Developing a Wordle Solver and Analyzer using Artificial Intelligence

Emily Costa[1], Nilay Barde[2], Priyank Shelat[3], Pranav Bansal[4]
costa.em@northeastern.edu | barde.n@northeastern.edu | shelat.p@northeastern.edu | bansal.pran@northeastern.edu

***Abstract* – Wordle is a game where the player has to guess a random five letter word in six guesses or less. After each guess the player receives feedback on each letter of the word that was guessed: a correct letter in the correct position returns green, a correct letter in the wrong position returns yellow, and an incorrect letter returns gray. This project remakes the current game and uses different artificial intelligence algorithms to beat the game.**

***Keywords* – Artificial Intelligence, Wordle**

I. INTRODUCTION

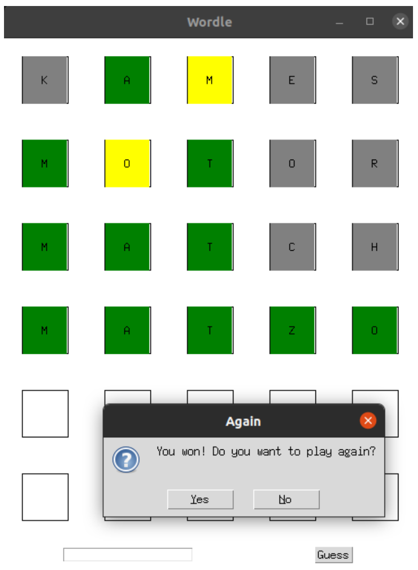
In this project, we created a game similar to Wordle, a popular game recently acquired by the New York Times. This game has captivated many of us, our friends, and family. We were inspired to do this because we wanted to know how well you can beat the game. We believe that several of the algorithms and techniques that we learned throughout this course would be applicable to beating this game in an efficient way.
The general rules are as follows:

1. There are six chances to guess the word.
2. The words you enter as guesses have to be in the word list, which consists of the most common five letter English words.
3. A correct letter in the correct position turns green.
4. A correct letter in the wrong position turns yellow.
5. An incorrect letter with no correct position turns gray.
6. Letters can occur in the word more than one time. The game won't tell you if a letter you guess is in the secret word twice, unless you guess a word with the two letters in it.

In our implementation of the game, we allow for the flexibility of changing some of these rules in the game. We are able to change the number of guesses a player has, we are able to change the entire word list, and we are able to change the size of the words.

II. USE CASE

The first use case for our project is the ability to solve the Wordle game in the most optimal amount of guesses. However, our project can do more than just solving the current Wordle game. We also allow for the flexibility to make game configuration changes (larger words, more guesses, etc) and have our model still be able to come up with an optimal solution. Our Wordle solver can also be applied in solving future problems and games that involve searching for a specific set of characters or trying to find a specific word from a word list.



You won! Do you want to play again?

Fig. 1. The Graphical User Interface of our Wordle implementation.

III. IMPLEMENTATION

A. Game Implementation

In constructing our project, we first needed to create our implementation of Wordle such that it replicated the original game by default and allows us to modify certain parameters. To do this, we create a class that outlines an instance of the "game". We then can initiate this class with parameters such as number of guesses allowed, length of the word, and the word list from which words of the correct length are pulled from. This Game class will essentially serve as the framework in which we create the game, track the moves made, give feedback to the algorithms and GUI, and determine the results of the game.

When the game class is instantiated, we generate a random "word of the day" which, in the real game of Wordle, is a selected word which people must deduce that day. This is something that is set automatically when a new instance of a Game is created by randomly selecting from our word list. The game has a few main functions for the GUI interface and AI algorithm in use: one to retrieve previous guesses, get the game status (WIN, LOSE, or IN PROGRESS), make a guess, get the word list, and restart the game. The algorithm will have the necessary information that a human would to make guesses with the GUI and play the full game

B. GUI Implementation

After completing the game implementation, we next built a GUI similar to the actual Wordle game to allow the user to play the game. We used a Tk GUI toolkit called Tkinter to build the game in Python. The game board starts in a grid of empty boxes. The number of rows indicates the number of guesses allowed until you lose and the columns indicate the length of the word that is being guessed. The user will be able to type into an input box and press the guess button to make guesses of the word. The GUI will then show the guess with each letter filling a box and a color indicating the information about the letter based on the actual word. If a user tries to input an invalid word, the GUI will respond with an error message using Tkinter message boxes informing the user of the invalid input. At the end of the game if the user had lost the game the GUI will tell the user what the word was and then ask if the user would like to play again. If the user had guessed the word correctly, they will just be asked if they would like to play the game again. Playing the game again will reset the GUI to a blank grid once again. This scenario is displayed in Fig. 1.

IV. ALGORITHMS

A. "Human" Baseline

For our baseline algorithm, we implemented a methodology typically done by a human. We randomly select words based on the feedback received by our most recent guess. We pick a word with letters we know exist in it that have shown as YELLOW or ones with letters in a certain position that have shown as GREEN. We generally avoid letters that are GREY but not always because humans choose imperfectly and sometimes reselect letters they know are not in the word.

B. Aggregated Frequency

This algorithm selects a word as the guess by calculating the frequency of every character in the remaining list of words after each guess is made. Once the character frequency is calculated, the algorithm then iterates again over the remaining words to give each word a score which is basically the aggregated character frequency for every word. Note that the algorithm avoids duplicates and if a character occurs more than once in a word then it is only counted once. The final guess is the word with maximum aggregated frequency. Intuition behind this approach is that the word that contains most occurring characters will help in eliminating a larger set of words after every guess and this way the space of possible words will shrink at a higher rate than if the words were picked randomly. It can be seen as a goal based agent mentioned in Russell and Norvig's AI book Chapter 2, section 2,4 [5]. This agent takes actions based on the goal information. Let's assume that the agent's goal is to minimize the size of remaining word list to 1 and in order to do that it is using the character frequency values as information. Implementation is inspired by Ido Frizler's Blog [2].

C. Max Entropy

This approach is based on information theory. A word is guessed by selecting a word with the highest entropy from the list of remaining words. The aim is to reduce the number of possible words matching the existing pattern. The smaller the space of possible words is after a guess, the greater the information gain will be. Whenever a word is guessed there can be several possible patterns that can result from it. For example, all the squares can be green, or we can have 3 yellow and 2 grays or 1 gray and 4 greens and so on. Since we have 3 colors for every square and there are a total of 5 squares in the standard game, therefore the total number of possible patterns is 3^5=243. Some might argue that 4 greens and 1 yellow is not a possible pattern so 5 such patterns must be deducted from 243 and the actual number of possible patterns is 238 which is true. We have taken this into account while implementing the approach.

To better understand the algorithm, we first need to know what is information in this context. The standard unit of information is a bit. The total number of 5 letter words that are valid is approximately 13000 (the list is available with our code). If we have an observation that can eliminate half of the words as the possible solution i.e. the probability of a word being the solution from the remaining words is 1/2 then we can say that it contains 1 bit of information. Similarly, if it can eliminate 3/4 of the words which basically means that the space has been divided into 4 parts and the probability is 1/4 then it will have 2 bits of information. Conversely we can also say that if an observation has n bits of information then it means that the probability of its occurrence is $1/2^n$. This gives the following formula for information:

$$I = \log_2(1/p) = -\log_2(p)$$

Here p is the probability of a word from solution space being the correct guess. We will get different amounts of information from different patterns when a word is guessed. In order to calculate the expected information that we will get from a specific word we can calculate the weighted sum of all the information that we will get from different patterns using the formula:

$$E[I] = \sum_x p(x).\log_2(1/p(x))$$

Here x is a possible pattern that may be produced when a word is guessed which can have around 238 different values. This expected value is called the Entropy for the observation. It tells us about the amount of information, on average, that we will receive by choosing that word as the guess. The more information that we get from a word, the smaller will be the solution space, i.e. the number of remaining words will be lower. To achieve our goal of reducing the number of remaining words we need to maximize the entropy i.e. find the word with maximum entropy. The algorithm works iteratively by calculating entropy values for all the guesses and picks the one with the highest. Once a guess is made, then the same steps are taken again in which the entropy values for that restricted set of words is calculated. This algorithm is modeled after the Utility Based Agent described in Russell and Norvig's AI book chapter 2, section 2.4.5 [5]. In such agents the performance measure of an action depends on how happy that action will make the agent. In our case, the agent will be happy when the size of list of remaining words is as small as possible. The Expected Information or the Entropy function is the utility function of the agent and it tries to maximize the expected utility. We can assume that the list of valid 5 letter words is the knowledge base for this agent and the response from the environment helps it in eliminating the words that can not be the goal state.

The implementation for the algorithm has been inspired by Grant Sanderson's Youtube video on solving wordle using AI

[6] and Diego Unzueta's Medium Blog [7].

D. Genetic

This algorithm is based on genetics and how genes combine and mutate. Our approach comes from the *Genetic Algorithms Introduction* file from lecture set two in class [4]. There are five main parts to this method - initialization, evaluation, selection, crossover, and mutation. We initialize by picking the first two guesses randomly. From here we loop the rest of the steps. Each word is given a fitness based on how many correct letters are guessed. For each grey letter, 1 fitness point is removed. For each yellow letter, 1 fitness point is added. For each green letter, 2 fitness points are added. Each new guess word has its fitness calculated. Then we select the 2 fittest, randomly combine them, and mutate each letter with some small probability.

There are some important technical work-arounds and assumptions that our implementation uses and abuses.

- While selecting two words, there is a check_selections function which checks all ordered combinations of the selections to ensure they can create at least one new word in the remaining word list. If no such word can be created then a random word is chosen. We used this approach primarily for speed and after some testing was performed. Alternatively, we could have:
    1. turned up the mutation threshold in general and removed the check_selections function or
    2. turned up the mutation threshold only if no new words can be created i.e. check_selections is False, or
    3. let it not converge and return one of the previous picks or
    4. selected another pair of fit words.
- The remaining word list is consistently used so this cannot be considered a pure genetic algorithm. It incorporates a process of elimination to make it more competitive with regard to speed. The alternative approach would be to use the full word list every time. This could possibly give a greater accuracy and win rate but it would likely slow the algorithm down substantially as there are many searches performed on the remaining words list. Using the full word list might also create new problems like guessing previous words again although this is fairly trivial.
- The default implementation does allow for no crossover, meaning either of the 2 selected words may be used again however there are other constraints placed, such as on the while-loop, that would force it to generate a new guess if this occurred.
- Lastly, the threshold for mutating each letter is set to 0.1. This was chosen arbitrarily and the idea was that for any threshold probability, p, there would be p^n probability of the word being mutated where n is the length of the word.

E. Q Learning

We also attempted to use Q-learning to decide optimal policy. Our state is the previously guessed word. The reward for a correct word is 10 and for an incorrect word is -1. Our actions will be picking which word to guess next. This gives a Q-value matrix of size n^2 where n is the size of the word list. We used the Q-learning update policy as seen in [1] which is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \lambda(max_{a'}(Q(s',a')) - Q(s,a)))$$

Our approach was to learn a policy once then quickly make guesses by reusing the learned policy. We also sped up the process and reduced the Q-matrix size by only calculating the values of the state-action pairs that were seen during training rather than for all pairs. Since a single policy with this idea of state would align with a single answer word, the idea is to update the policy with multiple target words. This means the policy will not converge, but our goal is to find the words that have the highest possible utility for the most answers. Most of the reinforcement learning(RL) agents we learned about rely on the problem being a Markov decision process(MDP) however this formulation is not and thus this could have ended very badly. Other approaches would be to have a policy for each answer word and narrow down the policy list with each guess however, this is more efficiently done with a tree. Another idea is to follow suit from Andrew Ho's Wordle solver [3] and define the state as, for each letter, track whether it's been attempted, and if it has, which spaces it's still possible for (i.e. yes, maybe, no for each of the 5 spots). His method is also constrained by one Q matrix per answer. Our method is similar to other methods that find the expected values of letters. The difference is the agent will find the values via reinforcement and it will be for words rather than letters.

V. LESSONS LEARNED

A. Algorithms Performance in Original Wordle

In this first section, we explore indicators of performance when the algorithm plays the original Wordle game. This includes the following game and evaluation setup:

(a) The length of the word is 5 letters.

(b) The max number of guesses is 6.

(c) The original Wordle list is used as the only valid words to guess and be the answer which is approximately 13,000 words long.

(d) The metrics measured to indicate performance are win rate, run time, letter accuracy, perfect letter accuracy, number of guesses, and number of guesses if won. The following are the definitions of the metrics identified and used for Fig. 2.

  (i) **Win rate** is the only metric that is not the average of all games played (trials) but rather the ratio of wins to loses; loses being the word not guessed once the maximum number of guesses is done.

  (ii) **Run time** is the average seconds taken for the algorithm to complete one game.

  (iii) **Letter accuracy** is the ratio of yellow and green letters to grey letters once final guess is given while **perfect letter accuracy** is the ratio of green letters to all other non-green letters in the final guess.

  (iv) **Number of guesses** is the average number of guesses the algorithm takes to guess the correct

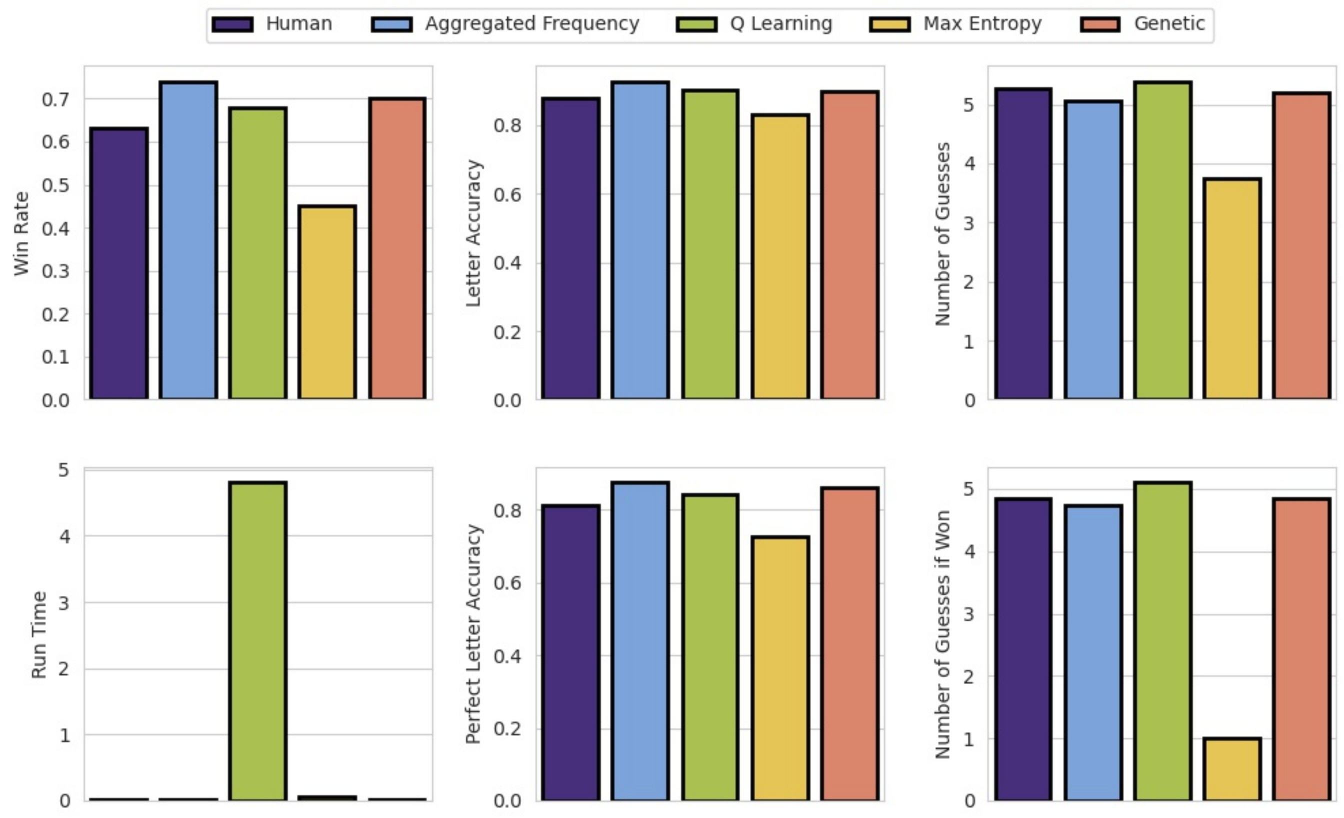


Fig. 2. Key indicators of the algorithms performance as games of Wordle are completed.

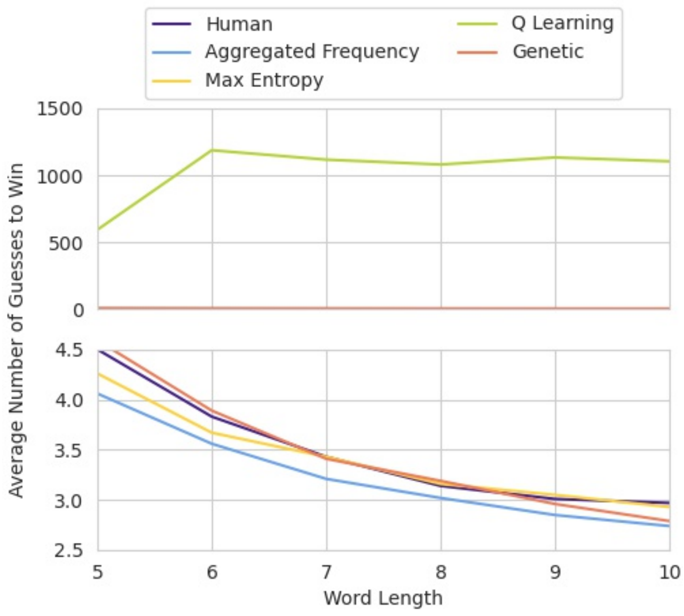


Fig. 3. The top figure is the average number of guesses an algorithm needs to guess the correct word. Because Q Learning took exponentially more guesses, the bottom figure is added to compare the other algorithms at proper scale.

word while the **number of guess if won** is the average number of guesses taken if the algorithm ends of winning the game.

(e) The number of games played, trials, for each algorithm is set to 100.

With this configuration, we use the algorithms to run trials and determine the key indicators of their performance. In Fig. 2 we plot those metrics using a barplot. From those plots, we make the following observations:

1. **Max Entropy either performs really well or really poorly.** Though this algorithm has the lowest win rate, it also takes the least amount of guesses if the algorithm wins that round. This accuracy variation is unique to this algorithm.
2. **Aggregated Frequency is the top performer.** This algorithm performs well across the board while maintaining one of the lowest average run times so it is determined to be efficient. It also has the highest win rate.
3. **The Genetic algorithm performs second best.** The metrics for Genetic algorithm performance are in line with Aggregated Frequency except slightly lower. It is a good alternative solver and still much better than human performance.
4. **The "Human" performs decently.** This is consistent with what most of us experience in a given game of Wordle when solving it ourselves. We win most games after 4-5 guesses.
5. **Q Learning has suboptimal run time performance.** This makes Q Learning not a viable option for solving the game as it takes significantly more time to run than all other algorithms to the extent that we are barely able to visual all other average run times for the algorithms. Additionally, we note that Q Learning is only 3rd best in terms of win rate and takes the most number of guesses

whether the game is won or lost. Hence, Q Learning is the least ideal to apply to the game of Wordle.

After applying the algorithms to a traditional game of Wordle, another question comes to mind; what happens when we change the foundational rules to Wordle?

B. Guess Performance in Varying Word Length

Next, we push test the performance of the algorithms as the traditional rules in Wordle are modified. This is shown in the following game and evaluation setup:

(a) The length of the word is varied and increased to up to 10 letters from the original 5 letters.

(b) The max number of guesses is unlimited so the algorithm can guess until a solution is found.

(c) The allowed word list is always 1750 words in length. The words themselves will be different as the word length changes, but the length being the same eliminates a factor that may effect the performance of the algorithms.

(d) The metric used to indicate performance is the average number of guesses taken to find the correct word as the win rates will technically be 100% for all algorithms given that they are permitted infinite guesses.

(e) The number of games played, trials, for each algorithm is set to 100.

In Fig. 3, we plot the average number of guesses for each algorithm as the word length is increased. On the top plot, we observe that Q Learning is the only algorithm to performance worse as the word length increases. Additionally, we observe that Q Learning takes an exceptionally higher number of guesses on average so we added an additional plot that scales much smaller on the bottom in order to compare the other algorithmic performance. In the bottom plot, we observe that all other algorithms improve as the word length increases.

Though Max Entropy performs second best with words of length 5, it does not see performance improvements as dramatic as Genetic, which overcomes Max Entropy as second best. Also, as previously found, Max Entropy indicates more variation in its performance from game to game. Hence, especially as the word length increases, the Genetic algorithm is a better alternative. However, overall, Aggregated Frequency still performs best as the average number of guesses taken to solve is the least thorough all the experiments.

VI. CONCLUSION

In this paper, we designed a game similar to Wordle in which algorithms can solve the game and the original rules can be modified. We discussed and implemented 5 methods/algorithms including a baseline human-like method, aggregated frequency, max entropy, Q learning, and a genetic algorithm. We then determined the performance of these methods by using them to play games while aggregating key indicators of performance such as win rate and average number of guesses. We show that it is possible to outperform the typical human at Wordle if certain algorithms are used and that the algorithms are versatile in their ability to complete a game if the rules are modified.

Our implementation and code is open-source and available at: https://github.com/emilyjcosta5/AI-Final-Project

VII. ACKNOWLEDGMENT

Thank you to Professor Prasad Saripalli and the TA's for helping throughout the course and assisting us in our assignments and guidance on our project.

REFERENCES

[1] Shweta Bhatt. Explaining reinforcement learning: Active vs passive, Apr 2019.

[2] Ido Frizler. The science behind wordle, Jan 2022.

[3] Andrew Ho. Wordle solving with deep reinforcement learning, Jan 2022.

[4] Vijini Mallawaarachchi. Introduction to genetic algorithms - including example code, Mar 2020.

[5] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* CreateSpace Independent Publishing Platform, 2016.

[6] Grant Sanderson. Solving wordle using information theory, Feb 2022.

[7] Diego Unzueta. Information theory applied to wordle, Feb 2022.