Wordle Stats Twitter Bot

By Andre Cardoso

Introduction

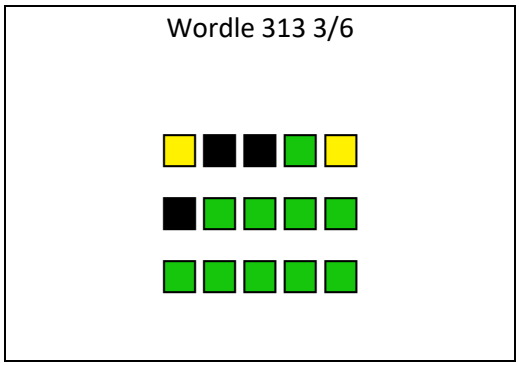
For this project I decided to look into make a twitter bot. I choose a twitter bot because I enjoy learning skills that I can apply later in my life, and because I wanted to practice more data analysis using public data. This project has been very interesting and fun for me to program, using a mixture of articles and Twitter own developer documentation I have learned a lot about how to use their data. I have been enjoying playing the very popular game wordle, and it seemed like a great topic to look into as it is very popular for twitter users to post their results. My twitter bot searches for tweets related to wordle and then calculates the average number of guesses it takes users to solve that particular day's wordle, along with calculating how many green, yellow, and grey letters each number of guesses has on average.

Twitter is a massive social media company with over 200 million registered users, from all around the world. These users can post whatever they choose to the site (or mobile app) in the form of a tweet, these tweets can be seen by other twitter users. Users can interact with tweets in the form of likes, replays, and retweets. Twitter has a developer program which allows programmer to request access to the Twitter API. There are several different levels of access to the Twitter API, the first level is essential access which is the most basic and only lets users request tweets. The second level is elevated access, this is the one I used. Elevated access allows the developer to request and create tweets, it has a limit of 2 million tweets per month. The next level is research access, to get access to this level a developer must prove that they are a researcher in the form of a published academic article. Research access allows the developer to request unlimited tweets. This level of access could be beneficial to my project, but I did not get approved by Twitter, and as a result I worked with elevated access.

Wordle is a very popular puzzle game in which the player has 6 chances to guess that day's word. Each day had a different 5 letter word. Each time the player makes a guess the letters will turn green if it is the correct letter in the correct location. Yellow if that letter is in the word but in the wrong location. And grey if that letter is not in the word. With this information the player should be able to figure out what the correct word. World has a built-in share result feature which is often how users post on twitter, the format of this feature can be seen in figure 1. The wordle attempt in figure 1, took 3 guesses to get the wordle word. At the top it can be seen that the format includes the current wordle number, which in this case is 313, and the number of guesses out of 6 it took, which in this case is 3.

Andre Cardoso

Figure 1: Wordle Share Format: 3 Guess attempt



Files and Libraries

My project is coded in python and uses two classes and many methods. Figure 2 contains an overview of how my code runs in the form of a flow chart. The list of files can be found in table 1. The file that is run is called main.py, it then runs the rest of the files. credentails.py is the next file in the hierarchy it contains a class called Credentails. This class contains 15 functions, inside of the __init__ function initiates the class, and also authenticates access to the twitter API using the keys, and tokens I got by signing up to be a twitter developer. I also initialize the Analize class, as that is called in several functions within the Credentails class. The SolveMain class is only called in one function so it doesn't need to be initiated unless it is being used. Wordle_Solver class is initialized and only used by SolveMain.

Table 1:

File Names and Hierarchy
main.py
credentails.py
analize.py
SolveMain.py
Wordle_Solver.py

The twitter API is an important part of how my project works, as it gives access to tweets, and allows my code to post to twitter. I used a library called tweepy simplify some interactions with the twitter API. Specifically, I used it in the authentication with twitter, as it makes that process much simpler. I also used a library for the date, called datetime, I also used timedelta to get the previous days date. These were used within the credentials.py file, but there are also more libraries that are a part of the SolveMain.py file. These libraries are PIL, which I use to import ImageGrab, pyautogui for mouse and keyboard control, and time for the sleep function. ImageGrab is used to check the color of each letter after the wordle solver makes a guess. As the wordle solver opens the official wordle site it requires keyboard and mouse control which is done through pyautogui. Time is important as the wordle site has animations and loading times which the program must wait to be completed before proceeding to the next guess. Pygal is used in the analize class to create a graph of the data if the user requests it.

Table 2:

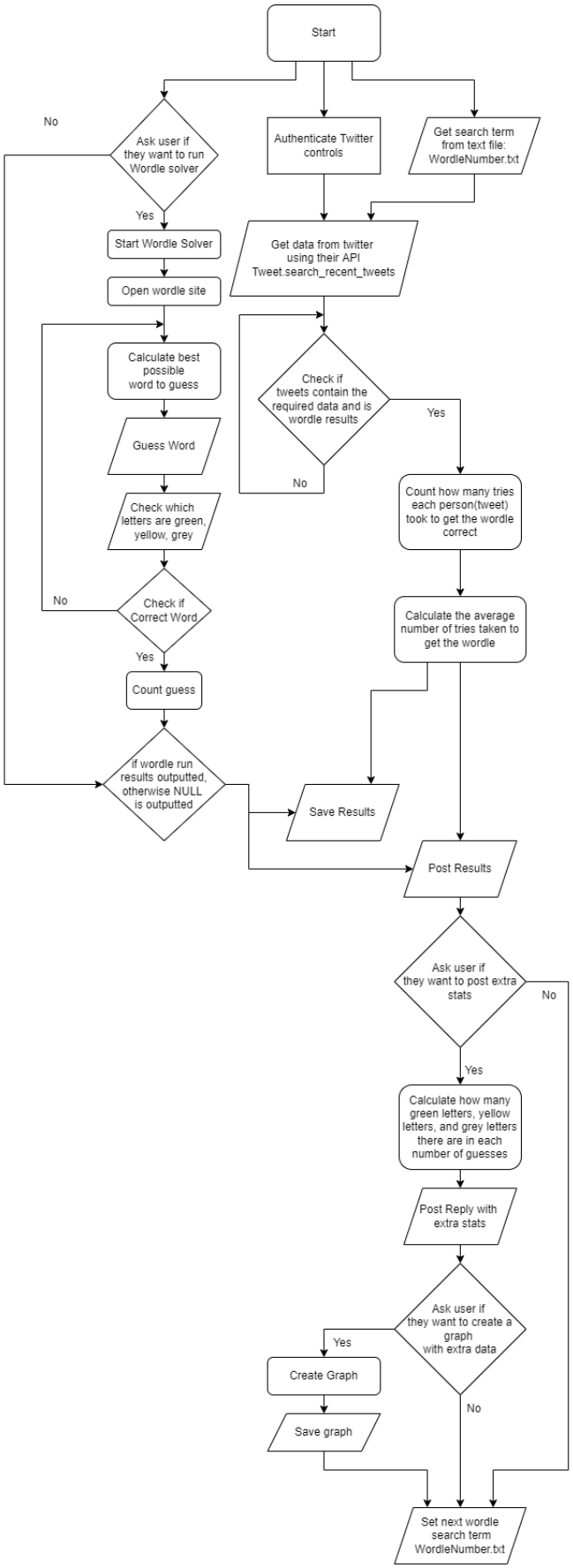
Libraries and APIs
Twitter API
import tweepy
from datetime import datetime, timedelta
from PIL import ImageGrab
import pyautogui
import time
Import pygal

My project uses several text files to both save, and load data. There are 5 text files in total, they are called in different places and each used for different purposes. WordleWords.txt is used to hold all possible wordle answers, this is used for the SolveMain.py as it needs to filter down all possible words to the correct word for that day. WordleNumber.txt contains the current days wordle number, an example of this would be “Wordle 314”, this is used as the search term when looking for wordle tweets. The wordle number is also posted in all tweets to make it clear which wordle the tweet is about. TwitterRecords.txt contains the raw data from how many guesses it takes users to solve that days wordle. This file is for long-term averaging of how twitter users performed in wordle. SolverRecords.txt contains the raw data from how many guesses the wordle solver took to guess the wordle. This is also used for long-term averaging. WordleRecords.txt contains a copy of what is posted to twitter incase anything goes wrong with the posting process.

Table 3:

File Name	Content
WordleWords.txt	This stores all possible wordle words
WordleRecords.txt	This stores a copy of each tweet in case there is a problem posting
WordleNumber.txt	This stores the current wordle’s number to then search for tweets
TwitterRecords.txt	This stores how many guesses on it took twitter user to guess the wordle
SolverRecords.txt	This stores how many guesses the wordle solver took to guess the wordle

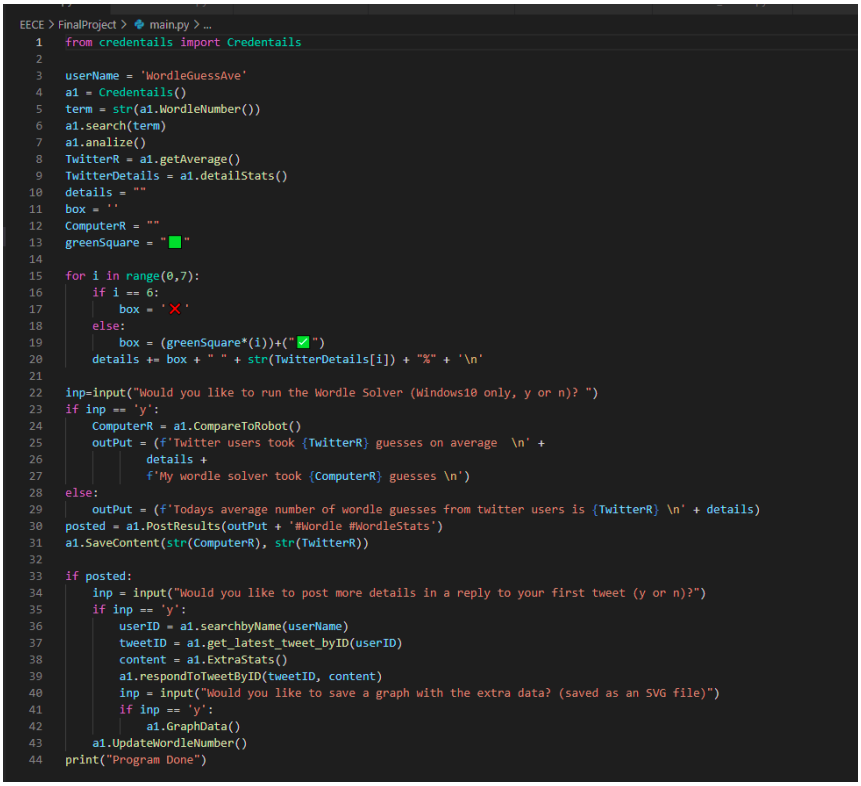
Figure 2: Over all Code Flow



main.py

The primary file for my project is called main.py, the full code in this file can be seen in figure 3. It starts with initializing the credentials class, and then uses methods in that class to get the wordle number, search for tweets containing that number, and analyzing those tweets to see how any guesses it takes. It then creates the string that will be the context of the tweet, after that string is created the user is asked if they would like to run the wordle solver. If they choose to run the wordle solver then that script will be run, and the result will be added to the tweet string. Then the string is posted on twitter. Next the user is asked if they would like to post more details in replies to the first tweet. If they choose to do so the program calculates that those extra details. After that the program asks the user if they would like a graph of the extra details that have been calculated, this graph is not posted to twitter, as twitter doesn't support the SVG file format. Lastly the program updates the wordle number.

Figure 3: main.py file



```
EECE > FinalProject > main.py > ...
from credentails import Credentails

userName = 'WordleGuessAve'
a1 = Credentails()
term = str(a1.WordleNumber())
a1.search(term)
a1.analize()
TwitterR = a1.getAverage()
TwitterDetails = a1.detailStats()
details = ""
box = ''
ComputerR = ""
greenSquare = "🟩"

for i in range(0,7):
    if i == 6:
        box = '❌'
    else:
        box = (greenSquare*(i))+("✅")
    details += box + " " + str(TwitterDetails[i]) + "%" + '\n'

inp=input("Would you like to run the Wordle Solver (Windows10 only, y or n)? ")
if inp == 'y':
    ComputerR = a1.CompareToRobot()
    outPut = (f'Twitter users took {TwitterR} guesses on average  \n' +
              details +
              f'My wordle solver took {ComputerR} guesses \n')
else:
    outPut = (f'Todays average number of wordle guesses from twitter users is {TwitterR} \n' + details)
posted = a1.PostResults(outPut + '#Wordle #WordleStats')
a1.SaveContent(str(ComputerR), str(TwitterR))

if posted:
    inp = input("Would you like to post more details in a reply to your first tweet (y or n)?")
    if inp == 'y':
        userID = a1.searchbyName(userName)
        tweetID = a1.get_latest_tweet_byID(userID)
        content = a1.ExtraStats()
        a1.respondToTweetByID(tweetID, content)
        inp = input("Would you like to save a graph with the extra data? (saved as an SVG file)")
        if inp == 'y':
            a1.GraphData()
    a1.UpdateWordleNumber()
print("Program Done")
```

The credentials class contains a lot of functions, and the following figures will go through them one by one. Starting with figure 4.0, which contains the imported libraries, and the __init__ function. The most important part of the __init__ function is authenticating access to the twitter API, without that the rest of the code will not work. This is done using a series of access tokens, API keys, and client ID. These are unique to my twitter developer account and **should not be shared with the public**. Other than authenticating access to the twitter API, there is not much in this function, it creates some arrays which are used in other functions and initializes the Analyze Class so it can be used in later functions.

Figure 4.0: credentails.py file: __init__ function

```python
import tweepy
from datetime import datetime, timedelta
from analize import Analize
from SolveMain import Solve

class Credentails:
    def __init__(self):
        self.Access_Token = [redacted]
        self.Access_Token_Secret = [redacted]
        self.API_key = [redacted]
        self.API_secret_key = [redacted]
        self.Bearer_Token = [redacted]
        self.Client_ID = [redacted]
        self.Client_Secret = [redacted]
        self.currentDate = datetime.today().strftime('%Y-%m-%d')
        self.client = tweepy.Client(
                        bearer_token=self.Bearer_Token,
                        consumer_key=self.API_key,
                        consumer_secret=self.API_secret_key,
                        access_token=self.Access_Token,
                        access_token_secret=self.Access_Token_Secret)
        self.WordleN = (open("W:\CODE\python\EECE\FinalProject\WordleNumber.txt", "r")).read()
        self.Analyze = Analize(self.tweets, (['🟩', '🟨', '⬛', '⬜']), self.currentDate,self.WordleN)
        self.tweets = []
        self.data = []
        self.color = []
        self.ColorDistrbution = []
```

The first function in the credeintails.py file is a search function that takes a term and searches twitter for tweets containing that term created within the last day. As seen in figure 4.1 ti is currently set to get 1000 tweets, that could be increased more, but that does increase run time and in testing hasn't changed the results significantly. At the end of the function, it prints the number of tweets found, as a confirmation that 1000 tweets have been found.

Figure 4.1: credentails.py file: search function

```python
    def search(self, term:str):
        query = term
        enddate = self.currentDate + "T00:00:00Z"
        startdate = datetime.strftime(datetime.now() - timedelta(1), '%Y-%m-%d') + 'T00:00:00Z'
        for tweet in tweepy.Paginator(self.client.search_recent_tweets, query=query, start_time=startdate, end_time=enddate, max_results=100).flatten(limit=1000):
            self.tweets.append(tweet)
        print(len(self.tweets)," Tweets Found")
```

The second function seen in figure 4.2 is also a search function but used for a very different purpose compared to the first one. It's purpose it to take a username and search twitter for their twitter ID. This is used if the user wants to post extra details in a reply to the original tweet. To find that original tweet the twitter ID is required, and thus this function was made. It returns the user Id it finds.

Figure 4.2: credentails.py file: searchbyName function

```python
    def searchbyName(self, userName:str):
        user = self.client.get_user(username=userName)
        print("User ID Found")
        return user.data.id

```

The following function (figure 4.3) is used in conjunction with the previous function as it takes a twitter ID and returns the ID of that user's latest tweet. This again is used to reply to the original tweet with extra details, this tweet ID is what is used to identify which tweet the replies should be posted to.

Figure 4.3: credentails.py file: get_latest_tweet_byID function

```
42      def get_latest_tweet_byID(self, userID):
43          tweet = self.client.get_users_tweets(userID)
44          data = tweet.data
45          id = data[0].id
46          print("Tweet ID Found")
47          return id
```

Continuing on posting replies with extra details, the following function in figure 4.4 is the function used to post the replies. It takes the tweet ID and the content for that tweet. The tweet ID would be given by the previous function and would give tell the twitter API which tweet the program wants to respond to. The content for this function is expected to be an array where each element of the array will be posted as an individual replay to the original tweet. This function also has an exception in the even there is an error posting the replies.

Figure 4.4: credentails.py file: respondToTweetByID function

```
49      def respondToTweetByID(self,TweetID, content):
50          try:
51              for t in content:
52                  text = self.WordleN + ' \n' + t
53                  self.client.create_tweet(text=text, in_reply_to_tweet_id=TweetID)
54              print("Replays Posted")
55          except:
56              print("Replays Failed to post")
57
```

The figure in 4.5 shows the function to the main post my project will make. It takes the content and adds some formatting at the beginning then posts it. It is quite straightforward, but it is critical to the project. This is also the tweet which is then replied to by the function in figure 4.4. This function has a print statement to print the length of the tweet that is going to be posted, this is important as twitter has a character limit of 280. This also relates to the exception in the event something goes wrong with posting the tweet. The function returns true if the post is successfully posted and false if something goes wrong. This is important to stop the rest of the code from running and trying to post replies to a tweet that doesn't exist.

Figure 4.5: credentails.py file: PostResults function

```
58      def PostResults(self, content):
59          t = self.currentDate + '   ' + self.WordleN + '\n' + content
60          print(len(t))
61          try:
62              self.client.create_tweet(text=t)
63              print("Tweet Posted")
64              return True
65          except:
66              print("Tweet failed to post, Check tweet is not too long")
67              return False
```

The figure in 4.6 shows the analize function, which is used to analyze the tweets returned by the search function seen in figure 4.1. This is don by passing the information to the Analyze class specifically the countGuesses method in that class. That method returns an array where each element is the number of guesses one tweet took. So, if there are 1000 tweets of data there should be 1000 elements in the array with how many guesses they took. But the tweets returned by the search function are not always the correct format or related to wordle. Often people will talk about wordle when posting about wordle derived games. As a result, the countGuesses function doesn't usually return the same number of elements as it receives, and as it analyses the tweets it removes any that are not relevant to wordle. To show the user how many tweets fit the expected format this function prints the number of tweets that have passed.

Figure 4.6: credentails.py file: analize function

```
69      def analize(self):
70          self.data = self.Analyze.countGuesses()
71          print('Data Analyzed. ', len(self.data),'Tweets passed')
72
```

The function seen in figure 4.7 is used to average the data collected in the analize function. It needs an if statement to check if the number of guess is “X” as that would mean that twitter user took more than 6 guesses to solve the wordle. I then then returns the average.

Figure 4.7: credentails.py file: getAverage function

```
73      def getAverage(self):
74          ave = []
75          Fail = []
76          for i in self.data:
77              if i == 'X':
78                  Fail.append(1)
79              else:
80                  ave.append(int(i))
81          average = round(sum(ave)/len(ave),3)
82          print("Averages Calculated")
83          return average
```

The function in figure 4.8 calculates the distribution of guesses for a given set of data. This means that if most people take 4 guesses to get the wordle then it will be seen in this data. It returns an array where each element is the percentage of users that took that many guesses.

Figure 4.8: credentails.py file: detailsStats function

```
85      def detailStats(self):
86          stats = [[],[],[],[],[],[],[]]
87          averages = []
88          for i in self.data:
89              for x in range(1,7):
90                  if i == str(x):
91                      stats[x-1].append(1)
92              if i == 'X':
93                  stats[-1].append(1)
94          for x in range(0,7):
95              averages.append(round(((len(stats[x])/len(self.data))*100), 2))
96          print("Detailed Stats Calculated")
97          return averages
```

The function seen in figure 4.9 runs a method in the Analyze class classed countcolers. That method counts how many green, yellow, and grey letters each user has. Then it calls the extraCalculations method also from the Analyze class organizes it based on how many guesses it took that user. This function takes the data returned from those methods and formats it into strings that can be posted to twitter. One string is made for each number of guesses, and all put into an array. That array is returned.

Figure 4.9: credentails.py file: ExtraStats function

```
99    def ExtraStats(self):
100       self.Analyze.countcolors()
101       outPut = []
102       aveG = [[],[],[],[],[],[],[]]
103       aveY = [[],[],[],[],[],[],[]]
104       aveGrey = [[],[],[],[],[],[],[]]
105       GLetters, YLetters, GREYLetters = self.Analyze.extraCalculations()
106       for x in range(0,7):
107           aveG[x] = round(sum(GLetters[x])/(len(GLetters[x])),2)
108           aveY[x] = round(sum(YLetters[x])/(len(YLetters[x])), 2)
109           aveGrey[x] = round(sum(GREYLetters[x])/(len(GREYLetters[x])), 2)
110           self.ColorDistrbution.append([])
111           self.ColorDistrbution[-1].append(aveG[x])
112           self.ColorDistrbution[-1].append(aveY[x])
113           self.ColorDistrbution[-1].append(aveGrey[x])
114           if x == 6:
115               outPut.append("Users that Failed to guesses the wordle had on average:\n" + str(aveG[x]) + '🟩 letters \n' + str(aveY[x]) + '🟨 letters \n' + str(aveGrey[x]) + '⬛ letters \n\n')
116           else:
117               outPut.append("Users that took "+ str(x+1) +" guesses to get the wordle had on average:\n" + str(aveG[x]) + '🟩 letters \n' + str(aveY[x]) + '🟨 letters \n' + str(aveGrey[x]) + '⬛ letters \n\n')
118       print('Extra Stats Calculated')
119       return outPut
```

The function is figure 4.10 is only run if the users specifies it, this function creates a graph based on the data calculated in the previous function in figure 4.9. The graph is created in the createGraph method in the Analyze class.

Figure 4.10: credentails.py file: GraphData function

```
121     def GraphData(self):
122         self.Analyze.createGraph(self.ColorDistrbution,self.WordleN)
123         print("Graph Created and Saved")
124
```

The function seen in figure 4.11 initialize and runs the wordle solver. It returns the number of guesses the computer takes to get the wordle.

Figure 4.11: credentails.py file: CompareToRobot function

```
125     def CompareToRobot(self):
126         solving = Solve()
127         return solving.ComputerGuesses()
```

The function in figure 4.12, saves a copy of what is posted to twitter, along with a copy of the raw data of the twitter results, and the solver results. It has exceptions in the event the file save doesn't work. And prints individually confirmations of the file being saved.

Figure 4.12: credentails.py file: SaveContent function

```
129     def SaveContent(self,SolverR:str,TwitterR:str):
130         try:
131             f = open('W:\CODE\python\EECE\FinalProject\WordleRecords.txt', 'a')
132             f.write('\t' + str(self.currentDate) + '   ' + self.WordleN + '\n' +
133             ('Twitter Results'+ TwitterR) + '\n' +
134             ('Computer Solver Results' + SolverR) +
135             '\n\n')
136             f.close
137             print("Record Saved")
138         except:
139             print("Record Failed to Save")
140         try:
141             SRecord = open('W:\CODE\python\EECE\FinalProject\SolverRecord.txt', 'a' )
142             SRecord.write(SolverR+' \n')
143             SRecord.close
144             print("Solver Record Saved")
145         except:
146             print("Solver Record Failed to Save")
147         try:
148             TRecord = open (parameter) TwitterR: str Project\TwitterRecord.txt', 'a')
149             TRecord.write(TwitterR+' \n')
150             TRecord.close
151             print("Solver Record Saved")
152         except:
153             print("Solver Record Failed to Saved")
```

The function in figure 4.13 returns the wordle number.

Figure 4.13: credentails.py file: WorldeNumber function

```
155     def WordleNumber(self):
156         return self.WordleN
```

The function in figure 4.14 updates the wordle number in preparation for the following day. It does so by opening the wordleNumber text file and rewriting it with the new world number.

Figure 4.14: credentails.py file: UpdateWordleNumber function

```
158     def UpdateWordleNumber(self):
159         search = self.WordleN
160         n = search[7:10]
161         number = int(n) + 1
162         try:
163             Wf = open("W:\CODE\python\EECE\FinalProject\WordleNumber.txt", "w")
164             Wf.write('Wordle '+str(number))
165             Wf.close
166             print("Wordle Number Updated")
167         except:
168             print("Wordle Number Failed to Update")
169         return search
```

analize.py

The second class used is called Analize. It is used to analyze the data gathered from twitter and calculate different percentages. It can also make a graph using a library called pygal. All figures number 5 are from the analize.py file.

Figure 5.0 shows the __init__ function this function takes tweets, required data within the tweets, the current date, and the number of that dates wordle number. It also creates some arrays that are used in later functions.

Figure 5.0: Analize.py file: __init__ function

```
1   import pygal
2   from pygal.style import Style
3
4   class Analize:
5       def __init__(self, tweets, Required, date, number):
6           self.tweets = tweets
7           self.required = Required[:]
8           self.date = date
9           self.guesses = []
10          self.number = number
11          self.GreenLetter = []
12          self.YellowLetter = []
13          self.GreyLetter = []
```

The function in figure 5.1 is used to count the number of guesses each tweet takes. This is done in two ways, and then they are compared, if the numbers match that tweet is considered to be valid other wise it is removed from the list of tweets. The first check is done by searching for the wordle number in the tweet, the characters following that have a very specific format. An example of that format is “Wordle 314 3/6”, by first checking that the format is current, it can be assumed that the character before the slash is the number of guesses. Once that number is saved a second check is performed. The second check also uses the wordle sharing format by counting how many rows of colored squares there are, those colored squares are saved in the required array. Then these two checks are compared and if they match that number is saved. If the first check produces an ‘X’ that overwrites the second check as they second check can’t account for more than 6 guesses. This function returns the number of guesses.

Figure 5.1: Analize.py file: countGuesses function

```
15      def countGuesses(self):
16          index = 0
17          l1 = []
18          for twet in self.tweets:
19              doubleCheck = False
20              g = 0
21              gCheck= 0
22              tx = twet.text
23              for char in range(0, len(tx)):
24                  if tx[char] == 'W':
25                      if tx[char+0:char+10] == self.number:
26                          temp = char
27                          while tx[temp] != '/':
28                              if temp+1 < len(tx):
29                                  temp+=1
30                              else:
31                                  break
32                          if tx[temp] == '/' and tx[temp+1] == '6':
33                              g = tx[temp-1]
34                  temp = char
35                  while tx[temp] in self.required and not doubleCheck:
36                      gCheck+=1
37                      if (temp+6) < len(tx):
38                          temp+=6
39                      else:
40                          break
41                  if gCheck > 0:
42                      doubleCheck = True
43              if g == str(gCheck) or g == 'X':
44                  self.guesses.append(g)
45                  l1.append(tx)
46              else:
47                  self.tweets.pop(index)
48                  self.tweets[index]
49              index+=1
50          self.tweets = l1
51          return self.guesses
```

The function seen in figure 5.2 is used to count the number of green, yellow, and grey letters there are in each of the tweets. These calculated results are then returned all in one array of arrays.

Figure 5.2: Analize.py file: countcolors function

```
53      def countcolors(self):
54          output = []
55          for twet in self.tweets:
56              yellow = 0
57              grey = 0
58              green = 0
59              for i in twet:
60                  if i in self.required:
61                      if i == '🟩':
62                          green+=1
63                      elif i ==  '🟨' :
64                          yellow+=1
65                      elif i == '⬛' or i == '⬜':
66                          grey+=1
67              self.GreenLetter.append(green)
68              self.GreyLetter.append(grey)
69              self.YellowLetter.append(yellow)
70          output.append(self.GreenLetter)
71          output.append(self.GreyLetter)
72          output.append(self.YellowLetter)
73          return output
```

The function in figure 5.3 is supposed to be used in connection with the function found the figure 5.2. It splits the data into how many guesses that tweet took to solve the wordle, thus counting how many green, yellow, and grey letters there are in when a tweet takes 4 guesses to solve the wordle for example. Tweets that take one guess to solve the wordle will always have 5 green letters, 0 yellow letters, and 0 grey letters, as on their first guess they got every letter correct and in the current location. This function then returns the 3 arrays of arrays which can be formulated into a string and posted in a reply on twitter as seen earlier in figure 4.4.

Figure 5.3: Analize.py file: extraCalculations function

```
75      def extraCalculations(self):
76          GLetters = [[],[],[],[],[],[],[]]
77          YLetters= [[],[],[],[],[],[],[]]
78          GREYLetters = [[],[],[],[],[],[],[]]
79          green = self.GreenLetter
80          yellow = self.YellowLetter
81          grey = self.GreyLetter
82          for i in range(0,len(self.guesses)):
83              if self.guesses[i] == 'X':
84                  GLetters[-1].append(green[i])
85                  YLetters[-1].append(yellow[i])
86                  GREYLetters[-1].append(grey[i])
87              else:
88                  for num in range(1,7):
89                      if self.guesses[i] == str(num):
90                          GLetters[num-1].append(green[i])
91                          YLetters[num-1].append(yellow[i])
92                          GREYLetters[num-1].append(grey[i])
93                          break
94          return GLetters, YLetters, GREYLetters
```

The user has the option to create a graph, which is done in the function seen in figure 5.4, the graph is created using pygal, and incorporates wordles official colors to keep it on theme. The graph can be saved as a SVG file, and I attempted to save it as a PNG but was unsuccessful. The graph's name is the current days wordle number.

Figure 5.4: Analize.py file: createGraph function

```
96      def createGraph(self, data,Filename):
97          custom_style = Style(
98              font_family='googlefont:Roboto',
99              background='white',
100             plot_background='white',
101             foreground='black',
102             foreground_strong='black',
103             foreground_subtle='black',
104             opacity='1',
105             colors=('#1E9B52', '#b7a93e','#3a3a3c')
106             )
107         bar_chart = pygal.Bar(fill=True, style=custom_style)
108         bar_chart.title = 'Number of Green Letters, Yellow Letters, and Grey Lettters \n by Number of Guesses'
109         bar_chart.x_labels = map(str, ('Guess 1', 'Guess 2', 'Guess 3', 'Guess 4', 'Guess 5','Guess 6','Failed',))
110         bar_chart._y_title = 'Number of Letters'
111         bar_chart.add('Green Letters',  [data[0][0], data[1][0], data[2][0], data[3][0], data[4][0], data[5][0],data[6][0]])
112         bar_chart.add('Yellow Letters', [data[0][1], data[1][1], data[2][1], data[3][1], data[4][1], data[5][1],data[6][1]])
113         bar_chart.add('Grey Letters',   [data[0][2], data[1][2], data[2][2], data[3][2], data[4][2], data[5][2],data[6][2]])
114         fileSvg = f'W:\CODE\python\EECE\FinalProject\{str(Filename)}.svg'
115         filePng = f'W:\CODE\python\EECE\FinalProject\{str(Filename)}.png'
116
117         bar_chart.render_to_file(fileSvg)
```

The function in figure 5.5 is used to print the raw data of how many guesses each tweet has taken to solve the wordle, this is only used for debugging and is not called in the final version of the code.

Figure 5.5: Analize.py file: __str__ function

```
119     def __str__(self):
120         return self.guesses
```

SolveMain.py

The wordle solver I made is quite good at guessing the currect word, but by no means does it compete with the best wordle solvers that exist. It does use mouse and keyboard control through pyautogui and opens the official wordle site playing along as any human would. To check the color of each letter I use PIL's imageGrab method and check it against specific RGB values. After each guess the program filters down the possible words and makes a new guess.

Figure 6 shows the __init__ function for SolveMain.py. This is mostly creating arrays, opening the file with all the words, saving the RGB values, and importantly initializing the wordleSolver Class

Figure 6.0: SolveMain.py file: __init__ function

```
1  from Wordle_Solver import wordleSolver
2  from PIL import ImageGrab
3  import pyautogui
4  import time
5  class Solve:
6      def __init__(self):
7          self.words = (open("W:\CODE\python\EECE\FinalProject\WordleWords.txt", "r").read()).split(',')
8          self.a1 = wordleSolver(self.words)
9          self.guesses = []
10         self.greenL = []
11         self.yellowL = []
12         self.locationY = []
13         self.blackL = []
14         self.locationG = []
15         self.saveY = [[],[],[],[],[],[]]
16         self.saveG = [[],[],[],[],[],[]]
17         self.yellow = (183, 169, 62)
18         self.green = (30, 155, 82)
19         self.black = (58, 58, 60)
20         self.colors = [[],[],[],[],[],[]]
```

The function seen in figure 6.1 is the only other function in the Solve class. It runs everything related to the wordle solver. Starting with using the keyboard and mouse to open a google chrome incognito tab and going to the official wordle site. Once there it starts to make guesses, guesses start at the for loop on line 47. It loops a maximum of 6 times as that is the maximum number of words that can be guessed. After each guess is made the program checks the location of each letter and if that color matches the green, yellow, or grey values that was set when the class was initializing. Once it has that data it sends that data to the wordleSolver which filters down can calculates the best word to play next. It also checks if the wordle has been solved in which case it ends the program. The program returns the number of guesses it takes to guess the wordle.

Figure 6.1: SolveMain.py file: ComputerGuesses function

```
    def ComputerGuesses(self):
        x, y = pyautogui.size()
        pyautogui.moveTo(10, (y-10), 0.1)
        pyautogui.click()
        time.sleep(1)
        pyautogui.typewrite("Google Chrome",0.1)
        time.sleep(2)
        pyautogui.hotkey("enter")
        time.sleep(1)
        pyautogui.moveTo(x/2, ((y*2)/3), 0.1)
        pyautogui.click()
        pyautogui.hotkey("ctrl", "shift", "n")
        time.sleep(1)
        pyautogui.typewrite("https://www.nytimes.com/games/wordle/index.html")
        pyautogui.hotkey("enter")

        time.sleep(1)
        pyautogui.hotkey("f11")
        pyautogui.moveTo(1180, 297, 0.1)
        pyautogui.click()
        pyautogui.moveTo(800, 270,0.1)
        pyautogui.click()
        self.a1.SortWords()
        Ly = 270
        T = 2
        for j in range(0,6):

            wordpool = self.a1.getWords()[:]
            best = 0
            if best < len(wordpool)-1:
                while wordpool[best] in self.guesses and best <= len(wordpool)-2:
                    best +=1
            try:
                pyautogui.typewrite(wordpool[best],0.1)
                self.guesses.append(wordpool[best][:])
            except:
                print("Something went wrong, Check that wordle site did not have problems")
            pyautogui.hotkey('enter')

            time.sleep(T)
            T += 0.5
            Lx = 800

            for i in range (0,5):
                pyautogui.moveTo(Lx, Ly,0.1)
                pixel1=ImageGrab.grab().load()
                pxcolor1=pixel1[pyautogui.position()]
                if pxcolor1 == self.yellow:
                    self.yellowL.append(self.guesses[j][i])
                    self.locationY.append(i)
                elif pxcolor1 == self.green:
                    self.greenL.append(self.guesses[j][i])
                    self.locationG.append(i)
                elif pxcolor1 == self.black:
                    self.blackL.append(self.guesses[j][i])
                else:
                    self.blackL.append(self.guesses[j][i])
                Lx = Lx+67
            Ly = Ly + 67

            #calculate the next word here
            self.a1.addGreenLetters(self.greenL, self.locationG)
            self.a1.addYellowLetter(self.yellowL,self.locationY)
            self.a1.addBlackLetter(self.blackL)
            self.a1.CalculateWords()
            self.a1.SortWords()
            self.saveG[j] = self.greenL[:]
            self.saveY[j] = self.yellowL[:]
            time.sleep(2)
            pixel1=ImageGrab.grab().load()
            pxc=pixel1[1010,665]
            if pxc == self.green:
                print('Wordle Solved')
                break
            self.yellowL.clear
            self.locationY.clear
            self.greenL.clear
            self.locationG.clear
        pyautogui.hotkey('ctrl', 'w')
        pyautogui.hotkey('ctrl', 'w')
        return len(self.guesses)
```

Wordle_Solver.py

The wordleSolver Class is designed to take data given by the SolveMain Class to filter down the possible wordle words to get the correct answer. It does this using several arrays and functions to go over the words and make sure that they meet the criteria for the next guess. It then calculates which of the possible words it should play based on the number of times each letter is in each position. This has resulted in a highly effective process for choosing a word. All of the following figures are taken from the Wordle_Solver.py file

The first figure (7.0) shows the __init__ function along with two global variables that are uses later in the file. In this function five arrays are made and the wordpool is set. I programed this before I made the rest of the program and I name grey letters black letters, but they are the same, both represent letters that are not in the word.

Figure 7.0: Wordle_Solver.py file: __init__ function

```
1   alphabet = ("a",'b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z')
2   vowels = ('a','e','i','o','u','y')
3
4   class wordleSolver:
5       def __init__(self, wordpool):
6           self.wordpool = wordpool
7           self.allwords = wordpool
8           self.blackLetters = []
9           self.yellowLetters = []
10          self.yellowLocations = []
11          self.greenLetters = []
12          self.greenLocations = []
```

Figure 7.1 shows a function to add black/grey letters to the array for those letters. First it checks that those letters are not already in the yellow letters, or green letters as that would cause problems when filtering down the words, and then appends that letter to the blackLetters array.

Figure 7.1: Wordle_Solver.py file: addBlackLetters function

```
14      def addBlackLetter(self, letters):
15          temp = []
16          for x in range(0,len(letters)):
17              if letters[x] not in self.yellowLetters:
18                  if letters[x] not in self.greenLetters:
19                      temp.append(letters[x])
20          self.blackLetters += temp
```

The function in figure 7.2 is similar to that of 7.1 but it is for yellow letters. It again checks if that letter is already in the green letters and only appends letters that are not already in the green letters. It also saves not only the letter itself but also its location because if a letter is yellow it is in the word but in the wrong location and shouldn't be put in that same spot.

Figure 7.2: Wordle_Solver.py file: addYellowLetters function

```
22      def addYellowLetter(self, letters, locations):
23          temp = []
24          temp2 = []
25          for x in range(0, len(letters)):
26              if letters[x] not in self.greenLetters:
27                  temp.append(letters[x])
28                  temp2.append(locations[x])
29          self.yellowLetters += temp
30          self.yellowLocations += temp2
```

Figure 7.3 shows the addGreenLetters function which adds green letters and their locations to their respective arrays.

Figure 7.3: Wordle_Solver.py file: addGreenLetters function

```
32      def addGreenLetters(self, letters, locations):
33          self.greenLetters += letters
34          self.greenLocations += locations
```

The CalculateWords function that can be seen in figure 7.4 is used to filter down the wordpool to only words that meet the criteria of green, yellow, and grey/black letters. It starts by taking out any words that contain a grey/black letter. Next taking out any word that does not contain the yellow letters in a different location than their original location. Lastly removing any words that do not contain the green letters in the correct location. This filters done the word pool very effectively and while the loops could possibly be made more efficient this system has been working well.

Figure 7.4: Wordle_Solver.py file: CalculateWords function

```
36      def CalculateWords(self):
37          out = []
38          for i in range(0,len(self.wordpool)):
39              c = 0
40              for x in range(0,5):
41                  for j in range(0,len(self.blackLetters)):
42                      if self.wordpool[i][x] == self.blackLetters[j]:
43                          c=c+1
44              if c == 0:
45                  out.append(self.wordpool[i])
46          self.wordpool=out[:]
47
48          if len(self.yellowLetters) >= 1:
49              out2 = []
50              for i in range(0,len(self.wordpool)):
51                  c=0
52                  for x in range(0,5):
53                      for j in range(0,len(self.yellowLetters)):
54                          if self.yellowLocations[j] == x:
55                              c=c
56                          elif self.wordpool[i][x] == self.yellowLetters[j]:
57                              c=c+1
58                  if c >= len(self.yellowLetters):
59                      out2.append(self.wordpool[i])
60              self.wordpool = out2[:]
61
62          if len(self.greenLetters) > 0:
63              out3 = []
64              for i in range(0,len(self.wordpool)):
65                  c=0
66                  for j in range(0,len(self.greenLetters)):
67                      if self.wordpool[i][self.greenLocations[j]] == self.greenLetters[j]:
68                          c=c+1
69                      else:
70                          c=c
71                  if c > len(self.greenLetters)-1:
72                      out3.append(self.wordpool[i])
73              self.wordpool = out3[:]
```

The function in figure 7.5 returns the remaining wordpool. This is used to get the next word to guess. As well as for debugging any problems that could come up.

Figure 7.5: Wordle_Solver.py file: getWords function

```
76      def getWords(self):
77          return self.wordpool
```

This function is used to get a word based on its index this is only relevant if the first word the program tried to guess had already been guessed, it will use this function to get the next word.

Figure 7.6: Wordle_Solver.py file: getWordsIndex function

```
79      def getWordsIndex(self, index):
80          return self.wordpool[index]
```

Sorting the wordpool is a particularly crucial step in guessing a wordle word, as it does not matter if the words are filtered if the one at the top of the list is wrong. To sort the words, I came up with my own algorithm to find the best word to guess. First my algorithm counts how many times each letter is in each position of a word. That number is the value of that letter in that location. By adding up the value of each letter in a word that word is given a value. Once the values are calculated, the wordpool is sorted based on value. I also experimented with having vowels have a higher point value but that proved not to be effective.

Figure 7.7: Wordle_Solver.py file: SortWords function

```
82      def SortWords(self):
83          words = self.wordpool[:]
84          one = [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ]
85          two = [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,]
86          three = [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,]
87          four = [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,]
88          five = [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,]
89          for i in range(len(words)):
90              for x in range(0,5):
91                  for j in range(len(alphabet)):
92                      if words[i][x] == alphabet[j]:
93                          if x == 0:
94                              one[j] = one[j]+1
95                          if x == 1:
96                              two[j] = two[j]+1
97                          if x == 2:
98                              three[j] = three[j]+1
99                          if x == 3:
100                             four[j] = four[j]+1
101                         if x == 4:
102                             five[j] = five[j]+1
103
104         out = []
105         for i in range(len(words)):
106             out.append(0)
107             for x in range(0,5):
108                 for j in range(len(alphabet)):
109                     v=0
110                     if words[i][x] == alphabet[j]:
111                         # if words[i][x] in vowels:
112                         #     v = 100
113                         if x == 0:
114                             out[i] = out[i] +  one[j] + v
115                         if x == 1:
116                             out[i] = out[i] +  two[j] + v
117                         if x == 2:
118                             out[i] = out[i] +  three[j] + v
119                         if x == 3:
120                             out[i] = out[i] +  four[j] + v
121                         if x == 4:
122                             out[i] = out[i] +  five[j] + v
123         rating = out[:]
124         n = len(rating)
125         for i in range(n):
126             for j in range(0, n-i-1):
127                 if rating[j] < rating[j+1] :
128                     rating[j], rating[j+1] = rating[j+1], rating[j]
129                     words[j], words[j+1] = words[j+1], words[j]
130         self.wordpool = words[:]
131         return words
```

The __str__ function seen in figure 7.8 was used while debugging the wordle solver and is not used any more. It returns a string of the green letters, yellow letters, and black/grey letters. This can be helpful in understanding why the solver could fail to guess the wordle.

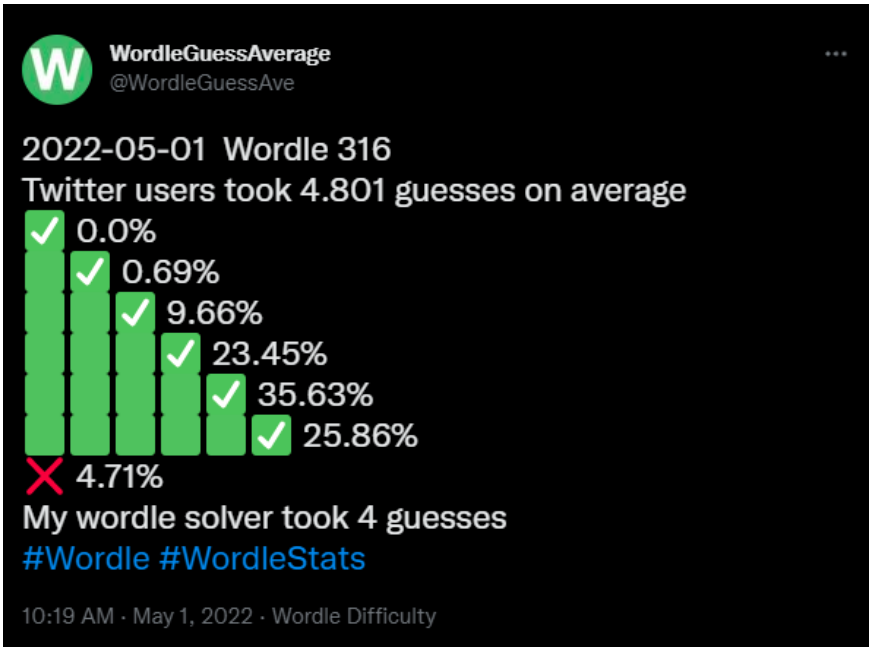
Figure 7.8: Wordle_Solver.py file: __str__ function

```
135     def __str__(self):
136         return f'Green Letters: {self.greenLetters}, Yellow Letters: {self.yellowLetters}, Black Letters: {self.blackLetters}'
137
```

Analysis

Once all of the averages are calculated the program formats and organizes it into a string that can be posted it to twitter. I named the twitter account WordleGuessAverage. Figure 8 shows the format the tweets are posted. It starts with the current date, followed by the wordle number and then the calculated data. First is the overall average which for this day is 4.8 guesses, followed by the guess distribution. As the program was run in the morning the data was more limited, the later in the day the program is run the more tweets there are about wordle. As a result, 0.0% of users guessed the wordle on the first try, 0.69% on the second try, 9.66% on the third, 23.45% on the fourth, 35.63% on the fifth, 25.86% on the sixth, and 4.71 failed. This shows that this wordle was more difficult than past ones, as most of them have an average bellow four guesses. Below the percentage that failed is the number of guesses my wordle solver took, which in this case was 4. Followed by 2 hashtags to help the tweet be shared with the target audience. As more of these tweets are posted a clearer idea of wordle difficulty is created. The distribution also shows how effective twitter users first guess is. At the bottom of the tweet there is the time and date it was posted, and where it was posed from. It says it was posted from Wordle Difficulty because that is what I named my project in the twitter developer dashboard.

Figure 8: Main Tweet

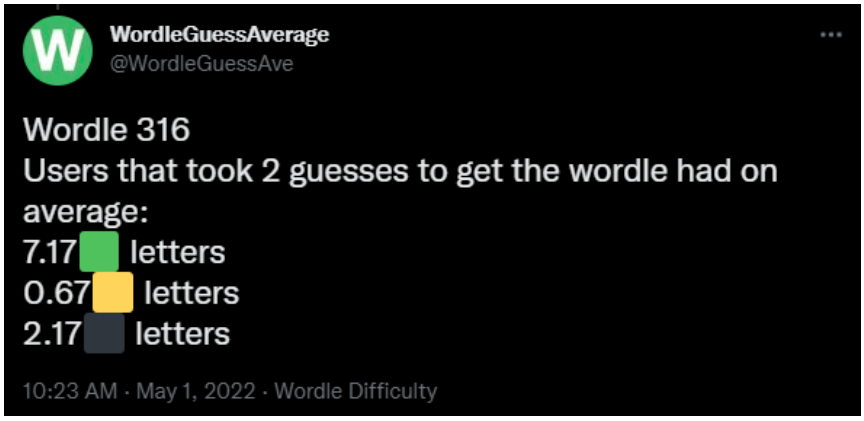


An example of the extra stats posted in a reply to the first tweet can be seen in figure 8.1. This figure shows the number of green, yellow, and grey letters users that got the wordle in one guess had. But as seen in the first tweet 0.0% of users guessed the wordle in one guess and as a result there are zero green, yellow, or grey letters. Under normal circumstances users that take 1 guess can will have 5 green letters.

Figure 8.1: One Guess Reply

Figure 8.2 shows the number of green, yellow and grey letters for users that took 2 guesses to solve the wordle. As seen in both figure 8.2 and 8.1 at the top of the tweet the wordle number is posed, this is to ensure that there is no confusion about which wordle this data is from. In figure 8.2 it can be seen that users appear to have very few yellow letters, and mostly green letters, this leads me to believe that these users are getting at least 2 green letters in there first guess, and then guessing the word correctly in there second guess.

Figure 8.2: Two Guess Reply



The number of green, yellow, and grey letters for users that took 3 guesses can be seen in figure 8.3. It shows a slight increase in yellow letters and a large increase in grey letters in comparison to figure 8.2. This leads me to believe that these users first guess is a lot less effective than that of users who solved it in 2 guesses.

Figure 8.3: Three Guess Reply

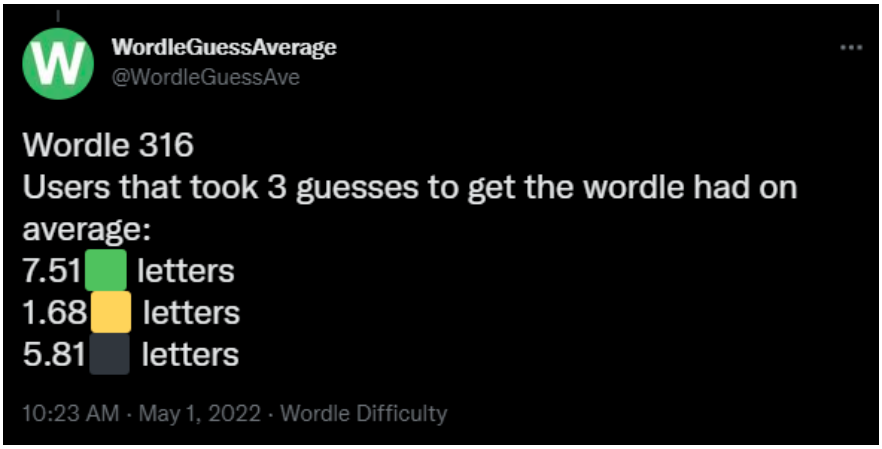
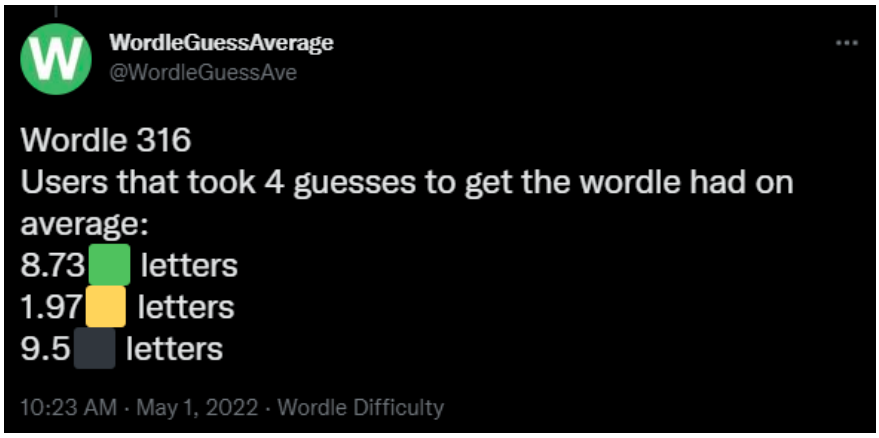


Figure 8.4 shows the number of green, yellow and grey letters users who solved the wordle in 4 guesses had. Again there is an increase across the board in the colors, but that is expected when there are 5 more letters to have colors compared to figure 8.3. Far more notable is the dramatic increase in the number of grey letters, increasing by most 4 letters.

Figure 8.4: Four Guess Reply

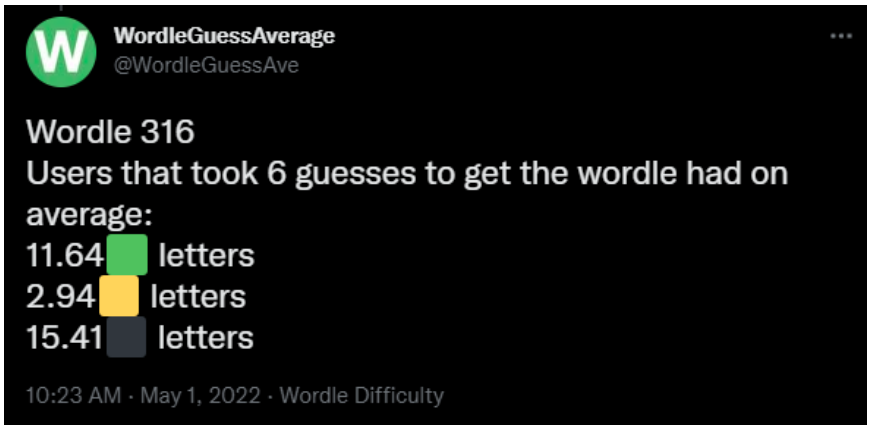


The following figure (8.5) shows the number of green, yellow, and grey letters for users that solve the wordle in 5 guesses. There is again a dramatic increase in grey letters, but interestingly there is a much larger increase in yellow letters than the last few figures.

Figure 8.5: Five Guess Reply

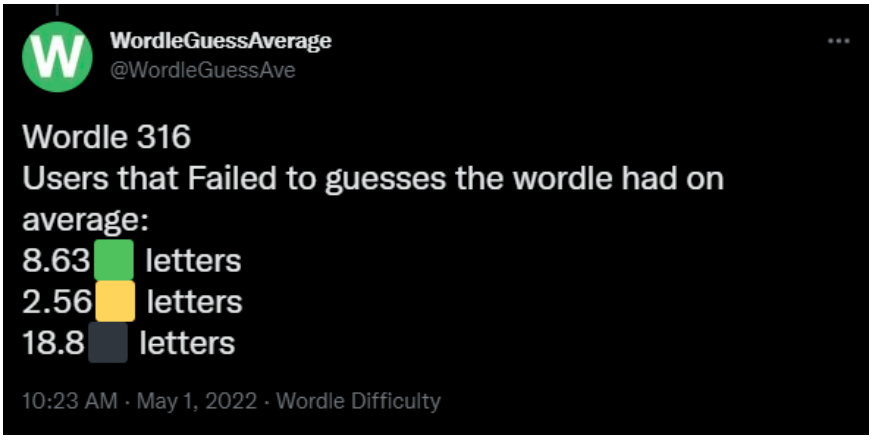
Below is figure 8.6 which contains the detailed stats for the 6 guesses users. This along with figures 8.1 to 8.7 is posted in a reply to the tweet seen in figure 8. There is another dramatic increase in grey letters, with a slight increase in green and yellow letters.

Figure 8: 6 Guess Reply



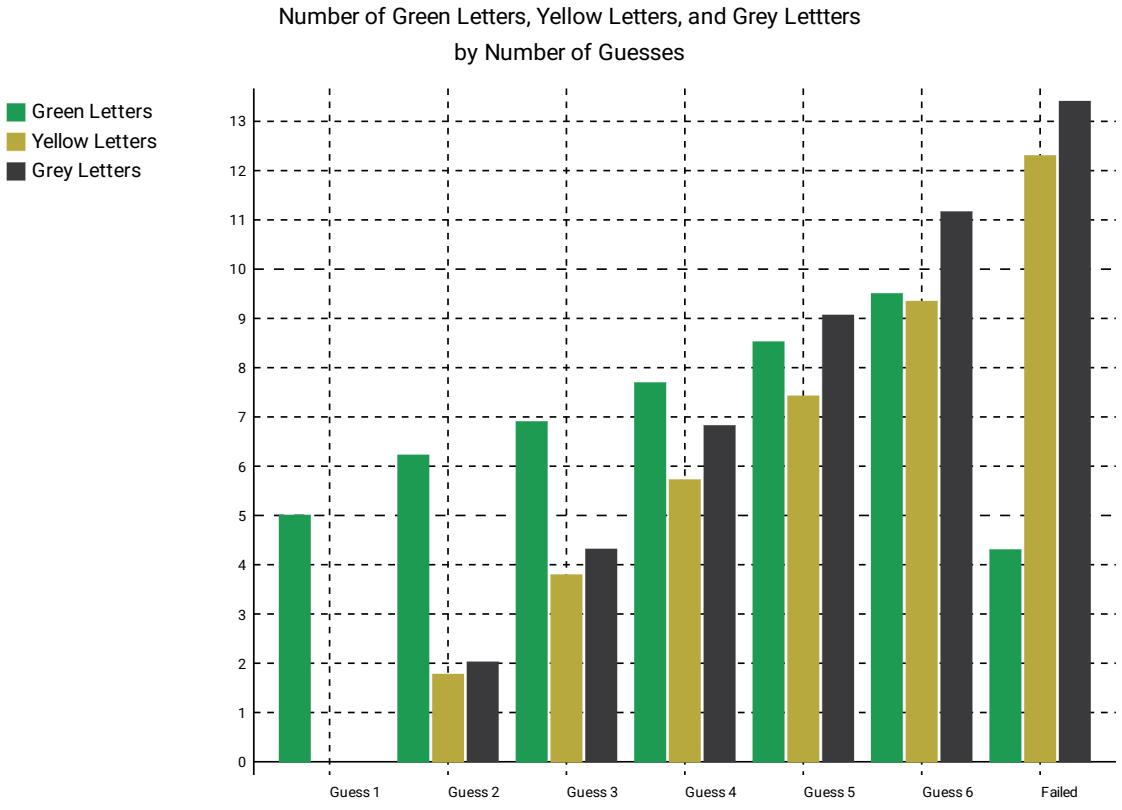
The detailed stats for users that failed to guess the wordle is very interesting as it often shows a much lower number of green letters compared to any of the other detailed stats, with far more grey letters than the other detailed stats. This all leads me to believe these users are guessing words that do not help narrow down the possible words, that also means they are likely to guess words containing letters they already know are not in the final word.

Figure 8: Failed Guess Reply



The user has the choice to create a graph with the detailed stats. I attempted to post this to twitter but, I have been unable to save this graph as a PNG so it can't be posted to twitter. The graph uses wordle colors to keep it on theme and gives a very interesting representation of the data seen in the previous tweets. This graph is not related to the data above, and was created for wordle 311. It shows that the better a user is at guessing the bigger the discrepancy between green letters and yellow letters. Interestingly it also shows that users on average always have more grey letters than yellow letters.

Figure 8: Graph for Wordle 311



This data shows interesting patterns in how people solve the wordle and help explain why a particular days wordle can be easier or harder to solve. By understanding one days wordle makes many peoples first word ineffective it can show why that wordle was more difficult. On the other hand, if the first guess is quite effective but it still takes many attempts to solve it, that days wordle could be an obscure word that many people don't think to guess. This data is very interesting to look to, and I plan on continuing to look into it.

conclusion

This project has been very enjoyable, and I believe it to be successful. I accomplished my original goal of calculating the difficulty of a wordle word and added more in the form of my wordle solver, and the details stats seen above. While I have sorted most of the errors there the program still runs into trouble if the wordle number it is searching for is too old, or in the future and as a result does not have any tweets. If the program fails to get tweets several errors occur, but I plan to add an exception that simply stops the code if it does not find enough tweets. I also plan to update the world solver to use a library called selenium to interact with the official wordle site instead of mouse and keyboard control. This is the only step holding the code back from being fully functional on Linux. I would like it to be fully run on Linux to allow me to upload it to a server and have it automatically run once a day every day without fail. I would also like to understand how to convert the SVG graph to a PNG image to allow it to be posted to twitter as it is a better representation of data than the twitter replies which can be confusing for users. Those changes will be made some time in the summer, but overall, I am very satisfied with how my project as turned out.

GitHub link: https://github.com/Acardoso21/FinalProjectEECE-WordleTwitterBot.git