Bilkent University

Computer Science Engineering Department

CS 461 - Artificial Intelligence

TERM PROJECT REPORT

Group Name: Suzzle Povler

Muhammed Safa Aşkın - 21502860

Burak Korkmaz - 21601296

Alp Ertürk - 21602659

Bahadır Tubay - 21501435

Fırat Ege Akın - 21602166

1.0 Introduction

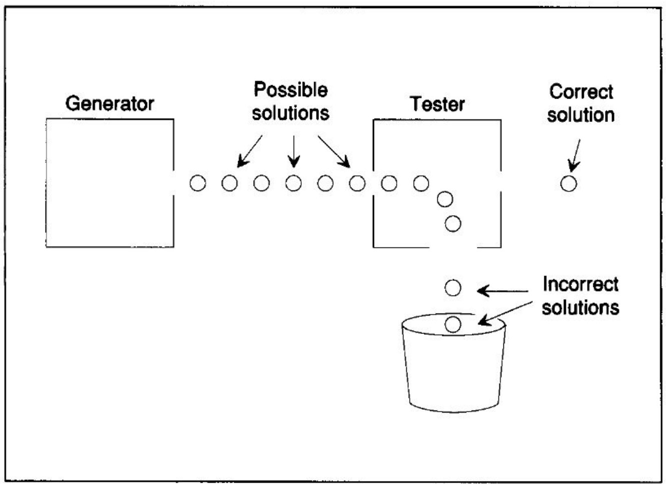
Puzzles are very popular around many people as they are mysterious, challenging and fun. Solving puzzles can enhance vocabulary knowledge and improve writing and speaking skills. That's why many people enjoy solving puzzles on a daily basis. One of the most famous newspaper, The New York Times, does publish an online puzzle called “Mini crossword puzzle”. It is designed by Joel Fagliano and it is released every day. This Puzzle contains 5 rows and 5 columns and it comes with 5 across and 5 down clues. Our team, Suzzle Polver, developed a program that is based on artificial intelligence (AI) technology that is designed to generate different clues over the available ones for any given mini crossword puzzle. The program takes advantage of many online resources, such as dictionaries, to come up with different clues while keeping the same answers to the puzzle. This way, users will be able to obtain another set of clues if they fail to solve the puzzle with existing ones. As Suzzle Polver, we believe that our program will help the future puzzle solvers by providing them with a few more changes so they can improve in this beneficial habit and continue to perform this rewarding hobby.

2.0 AI Aspect

In this section, problem-solving techniques and models of the Suzzle Polver are explained. Our problem-solving methods start with selecting sources for generating new clues. In the beginning, we decided to use qualitative sources as Merriam Webster, Wikipedia, Urban Dictionary, Word Hippo, and Anagram Smith generator. In terms of generating qualitative clues, we decided not to use sentence paraphrasing tools such as Free Article Spinner, and Seo Magnifier. These tools generate similar sentences with the original clues, and they contain security check services which cause delays, so we did not use paraphrasing tools. We implemented the Goal Tree method to indicate the steps in the clue generating process. We decided to get shorter clues as Joel Fagliano’s style. As a problem reduction method, we eliminated sentences longer than 120 words and we underscored the words that matched with the solution in these sentences. Diving the problem into smaller subproblems helped us to generate meaningful new clues for many word types. These methods and sources that will be investigated in the 2.1 Generate and Test, and 2.2 Goal Tree subsection in detail.

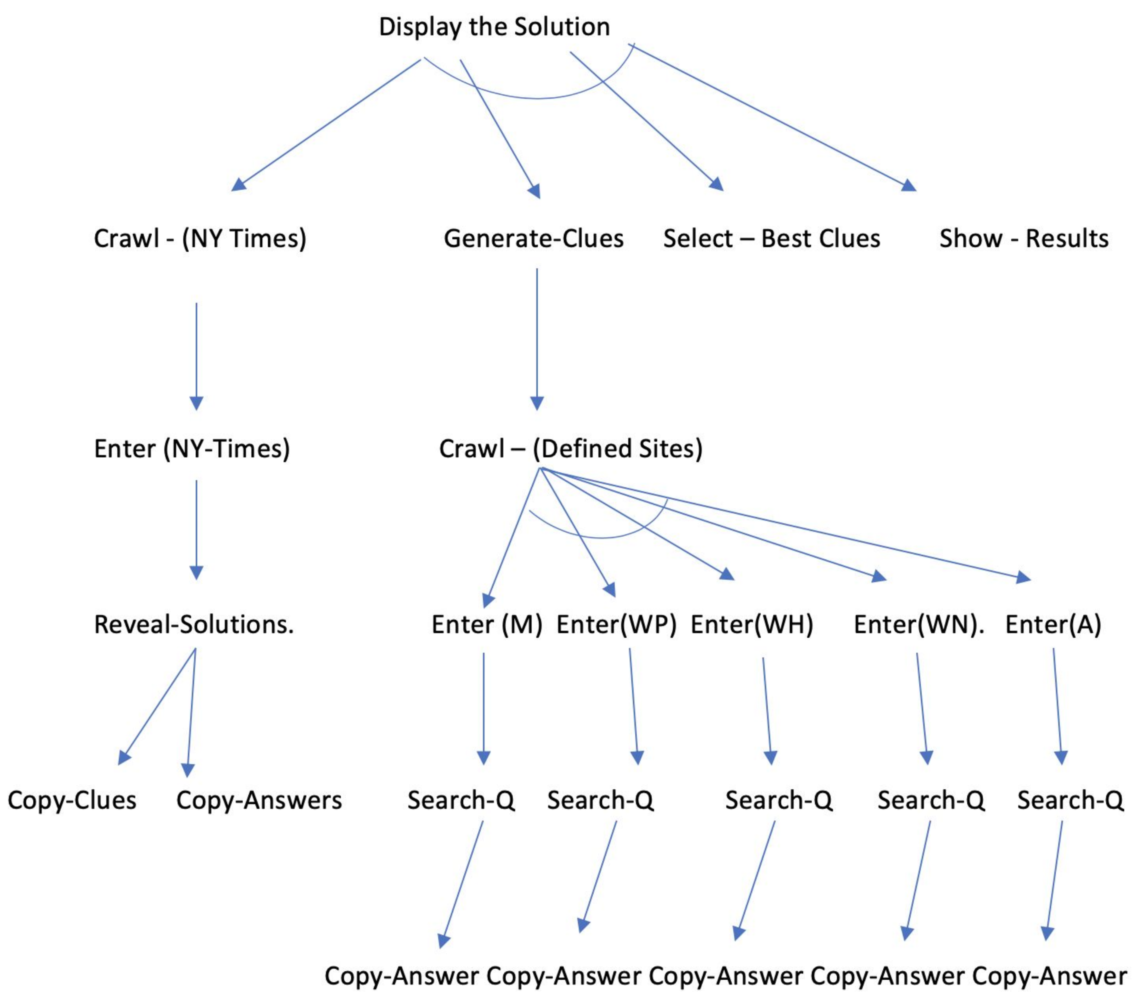
2.1 Generate and Test

The main procedure that we used in our project was to Generate and Test. Firstly, we generate possible clues from different websites. These websites are Merriam Webster, Wikipedia, Wordhippo, Urban Dictionary, and Anagram Smith. Since Merriam Webster is a dictionary, sometimes it can not find special names, plural words or place names. Another issue of Merriam Webster was, it sometimes finds very long definitions. Very long definitions are contrary to the logic of the puzzle. Therefore they are not proper clues for answers. We decided that a clue of an answer to the puzzle is long if it is larger than 120 characters. Additionally, we filtered parentheses in sentences. If resulted definitions are long or it can not find any result, we continue to test other clues that we have obtained from the mentioned websites. After Merriam Webster, we test the result that we obtained from Wikipedia. While doing that we get the first sentence in the summary part of the Wikipedia page of the searched answer. Wikipedia is useful for special names, and abbreviations. The first sentence contains the answer to the puzzle. So we eliminated the answer from the sentence and converted the position of the answer to an empty slot. In most of the cases, special name issue that Merriam Webster cannot solve is solved by Wikipedia**.** Like Merriam Webster, Wikipedia sometimes produces long sentences. If the clue we got from Wikipedia is long or it can not find the result, we continue to search the clues from Wordhippo. Wordhippo is a sentence generator tool. We searched the answer in this tool and get the sentence that Wordhippo produced. Wordhippo produces multiple sentences from the searched answer. We picked the shortest sentence from the sentence list. The picked sentence was containing the actual answer, so we eliminated the answer by converting its position to an empty slot. If we cannot find an answer from Wordhippo, we continued to search the clues from the Urban Dictionary. Urban Dictionary provided us to find clues about the slang words and cultural phrases. The last step that we used to find clues is Anagram Generator. Anagram Generator generates all anagrams of the searched word. If we cannot find any clues from previous steps we from clue by using the anagram of the word.



In each step that we used to generate a clue, if we observe a satisfactory solution, we use that solution as a clue for our puzzle. If the solution that a step is not satisfying our constraints for a clue we've eliminated these solutions and look candidate solutions that other websites produced. Therefore by using the Generate and Test Procedure, we generate the best clues for our puzzle.

2.2 Goal Tree



Abbreviations used above:

- M: Merriam Webster
- WP: Wikipedia
- WH: Word Hippo

- WN: Wordnet
- A: Anagram
- Q: Query

Explanation of Subproblems:

- Crawl - (NY Times): Crawls original clues and answers published at NY-Times crossword.
- Enter: Indicates the entering of a website.
- Generate-Clues: Generate new clues based on original ones and their answers.
- Crawl - (Defined Sites): Crawls predefined five websites.
- Search-Q: Searches the relevant query in a website.
- Copy-Answer: Copy the result of the query.
- Select - Best Clues: The selection of best clues based on the priorities of websites and character length of clues.

Detailed Explanation:

Our project was composed of lots of subproblems that we had to deal with. As our project demos directed us, we've divided it into 3 major and lots of minor subproblems. In the first project demo, we had to crawl original clues and answers from the NY-Times web site, show this crawling process clearly and appropriate user interface.

The Crawling NY Times web site was the easiest subproblem. We were already capable of using tools to crawl websites and in a couple of days, this part has accomplished. Before solving other problems we were saving all clues and answers to a text file for future use.

The Generate Clues subproblem was the hardest among there major problems as expected. First of all, we've focused on paraphrasing the original clues, but the paraphrased versions were irrelevant or very similar to the original clue. Therefore, we've eliminated this solution without losing time and started focusing on answers.

Secondly, we've focused on generating clues by using the answers instead of original clues. A lot of time spent on finding accurate websites to crawl to generate meaningful and convenient to the structure of the NY Times puzzle. In the end, we've decided 5 different web sites to search for new clues. A dictionary, Merriam Webster, encyclopedia, Wikipedia, one of the largest word database, Wordnet, an anagram generator, inges-anagram, and sentence generator, Wordhippo. After deciding which websites will be used crawling them and getting new clues were straightforward.

Our third major subproblem was selecting the best clues. After successfully implementing the crawling mechanism and taking all the results from defined websites we've started doing experiments about the quality of clues that are gathered from these sites. We've tried to increase the quality of resulting clues. In that aspect we've spent time understanding the structure of the original NY Times clues structure, it is seen that best clues in terms of similarities to the original clues were coming from Wikipedia and Merriam Webster. Therefore, they are assigned as higher priority clue sources. Although it came up with a solution in every trial, anagram generator has assigned as a lower priority because the structure of anagram clues was not fitting the original structure of clues. After completing these decisions, the resulting priority order of websites as follows, Merriam Webster, Wikipedia, Word Hippo, Wordnet, Anagram generator. After the first demo, we've designed this selecting mechanism again with respect to the feedback. Because even though the system was working perfectly and the resulting clues were not perfect yet. As the opposite of original clues, the length of recently created clues was very long and the crawling mechanism was not working perfectly. We've decreased the length of sentences by extracting the parentheses. We've added a length constraint in order to force the system to eliminate clues longer than 120 characters and the resulting clues reached a better form. Lastly, we've fixed the bugs of crawling mechanism and limited word filling type of questions with three.

The showing results subproblem has been solved faster than crawling and selecting the best clues subproblems. Since that part requires less effort the expectations were higher, we've tried place the original clues and answers similar to the original NY Times website and after first feedback, we've spent time on making our grid composed of perfect squares instead of rectangular boxes. After locating the recently generated clues to the bottom of the original ones this subproblem has solved. That was the last subproblem of the supreme problem "Displaying Solution". Therefore, our term project is concluded. The screenshots of the resulting problem are listed in the appendix.

3.0 Future Work

Different methods which are related to described AI aspects in recent sections used in Suzzle Polver in order to generate clues by using answers and original clues of the New York Times Crossword

Puzzle by checking different sources from the internet. This section contains information about how Suzzle Polver can be improved to generate clues that are more accurate and similar to the original clues in terms of length, language and style. In order to increase the accuracy of the new clues, examining a large data set that contains crossword puzzles both clues and answers from different sources can be done as a first step [2]. Crossword puzzles mostly include clues and answers which are related to the daily life of people. By examining older clues and answers generating similar and related clues as possible. Separate modules should be implemented to examine data and create new clues.

Generated clues may be full sentences, abbreviations, incomplete sentences, synonyms, etc just like an original clue. In most basic sense implementation of two different modules can be optimal for improvement of this project. The first module should deal with large data sets obtained from other crossword puzzles where the second module should be able to create sentences, search for synonyms, opposite meanings of words and also check corner cases like words only used in idioms and daily life language. The second module may use Frame Theory which is an AI approach to generalize linguistic notions with the case of grammar. Which makes a possible representation of both syntactic and procedural knowledge fro parsing and understanding related semantic, contextual sentences [3]. So the second module should able to understand close sentences, clues in terms of domain and context by parsing clues and answers obtained by using the first module. In Suzzle Polver we used different methods for clue generation but it is believed that the accuracy of clues will be increased by implementing and using these two new modules.

4.0 Conclusion

The power of artificial intelligence should not be underestimated as if they are designed and implemented in a proper way. AI’s can accomplish many tasks in a very short period of time compared to humans. In order to be sure about the accuracy of any AI, designers should create a program such that it attaches importance to 2 factors: methods and resources. There are many available methods to solve a problem, in our project we’ve focused on generate and test or goal tree . However, there will always be a dead-end for certain situations. A designer should be aware of this by implementing backup methods so to increase the accuracy of a program. Moreover, benefiting from a few numbers of resources can be restrictive. Searching many more sources and comparing results will assure designers about the legitimacy of the program. This project was very informative about AI so that, as Suzzle Polver, we comprehended what we can achieve with AI technology. However, this is just the tip of the iceberg.

This project reports work done in partial fulfillment of the requirements for CS 461 -- Artificial Intelligence. The software is, to a large extent, original (with borrowed code clearly identified) and was written solely by members of the SUZZLE POLVER.

Word Count: 2044

References

1. Blockeel, Hendrik, and Luc De Raedt. "Top-down induction of first-order logical decision trees." *Artificial intelligence* 101.1-2 (1998): 285-297.
2. Littman, Michael L., Greg A. Keim, and Noam Shazeer. "A probabilistic approach to solving crossword puzzles." *Artificial Intelligence* 134.1-2 (2002): 23-55.
3. Goldstein, Ira, and Seymour Papert. "Artificial intelligence, language, and the study of knowledge." *Cognitive science* 1.1 (1977): 84-123.

APPENDIX

Test Runs:

Source Code:

find_sol.py

```
from selenium import webdriver
```

```
import os
import numpy as np
from get_clues import *
from operator import itemgetter
import time
chrome_options = webdriver.ChromeOptions()

#To hide chrome comment out the following line
#chrome_options.add_argument('--headless')
browser = webdriver.Chrome(os.path.abspath(os.curdir)+'/chrome_driver/chromedriver
78',options=chrome_options)

username = ""
password = ""
#Log in
browser.get('https://www.nytimes.com/crosswords/game/mini')
browser.find_element_by_xpath("//*[@id=\"root\"]/div/div/div[1]/header/div[3]/div/a[5]").click()
browser.find_element_by_xpath("//*[@id=\"username\"]").send_keys(username)
browser.find_element_by_xpath("//*[@id=\"password\"]").send_keys(password)
browser.find_element_by_xpath("//*[@id=\"myAccountAuth\"]/div[1]/div/form/div/div[5]/button").click()
time.sleep(30)
browser.get('https://www.nytimes.com/crosswords/game/mini/2019/11/04')

try:

browser.find_elements_by_xpath('//*[@id="root"]/div/div/div[4]/div/main/div[2]/div/div[2]/div[3]/div/article/div[
2]/button/div/span')[0].click()

browser.find_elements_by_xpath('//*[@id="root"]/div/div/div[4]/div/main/div[2]/div/div/ul/div[2]/li[2]/button')[0]
.click()

browser.find_elements_by_xpath('//*[@id="root"]/div/div/div[4]/div/main/div[2]/div/div/ul/div[2]/li[2]/ul/li[3]/a')
[0].click()
  browser.find_elements_by_xpath('//*[@id="root"]/div/div[2]/div[2]/article/div[2]/button[2]/div/span')[0].click()
  browser.find_elements_by_xpath('//*[@id="root"]/div/div[2]/div[2]/span')[0].click()
except:
  print("Already revealed")
try:
  today =
browser.find_element_by_xpath("//*[@id=\"root\"]/div/div/div[4]/div/main/div[2]/header/div/div/div/div[1]").text
except:
  browser.quit()
```

```
clues_across = list()
clues_down = list()

clues_across =
browser.find_elements_by_xpath('//*[@id="root"]/div/div/div[4]/div/main/div[2]/div/article/section[2]/div[1]/ol')[
0].text.split('\n')
clues_down =
browser.find_elements_by_xpath('//*[@id="root"]/div/div/div[4]/div/main/div[2]/div/article/section[2]/div[2]/ol')[
0].text.split('\n')

print("Getting Original Clues from: nytimes.com\n")
print("Across Clues\n",clues_across)
print()
print("Down Clues\n",clues_down)

solutions = []



for i in range(25):
  css_link = '#xwd-board > g:nth-child(5) > g:nth-child('+str(i+1)+')'
  #print(css_link)
  if not browser.find_elements_by_css_selector(css_link):
     solutions.append('')
  else:
     solutions.append(browser.find_elements_by_css_selector(css_link)[0].text.split("\n"))

sol = list()
loc = list()
counter = 0
for i in solutions:
  if i[0].isdigit():
     sol.append(i[1])
     loc.append(i[0])
  else:
     sol.append(i[0])
     loc.append('')
  counter += 1

sol = np.array(sol).reshape(5,5)
loc = np.array(loc).reshape(5,5)

accross_solution = list()
```

```
for i in range(5):
  for j in range(5):
     if loc[i][j].isdigit() == True:
        accross_solution.append(loc[i][j])
        accross_solution.append(''.join(map(str,sol[i])))
        break

print("\nGetting Solutions from: nytimes.com\n")
print("Accross solutions:\n",accross_solution,"\n")

down_solution = list()
for i in range(5):
  for j in range(5):
     if loc[j][i].isdigit() == True:
        down_solution.append(loc[j][i])
        down_solution.append(''.join(map(str,sol[:,i])))
        break

temp = list()
for i in range(0,len(down_solution),2):
  temp.append((int(down_solution[i]),down_solution[i+1]))


temp = sorted(temp,reverse = False)

for i in range(0,len(down_solution),2):
  down_solution[i],down_solution[i+1] = str(temp[int(i/2)][0]),temp[int(i/2)][1]

print("Down solutions:\n",down_solution)




output = open("output.txt",'w')
output.writelines(str(clues_across))
output.write('\n')
output.writelines(str(clues_down))
output.write('\n')
output.writelines(str(solutions))
output.write('\n')
output.close()

new_clues_accross = list()
```

```
new_clues_down = list()

for i in range(0,len(accross_solution),2):
    print()
    print("Step1 - From Meriam Webster Finding clue for ", accross_solution[i+1])
    new_clues_accross.append(accross_solution[i])
    clue = merriam_webster(browser,accross_solution[i+1])
    if clue!=False and (accross_solution[i+1][-1] != "S" or len(accross_solution[i+1])<=3) and len(clue) < 150:
        new_clues_accross.append(clue)
    else:
        print()
        print("Step2 - From Wikipedia Finding clue for ", accross_solution[i+1])
        clue = wiki(accross_solution[i+1])
        if clue != False and (accross_solution[i+1][-1] != "S" or len(accross_solution[i+1])<=3)  and len(clue) < 150:
            new_clues_accross.append(clue)
        else:
            print()
            print("Step3 - From Word Hippo Finding clue for ", accross_solution[i+1], "By creating sentences with
answer")
            clue = word_hippo(browser,accross_solution[i+1])
            if clue != False:
                new_clues_accross.append(clue)
            else:
                print()
                print("Step4 - From Anagram Smith Finding clue for ", accross_solution[i+1])
                clue = anagramSmith(browser,accross_solution[i+1])
                if clue != False:
                    new_clues_accross.append(clue)
                else:
                    new_clues_accross.append(" ") #Wordnet


    print("New clue is: ",clue)

for i in range(0,len(down_solution),2):
    print()
    print("Step1 - From Meriam Webster Finding clue for ", down_solution[i+1])
    new_clues_down.append(down_solution[i])
    clue = merriam_webster(browser,down_solution[i+1])
    if clue!=False and (down_solution[i+1][-1] != "S" or len(down_solution[i+1])<=3) and len(clue) < 150:
        new_clues_down.append(clue)
    else:
        print()
        print("Step2 - From Wikipedia Finding clue for ", down_solution[i+1])
        clue = wiki(down_solution[i+1])
        if clue != False and (down_solution[i+1][-1] != "S" or len(down_solution[i+1])<=3) and len(clue) < 150:
            new_clues_down.append(clue)
        else:
            print()
            print("Step3 - From Word Hippo Finding clue for ", down_solution[i+1], "By creating sentences with
answer")
            clue = word_hippo(browser,down_solution[i+1])
            if clue != False:
                new_clues_down.append(clue)
            else:
                print()
                print("Step4 - From Anagram Smith Finding clue for ", down_solution[i+1])
                clue = anagramSmith(browser,down_solution[i+1])
                if clue != False:
                    new_clues_down.append(clue)
                else:
                    new_clues_down.append(" ") #Wordnet

    print("New clue is: ",clue)

browser.quit()
```

get_clues.py

```
from selenium import webdriver
import os
import numpy as np
import textwrap
import time
import wikipedia
import time
import nltk as ntlk
from nltk.corpus import wordnet
import re

def clear(sentence, word):
```

```python
    newsentence = ""
    for i in sentence.split(" "):
        if word.lower() == i.lower() or (i[:-1].lower() == word.lower() and (i[-1] == "." or i[-1] == ",")):  # see if one of the words in the sentence is the word we want
            i = i.lower().replace(word.lower(), "____")
            #print(replaced)
        newsentence += i + " "
    #print(newsentence)
    if newsentence.find("(") != -1 and newsentence.find(")") != -1:
        newsentence = newsentence[0:newsentence.find("(")] + newsentence[newsentence.find(")")+2:]
    if newsentence.find("\n") != -1:
        newsentence = newsentence[:newsentence.find("\n")+1]
    return newsentence.capitalize()



def word_hippo(browser,word):

    #chrome_options = webdriver.ChromeOptions()
    #browser = webdriver.Chrome(os.path.abspath(os.curdir)+'/chromedriver 78',options=chrome_options)

    new_clues = list()

    try:
        browser.get('https://www.wordhippo.com/what-is/sentences-with-the-word/' + word + '.html')
        sentence = browser.find_element_by_xpath("/html/body/div[3]/table/tbody/tr[2]/td/table/tbody/tr/td/div/table/tbody/tr[1]/td[2]/div/table/tbody/tr/td/div[2]/table/tbody/tr[2]").text.splitlines()
    except:
        print("No solution in word hippo")
        return False
    for i in range(len(sentence)):
        sentence[i] = clear(sentence[i], word)
        if len(sentence[i]) < 140:
            return sentence[i]
    return sentence[0]



def wiki(word):
    try:
        results_of_search = wikipedia.search(word)
        if("disam" in results_of_search[0] or len(results_of_search)>1):
            first_result = wikipedia.summary(results_of_search[1], sentences=1)
        else:
            first_result = wikipedia.summary(results_of_search[0], sentences=1)

    except:
        print("No solution in wikipedia")
        return False

    first_result = first_result.lower()
    word = word.lower()
    first_result = clear(first_result,word)
    return first_result



def anagram(browser,word):
    #Setting browser
    #chrome_options = webdriver.ChromeOptions()
    #To hide chrome comment out the following line
    #chrome_options.add_argument('--headless')
    #browser = webdriver.Chrome(os.path.abspath(os.curdir)+'/chromedriver79',options=chrome_options)

    browser.get('https://ingesanagram.appspot.com')
    browser.find_element_by_xpath("/html/body/div/div/div[1]/div[1]/div/div[2]/div/input").send_keys(word)
    browser.find_element_by_xpath("/html/body/div/div/div[1]/div[1]/div/div[3]/div[1]/button").click()
    time.sleep(2)
    clues = browser.find_element_by_xpath("/html/body/div/div/div[1]/div[3]/div").text.splitlines()
    print (clues)




def findIndex(word, list):

    for element in list:
        if(word in element):
            return list.index(element)

    return -1
```

```python
def anagramSmith(browser,word):

    #Setting browser
    #chrome_options = webdriver.ChromeOptions()
    #To hide chrome comment out the following line
    #chrome_options.add_argument('--headless')
    #browser = webdriver.Chrome(os.path.abspath(os.curdir)+'/chromedriver 78',options=chrome_options)

    try:
        browser.get('https://new.wordsmith.org/anagram/anagram.cgi?anagram=' + word)
        #To disable popup
        try:
            browser.find_element_by_xpath("//*[@id=\"t402-prompt\"]/div[2]/div[3]/a[2]").click()
        except:
            print("No pop up")

#browser.find_element_by_xpath("/html/body/topbar/blockquote/form/table/tbody/tr[1]/td[1]/input").send_keys(word)
        #browser.find_element_by_xpath("/html/body/topbar/blockquote/div/div").click()
        clues = browser.find_element_by_xpath("/html/body/topbar/blockquote/div").text.splitlines()
    except:
        print("No solution in anagram smith")
        return False

    #finds the position of Displaying elements label
    index = findIndex("Displaying", clues)

    #prints out the first element after displaying label on page
    return "Anagram of " +  word + " is " + clues[index+2].lower()



def wordnetFunction(clue):
    for word in clue:
        synset = wordnet.synsets(word)
        if(len(synset) != 0):
            print('Word and Type : ' + synset[0].name())
            print('Synonym of ' + word + ' is: ' + synset[0].lemmas()[0].name())
            print('The meaning of the word : ' + synset[0].definition())
            print('Example of ' + word + ' : ' + str(synset[0].examples()))



def merriam_webster(browser,word):
    new_clues = list()

    try:
        browser.get('https://www.merriam-webster.com/')
        browser.find_elements_by_xpath('//*[@id=\"s-term\"]')[0].send_keys(word)
        browser.find_element_by_xpath("//*[@id=\"mw-search-frm\"]/div[1]/div[2]").click()
        new_clues =
list(browser.find_element_by_xpath("//*[@id=\"dictionary-entry-1\"]/div[2]").text.splitlines()[1:])

new_clues.append(browser.find_element_by_xpath("//*[@id=\"dictionary-entry-1\"]/div[2]/div/span/div/span/spa
n").text)
        try:
            new_clues.append(browser.find_element_by_xpath("//*[@id=\"dictionary-entry-1\"]/div[2]/div").text)
        except:
            pass
        try:

new_clues.append(browser.find_element_by_xpath("//*[@id=\"dictionary-entry-1\"]/div[2]/div/span/div/span/spa
n/span/span[2]").text)
        except:
            pass
    except:
        print("No solution in marrian webster")
        return False


    #print(new_clues)

    for i in range(len(new_clues)-1,-1,-1):
        """if new_clues[i][0]=="—" and i!=0:
            new_clues.remove(new_clues[i])
        else"""

        for k in new_clues[i]:


            if new_clues[i].count(" ")<3 :
                new_clues.remove(new_clues[i])
                break
```

```
  for i in range(len(new_clues)):
    if new_clues[i].find(":") != -1:
      new_clues[i] = new_clues[i][new_clues[i].find(":")+2:]

    if new_clues[i].find("sense") != -1:
      new_clues[i] = new_clues[i][:new_clues[i].find("sense")]

    new_clues[i] = new_clues[i].replace(":",". ")



  for i in range(len(new_clues)):
    new_clues[i] = clear(new_clues[i],word)

   if len(new_clues) == 0:
    return False

  for i in range(len(new_clues)):
    if len(new_clues[i]) < 120 and new_clues[i][0].isalpha()==True and new_clues[i].count(".") == 0 and word
not in new_clues[i] and "_" not in new_clues[i]:
      new_clues[i] = new_clues[i][:-1] + "."
      return new_clues[i].capitalize()  #textwrap.fill('. '.join(map(str, new_clues)),50)

  return False
# Test cases
"""
chrome_options = webdriver.ChromeOptions()
#To hide chrome comment out the following line
#chrome_options.add_argument('--headless')
browser = webdriver.Chrome(os.path.abspath(os.curdir)+'/chromedriver 78',options=chrome_options)
print(merriam_webster(browser,"vegan"))
#print(word_hippo(browser,"OBAMA"))
#print(wiki("nudes"))
#print(anagramSmith(browser,"nudes"))
"""
```

displayPuzzle.py

```
from tkinter import *
from find_sol import *
from datetime import date
```

```
import textwrap

mainwindow = Tk()
mainwindow.title("Suzzle Polver")
mainwindow.geometry("2000x1000")


leftFrame = Frame(mainwindow)
leftFrame.pack(side = LEFT,padx = 50)
rightFrame = Frame(mainwindow)
rightFrame.pack(side = LEFT)



for i in range(25):

  #print(solutions[i][0])
  if solutions[i][0] == '':
     l1 = Label(leftFrame,bd=1,font="Times 30",height = 3,width =7,background="black",justify=LEFT)
  else:
     try:
        int(solutions[i][0])

        sol = " " +solutions[i][0] +"\n      " + solutions[i][1]
        l1 = Label(leftFrame,bd=1,font="Times 30",text=sol,relief="solid",height = 3,width
=7,background="white",justify=LEFT,anchor=NW)

     except:
        sol = "\n      " + solutions[i][0]
        l1 = Label(leftFrame,bd=1,text=sol,font="Times 30",relief="solid",height = 3,width
=7,background="white",justify=LEFT,anchor=NW)
  l1.grid(row=1+int(i/5),column=1+int(i%5))


clues_accross_txt = ""
for i in range(0, len(clues_across), 2):
  clues_accross_txt += clues_across[i] + " - " + textwrap.fill(clues_across[i+1],60) + "\n\n"


clues_down_txt = ""
for i in range(0, len(clues_down), 2):
  clues_down_txt += clues_down[i] + " - " + textwrap.fill(clues_down[i+1],60) + "\n\n"
```

```
new_clues_accross_txt = ""
for i in range(0, len(new_clues_accross), 2):
  new_clues_accross_txt += new_clues_accross[i] + " - " + textwrap.fill(new_clues_accross[i+1],60) + "\n\n"



new_clues_down_txt = ""
for i in range(0, len(clues_down), 2):
  new_clues_down_txt += new_clues_down[i] + " - " + textwrap.fill(new_clues_down[i+1],60) + "\n\n"

lab_across = Label(rightFrame,text="Accross",justify=LEFT,font ="Times 30 bold",anchor=N)
lab_across.grid(row = 0, column = 1,sticky=NW)

lab_across = Label(rightFrame,text=clues_accross_txt,justify=LEFT,font ="Times 16",anchor=N)
lab_across.grid(row = 1, column = 1,sticky=NW)

lab_down = Label(rightFrame,text="Down",justify=LEFT,font="Times 30 bold",anchor=N)
lab_down.grid(row = 0, column = 2,sticky=NW)

lab_down = Label(rightFrame,text=clues_down_txt,justify=LEFT,font="Times 16",anchor=N)
lab_down.grid(row = 1, column = 2,sticky=NW)

new_lab_across = Label(rightFrame,text="New Accross",justify=LEFT,font ="Times 30 bold",anchor=N)
new_lab_across.grid(row = 2, column = 1,sticky=NW)

new_lab_down = Label(rightFrame,text=new_clues_accross_txt,justify=LEFT,font="Times 16",anchor=N)
new_lab_down.grid(row = 3, column = 1,sticky=NW)

new_lab_across = Label(rightFrame,text="New Down",justify=LEFT,font ="Times 30 bold",anchor=N)
new_lab_across.grid(row = 2, column = 2,sticky=NW)

new_lab_down = Label(rightFrame,text=new_clues_down_txt,justify=LEFT,font="Times 16",anchor=N)
new_lab_down.grid(row = 3, column = 2,sticky=NW)

#today = date.today()
#today = today.strftime("%d %B, %Y")
today += " Suzzle Polver"
dayLabel = Label(mainwindow,text = today, bd = 1,font="Times 22",height = 2,width =40,background="white")
dayLabel.place(x=150,y=750,in_=mainwindow)
mainwindow.mainloop()
```