Attempting to Solve Wordle Through the Use of Artificial Intelligence Processes With Varying Reward Functions

Joseph Kwilman, Michael Ruberto

December 15, 2022

Abstract:

For our project, we attempted to create an artificial intelligence agent that can solve the popular word guessing game Wordle. We determined that the best way to solve this problem would be through the use of Markov Decision Processes (MDPs) as well as Hidden Markov Models (HMMs) due to the random, non-deterministic nature of a guessing game. As a way to compare our results, we decided to construct agents with two different ways of evaluating guesses to see which would perform better. One MDP based agent used rewards based on how few possible solutions there were remaining, while the other was inspired by forward filtering, a HMM technique, coupled with information on how many individual letters it could either place or rule out. After constructing the agents and running tests, it was determined that the "Letter Based" Agent attained a higher score on average, but the "Fewest Words" Agent had a higher win rate. Further, both agents were able to perform comparably to humans, tying or exceeding the best average score by country.

Introduction

For our project, we aimed to create an agent that could optimally solve the game Wordle. Wordle is a guessing game where each day, a new five-letter english word is randomly selected. Players are then given six attempts to guess the word. After each attempt, every letter in the guessed word is marked either green (meaning this letter is in the correct location), yellow (meaning this letter is in the word, but in a different location), or gray (meaning this letter is not in the word). Each of the player's guesses must also be a valid word.

Wordle exploded in popularity at the beginning of this year, going from 90 daily players in November 2021 to over 2,000,000 by January 8th, 2022[1]. The game became a cultural phenomenon and was ultimately acquired by the New York Times. We believed this project would be an interesting endeavor due to this immense popularity. With so many people familiar with the game, there would no doubt be people interested in finding optimal strategies or ways to improve their scores.

Wordle is based on a finite dictionary of five-letter English words. Some words have been removed from this dictionary for various reasons (e.g. profanity) to keep the game friendly for all players. In total, there are 12,972 valid words a player can enter, with a subset of 2,315 of those words that could be a possible solution[2,3]. These dictionaries are available to our agents. For one individual run of the game, the input would be a word drawn randomly from that set of 2,315, and the output would be the series of guesses the agent made in order to arrive at the solution.

Background

Our first step for designing these agents was to decide on an algorithm to use to formulate our Wordle game. We first settled on using Markov Decision Processes (MDPs), which would allow us to represent Wordle by defining states, actions, transitions, and, importantly, rewards. While we initially considered using some sort of search algorithm, as many similar problems are formulated, we ultimately decided MDPs would be better due to the stochastic elements inherent in a guessing game. In our implementation of Wordle, guessing a certain word is considered to be an action from the agent. As such, actions in this setup are non-deterministic. If you were to start a game from the initial state and guess the word "CRANE," for instance, your game would not always end up in the same state each time, as the transition to the next state would be dependent on the randomly selected target word which is unknown to the agent. Therefore, an MDP was considered the best choice due to its tolerance of randomness[4].

We decided on two main implementations of agents to use so that we may compare and contrast the results. Our first idea was to make a reward function that prioritizes states where there are fewer possible valid solutions based on the information we've already received. Our second idea was to make something similar to a forward filtering Hidden Markov Model that handled a probability distribution of potential solutions, updating after observing the response pattern to a guess. There are other possible functions one could come up with, but these seemed the most intuitive to us.

Related Work

The biggest area for further research as far as our project is concerned is finding a way to speed up the "Fewest Words" Agent. As is, it takes so long to run that we were unable to run our tests on the full set of possible Wordle solutions, which contains about 2300 words. For our testing purposes, we randomly sampled 100 words from the list and ran the agents on those. Even still, one game took that agent about 13 minutes to complete. We believe that the results we obtained are still indicative of the results we'd see on the full dataset, however we were unable to test that for certain on our machines. Either a more efficient algorithm or a more powerful machine may be needed.

Outside of that, another possible area for further research would be to try and devise new reward functions to see how they compete with the "Fewest Words" Agent and the "Letter Based" Agent. As we said in the introduction, those two reward schemes felt the most intuitive to us, but further research may uncover a creative solution.

Another potential feature we could've considered is the static, normalized distribution of each letter's usage amongst the solution words. Some letters are much more common than others. For example, a guess such as "SALET" is more likely to reveal additional information than a guess such as "JAZZY". Although not enough for an agent itself, this idea could influence the reward/assessment of our other agents.

Approach

Fewest Words Agent:

For the "Fewest Words" Agent, our general approach for our agents was to formulate a Wordle game as an MDP. Parts of our code were adapted from UC Berkeley's CS 188 Pacman Project 6[5]. This code was then tailored to fit our specific needs. Our actions were simply guessing a word. Gamma was set to 0.9. The state was defined by two pieces of information: the number of guesses remaining, and a list of all the words the target could possibly be based on the known information. As far as the reward function is concerned, it is simply calculated by subtracting the number of possible words remaining in this state from the total number of words in the initial list. This way, the initial state would be worth 0, and the states with only one possible word remaining would be worth the most. The hardest aspect to figure out was how to model possible next states and their respective transition probabilities. Eventually, we determined that for every action, there would be at most 243 possible states you could result in. The reason for this is that when you make a guess, there are five letters, and each letter is assigned one of three colors. As such, there are $3^5 = 243$ possible patterns of colors that can result from each guess, with each pattern leading to a distinct list of possible remaining answers, and therefore a distinct state. Whenever an action needs to be chosen, all 243 of these possible lists of remaining answers are calculated. The probabilities of each state are then calculated by summing up the lengths of each list, then dividing each list's length by that total sum, effectively normalizing each state to a value between 0 and 1.

With the MDP fully defined, making a guess became fairly simple. When the agent needs to pick a word, it computes the optimal action based on the Q-Values of each possible state-action pair, defined by the Bellman Equation $Q(s, a) = R(s) + \gamma \sum_{s'} T(s, a, s')V(s)$. The action associated with the highest Q-Value is then selected.

Letter Based Agent:

The "Letter Based" agent was inspired by the forward filtering algorithm in Hidden Markov Models. It starts with an even distribution of probabilities for each word. With this, it makes a random guess to start the game. The agent then observes the response pattern and updates the probability distribution based on the new information. At each observation, the agent updates three data structures:

- Bad Letters: a set of letters we know are not in the word
- Known Letters: a set of letters that we know are in the word

- Confirmed Letters: an array of length 5. If a letter has been confirmed at a position, it will be added to the array accordingly.

From this information, the agent updates the probability distribution from the equation:

P(*word* | *all observations*) ∝ P*(new observation* | *word*) P(*word* | *previous observations*)

The interesting part of this algorithm comes from how exactly we calculate P*(new observation* | *word*). If the word contained a "bad letter", it's probability went to 0. For the other words, we assigned weights to two different features: the number of indices that have confirmed letters and whether the word had any known letters. After the new probability distribution was calculated, we normalized the distribution before making the next guess. Each guess was whichever word the agent thought the solution was most likely to be.

Results

For our experiments, we put the "Fewest Words" Agent and the "Letter Based" Agent against one another. We made each agent play 100 games of Wordle with the intention of comparing both their success rates (what percentage of games the agents successfully find the right answer) and their average scores (how many guesses it takes the agents to find the right answer on average). As mentioned previously, due to the amount of time it takes the "Fewest Words" Agent to run, we had to randomly sample 100 words from the full Wordle list rather than using the full list. Although the "Letter Based" Agent runs faster, we wanted to apply this restriction to both agents to make it a fair test.

After running the experiments, we found that the Fewest Words agent had a 100% success rate under these conditions, meaning that it always managed to find the correct answer. The Letter Based agent, on the other hand, had a win percentage around 76%. Furthermore, we determined that the Letter Based agent had a better average score, despite not winning as many. "Fewest Words" got the correct answer in 3.72 words on average, while "Letter Based" averaged 3.1 words. We theorize that the Fewest Words agent won more often as it planned out its guesses more thoroughly, almost brute-forcing a safer solution. Reaching a state with fewer potential solutions allowed it to prioritize narrowing in on an answer, while the Letter Based agent acted more greedily. It consistently chose what it thought to be the best answer, which might not always narrow the potential solutions by a considerable degree. This allowed the Letter Based agent to run considerably faster, as it was only updating the state information after each observation instead of predicting the entire game.

Overall, we were very satisfied with these results. For comparison, the average American takes 3.92 guesses to solve a game of Wordle. Worldwide, the best country at Wordle is Sweden, whose people can solve the game in an average of 3.72 guesses[6]. This ties our "Fewest Words" Agent and is beaten by our "Letter Based" Agent.

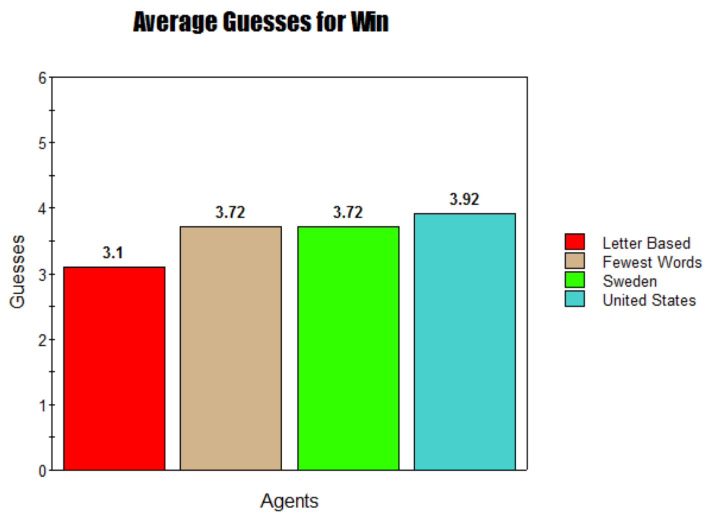

Figure 1: Average Guesses per Win

Additionally, we were curious about the effect our random sampling was having on our results. To test this, we ran our "Letter Based" model multiple times, each time increasing the amount of words we used to see how our success rate and scores would be affected. Though we couldn't do this with our "Fewest Words" agent due to runtime, we still felt this would be a worthwhile exercise. Figure 1 shows the average guesses for a win plotted against the sample size of words. Figure 2 shows the win percentage plotted against the sample size. For both data sets, we found the relationship to be logarithmic. It makes sense that the agent would do worse having to keep track of more words, but we found it interesting that the amount its effectiveness dropped slowed with each additional word.

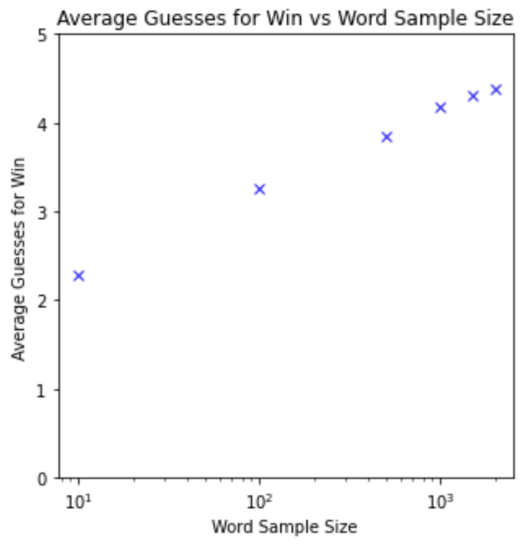

Figure 2: Average Guesses vs. Sample Size

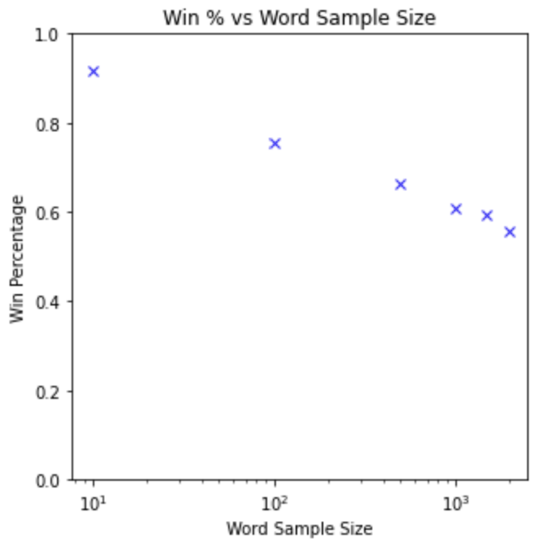

Figure 3: Win% vs. Sample Size

Conclusion

In conclusion, we determined that both of our agents are capable of playing Wordle reasonably well, although both have their strengths and weaknesses. For our “Fewest Words” Agent, which employs an MDP, we generally found a higher win rate due to its safer, brute-force style evaluations. The drawbacks to this were that the model played the game very slowly due to the amount of computations it needed to make, and its conservative style led to a slightly worse average score. The HMM inspired “Letter Based” Agent, on the other hand, generally scored better and ran quicker, although this also led to a slightly lower win rate. Despite these small differences, both agents performed well relative to human players, which was reassuring to see. Our tests were certainly impacted by the slow speed at which our MDP runs as we had to randomly sample a subset of words to make the agent run faster. While our evidence shows that these metrics are likely to decrease logarithmically as we add more and more possible words to the dataset, we are still satisfied that even in the worst case scenario, our agents will still perform sufficiently.

Work Breakdown

We tried to split the work required for this project as evenly as possible. The initial implementation of the Wordle game in Python was done together with both group members contributing. This made it so we had a fully functional game. From there, both group members split up, with Michael handling the “Fewest Words” Agent and Joseph taking the “Letter Based” Agent. Overall, we are both pleased with the final product and each member’s individual contributions.

References

1. “Wordle Creator Overwhelmed by Global Success of Hit Puzzle.” *The Guardian*, Guardian News and Media, 11 Jan. 2022, https://www.theguardian.com/games/2022/jan/11/wordle-creator-overwhelmed-by-global-success-of-hit-puzzle.
2. Dracos, valid-wordle-words, (2021), GitHub repository, https://gist.github.com/dracos/dd0668f281e685bad51479e5acaadb93
3. “Wordle Words List.” *Word Unscrambler*, https://www.wordunscrambler.net/word-list/wordle-word-list.
4. Jagtap, Rohan. “Understanding Markov Decision Process (MDP).” *Medium*, Medium, 10 Feb. 2021, https://rojagtap.medium.com/understanding-the-markov-decision-process-mdp-8f838510f150.
5. University of California Berkeley. “Project 6: Reinforcement Learning.” *Project 6 - Reinforcement Learning - CS 188: Introduction to Artificial Intelligence, Spring 2022*, https://inst.eecs.berkeley.edu/~cs188/sp22/project6/#mdps.

6. “Where in the World Is the Best at Solving Wordle?” *Where in the World Is the Best at Solving Wordle? - Word Tips*, https://word.tips/wordle-wizards/.