Solving Wordle Puzzles using Regular Expression and Greedy Algorithm in Python

Gede Prasidha Bhawarnawa - 13520004
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail : 13520004@std.stei.itb.ac.id

***Abstract*—This paper aims to solve a popular word puzzle named "Wordle", a web-based word game. The goal of the game is to find the keyword with the least amount of guesses. The solution proposed by the researcher is by implementing "greedy" algorithms with three different heuristic bases: <MASUKIN NANTI>. The result is a comparison that shows the most optimal heuristic to win a Wordle game.**

Keywords—Wordle, greedy algorithm, priority queue, regex, weighted letters.

I. Introduction

Since the 1800s, word puzzles have been popular among the masses, especially crosswords. Although the most popular form of crosswords needs to wait until 1913 in the U.S. for its most popular distribution form, the newspaper, people around the world have shown intrigue in word puzzles such as hangman, where the guesser must correctly guess the secret word before the drawn man gets hanged from too many wrong guesses; anagrams, where letters in a word are re-ordered to create a new word i.e. "space" is an anagram of "paces"; and word scramble games, a similar game to anagrams but the player are given a set of rules and wildcards that let them use whole or partial letters.

Figure 1 Hangman Word Puzzle
(Source: Personal Library)

As civilization progresses, so does our way of playing games. While some ways of playing word puzzles persist through time like doing crosswords on a newspaper, many people prefer to use the current technological advancement for their gaming experience, word puzzles included. Game companies also saw this and adapt their products to accommodate the current players' needs. Instead of physical board games, now word puzzle executable applications can be seen mass distributed through online distribution services such as Google Play Store, Apple App Store, and Steam.

Alternatives to executable files for games right now are web-based games. Web-based games are games that can be accessed through web browsers such as Mozilla Firefox, Internet Explorer, Google Chrome, or other similar applications. This is preferable for many people as it does not require complicated installation and these formats are playable anywhere and at any time, provided they have the internet connection. One such web-based word puzzle game that will be discussed more thoroughly in this paper is Wordle, a web-based game from the New York Times.

II. The "Wordle" Game And English Vocabulary

"Wordle" is a web-based word puzzle owned by the New York Times Company since 2022. This game is created and developed by Welsh software engineer named Josh Wardle. Originally, the game was created by Wardle for himself and his partner to play in 2013. However, they decided to release it to the public in October 2021. The game later gained a lot of popularity thanks to a feature that enables the players to share their results to social media like Twitter. After that, The New York Times Company proceeds to buy the game for an undisclosed seven-figure sum (in U.S. dollars) with plans to keep it free for all players. As of right now, the game can be visited in their official website through www.nytimes.com/games/wordle/index.html.

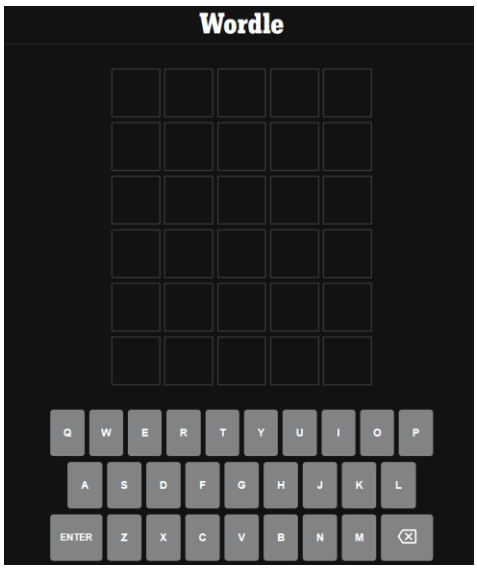


Figure 2 New Wordle Game GUI and Template
(Source : Personal Library)

The figure above is a new game of Wordle. The players can guess up to six times to figure the word-of-the-day. For every attempted guess that is not an English word, the game denies it and asks the player to change their guess. In the case that the guess is a valid English word, the program then checks letter by letter whether the inputed word is correct or not. There are three possible outcomes for each letter in a word: the letter is incorrect and it doesn't exist anywhere in the secret word, the letter exists but it is placed incorrectly in the secret word, and the letter exists in the final word and also placed correctly. An example game is shown in the following figure.

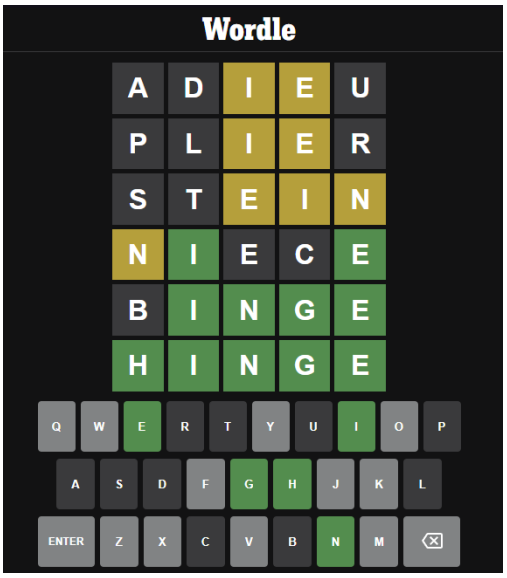


Figure 3 Example Wordle Game
(Source: Personal Library)

As can be seen from the figure above, the secret word is "HINGE". Letters that are highlighted in yellow means that the letters exist on the final answer but is placed incorrectly. Letters that are highlighted in green, however, means that those letters are correctly placed in the final answer. The remaining unhighlighted letters mean that they don't exist in the final answer.

The game refreshes every 24 hours. This means that to play a new game, a player must wait for 12.00 AM their local time. The game also records past gameplays and track the statistics, including the total game played, win rate, win streak, and maximum win streak.

On to deciding the best strategy, first it is important to understand the English vocabulary. Vocabulary, cited from the Oxford Languages dictionary, is "the body of words used in a particular language". In other words, vocabulary is the complete list of words that may be used in a language. Now English words are constructed with Roman alphabet, alternatively known as Latin alphabet. Roman alphabet consists of alphabetical characters starting from A to Z.

In English dictionary, there exists more than 171.146 words according to the Oxford English Dictionary with all sizes and length. Since this research will only focus on the Wordle game, we can filter the words to those that have a length of five letters. There are more than 158.000 five-letter words in English dictionary but Wordle does not use every word in the dictionary. Wordle has only 2.315 possible solution words and an additional 10.657 words that will not be a possible solution but still accepted as guesses.

Second thing to understand about the English vocabulary is the types of letters that are used to construct words. Commonly there are two types based on the sound that are represented: vowels and consonants. Vowel letters are "a", "i", "u", "e", "o" with the remainder of the letters classified as consonants. Out of the 158.000 five-letter words in the English dictionary, there are less than a percent words that has no vowels, a total of twenty to be exact. With that being said, it is safe to asssume that vowels are almost guaranteed to exist in a secret word every time.

III. Greedy Algorithms and Implementation

Greedy algorithms are algorithms that solve problems by picking out the currently (local) best option available with the hope that it will lead to an global optimal solution. Greedy algorithms never reverses an earlier decision and only progresses forwards until the solution is reached, regardless the solution being most optimal or not.

Greedy algorithms are commonly used for optimization problems or problems that require an optimal solution. There are only two types of optimization problems: maximazation problems and minimization problems.

In a greedy algorithm, there are elements that are required for the functionality of the algorithm. In total there are six: candidate set, solution set, solution function, selection function, feasibility function, and objective function. Candidate set is a set which contents are filled with possible choices to pick at each step. Solution set contains candidates that have been chosen from previous steps. Solution function is a function that

determines whether the previously mentioned solution set is enough to satisfy as a solution. Selection functions are functions that serve as a filter for the candidate set on a given step based on heuristic strategies. Feasibility function is a function that determines whether a candidate solution is feasible to be inserted into the solution set. Lastly, objective function is a function that serves as an optimization, either maximizing or minimizing the solution set.

In this research, the researcher's final aim is to determine the best strategy to win a Wordle game. To do this, the researcher create a simulation of a Wordle game using Python programming language. To simulate the dictionary used in Wordle, the simulation uses a Python library called NLTK or Natural Language Toolkit. Using this library, the simulation can extract five-letter English words and save that list to be used later. It is important to note that some words in the NLTK library has upper-cased letters and since the simulation wants to mimic Wordle as close as possible, these words are eliminated from the list.

```
# Importing the list of english words and limit the length of the words to five (as in Wordle)
# All words here is assumed to have a meaning and is a valid answer to a wordle (lowercase).
raw_english_words = nltk.corpus.words.words()
print(len(raw_english_words)) # Returns 236726 words in total
wordHashList = PriorityHashQueue()
for word in raw_english_words:
    if len(word) == 5 and not hasUpperCase(word):
        wordHashList.enqueue(word, 0)
print(wordHashList.returnLength()) # Returns 8689 five-lettered words in total
```

Figure 4 Importing Words to Simulation
(Source : Personal Library)

In total, there are 8.689 five-lettered words extracted from the NLTK library, which is almost four times the amount of possible solutions in Wordle dictionary. As a result, it may need more than six guesses to find the final answer and more computation time to guess the answer.

These 8.689 serve as members of the candidate set, as every single one of them has the possibility of being the final solution. As for the solution set, it is not necessary to store every word that has been attempted. In exchange, the solution set can be replaced with a list of five elements with each item stored sequentially and can be accessed directly or through random access. The reason being we only need to store the correct letters with the correct position in the set. Note that to input letters into the solution set, it is required to do sequential comparison between the candidate and the secret word or the solution. Therefore, the corresponding solution function can be pseudocoded as follows: "If every cell in the solution set is full, then the solution function returns true". The feasibility function for this case must consult the current contents of the solution set and joins all contents of the list as a string, with empty cells replaced with regex wildcard (.). If a candidate pass the regex, then it passes the feasibility function.

For the optimization function, there will be two parameters used in this simulation: the computation time and the number of guesses made. This problems falls into the minimization problem category, as the strategies involved and used are expected to minimize the computation time and the number of guesses made.

IV. Wordle Heuristic Strategies and Results

There will be two different specific-use heuristic strategies that will be implemented and compared with each other for this simulation. There will also be two general-use heuristic strategies that will be used in every simulation.

The first general-use strategy is to test words that has the most variative vowels on the first guess. Those words are contained in a list named "first_words" as those will be the words that will be tested first. The contents of this list are the words "adieu", "audio", "auloi", "aurei", "louie", "miaou", "ouija", "ourie", and "uraei". All of the above letters has four different vowels and this strategy is expected to eliminate as many candidate members in the first guess.

The second general-use strategy is to use a priority queue. The reason for using this specific type of data structure is so that the simulation knows what word to be used for the next guess. As previously mentioned, greedy algorithms always attempts to pick the best option at each step. By using priority queue with degree of correctness as the priority number, the simulation may pick the best option at all times. The degree of correctness for each word is equal to the amount of misplaced letters in the corresponding word and add that with the amount of correct letters times six. The reason six being used is that it's higher than five, which is the length of the word, and it avoids a problem where a word with five misplaced letters is used as a guess instead of a word with three correctly placed letters. To implement this in Python, the researcher construct a class using object-oriented programming. The researcher created two types of class: one to store word and priority number named HashMap and the other for the priority queue itself named PriorityHashQueue.

```
class HashMap:
    # This class is used to create a hashmap
    def __init__(self, word, match_value):
        self.word = word
        self.match_value = match_value

    def get_word(self):
        return self.word

    def get_match_value(self):
        return self.match_value

```

Figure 5 HashMap Class Implementation
(Source : Personal Library)

```python
class PriorityHashQueue:
    def __init__(self):
        self.queue = []

    def returnWordIdx(self, index):
        return self.queue[index].get_word()

    def returnMatchValueIdx(self, index):
        return self.queue[index].get_match_value()

    def returnLength(self):
        return len(self.queue)

    def returnContents(self):
        for i in self.queue:
            print(i.get_word(), i.get_match_value())
        print()

    def returnRandomValue(self):
        return self.queue[random.randint(0, self.returnLength() - 1)]

    def returnTopValue(self):
        return self.queue[0]

    def enqueue(self, word, match_value):
        if (self.returnLength() == 0):
            self.queue.append(HashMap(word, match_value))
        else:
            newEntry = HashMap(word, match_value)
            for i in range(self.returnLength()):
                if (i == self.returnLength() - 1 or match_value == 0):
                    self.queue.append(newEntry)
                    break
                elif (match_value > self.queue[i].get_match_value()):
                    self.queue.insert(i, newEntry)
                    break

    def dequeue(self, index):
        return self.queue.pop(index)
```

Figure 6 PriorityHashQueue Class Implementation
(Source : Personal Library)

The first specific-use heuristic strategy is by using a "banned letters" list. This list contains letters, can be vowels or consonants, that have been confirmed to not exist in the secret word. The goal here is to eliminate as many words as possible with every guess. Theoritically speaking, this strategy is good, but has its flaws. Assuming that the secret word has five distinct letters, it means that there are 21 incorrect letters. To eliminate every word from the candidate list with at least one of the 21 incorrect letters, it requires at least 5 guesses, which is 5/6 of the total guesses the game provides.

```python
def filterWordsBannedLetters(word_list, banned_letters, baseline_answers, misplaced_letters):
    # This filter uses list of banned letters to eliminate words from the candidate list (word_list)
    wordHashList = PriorityHashQueue()
    try:
        for i in range(word_list.returnLength()):
            word = word_list.dequeue(0).get_word()
            isBanned = False
            matching_letters = 0  # Priority value score
            for letter in banned_letters:
                if letter in word:
                    isBanned = True
                    break
            if not isBanned:
                index = 0
                for letter in misplaced_letters:
                    if letter == baseline_answers[index]:
                        matching_letters += 6
                    elif letter in word:
                        matching_letters += 1
                    index += 1
                wordHashList.enqueue(word, matching_letters)
    except Exception as e:
        print(repr(e))
    return wordHashList
```

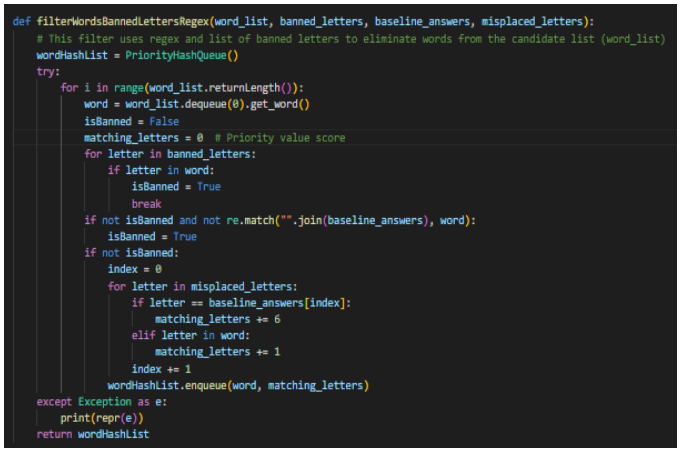
Figure 7 First Heuristic Strategy Implementation
(Source : Personal Library)

The second specific-use heuristic strategy is by using regular expression (regex). As previously mentioned, the feasibility function can extract the solution set into a regex with the empty cells replaced with a wildcard. The idea here is to eliminate every word in the candidate list that gets rejected from the regex test. In theory, there should be an improvement from the previous strategy because assuming that a five-letter word can be made from all 26 characters in the alphabet, regardless whether there exists a void in meaning or not, that means there are 26^5 words. Assuming the regex consists only 1 non-wildcard element, like /^…a.$/, this would make the candidate set contents size at a 26^4 words, a decrease of 96%.

```python
def filterWordsRegex(word_list, baseline_answers, misplaced_letters):
    # This filter uses regex to eliminate words from the candidate list (word_list)
    wordHashList = PriorityHashQueue()
    try:
        for i in range(word_list.returnLength()):
            word = word_list.dequeue(0).get_word()
            isBanned = False
            matching_letters = 0  # Priority value score
            if not re.match("".join(baseline_answers), word):
                isBanned = True
            if not isBanned:
                index = 0
                for letter in misplaced_letters:
                    if letter == baseline_answers[index]:
                        matching_letters += 6
                    elif letter in word:
                        matching_letters += 1
                    index += 1
                wordHashList.enqueue(word, matching_letters)
    except Exception as e:
        print(repr(e))
    return wordHashList
```

Figure 8 Second Heuristic Strategy Implementation
(Source : Personal Library)

The third specific-use heuristic strategy is by combining the first and the second strategy with the hopes of achieving better results than the result of one specific-use strategy alone.



```
def filterWordsBannedLettersRegex(word_list, banned_letters, baseline_answers, misplaced_letters):
    # This filter uses regex and list of banned letters to eliminate words from the candidate list (word_list)
    wordHashList = PriorityHashQueue()
    try:
        for i in range(word_list.returnLength()):
            word = word_list.dequeue(0).get_word()
            isBanned = False
            matching_letters = 0  # Priority value score
            for letter in banned_letters:
                if letter in word:
                    isBanned = True
                    break
            if not isBanned and not re.match("".join(baseline_answers), word):
                isBanned = True
            if not isBanned:
                index = 0
                for letter in misplaced_letters:
                    if letter == baseline_answers[index]:
                        matching_letters += 6
                    elif letter in word:
                        matching_letters += 1
                    index += 1
                wordHashList.enqueue(word, matching_letters)
    except Exception as e:
        print(repr(e))
    return wordHashList
```

Figure 9 Third Heuristic Strategy Implementation
(Source : Personal Library)

The result can be shown with the table below for the time and the number of guesses required. Strategy 1 is for "banned letters" list, strategy 2 is for the regex method, and strategy 3 is the combination of the first two strategy

Word	**Strategy 1**	**Strategy 2**	**Strategy 3**
Cully	0.02678 s	0.07329 s	0.02742 s
Rever	0.05221 s	6.90094 s	0.03840 s
Slurp	0.08432 s	10.84957 s	0.10699 s
Abode	6.66904 s	0.11388 s	0.05123 s
Fenny	0.14661 s	14.55807 s	0.04644 s
Nokta	0.97845 s	2.11133 s	0.61277 s
Dodgy	0.03228 s	0.83489 s	0.06668 s
Maybe	2.52278 s	0.48531 s	0.16146 s
Average	1.31406 s	4.49091 s	0.13892 s

Table 1 Computation Time for Three Heuristic Strategies
(Source : Personal Library)

Word	**Strategy 1**	**Strategy 2**	**Strategy 3**
Cully	9 guesses	7 guesses	4 guesses
Rever	10 guesses	16 guesses	6 guesses
Slurp	6 guesses	16 guesses	5 guesses
Abode	11 guesses	12 guesses	8 guesses
Fenny	11 guesses	21 guesses	4 guesses
Nokta	11 guesses	11 guesses	5 guesses
Dodgy	5 guesses	21 guesses	5 guesses
Maybe	14 guesses	11 guesses	5 guesses
Average	9.625 guesses	14.375 guesses	5.25 guesses

Table 2 Number of Guesses for Three Heuristic Strategies
(Source : Personal Library)

It can be deduced from the two tables that the third strategy is better than the first two strategy alone. The reason why the first strategy fails is for the reason mentioned previously. There may exists a chance that many guesses have to be made until every candidate that contains the letters not in the final answer are eliminated from the set. This takes time and guess attempts.

Another reason that can make the first strategy slow is for cases where the answer word has more than one repetition of letters. Like the word "Fenny" that has two n. This makes it harder for strategy 1 to work efficiently because it focuses more on eliminating the wrong answer rather than finding the correct answer.

The reason why the second strategy doesn't work efficiently, or worse than the first strategy in multiple cases, is that the strategy focuses too much on finding the correct answer rather than searching and eliminating the incorrect answers. In English, there are multiple words that follow vowel patterns like CVCCV with C being a consonant letter and V being a vowel letter. This makes the next guess attempts purely guessing and adding the risk of longer computation time and guess attempts. It will also difficult the algorithm to reduce the length of candidate list later in the program and since priority queue enqueue and dequeue process uses linear scan, the longer the list, the longer the time it will take.

The reason why the third strategy is better is because it focuses on both finding the correct answer and eliminating the incorrect answers. By decreasing the length of the candidate set with the banned_letters list and finding the correct answer with regex, the program finds the ansswer more efficiently even in cases where there exists double letters or more in the answer word. Even though the dictionary has four times the word list compared to the official Wordle dictionary, the current algorithm still manages to, in average, answer the daily puzzle without running out of guesses.

V. SUMMARY

Wordle is a web-based word-puzzle game where the player has limited guesses on the answer word. Hints are given to the player in the type of letter coloring. Green if the letter is correctly placed. Yellow if the letter exists in the answer but is currently misplaced. Gray if the letter doesn't exist. This research successfully determines that by implementing a greedy algorithm with a priority queue, a "banned_letters" list, and a regular expression to identify the possible correct words and eliminate incorrect words from the word list using a simulation with Python, an optimal solution is reached, even for a diictionary that is significantly larger than the Wordle dictionary.

VIDEO LINK AT YOUTUBE

https://www.youtube.com/watch?v=vOTjM04gdeQ

GITHUB LINK

https://github.com/LordGedelicious/Wordle-Puzzle-Solver-using-Regex-and-Greedy-Algorithm-in-Python

REFERENCES

[1] Munir, Rinaldi. Algoritma Greedy (Bagian 1). Institut Teknologi Bandung: Bandung, 2021, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf

[2] Munir, Rinaldi. Algoritma Greedy (Bagian 2). Institut Teknologi Bandung: Bandung, 2021, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf

[3] Munir, Rinaldi. Algoritma Greedy (Bagian 3). Institut Teknologi Bandung: Bandung, 2021, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag3.pdf

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 3rd Edition, pp. 414-450, Massachustes Institute of Technology, 2009.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 23 Mei 2022

Gede Prasidha Bhawarnawa 13520004