Wordle Bot

Scott Pozder
Northeastern University
Boston, MA
pozder.sc@northeastern.edu

Katherine Poissant
Northeastern University
Boston, MA
poissant.k@northeastern.edu

Iman Moreira
Northeastern University
Boston, MA
moreira.i@northeastern.edu

ABSTRACT

Wordle is a popular guessing game in which the user tries to guess a five letter word in six chances. By viewing this as a constraint satisfaction problem, the authors were able to create a bot that performs as well as a typical human. This included the use of filtering and approaches similar to expecti-mini-max and minimum remaining variables. Future work is needed in dealing with picking a frequent word in order to have a better chance at guessing the word sooner.

CCS CONCEPTS

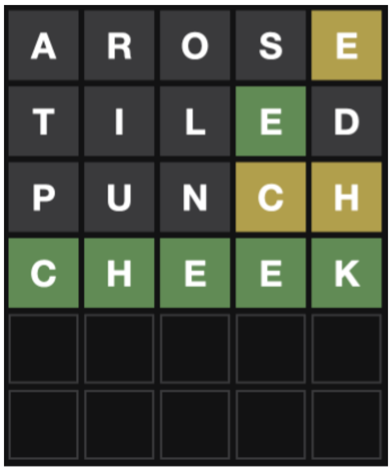
• Computing methodologies → Artificial intelligence; Constraint Satisfaction • Theory of computation → Solution concepts of game theory; Constraint Satisfaction

1. INTRODUCTION

This paper provides an overview of our process in developing a Wordle solving AI for autonomously completing games of Wordle in as few guesses as possible. Wordle is an online word game in which the goal is to guess the daily five letter word when given information about the letters each turn in a format similar to Mastermind.

The user is given six guesses to find the right answer word in order to win the game. After each guess the user is told how close their guess is to the right answer by corresponding each letter of the guess to a color: gray means not present, yellow is the right letter but wrong position, and green means that it is the correct letter and position. Players then use the information from their guesses in order to discern the correct answer word.

Figure 1: Wordle Game Board



Users can also choose to play in "hard mode", where they must use the information for each of the subsequent guesses. This means that they cannot reuse gray letters and they must keep yellow and green letters present in their guesses.

Combining the game information with linguistic rules, letter frequencies and patterns of words, it creates an updating Constraint Satisfaction Problem (CSP) that, when solved, finds the next optimal guess. Using a list of allowed guesses (12,972 words), our bot is able to make guesses, collect responses, and update our list of possible guesses using the CSP. For a human, playing Wordle is limited by their knowledge of english words, but for a computer it is simple to find the words that the answer could be. Using a dataset of all the allowed guesses in Wordle, our agent knows all of the, almost 13,000 words, that the answer can be. So while finding a word is no longer the problem, there is still an interesting process to developing an algorithm that can play the game as optimally as possible.

1.1 MOTIVATION

'Our main motivation in deciding to make a Wordle bot was the popularity surrounding the game. We started the project a few months after the creation of the game in October 2021, and its popularity picked up in early 2022. With hundreds of thousands of people playing such a new game it sparked our interest to try to solve the game using Artificial Intelligence. Having a problem that is well known and easily visualized made us think that it would be a perfect project to work on. The popularity surrounding the game motivated us to solve the game for ourselves before the optimality of the game was well known. Figuring out the best initial guess, doing an analysis of the lengths of words that made the game harder and easier and implementing linguistic rules in order to teach an agent are just a few of the problems that sparked our interest.

1.2 CHALLENGES

Our challenges from this project stem from what information we want our bot to have in order to produce the best guess that it can for any Wordle board. We tackled this problem by starting with the hardest board, or the initial guess, because it contains the least amount of information. When given no information about the guesses, the bot only has its knowledge of the English language and 5 letter words to work with.

Table 1: Letter Frequencies

Letter	Frequency
s	0.4576
e	0.4398
a	0.4109
o	0.3015
r	0.3013
i	0.2767
l	0.2401
t	0.2338
n	0.2148
u	0.1878
d	0.1772
y	0.1566
c	0.148
p	0.1453
m	0.144
h	0.1317
g	0.1189
b	0.1171
k	0.1113
w	0.0792
f	0.0763
v	0.052
z	0.0301
j	0.0223
x	0.0221
q	0.0086

This is where linguistic rules are important for teaching the agent about the English language. The information starts with the dataset of words that the bot can work with. From there it can find letter frequencies, common orderings, and commonality of words to judge the quality of potential guesses. That way the agent can make a ranking of guesses that would provide it with the most information in subsequent guesses.

In order to find the most informative guess we need to balance the likely and unlikely outcomes. For example, let's say that our agent guesses a first word that contains the letter "z". if the answer has the letter "z", then we gain a lot of information from that guess that significantly decreases the size of the search space. But, having "z" in the answer is so uncommon, that it is not worth guessing in the initial guess over a more common letter. More times than not, guessing "z" will only tell us that it is not in the word, which is not as much information. Thus, by balancing these likely and unlikely outcomes, we arrive at the combination of letters that are optimal to guess.

1.3 RESULTS OVERVIEW

Our final analysis of our algorithm included running our bot in various forms over previous Wordle answers. This analysis showed not only a great improvement to the starter code, but a superior average compared to a human as well. With each improvement to the code, our bot continued to improve and averages, at its best, 3.66 guesses per word (more than half a guess less than an average human player). This is when the bot filters words, uses a start word of “salet”, takes into account the frequency of words, and does not worry about the S_ILL problem.

2. RELATED WORKS

The first related work that we found that guided us through the project was a WordleBot competition that provided us with the initial framework for our code. This source is used to provide code for developing a Wordle bot and then registering that bot online to fight other competition participants. We used it for testing our bot locally and seeing how it solves Wordle as we add more methods to it. The code lets us see the performance of our bot by outputting to the console the guesses that we were making during the games. It also allowed for more than 6 guesses that the normal game of Wordle constrains the user to. That way we had the initial framework necessary to test and compare the bots that we would be developing.

The second related work that we found was a video by a popular mathematics youtuber called 3Blue1Brown. Around the same time that we started working on the project, he uploaded a video that outlines his approach to developing a Wordle bot. Using an information theory approach, he was able to calculate the expected amount of information that a guess would give him in order to rank the next optimal guesses. A guess would have 1 bit of information if it cut the search space in half. So, because about half of the possible guesses have an “s”, knowing whether or not the answer has an “s” provides about one bit of information. In addition, if a factor cuts the search space down to a fourth then it has 2 bits of information, to an eighth would have 3 bits of information, and so on and so forth. Basically, by gathering all of the information that one has about a Wordle board, they are then able to convert that information in a number of bits based on how much it cuts down the possible guessable search space. 3Blue1Brown was able to quantify the expected information value of a particular guess by using the entropy equation. He added several other features and ideas that were necessary for completing the Wordle Bot, but the information theory aspect is what really made it stand out. Having a way to quantify the information expected by a Wordle guess was an approach that none of us would have considered, and it gave us a lot of inspiration for how we should approach our own agent.

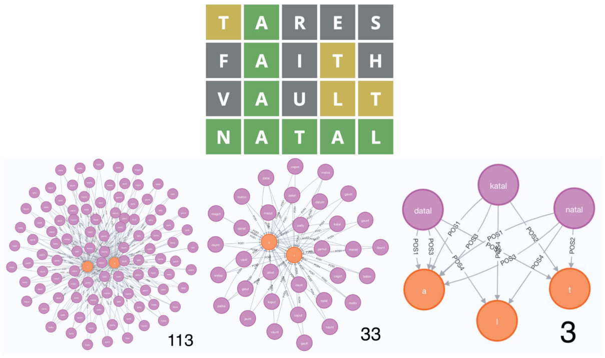
3. APPROACH

In order to consider the approach we had for this problem, we had to consider both the general approach we would use to determine which word to select next, as well as determining what would be the best starting word for our algorithm to use on each attempt. While the problems are different, we use similar algorithms for both to determine some potential start word candidates and then to determine the “next” Wordle guess based on the filtered options. For both of these, there needs to be some algorithm that will create some sort of metric on how to see which guess will both optimally filter down the word list based on any of the possible evaluation result for that word (which letters are green, yellow and gray) as well as consider what words are reasonable guesses within the english language.

Since for any game to run, we need a start word, our first attempt at getting a start word was to find all the letters with the highest frequency in the english language and tried to combine those into a word. From that, we got our first start word which was “arose”.

To start off with our first algorithm, what we implemented was a simple filter on words after each guess so that any words that were not valid after getting the evaluation from the previous word, and selecting randomly from those set of options. Even though this is a very intuitive next step, it was also a very effective first step as it was significantly more effective than the default state of randomly choosing words.

Figure 2: Displaying word filtering



From there onwards, since we were still pretty randomly choosing words from the many remaining words, there was still very little logic going into the approach, so for the next step we decided to select our next word based on some processing of our remaining words. For this approach we used a 2-d matrix of all remaining words that looked as follows:

Figure 3: Matrix approach graphic

		Potential guesses		
		w1	w2	w3
	w1	1	1	4
Answer	w2	3	1	4
	w3	5	7	1

Within this matrix, you would assume one of the axes is the "answer" and the other axes is a possible guess. From this, you would take the evaluation of the guess given that the other word was the answer, filter the remaining word list based on that evaluation, and see how many words remain after the filtering process. After all of those filtering values get assigned for the entire matrix, we use a minimax inspired approach to pick the guess whose maximum value is the lowest across all of the potential guess values in the matrix.

After out first attempt at this approach, we ended up with code that ran for far longer than any average person could wait for their code to complete due to the O(n^3) being run on 15k different words. In order to account for the runtime issue, we modified our algorithm to run at O(n^2). With this addition, the code was able to run significantly more smoothly and allowed us to continue to collect results much more effectively.

Once we had seen the results of our first approach, we also used the same algorithm to collect another start word to work with by running the min-max matrix algorithm on the entire word list and got the word "serai" to add to potential start word candidates.

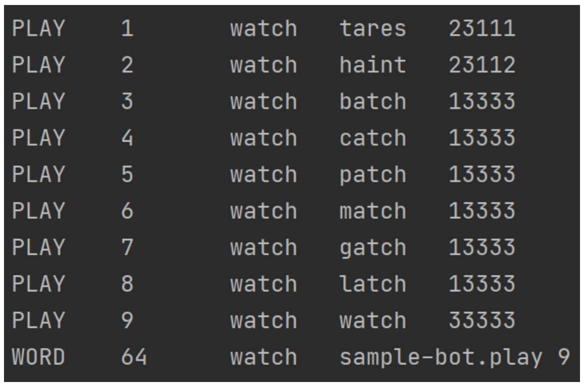
Following this min-max approach, we considered whether or not only factoring in the worst case was the best solution, since the odds of it ending up on the "worst case" isn't necessarily the most likely thing to happen. As a result, our next approach was to create the same matrix as in the min-max algorithm but instead of selecting the worst case for each word, we selected based on the average case for each word. Since the sum of all values under one of the guesses is always the total number of words available, the average value is essentially determining which words has the highest number of unique evaluations (permutations of green, yellow, and gray), since a higher number of unique evaluations means any single evaluation would have fewer words.

Similar to the other approaches, after running this average matrix approach a few times, we also ran it on the entire word list to find another potential start word. From the average case approach we found "tares" as the best average case word.

After continuing to understand what words continue to have high guess counts, we discovered there was an issue with words that have either three or four letters the same as each other. Since there are many sets of words that only differ in one letter from each other, it gave our AI a particularly hard time since it currently followed the “Hard mode” policy of wordle, which did not allow it to guess any word that didn’t line up with the previous evaluations.

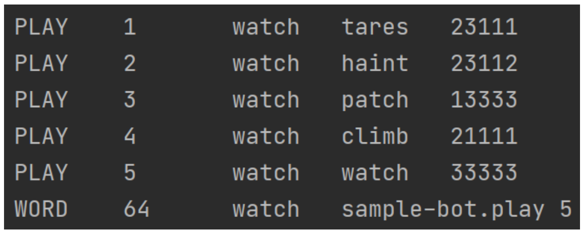
Figure 4: Resolving Similar Word Issues

Before:



```
PLAY    1       watch   tares   23111
PLAY    2       watch   haint   23112
PLAY    3       watch   batch   13333
PLAY    4       watch   catch   13333
PLAY    5       watch   patch   13333
PLAY    6       watch   match   13333
PLAY    7       watch   gatch   13333
PLAY    8       watch   latch   13333
PLAY    9       watch   watch   33333
WORD    64      watch   sample-bot.play 9
```

After:



```
PLAY    1       watch   tares   23111
PLAY    2       watch   haint   23112
PLAY    3       watch   patch   13333
PLAY    4       watch   climb   21111
PLAY    5       watch   watch   33333
WORD    64      watch   sample-bot.play 5
```

Due to this issue, it led us to decide to have our AI break “hard mode” rules. Instead of going through every possible word that is one letter apart, our AI would now take a different approach if there were more than 2 filtered words remaining after evaluating a word to have 3 or 4 green letters. When this case was hit, instead of continuing to choose from the filtered selection of words, it would select a word from the entire wordlist that had the most of the “missing letters” from our filtered selection of words. For the example in figure 4, instead of continuing to select catch after batch, it would select a word that had the most letters having c, p, m, g, l and w so we could gain more information about which remaining letters would be the best match. This approach reduced the number of guesses for these problematic words significantly as shown in figure 4.

At the end, we also decided it was important to factor in how common words were, since filtering down the “best” guesses out of the entire word list often ended up with the 3rd and 4th guess being completely unknown words. Since both our simulated wordle as well as the real wordle answers only have answers as commonly used words. Since so many of our middle game guesses would be uncommon words, it almost guaranteed that it would not be the correct answer for those guesses.

In order to correct this miscalculation, instead of purely selecting the “best” word based on the algorithm, we took a set of the best words and out of those, sorted them based on how common they were and only would select the most common words out of those as the final adjustment we made to our algorithm.

4. RESULTS

In order to analyze our bot’s performance, the bot was tested in many different forms. The code provided from botfights.ai included print statements that allowed for the code's performance to be analyzed. When the program was run, the program would print the current guess number, the answer word, the guess word, and feedback provided in the form of the numbers 1, 2 and 3 for each letter for each game iteration. A number 1 represented the gray shading in the game Wordle (the letter was not in the word). A number 2 represents the yellow shading (the letter is in the answer, but not in the spot guessed). Finally, a 3 represents the green shading (the letter is in the word and in the spot you guessed). After the word had been guessed, the program printed the game iteration number, the answer word, how many guesses that word took, and the average guesses per word so far. This information, specifically the average guesses per word at the end of the run, was used in order to analyze the performance of our bot.

Further, in order to accurately test the performance of the bot, past wordle answers were used. The provided code was updated to instead use a wordlist of all the answers words from the past 106 days. We were then able to compare all of the results to a human, as one of the group members had played the past 106 days as well and was able to view their results.

First, the original code from botfights.ai was tested in order to obtain a base test. This code provided a very weak bot that did a little bit of filtering so that the provided code would finish in a reasonable amount of time; however it was obvious this code did not perform well. As seen in Table 2, the original code guessed the word in an average of 8.556 guesses. In a game of Wordle, a player usually only has 6 guesses but this program allowed for an infinite number of guesses. This means the original code was failing to win Wordle most of the time.

After we added in filtering in order to ensure that all subsequent guesses would match the guess feedback we had received so far, our performance greatly improved. As seen in Table 2, our algorithm now only needed an average of 4.8648 guesses, which is almost half the number of guesses originally needed. Further, our code was also completing Wordle in under the 6 provided guesses most of the time. This proved our assumption that Wordle is really a constraint satisfaction problem and that treating it as such is an appropriate strategy to solving the game.

Next, we began analyzing start words and began using the start word “tares” after discovering it was the best start word with the original matrix approach. After implementing this, the average dropped further to 4.6648 guesses as shown in Table 2. Even though the average only decreased by .2 guesses, this is still a significant decrease and proved to us that picking a good start word matters.

Then, word frequency was added to the program so that only common words that had a good chance at being the actual answer were guessed. With this approach, the same words were guessed each time so each time the program was run, the average was the same. With frequency added, the average once again dropped to 3.726 guesses, as seen in Table 2. This showed the importance of factoring in the frequency when using past Wordle answers.

Finally, we attempted to solve the S_ILL problem in order to avoiding getting stuck on one letter; however, it was discovered that this approach actually increased the average number of guesses to 3.887, as seen in Table 2, showing that for our program it is actually better for the bot to run through each of the possible guesses. This was different to the results seen when working with guesses being chosen using the matrix approaches. In those cases, the average number of guesses remained about the same, but was more precise.

Table 2: Average number of guesses as new features were added

Trial #	original code	add filtering	add “tares”	add frequency	add S_ILL problem
1	8.571	4.648	4.714	3.726	3.887
2	8.724	4.886	4.438		
3	8.676	4.99	4.724		
4	8.219	4.924	4.848		
5	8.59	4.876	4.6		
Average	8.556	4.8648	4.6648		

While designing the bot, we experimented using the matrix approaches. In order to compare these approaches to the final approach of using frequency, each matrix approach was run using “tares” as the start word and including the solution to the S_ILL problem, as seen in Table 3. With the matrix algorithm using the average, the bot averaged 4.2192 guesses and with the matrix algorithm using the worst case, the average number of guesses was 4.202. This shows that these algorithms work relatively well in solving Wordle and win most of the time; however they still require more guesses than our bot that uses frequency, which only needs 3.887 guesses when using “tares” as the start word and including the solution to the S_ILL problem.

Table 3: Averages with the matrix approach using average and worst case

Trial #	Average	Worst Case
1	4.219	4.238
2	4.286	4.229
3	4.21	4.2
4	4.21	4.095
5	4.171	4.248
Average	4.2192	4.202

Now that the proper algorithm was identified, we began analyzing start words. The start words that were tested were “tares”, “salet”, “serai”, “arose”, and “qajaq” and the results are shown in Table 4. These words were run using the frequency algorithm as well as including the solution to the S_ILL problem. With an average of 3.755 guesses per word, “salet” was found to be the best start word for our final algorithm. This is compared to one of the worst start words “qajaq”, which had an average of 4.61 guesses per word. This means that with any start word, or algorithm will probably solve the Wordle on average between 3.755 guesses and 4.61 guesses. This shows that the start word does matter and can even cause you to use almost an entire extra guess per word, meaning the start word is pretty much a wasted guess; however, we did see that other popular start words such as “tares” and “arose” work decently well and still allow you to guess the word on average in under 4 guesses. This allowed us to conclude that picking a good start word for our algorithm is very important, but that there are multiple good start words that can be used that do not result in a large difference in average guesses.

Table 4: Start word averages with final algorithm

Word	Guesses
tares	3.887
salet	3.755
serai	4.19
arose	3.971
qajaq	4.61

Next, we compared the final algorithm (frequency approach with the S_ILL problem solution) with our two start words (“tares” and “salet”) to a human’s scores, as seen in Table 5. This was done using the wordlist of the past 106 days of Wordle. Katie played Wordle for all 106 days without losing Wordle, which allowed us to accurately compare the average number of guesses needed for a human to complete Wordle. Our bot averaged 3.887 guesses per word with the start word of “tares” and 3.755 guesses per word with a start word of “salet”. Both of these scores beat the human score of 4.208 guesses per word, proving that our bot can beat an average human at Wordle. Further, the margin between the human score and the score for each of the two start words is wide enough to confidently conclude that our algorithm is truly smarter than the average human and would beat most humans' average at Wordle.

Table 5: Averages from human and final algorithm

Human average	4.208
Algorithm using “tares”	3.887
Algorithm using “salet”	3.755

Next, we decided to observe the effects of word length on number of guesses. To do this, we obtained wordlists that had a list of four, five, six, and seven letter words; however we did not have a specific list of answers to use so the answers were random words in the lists. This meant we used our average matrix algorithm to analyze this data and we also used a random start word for each game. The S_ILL problem section of the code was also updated to properly match similar problems that would appear in different length words. Although there are a couple changes from our final algorithm, this still gives us a good understanding of how the length of the word affects the average. These results are shown in Table 6. As the words became longer and longer, the average consistently dropped. Looking at this we can assume that our program would continue to drop with larger words until it evens out around 3.5 guesses. This could be caused by many factors, the main one being that the code for the S_ILL problem probably plays a large impact and begins to have a smaller but more specific effect on the larger words. Another cause could be that with larger words you have a greater chance of guessing a letter that is in the word, and therefore more information. From this we can conclude that it really does matter how long the word is, although it appears as though it begins to matter less with longer and longer words.

Table 6: Results from different length words

Trial #	4 letters	5 letters	6 letters	7 letters
1	5.89	4.72	4.07	3.69
2	5.69	4.74	4.16	3.73
3	5.9	4.85	4.2	3.7
4	5.95	4.78	4.18	3.68
5	5.84	4.77	4.11	3.74
Average	5.854	4.772	4.144	3.708

Finally, we decided to compare the score distribution of our best algorithm to human score distributions on the last 106 days of Wordle answers. As seen in Figure 5, with the results provided by the Wordle site, the human won 100% of games, with almost half of the games taking 4 guesses. This was compared to the score distribution of our bot using the frequency approach, "salet" as the start word, and without the S_ILL problem solution, as seen in Figure 6. The only problem with our bot in this run is that it did lose one game, where it took the bot 7 tries to guess the word "foyer"; however, our bot clearly has a better average. With a much more even split between 3 and 4 guesses for a word as well as winning 12 games in only 2 guesses (compared to the human only accomplishing this feat once), our bot clearly outperforms. The average of this bot is also significantly better with the bot averaging 3.66 guesses per word and the human averaging 4.208 guesses per word. Final conclusions we can draw from these results are that our bot still has the bell curve shape for guess distributions that is seen in the human players distribution, but with a much lower average. This allows our bot's guess distribution to appear to be the guess distribution of a very intelligent human Wordle player's distribution.

Figure 5: Human distribution over 106 words

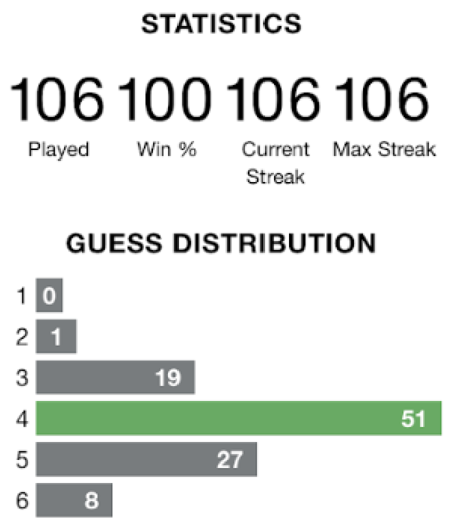
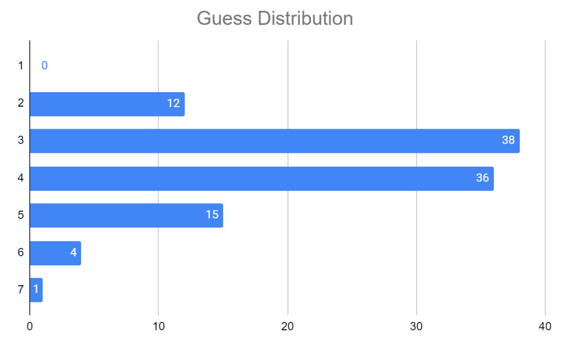


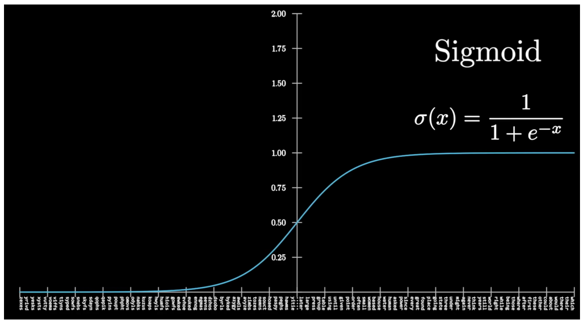
Figure 6: Best possible results using our algorithm



5. FUTURE WORKS

Some future works we intend to look into in order to further optimize our Wordle AI is to consider a way to more accurately balance the usage of common words with the best filtering words. Even though we were able to factor it in by selecting some values, it didn't feel wholly accurate to choose the most common word. For example, the word "which" is the most frequently used word in the English language with a .002061 frequency, but a word like "braid", which has a frequency of .000002, is just as likely to appear as a wordle word but has a very drastically different frequency. So rather than picking the most frequent word out of the set of "best picks" from the algorithm, it would be best to find a way similar to using the sigmoid function to define a more clear line of "common words", "maybe common words" and "unused words" and factor in that value to the value used in the matrix algorithm.

Figure 7: Sigmoid Function Applied to Word Frequencies



Along with this, we'd like to incorporate a better filtering approach to the matrix algorithm that also factors in the disparity in the words filtered across different evaluations. For example, a word that has similar remaining words across all evaluations versus a word with a wider range of how their different evaluations filter their words.

ACKNOWLEDGEMENTS

We would like to thank Stacy Marsella for his guidance in approaching this problem as well as refining solutions.

Similarly, we would like to thank Nutchanon Yongsatianchot for his help in refining solutions.

Our bot is built off the starter code provided by botfights.ai that allows other programmers to challenge themselves in creating their own bot. The code used from botfights.ai can be found at:

https://github.com/botfights/botfights-sdk

The division of work for this project was:

Katherine Poissant: algorithm for filtering; work with letter frequency within dataset; algorithm for solution to S_ILL problem; played and collected human Wordle results; abstract, CCS concepts, results overview, and results report section.

Iman Moreira: Min-max matrix algorithm, performance improvements, average case matrix algorithm, finding and incorporating word frequencies into the algorithm, and evaluation and future works sections.

Scott Pozder: Wordle Bot ideation and research, Neo4j graph representation of the CSP, finding word datasets for X-long words, introduction, motivation, challenges and related works sections

REFERENCES

[1] 3Blue1Brown. "Solving Wordle Using Information Theory." YouTube, 6 Feb. 2022, https://www.youtube.com/watch?v=v68zYyaEmEA.
[2] 3Blue1Brown. "Oh, wait, actually the best Wordle opener is not "crane"…" YouTube, 13 Feb. 2022, https://www.youtube.com/watch?v=v68zYyaEmEA.
[3] Unzueta, Diego. "Information Theory Applied to Wordle." Towards Data Science, 14 Feb. 2022, https://towardsdatascience.com/information-theory-applied-to-wordle-b63b34a6538e
[4] Lubin, Noa. "Wordle - What is Statistically the Best Word to Start the Game With?" Medium, 28 Jan. 2022, https://medium.com/mlearning-ai/wordle-what-is-statistically-the-best-word-to-start-the-game-with-a05e6a330c13
[5] DeMichele, Patrick. "A bot that solves Wordle with the frequency of English words in mind." Medium, 17 Jan. 2022, https://medium.com/@patd998/a-bot-that-solves-wordles-with-the-frequency-of-english-words-in-mind-1b6581c5a8d
[6] Hunger, Michael. "Discover AuraDB Free: Week 19 - The Wordle Graph." Neo4j, 14 Feb. 2022, https://neo4j.com/developer-blog/discover-auradb-free-week-19-the-wordle-graph/

[7] Peattie, Alex. “Establishing the minimum number of guesses needed to (always) win Wordle.” 25 Jan. 2022, https://alexpeattie.com/blog/establishing-minimum-guesses-wordle/
[8] Smith, Barry. “How to Guess Well in Wordle.” Towards Data Science, 26 Jan. 2022, https://towardsdatascience.com/how-to-guess-well-in-wordle-d21167aae444