the 2 player game can be applied to our Nerdle to find the optimal solution with alpha-beta pruning. Using alpha-beta pruning on black tiles reduces solution search space and improves solution search speed.

2.2 Data-sets and Data-mining

There are many equations that can be created by combining 10 numbers and 4 operators. To develop a heuristic about our dataset, we must understand what the best next guess would be that limits future guesses. When playing games like the Nerdle or Wordle, our brain can develop a guess that we know to include or cut out characters or numbers. To allow the algorithm to have the same logic, we have to train our algorithm to think similarly to us. For our Nerdle data set, there are 17,723 possible solutions we pulled from Christiano Grenco's possible solutions data set. To get the best next guess we would have to square that number equating to just around 314 million possible guesses. While this seems like a high number, our CPU's should be able to handle these simple computations rather quickly. Per BBC's report, "a clock speed of 2 gigahertz (GHz) can carry out two thousand million

(or two billion) cycles per second" [bbc]. This means that we can always estimate the next best guess at the largest data set in seconds.

In addition to simple processing, we can use data mining to speed up our performance against metrics. According to Rutgers, "Data mining is most commonly defined as the process of using computers and automation to search large sets of data for patterns and trends, turning those findings into business insights and predictions" [dat21]. We can apply this to our algorithm by logging every move or guess that has been made into a lookup table. This allows us to have pre-performed computations which reduce our computation time to a simple O(1) performance on those moves. The other benefit of this is that our algorithm only gets better with more games played because our lookup table will get further built out. With these computation methods being utilized, we can now turn to the construction of a heuristic to guide our computations.

2.3 Heuristics

Taking algorithms and data mining into consideration, we need to look at how to open and how play the next best guesses. When considering opening, many people try a random query and see what tiles turn black, yellow, and green. However, it has been proven time and time again that not having a strategy to reduce or promote that initial black-tile noise works against you. According to nerdschalk's Haripriya, "the most efficient way by default is to introduce all the characters in the initial guesses" [Har22]. By doing this, we can see what is completely disregarded, what can be moved, and what is in the correct place. The more tiles that are used (and not duplicated) the better for our initial starting point. With this, we can proceed to identify what might be our next best guess.

To truly develop a way to define the next best guess, we need to develop a heuristic that defines the measurement of the output of each guess. Our output in this case would be the colors of each square of input. For example, if we placed "11-2-9=0" into our Nerdle guess and received a yellow on 2 and 9, but black squares everywhere else, we need to identify the progress or score of that guess. What we are looking for is information theory, or measuring and identifying the points for each result to reach the fastest conclusion. Before we can develop a points system from the output, we have to label each number or operator given. In Peter Eckersley's technical analysis of Information Theory to identify a person, he defines his subject (a person in his case and a number/operator in ours) as a bit; where a 1 bit is 2 pieces of information or guesses about the next subjects: "That quantity is called entropy, and it's often measured in bits. Intuitively you can think of entropy being generalization of the number of different possibilities there are for a random variable: if there are two possibilities, there is 1 bit of entropy; if there are four possibilities, there are 2 bits of entropy, etc" [Eck11]. Eckersley goes on later to define how much entropy is needed to identify a subject. His calculations pointed to 33 bits

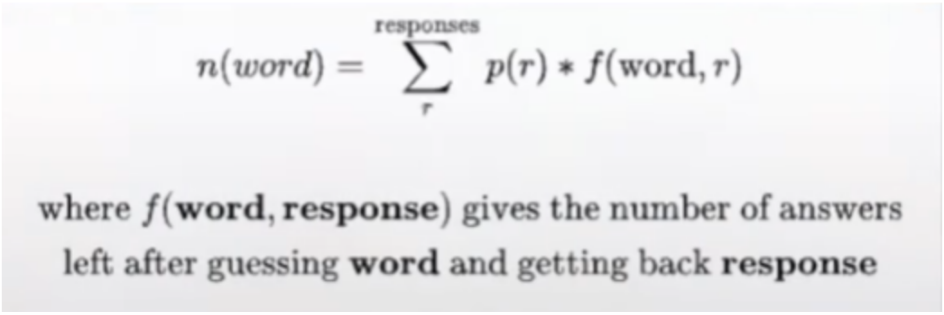

Figure 1: This the formula created by Zeb. [zeb]

of information: Bits = log2(1/World Population). For our case we would use a similar equation to calculate on average we would need to know 4.1475509267 bits of information to guess a subject.

With the knowledge of entropy in Information Theory, we understand that with each guess we want to get as many bits of information as possible. To better define our output, we will make a general points system of green squares equating to 2 points, yellow squares equalling 1, and grey squares equalling 0 similar to Tyler Glaiel's technical report for solving a similar game called Wordle [Gla22]. For each guess, to estimate the guess that creates the highest entropy, we follow a similar algorithm as Glaiel. For his report, for all 13,000 possible guesses, he took with all possible responses (3x3x3x3x3 = 273) and multiplied each of those probabilities of a response with the number of answers left after that response. After summing these together will designate the entropy of the guess inversely by showing how many average responses are left after each word guess. A PhD student Zeb, created a video explaining this heuristic and provides this helpful formula in Figure 1.

Another heuristic, we could use stems from a modified binomial distribution. According to the University of Notre Dame, "A binomial distribution gives us the probabilities associated with independent, repeated Bernoulli trials" [oND]. We can view our guessing for numbers as uniform probability or, in other words, every slot has an initial equal probability. After getting information from the initial drawing of numbers and operators, then we update the probability given the black, yellow, or green status. From there we can continue using combinations and updated binomial distributions to get this answer. This method has been used in nearly every gambling game ever invented and has even been used to count cards in Vegas. We use this as a heuristic for our Nerdle game problem.

2.4 Evaluation

To evaluate our given problem, we will need to look at a number of performance metrics based on our algorithms, heuristics, and human/machine competition. As discussed in Cooperative Path finding by David Silver, we have already familiarized ourselves with the performance metrics of algorithms [Sil]. Percentage success, average path lengths, initialization time, and computation time are some of the

best metrics to analyze our algorithm success. In addition to algorithms, we can use these metrics to evaluate our heuristics in combination with some new ones. Due to the addition of data mining in our algorithm and heuristic search, we need to factor in the addition of storage constraints for cached computations. Although we know that we will need to store a large number of computations, different algorithms and heuristics may poorly cache computations that have no reason to be stored. For example, if we have an algorithm that stores incompatible computations (i.e. 2+2=222), it leaves redundant data which could potentially slow down a machine. We can evaluate which metrics are taking up more storage space after playing a number of games and see how this affects our machine.

Another set of metrics we need to consider are ones that focus on human/machine competition of our artificial intelligence. One of the best parts about games like the Nerdle and Wordle is how we compete with one another to get the correct answer in any amount of steps. Saying, "'I got it in 4', 'Oh, well I got it in 3'" is one of the best parts and needs to be its own separate metric. Our algorithm, off of the inclusion of data mining, should get better over time and should be competing with human distributions and algorithm and heuristic distributions of itself and others.

3 Approach - Murphy

3.1 Representation

To start, our group knew that this game had not had an AI solution developed for it prior to our development. This meant there were no refined data sets, solutions, nor most importantly, proven heuristic functions that would accurately guess the 8 characters with proven theory. Due to the lack of formal research on this specific game, we only had one place to start. Data Mining. "Data mining is a process of extracting and discovering patterns in large data sets." [Cha] This portion was essential to creating our heuristic and ultimately our solving algorithm. We set out to compile every possible combination of possible guesses and answers based on the Nerdle ruleset: "Each guess is a calculation; You can use 0 1 2 3 4 5 6 7 8 9 + - * / or =; It must contain one '='; It must only have a number to the right of the '=', not another calculation; Standard order of operations applies, so calculate * and / before + and - eg. 3+2*5=13 not 25!; If the answer we're looking for is 10+20=30, then we will accept 20+10=30 too" [ner22]. Based on these restrictions, we were able to formulate 141,099 different combinations of possible guesses that would produce data for our next guess, allowing the data mining to begin.

3.2 Algorithms

From all the combinations we were able to extract how one guess would affect the next guess by the uniqueness of the combination. Yotam Gafini explains this well in his paper titled, "Automatic Wordle Solving" where he describes the Greedy-min-max algorithm and how it works against the Wordle game: "After each guess, the existing list of possible words is partitioned according to the feedback the adversary gives. Since eventually, we want to get to only one possible word, a good thing to aim for is to gradually reduce the size of the possible word list." [Gaf22] For our heuristic instead of the size of possible words, we developed a dynamic column entitled "unique", where each equation will have a counter for the number of unique characters remaining. For each iteration of the algorithm, we had a way to rank the effectiveness of the previous guess by how close it could get to the unique characters to zero, which would be our goal. The algorithm was greedy because it only looked to the next iteration and was not looking beyond. This implementation is effective and necessary because of the large amount of data our algorithm must be able to parse and edit. It would be unrealistic to look beyond the next best option, especially at the beginning when we have over 100,000 unique data frames.

Now that we had our dataset and our heuristic defined, our group outlined the goal of the algorithm. Since we knew the starting point of every game would be constant (no input or hints), we knew that our algorithm's success would be judged by how it performed against the Nerdle game. To win Nerdle, at minimum one should be able to solve the problem in 6 tries, and is said that the average human should be able to perform at this margin. Knowing this, our algorithm must be able to pass this at a high rate and average closer to a proficient rating of 3-4 guesses. Next, we had to define computationally what our algorithm would define as a success. Since the algorithm has no direct interaction with the GUI of the game, we had to define when the program could gain from each guess. We defined the output of grey to be "0", purple to be "1", and green to be "2". This way our algorithm knows the location of each digit as well as if the algorithm is successful.

4 Design - Devin

4.1 Description

The design of this experiment can be broken into four distinct sections: data mining, data cleaning, engine building, and simulation. The first function that is written "repeater" loops through every possible permutation of the symbols available to use and binds them to a string. It does so by spawning eight for loops to create every possible equation permutation and run them through the built-in python evaluation function, "eval". From there, we write out our possible results to a text

file, “a1.txt” in “all_possible_equations” and begin the second section of our experiment, data cleaning. Using “parser,” we open our first text file, “a1.txt”, and go line by line accepting and rejecting possible answers that passed the “eval” function, but may not pass Nerdle’s evaluation criteria. We did this by simply removing queries that contained invalid pairs of text such as “//” and “–”. Additionally, we seeked for queries that had multiple equal signs because Nerdle equation criteria dictates only one equation sign per guess. From there, we wrote the correct equations to another text file, “a2.txt”. We use “a2.txt” for one final data cleaning function, “builder”, which removes entries that have operation symbols after the equation symbol because Nerdle dictates that is improper for the answer. We do this by going character by character searching for the first equals sign and then if there exists an operation symbol from there on after. We can then write out the finally cleaned equations to a comma-separated value file, “a3.csv”. With that, we now can move on to the third section of our experiment, simulating.

The third section of our experiment is the manufacturing of a Nerdle guessing engine. “Solver”, the actual engine, reads “a3.csv” into a pandas dataframe with alphabetical columns for each index, a heuristic column, and a column for the equation. The heuristic column, “unique”, counts the number of unique characters remaining in the equation. For example, “9*8-7=65” has all unique characters and “000000=0” has just two unique characters, “0” and “=”. We prioritize equations that have the highest uniqueness to them because we can uncover more of the answer in a guess that way. From there we begin guessing, which comes to our first two hard-coded answers: “9*8-7=65” and “0+12/3=4”. We chose these answers because they are considered the best opening equations as they cover every possible character we can select for our answer which in turn can whittle down results, depending on (0) if the character is not in the answer at all, (1) if the character is in the answer but at a different index, and (2) if the character is in the answer and at the correct index [Har22]. In turn, any given equation will have a corresponding ternary code that tells us more about the correct answer which is written out through “result_mapper”. If the result is “22222222” then we can return, otherwise, we will need to see what went wrong and whittle down our possible answers. If we are given a “2” in our “result_mapper” result at index 0, then we can remove all other characters that exist at that index in column “a”. If we are given a “1”, then we can remove that character from that index, but append it to a list, “in_play” which keeps track of remaining possible characters. If we are given a “0”, then we will append it to a list, invalids, which will be used to remove that character from all columns since it does not exist in the equation. Once a character gets removed from a column, its entire row of characters, “unique”, and “equation” gets removed, whittling down the correct answer massively with each guess. After removal, we repeat this seven more times for every index and then proceed to confirm that all “in_play” characters are in the equation, removing ones that do not contain an “in_play” character. This is done by running a lambda function over the equation column and

looping through it to confirm every "in_play" character is accounted for. Once complete, we sort the dataframe on the column "unique" in a descending order and grab the top equation which contains the most unique equation. We continue this five more times since we are given six guesses and we continue to the simulator.

4.2 Experiments and Results

The simulator "real_simulator" tries every Nerdle answer given and records how many guesses it took our Artificial Intelligence to correctly guess the answer. This is done by scanning the file, "answers.txt", and individually running them through the engine. The engine returns the number of guesses it needed to correctly guess the answer and the simulator appended them to a master dictionary log. The statistics were overwhelmingly positive, with 63 answers guessed in three attempts, 32 answers guessed in four attempts, and 2 answers guessed in five attempts. For the 97 guesses that were given, it took 12.6 seconds to answer them all. Running all theoretical answers against the engine in the "theorectical_simulator" also produced interesting results. Besides the hard-coded guesses for getting the correct answer in one and two attempts, 37956 answers were guessed in three attempts, 69048 answers were guessed in four attempts, 25367 answers were guessed in five attempts, 7267 answers were guessed in six attempts, and 1459 answers were unable to be solved in six attempts. However, given one extra guess, all of the 1459 answers were able to be solved. Interestingly enough, all of those 1459 answers were either multiplications or division with a zero involved. Given the Nerdle layout and current answers, it appears that these answers will not happen, but we theoretically have found our next improvement on our code.

5 Analysis of Results - Murphy

Looking at our algorithm against the past 97 actual games we see that we reached our goal of beating the game in under 6 guesses. In fact, only 2 weren't after by the 4th guess and were solved the next iteration. It was not shocking to not see any guesses correctly guessed in the first two iterations as they were both hard coded to receive the most information about the problem and they would have to be exactly the input given. Knowing this, comparing iterations 3-6, we see a skew left in the dataset. We should expect this because by the 3rd iteration of guesses, the algorithm should have knowledge about all the numbers and operators within the solution because of the first two guesses, but it will not have the information about the repetition of a number or operator, nor will it have the exact positioning of the number. But as we can see, even with the uncertainty of all the positioning, the heuristic is still strong to limit the possible next guesses to better than a coinflip if it can solve in the next iteration.

Figure 2: Algorithm vs actual gameplay.

Our algorithm also processed 97 answers in 12.602 seconds, averaging the ability to solve any problem in under .13 seconds. It should be noted as well that this was run on a weaker processor than most standard laptops. The speed of this algorithm shows the speed of the greedy algorithm paired with the precompiled nature of our algorithm.

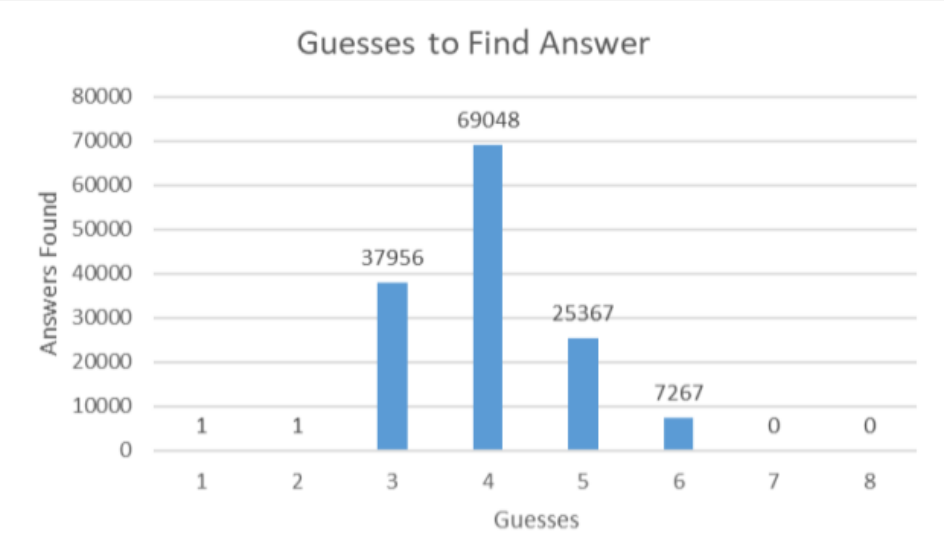

Figure 3: Algorithm vs theoretical dataset.

Looking at the algorithm vs the theoretical dataset we see less of the skewing trend, rather; we see more of a normal distribution. We estimate this is the case because in the actual game, the creators typically do not use multiple repetitions within their answers. This makes it easier on the algorithm as it does not have to waste its guess on answers that only vary with placement of numbers or operators.

Again here, we see that our algorithm is successful by the definitions of success we wrote prior to the implementation. We also see that the first two guesses, although hit because they are contained once in the dataset, do not have multiple answers found. The algorithm could be rewritten so the only the first guess is hard-coded, but if we were to do this implementation, the algorithm would be more spread out as it could fall down a greedy path prior to understanding which numbers and operators are within the equation.

6 Conclusion - Eunseo

Our project provides Nerdle game solver with added data mining in algorithm and heuristic search. It computes all possible combinations of possible guesses and answers to an exact 8 digits equation containing 10 numbers and 5 operators in 6 or fewer guesses. It then applies data mining by formulating possible combinations of guesses based on the constraints. After each guess, it reduces possible guesses based on feedback from the game. It checks the number of unique characters within a possible guess and tries the next attempt based on the effectiveness of the guess. AI keeps track of all symbols that are in the correct position, in the equation and that are not included in the equation. Our AI containing data mining, data cleaning, engine building and simulation works successfully for all 97 previous Nerdle games. For the most part, it can get answers within 4 times, only 2 games go through 5 trials. The first two equations, '9*8-7=65' and '0+12/3=4', contain all numbers and operators so we can get hints for the target equation. Except for the initial two predictions to get as many hints as possible, most of them are predictions within 2. Since we already know dataset and find to the optimal to go to the goal we use informed search. It takes 12.602 seconds to solve 97 problems and averages 0.13 seconds. This shows the speed of the greedy search algorithm. Our project focuses on a normal Nerdle game that predicts 8 tiles within 6 times, but it can also be applied to Mini Nerdle that predicts 6 tiles within 6 times.

References

[bbc] Cpu speed - the cpu and the fetch-execute cycle - ks3 computer science revision - bbc bitesize. *BBC News*.

[Cha] Soumen Chakrabarti. Data mining curriculum: A proposal. *KDD*.

[dat21] What is data mining? a beginner's guide (2022). *What Is Data Mining? A Beginner's Guide (2022)*, Dec 2021.

[Eck11] Peter Eckersley. A primer on information theory and privacy. *Electronic Frontier Foundation*, Oct 2011.

[Gaf22] Yotam Gafni. Automatic wordle solving. *Medium*, Jan 2022.

[Gla22] Tyler Glaiel. The mathematically optimal first guess in wordle. *Medium*, Jan 2022.

[Har22] Haripriya. Best nerdle start numbers and equations. *Best Nerdle Start Numbers and Equations*, Feb 2022.

[Mar22] Liam Martin. Nerdle tips and tricks: Strategies to help you solve those tricky daily maths puzzles. *Express.co*, Mar 2022.

[ner22] The daily numbers game, 2022.

[oND] University of Notre Dame. The binomial distribution.

[Pat14] Amit Patel. Introduction to the a* algorithm. *redblobgames*, May 2014.

[Reb22] Rebecca. Pathfinding algorithmsnbsp;: The four pillars. *Medium*, Feb 2022.

[Sil] David Silver. Cooperative pathfinding - association for the advancement ... *Cooperative Pathfinding*.

[zeb] Math of wordle solution.