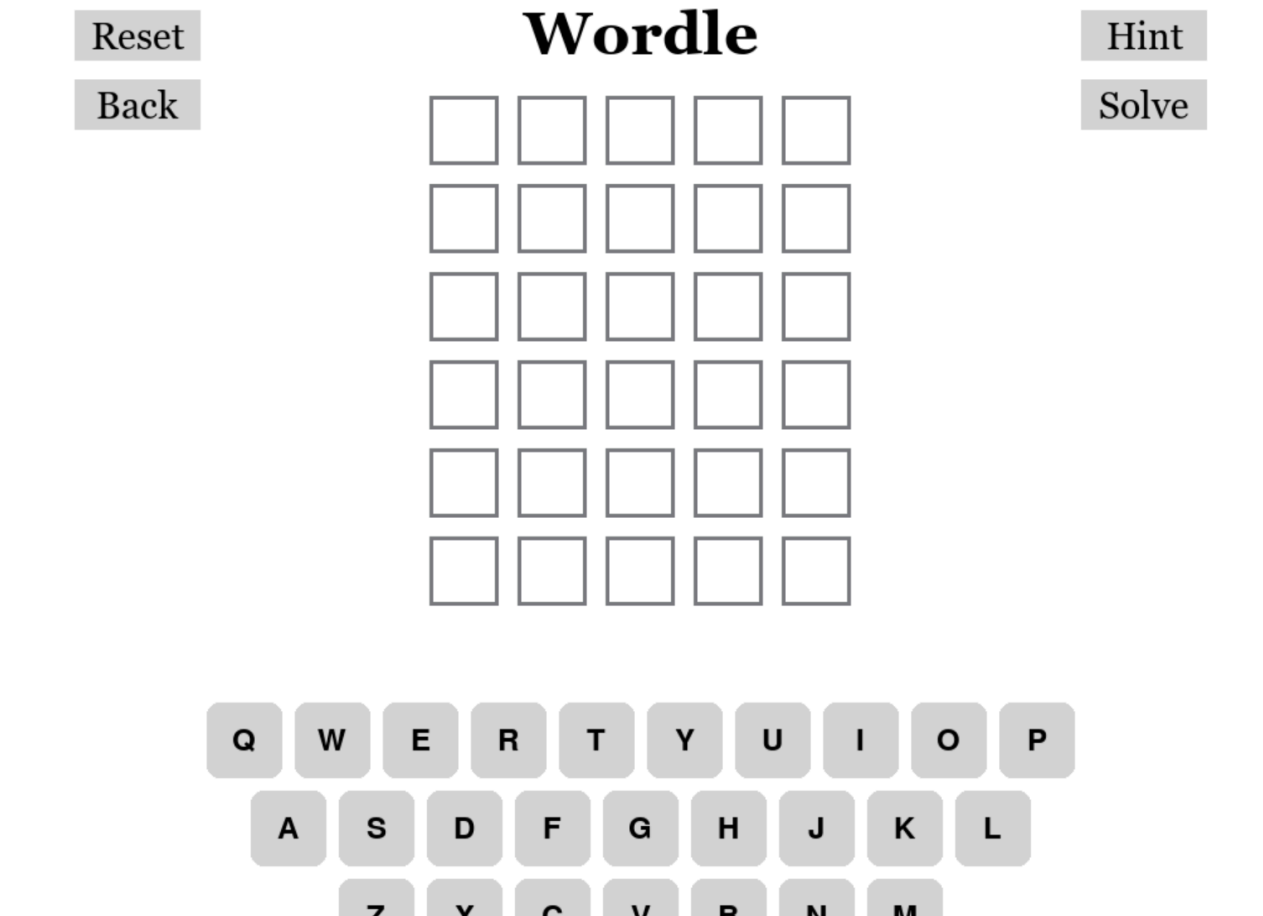


Wordle Solver

Group Members: Artur Kumik, Ethan Cecil, Gillian Rodgers

Introduction

For the capstone project, we created an all inclusive Wordle Solver and Wordle game. We created the game using PyGame, which is a set of open source Python modules that are used for creating video games. This allowed us to have a Graphical User Interface (GUI) in which the user can play the Wordle game, with integrated buttons that call the solver function.



The two functionalities that we implemented for the solver were the “Solve” button and the “Hint” button. Both buttons call the same function in our solver file, they just use the information differently. The “Solve” button calls the solver function and enters the entire word as

the next guess for the user. The “Hint” button calls the solver function as well, but it only inputs 1 letter of the word at a time. This allows the user to fill in the blanks, and input whatever they want, while using our algorithm to give them suggestions of common letters that will get them further in the game.

Solver

A large part of this project was creating the strategy and coding the algorithm for the Wordle solver itself. The overall idea was for us to create a function that takes in the current state of the Wordle, and outputs the word that includes the most common letters, while also following the Wordle rules. The rules that we are referring to include not using any letters that we know are gray and always reusing letters that are yellow and green. Some people tend to not follow this rule, but we thought it would be more realistic, and also give the best chance for us to guess clues in 2-3 guesses on average.

All of the functions that are required for this solver are included in the solver.py file. The main function within this file is `generate_guess`. This is the function that will be called from the game loop, and it takes in three inputs: the current guess, the guess array (which we will go over shortly), and a boolean value called `first`. The current guess is the word that was last inputted into the Wordle game, and if it is the first guess, an empty string is inputted. The guess array is our version of passing in the feedback that the Wordle game gives us. It is a string that is made up of either ‘g’, ‘y’, or ‘r’. If the letter in the current guess at the corresponding index is gray, there is a ‘g’ in the guess array. Similarly, if the letter is yellow, a ‘y’ is put in the place and if it is green an ‘r’ is put in the spot. Lastly, the boolean value “first” lets us see whether this is the first time that the solver is being run in the current game. This tells us whether we need to generate the csv from scratch or not.

Our `generate_guess` function first checks to see whether the “first” variable is true, and if it is, we call the `generate_csv` function. This function takes no inputs, and all it does is read in the list of words, and pass them as an array to our `rank_words` function. We have previously calculated the weight of each letter in the alphabet by counting the amount of times each letter occurs in the dataset and dividing it by the total number of letters in the dataset. Once we have this, we can calculate the weight of each word by simply adding up the letter weights for each letter in the word. Because every word has the same number of letters, we do not need to balance anything more than this. We also have a built in bias against repeated letters in a word, by cutting the weight of a repeated letter in third. This helps to keep our model from skewing towards guesses that include many of the same letter, even if that letter is one of the most popular ones in the dataset. Once these ranks are calculated for every word in the list, we sort them by their rank and send them back as a dictionary with the word as the key and the rank as the value. We then write this dictionary into a csv called `word_ranks.csv`.


```
arose: 42.851565129586
arise: 41.73274991585325
raise: 41.73274991585325
erase: 40.852911477616956
stare: 40.82531134298216
saner: 40.516997643890946
snare: 40.516997643890946
aisle: 40.475260854930994
easel: 39.59542241669472
lease: 39.59542241669472
least: 39.56782228205991
```

Once this file is written, we can filter the words based on the guess feedback. We do this the way anyone would logically do it if they were playing Wordle themselves, by going letter by letter and eliminating words that do not fit the criteria of the feedback for that letter. For example, if the feedback for a letter is ‘g’, we delete every word from the list that has that letter in it. If the feedback is a ‘y’, we remove any words that have the letter in that position, and also

delete any words that do not have the letter in it. Lastly, if the feedback is 'r', we delete all words that do not have the letter in that exact spot. We then override the word_ranks.csv file with this new list, so that we can save it for the next guess. Lastly, we take the first word on the list (which has the highest rank) and return that as the ideal guess in the given situation.

The way that this solver is set up ensures that all progress is saved as long as the first variable is passed in as false. This tells the function to skip the generation of the word_ranks.csv file, and therefore it will use the file that was edited on the previous run. This lets the program work with the already filtered data for each subsequent guess, so that it does not guess a letter that was marked as gray in a previous word.

All of the code that was written for the purposes of this project was original, but we did use data from a Wordle word dataset that we found from a public GitHub repository that included only the words as a text document. This dataset has 2,315 words that we use as all the possible answers and guesses that the solver (or the player) is allowed to enter into our game. The dataset is just a text document that includes one word per line with no other punctuation, as shown in the picture below. In addition, a website was used to find the RGB color codes for the proper wordle colors, in accordance with the New York Times game. The only thing taken from this website was the color codes, to incorporate into our game.



```
1    aback
2    abase
3    abate
4    abbey
5    abbot
6    abhor
7    abide
8    abled
9    abode
```

Our code was organized in a way that we shifted from broad to specific. Our first files are `wordle.txt` and `wordle_answers.txt`. These 2 files contain the list of all valid wordle words and all wordle answers respectively. Then we go more specific towards our goal with the creation of `wordle.py`, which is our recreation of wordle without pygame, and what we used to get a basic understanding of how the Wordle game works. Then, we go even more specific with `solver.py` which contains the code for our solver. This solver outputs a csv file that contains the rankings of all our words. This file is further supported by our 2 common letter sorters, `common_letter.py` and `common_letter_pos.py` which sort the list contained in our word answers by common letter in general and for each position within the word. Finally, we go very specific with our wordle game, contained in `main.py` and `welcome_screen.py`. These two files contain the culmination of all the other files, combining a recreation of the Wordle game and our solver into one product.

One of the main challenges that we experienced was using a data set that was not proper for the game. The first set of data that we used had upwards of 10,000 words, which had words that did not exist, or were not in English. We realized this after trying to play the game, and the answer was impossible for us to get, since it was not a real word that we knew of. This caused us to realize that we needed to change our data. We then found data based on real wordle answers, which we knew would be much better in our game. After incorporating this data, we found..... From our challenges, we have learned to check the data before using it, in order to ensure that it works properly for our project. In addition, we were able to become more familiar with Pygame, so that we can create more games or projects with a nice GUI in the future. This was something that we were not extremely experienced with, so it was interesting to get to learn about.

One way that our project changed over time was our data set. We first began with using the complete list of possible wordle guesses. However, once we created our game, we realized

that a lot of these words were not real words, with many not even being english. After realizing this, we imported a new data set of only possible wordle answers that not only made our game more user friendly, but more accurate to Wordle.

The idea of a programmatic Wordle solver has been completed before, so there were plenty of resources online which showed how other people approached the problem. We decided to go in our own direction for our solver, using a strategy based on finding the most common letters used in the dataset, and then ranking each word accordingly to give ourselves the best chance to at least have letters that are yellow or green. Other solvers took different approaches based on weighting letters that appear in closest to 50% of words, as that would help to eliminate more words. However, after testing multiple strategies, we determined that the most common letter approach solved the problem in the quickest and most memory efficient way. We also found resources that included Wordle tips for non-programmatic approaches, and we used these to help form our strategy as well.

Another resource we used was our dataset, which was a list of acceptable Wordle answers. This data was used both for the purposes of the game and also for the purposes of the solver. The words came from an open source dataset that we found on a GitHub repository.

In addition, the New York Times wordle game was a key resource. Playing their version helped us to ensure that ours was as similar as possible, in terms of both gameplay and aesthetics. It was important to play their game multiple times, and to even play the archives in order to ensure our project worked in all the ways we wanted it to.

Be sure to explain what mathematical concepts you used in your project.

Although there were no machine learning concepts that we used in this project, we did use statistics to give us the best chance of finding letters that are in the word. This allows us to find the word in an average of 3 to 4 guesses in most cases. Of course there are some words, which have almost exclusively rare letters, which our solver takes 5 or 6 attempts to solve, but this is not a common occurrence. We ran our model versus a solver that picks a random word following the wordle rules, and ours outperformed the random picker in every test we ran. Our model found the correct answer in 3 guesses 39% of the time, while the random picker only found it 27% of the time. Additionally, our model found the correct answer in 4 guesses 77% of the time, while the random solver only found it 66% of the time. This was tested in a simulation of 10,000 Wordle games where our solver and the random solver were tested on the same word for the game.

Did you find anything interesting? What applications are there for your project?

There were a few interesting aspects of the project. For one, we noticed that it was necessary to find a dataset of wordle answers, rather than just five letter words. After doing this, we noticed that we had much less data, however this data was much more beneficial for our project. The data set of random five letter words included words that did not exist, or were in another language. It was important to find a new data set, since players would never be able to guess the word if it was not a valid English word.

The project can be used in a variety of ways, with the main one being for fun and entertainment. Our solver allows users to get hints or to see the answer to the game, rather than getting stumped and frustrated. It makes the gameplay easier for users, but still allows players to have the option to play without having hints or having the game solved.

There are many alternative methods of finding the best possible next word. For example, though we used the most common letter in total, we also investigated the most common letter for position, which we decided not to use. However, the most common letter by position may produce different results, and is therefore worth investigating next in the project.

Additionally, we could also use different methods of sorting words that don't include common letters. These different methods of sorting could include things like maximizing the number of greens and yellows when plugging the word into Wordle. This method may be more efficient to find an answer, so it would be worth further investigation.

Further expansion of our project could be related to the solving of wordle spin off games, of which there are numerous. One example is Quordle, where the user is told to solve 4 different Wordles at once. Our project would fit perfectly with this game as a beginner would be very stumped and overwhelmed, while our hint system would help them enjoy the game. Another further game is Squirdle, which is a Wordle spinoff that tasks the player with guessing the name of a pokemon. There are plenty of spinoff games, others including ones based on guessing a sports player or a country based on certain hints. It would be interesting to see our solver logic working to solve these other games, and to see how we may have to tweak our project to make it fit the other games.

Further expansion of our game could also be incorporating it into a web browser, as an extension or simply a solver. With the extension, the user could ask for hints for solving Wordle, or get the answer outright with the extension placing answers directly into the Wordle game. This project would be of a much larger scale than our current project, but it would also increase the usability and relevance to those who play Wordle on a daily basis or those that are trying to get into the game.

Overall, our project was successful and we accomplished our main goals. We were able to recreate the Wordle game, and incorporate a solver into it. The hint and solve buttons allowed users to get letters of the word to use in their guess, or to see the answer. Despite encountering a few challenges along the way, we were able to get past them and finish our project. The game is fun to play, and now has added handicaps to help a player who may be struggling. The Wordle solver was a great success.

Works Cited

Palmer, Dan. "Programmatically Solving Every Wordle Answer - Dan Palmer - Medium."

Medium, 21 Mar. 2022,

medium.com/@d.palmer101/programmatically-solving-every-wordle-answer-93c5f6237e88.

"How to Solve Wordle (Without Spoiling the Fun) | 8th Light." *8th Light*,

8thlight.com/insights/how-to-solve-wordle-without-spoiling-the-fun.

Knight, Ste. "13 Wordle Tips and Tricks to Improve Your Score." *MUO*, 30 June 2023,

www.makeuseof.com/wordle-tips-hints-tricks.

"Wordle - a Daily Word Game." *Wordle*, www.nytimes.com/games/wordle/index.html.