BILKENT UNIVERSITY

CS 461 - ARTIFICIAL INTELLIGENCE

TERM PROJECT

Instructor: Varol Akman

GROUP: ASOFT

Taner Durmaz	21503008
Samir İbrahimzade	21701339
Ayberk Görgün	21201986
Ömer Olkun	21100999
Fırat Deniz Müftüoğlu	21301962

TABLE OF CONTENTS

1. Introduction

Artificial intelligence (AI) is the study and design of algorithms that perform tasks or behaviors that a person could reasonably deem to require intelligence if a human were to do it. [1] Similar to how machines take on the physical work of humans, today we rely on AI to cope with more sophisticated intellectual problems. Crossword puzzles are the most popular form of linguistic puzzles; for the solver they are intellectually challenging and entertaining as well. [2] In the scope of this project, an algorithm which is capable of solving New York Times' daily crossword puzzle is demanded. There are many studies on AI, for puzzle solving or puzzle generation. Our work focuses on a word-based approach to satisfy different constraints which are given us by the puzzle. The program will retrieve the required puzzle data like the grid and clues from New York Times website and it will find the best option through the candidate words; such that, the word should satisfy the meaning of given clue, have the required number of characters in it (according to the given space for it to fit in the puzzle grid) and it must be checked that the characters should be in a coherence with the other words. To simplify, it is a multi-layered constraint satisfaction problem.

Our approach to the problem consists of multiple searches on different platforms. For the implementation part, DataMuse, Google Search, Reverse Dictionary and Merriam Webster will be explained in the continuation of the report. DataMuse comes handy when it is needed to do constraint based word search which is significantly critical for this project. On the other hand, even though Google Search offers unrelated results for searching a word, it is practical to use as a verification. Reverse Dictionary and Merriam Webster offer different datas might be useful for the task. However, we were unable to utilize them in the given time which will be explained in detail, in section 2.1.3. By using these tools, a set of words will be obtained as candidates. However, to accomplish the task, it is needed to assess the resources and to apply results. For this reason, clues are sorted and categorized according to their types in a rule-based deduction system. Then according to their categories different search methods are being used. A* algorithm is used to decrease solving time in this procedure. This process and AI approaches will be

explained in detail in Section 2.2. Lastly, test runs and their results in a graphical user interface will be given.

2. Implementation

The implementation of the puzzle solver project included two separate parts: scraping the puzzle grid details with across and down clues and to solve the puzzle using the given clues. Getting details of the puzzle was done with the help of libraries such as BeautifulSoup and Selenium. In this section, the clue searching and our AI approaches in order to solve the puzzle are explained.

2.1 Searching Clues in Different Sources

We used 4 sources for the clue search: Datamuse, Google, Merriam Webster and Reverse Dictionary. For the three of them we found an API, which made it comfortable to send the requests and get the response data but for Reverse Dictionary, we used selenium library to send requests for each clue and then parse the data.The details of the sources which are used in clue search and the way how we used them are explained below.

2.1.1 DataMuse API

We chose DataMuse because it offers an API and accepts sentences as input to search similar words with their corresponding scores showing how much words related to the input. DataMuse also offers constraint based search, for example a search of words with 5 letters and starting with the letter “v”. We used both constraint based search and the similar words search. Thus we wrote two separate functions for each search method, searching similar words function takes the clue as an input then returns related words with less than 5 letters, among DataMuse’ suggestions. Constraint based search takes the string with already found letters as input such as “b?u?” then returns the list of words DataMuse suggests, in this case top suggested words are “blue” and “blur”. At first we tried to include clues too along with constraints in constraint based search, but most of the time DataMuse suggested too few or unrelated words so we decided to exclude the clues in constraint based search.

The use of api is straightforward, we are sending requests to "https://api.datamuse.com/words" with addition of function and parameter info such as "?ml=fantasy+football+deal" for similar word search by giving the clue of "fantasy football deal". For the constraint based search our request is "?sp=b?u?", which means suggest words with 4 letters whose first letter is "b" and third letter is "u".

2.1.2 Google Search

The Google search function makes a search request to Google, then processes the response using BeautifulSoup. After parsing the words using the BeautifulSoup, functions adds the words into a list and returns it. The issue with this method is, the resulting words were mostly unrelated and we couldn't determine which word is the possible answer. But we planned to use this function as a way of verification and kind of possible heuristic.

2.1.3 Reverse Dictionary and Merriam Webster

Reverse Dictionary uses WordNet as a main source but it also searches the given sentence in multiple other databases, therefore instead of using WordNet directly we decided to send html requests to the reversedictionary.org to search clues as sentences then parsed the response using BeautifulSoup.

For Merriam Webster we used their API for searching synonyms of each word in the clues separately. As we examined the clues and the answers, we found that the clues which are asking for the synonym or similar word include just a few words, therefore it was not beneficial to use Merriam Webster for the clues with 3 words or more.

Even though both Reverse Dictionary and Merriam Webster offered various data with promising potentials we were unable to utilize it properly. Kory Stamper, a Merriam Webster lexicographer says that, "Dictionaries job isn't to say what is, but to objectively and comprehensively catalog the many different ways words are used by real people." [3] Merriam Webster offers their extensive and unique archive for the public use, which enables alternative ways to solve problems including our mini puzzle. But since we failed to utilize the sources properly, outputs of Reverse Dictionary and Merriam Webster decreased the overall performance in our test runs.

To achieve better results we decided to exclude both search functionalities from the program, though they were included in our AI approach for solving the puzzle.

2.2 Using AI Approaches to Solve the Puzzle

In the previous section we illustrated the resources that we used for searching the words according to the given clues. However, it was not possible to guess the correct solution of the puzzle just collecting the set of words. Dem Amlen in *How to Solve the New York Times Crossword* article stated that, “A crossword puzzle is not a test of intelligence, and solving is not really about the size of your vocabulary. Becoming a good solver is about understanding what the clues are asking you to do.” [4] Therefore, our first aim was to make clues understandable to the software instead of collecting the large amount of data(candidate words) for each clue. Our approach to this problem was rule-based deduction systems. In the rule base given below, x represents the clue sentence as a string and C1, C2, C3 means clue type1, clue type 2, clue type 3, respectively.

If	?x has __	R1
Then	x is a fill the gap question	
If	?x has less than 3 words	R2
Then	x is a synonym or similar word question	
If	?x has more than 2 words	R3
Then	x is a general knowledge question	
If	?x has more/less/very	R4
Then	x is a synonym or similar word question	
If	?x is a fill the gap question	R5
Then	x is a C1	

If ?x is a synonym or similar word question R7

Then x is a C2

If ?x is a general knowledge question R8

Then x is a C3

After finding the types of all clues, our idea is to search clues in one or more sources, respective to its type. The clues which are type C1, are searched on Google and Datamuse, consequently for type C2 clues, we use Merriam Webster, Datamuse APIs and for C3 clues, Reverse Dictionary, Datamuse and Google are the search source for us.

At this point, after finishing the clue sorting and searching part, we get a list of words for each across and down clue which are returned from the source APIs and web sites.

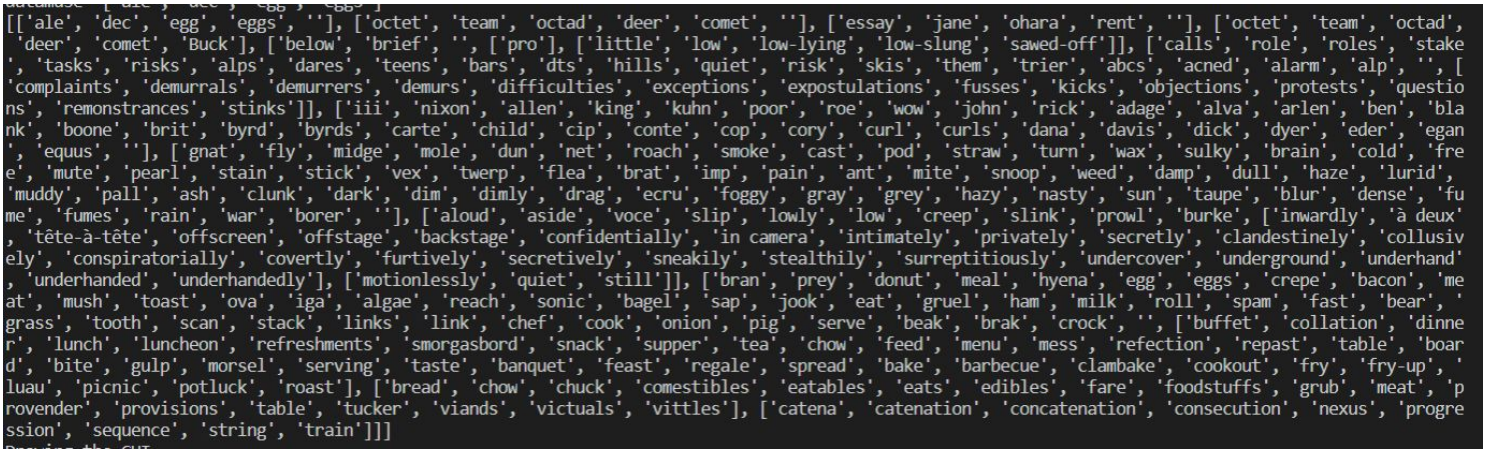
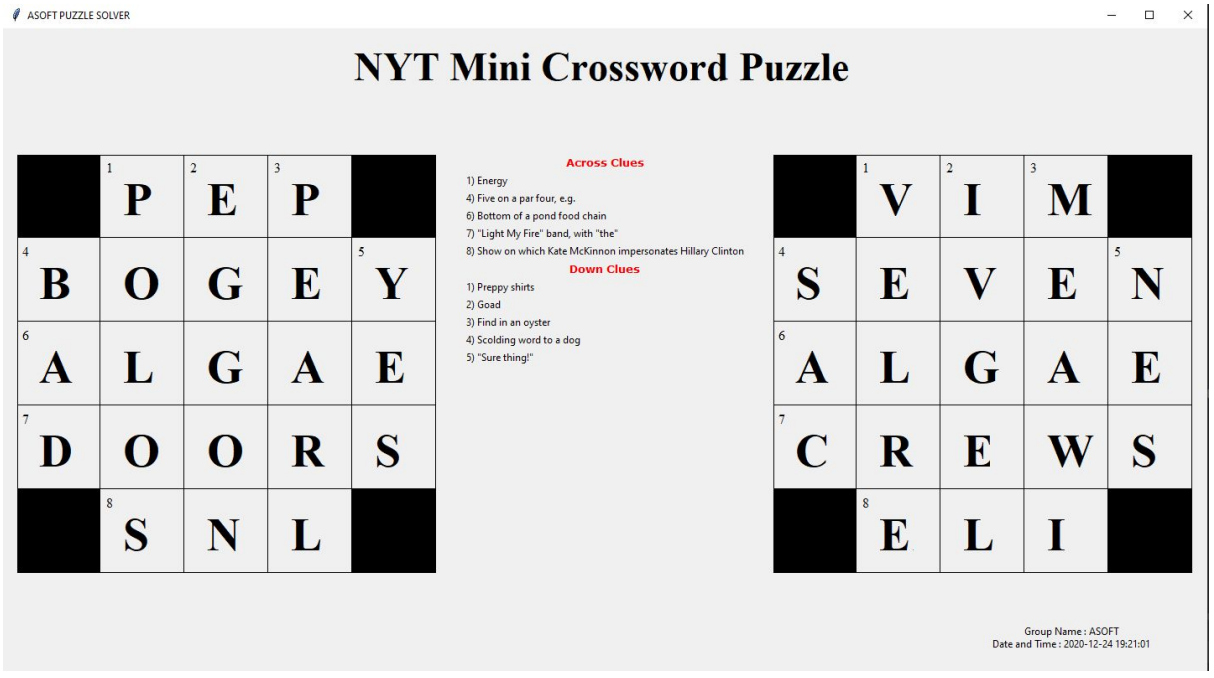


Figure 1: List of all candidate words

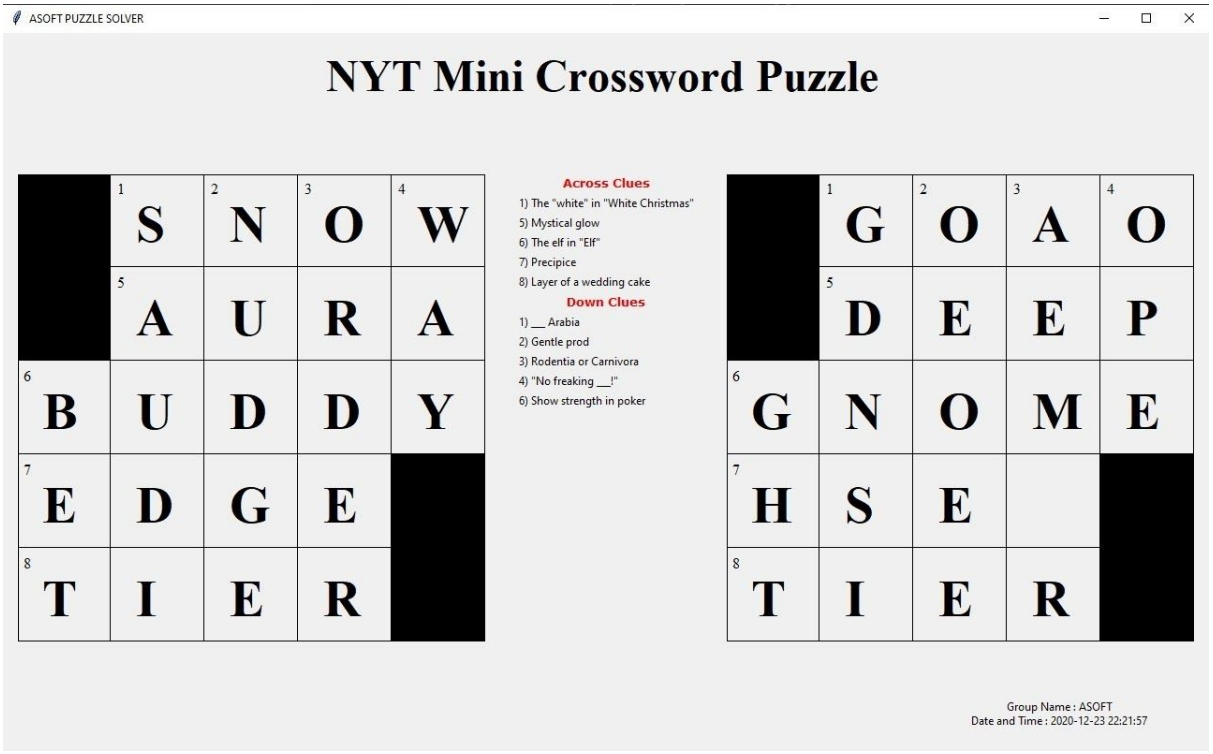
As it can be seen from the figure 1, the list can become larger, which makes the brute force solution very inefficient to implement for finding the words that together can fill the grid of puzzles. Therefore, we decided to use A* search algorithm to

decrease the number of compares and the solving time of the application. The important part of implementing A* algorithm was deciding how to distribute weights of the nodes in order to sort the queue with the least-cost path in front; our approach was to sort the paths in queue according to the sum of source weights(for each clue type we mentioned above, we assigned a source weights based on the search place – for example, Google search's match rate is higher than Datamuse's search for fill the gap questions, that's why the weight of C1 type of question that found in Google is more than the ones that found in Datamuse) and the rank inside the source(the sources – except Merriam Webster – that we use for searching clues, sort the found words by their relevance to the clue).

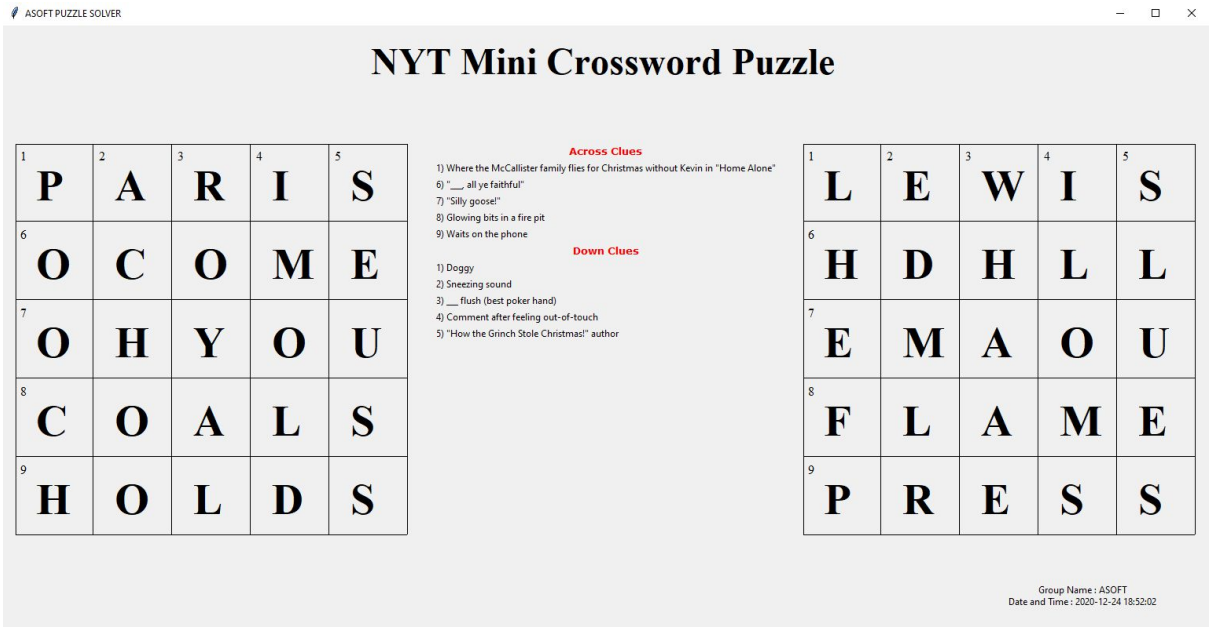
3. Test Runs and Results



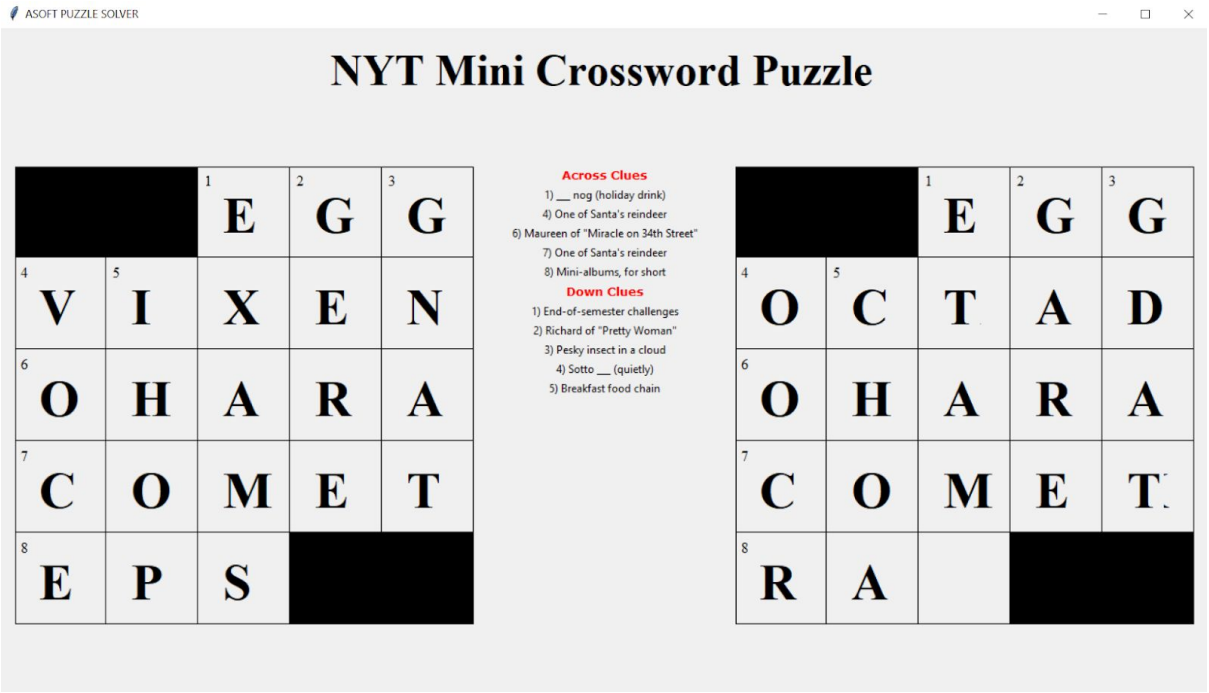
06.01.2016



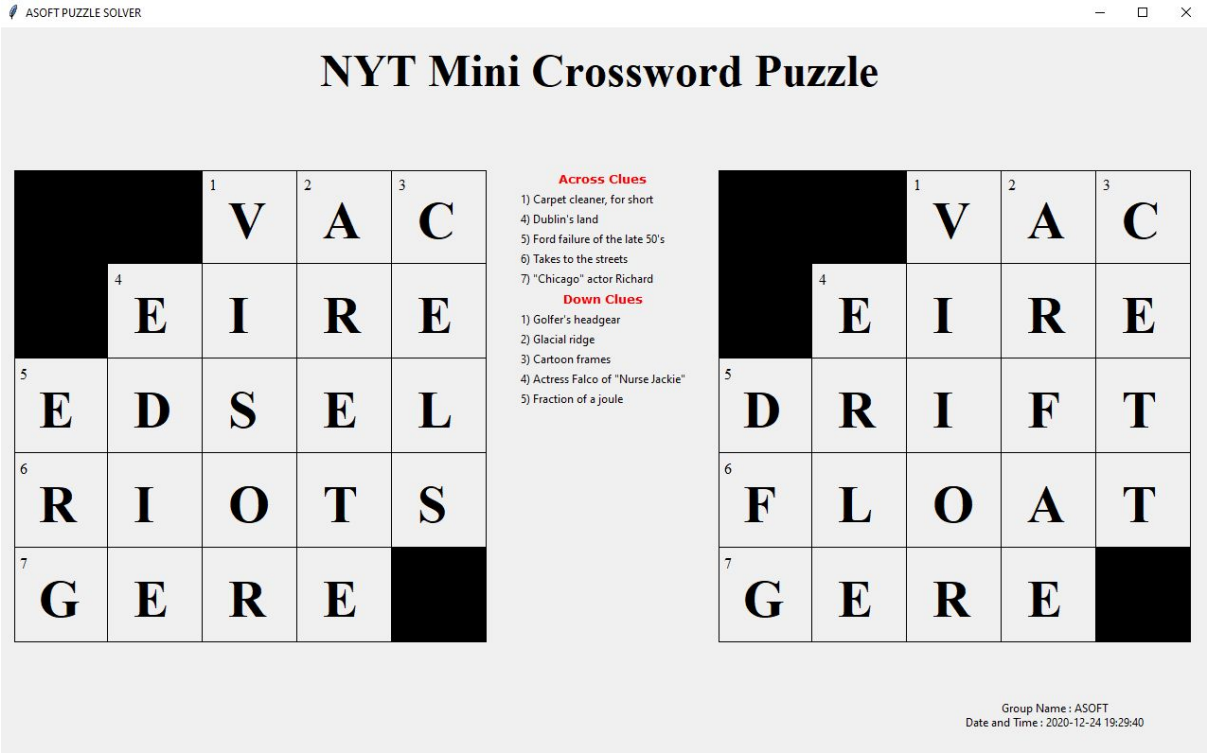
23.12.2020



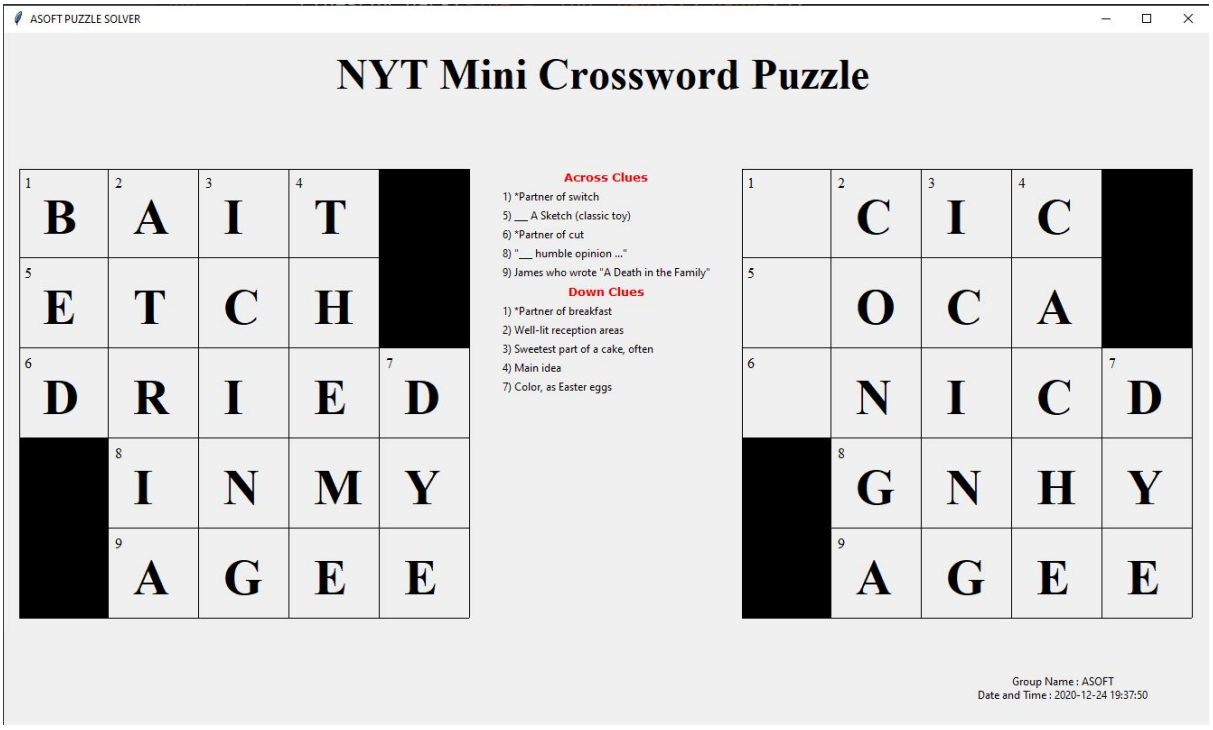
24.12.2020



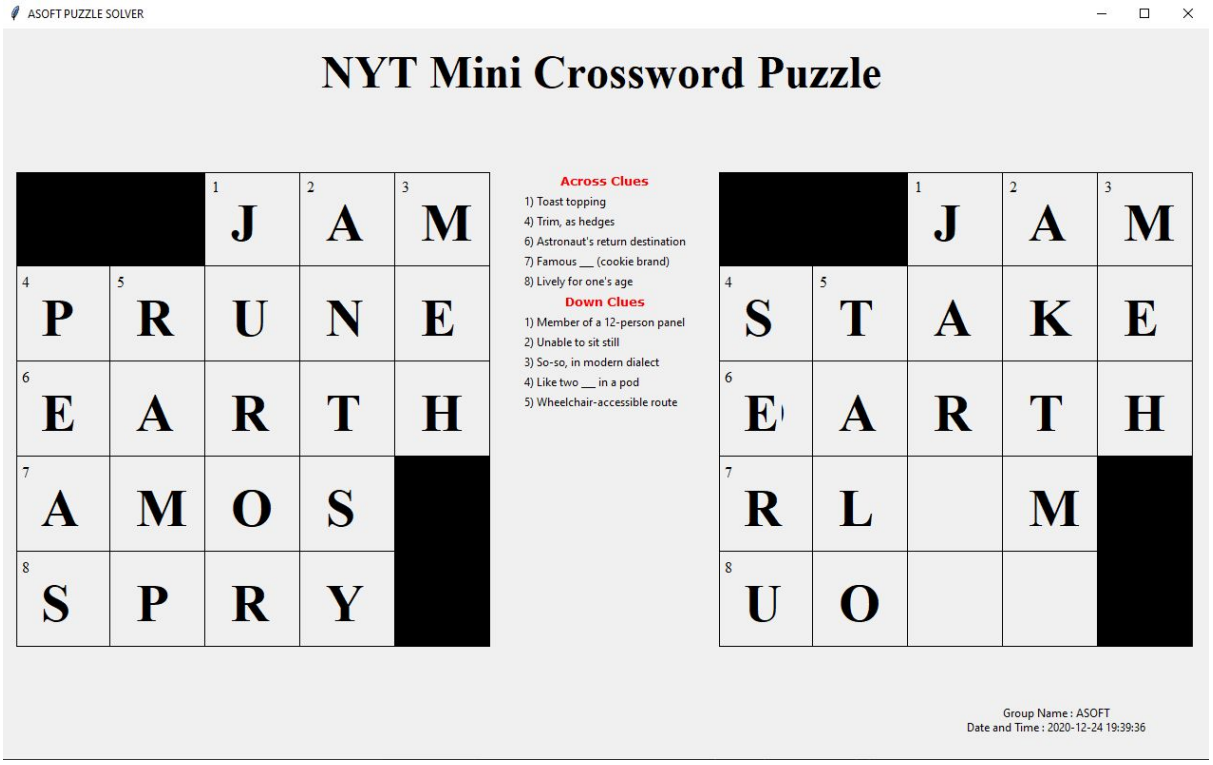
25.12.2020



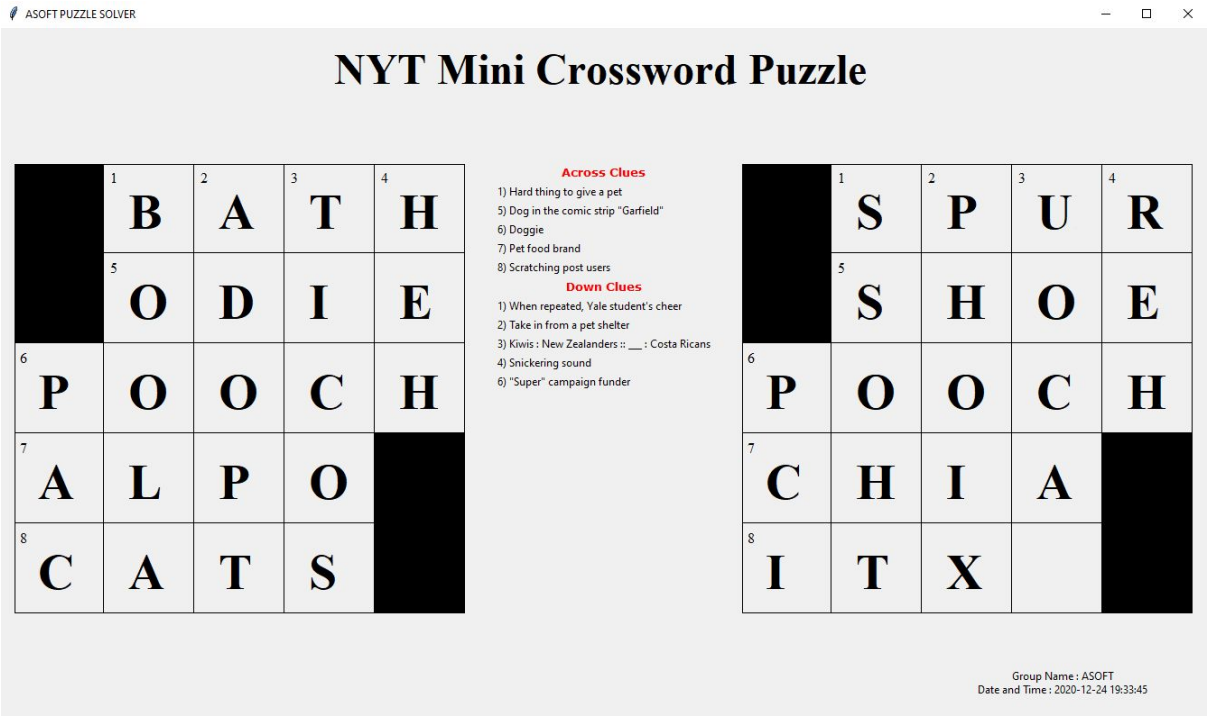
specials-crosswordese



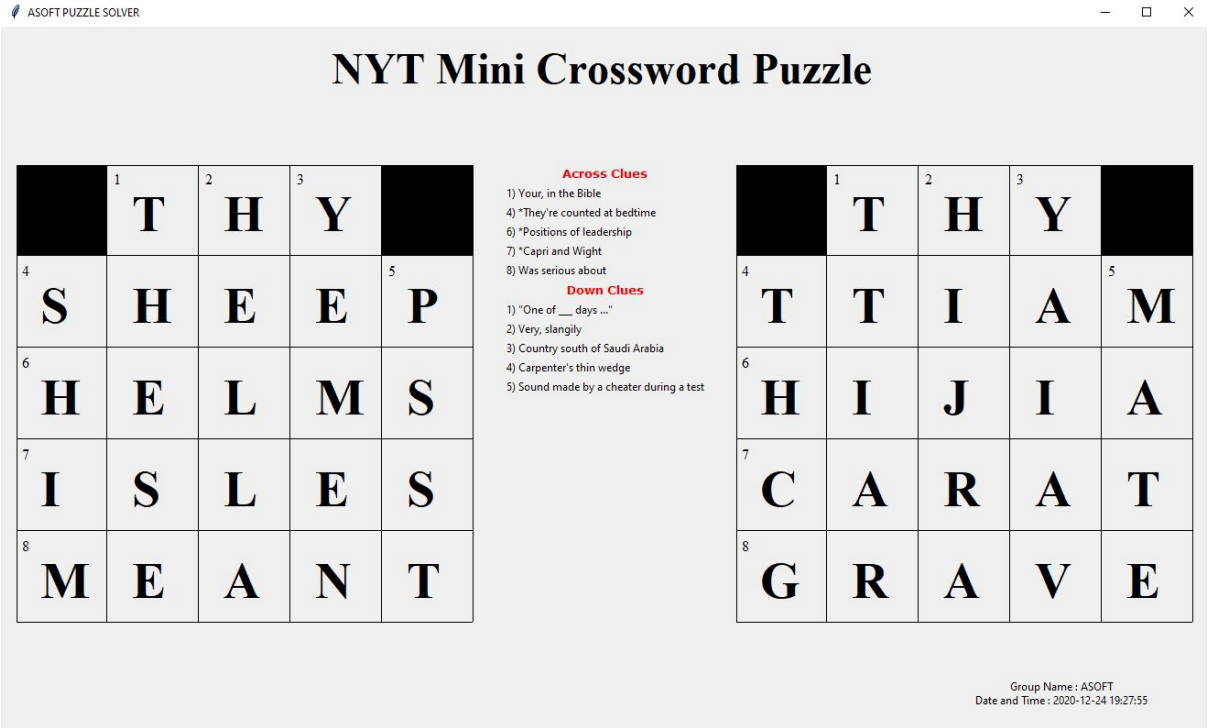
specials-partners



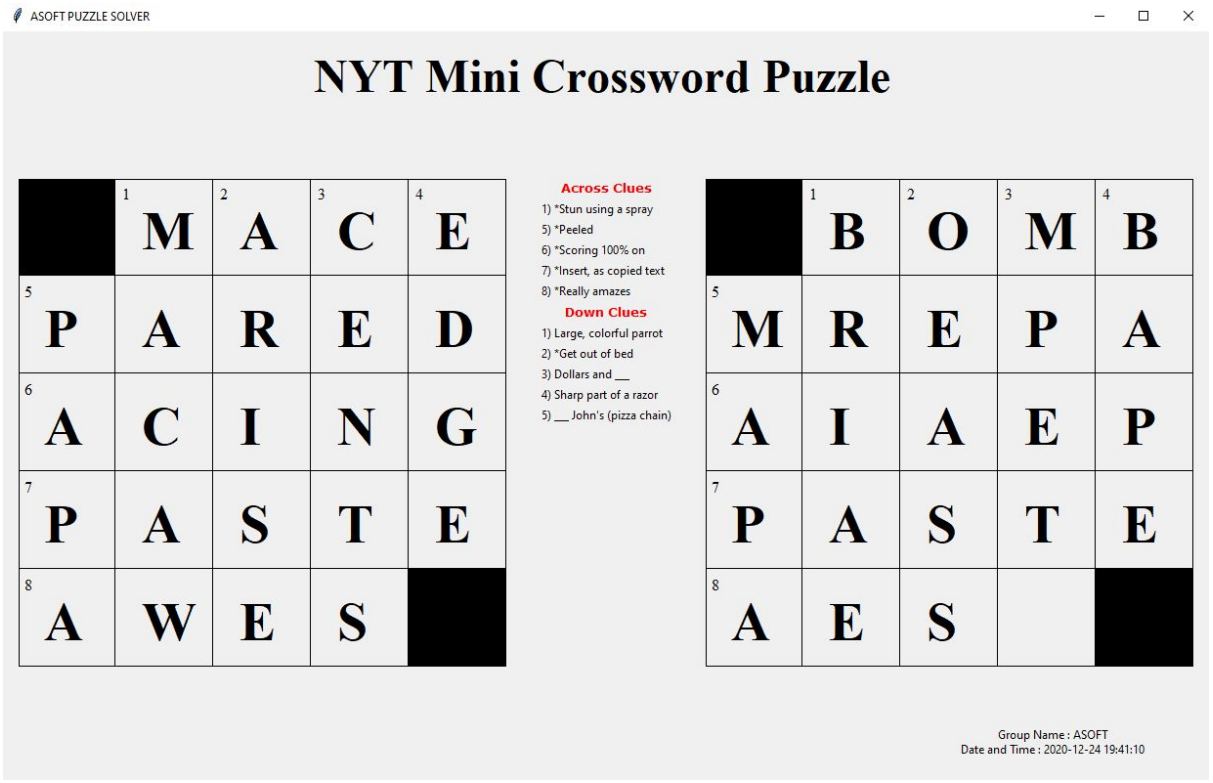
specials-parts-of-speech



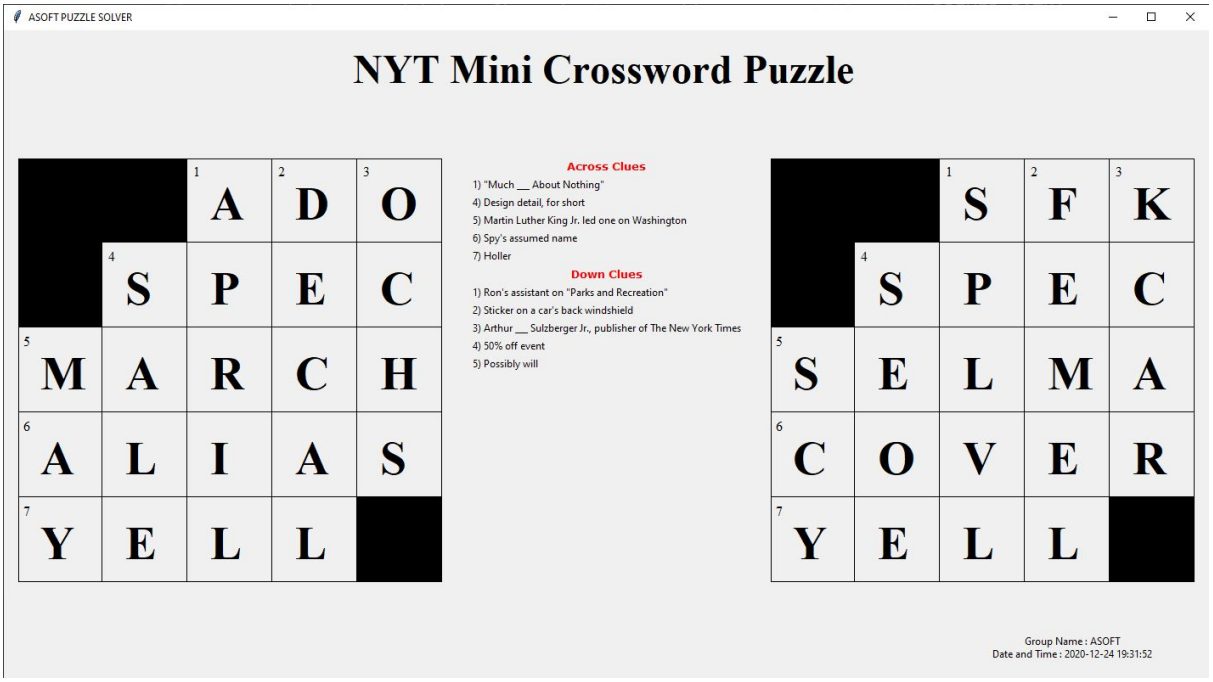
pets-mini



plurals-mini



specials-tenses



themed-mini

4. Conclusion

All in all, the project aimed to solve The Mini Crossword by Joel Fagliano from the New York Times. The puzzle gets an update everyday, such an algorithm that is capable of solving any puzzle which is given to it as an input is generated, as the Ho and Liausvia stated, “For an intelligent agent to be fully autonomous and adaptive, all aspects of intelligent processing from perception to action must be engaged and integrated.” [5] In order to create the candidate words for the puzzle, the different platforms are used. While assigning the clue types, the rule-based deduction system is considered. This system allowed us to make the clues understandable for the software. What’s more, the candidate keys are categorized with respect to their clues using this system. After the categorization is done, the platforms are used to determine the best fitting candidate for the puzzle. For every clue, different types of platforms are considered relevant as the clue suggests a different type of question in terms of structure and meaning.

After collecting all the candidate keys that can be sorted in the clue types, a list is formed. It is observed that such a list that is large is not suitable nor efficient for brute force thus a different search algorithm is used. The A* algorithm is selected to decrease the solving time of the application as well as the number of the compares. While implementing the A* algorithm the weights of the platforms are considered to approach the least-cost path. To sum up, the use of the algorithm is used for the solving of the crossword puzzle yet with effort and further study, the performance of the algorithm can be improved and implementation of Reverse Dictionary and Merriam Webster can be done.

Lastly, when the program, puzzle solver, is tested, it can be seen that the candidates are consistent with the clues and the correct answer’s length. At least one correct answer is generated. The program’s success rate would be increased by alternative improvements.

REFERENCES

[1] Riedl, MO. Human - centered artificial intelligence and machine learning. *Hum Behav & Emerg Tech*. 2019; 1: 33– 36. https://doi.org/10.1002/hbe2.117

[2] S. Naranan (2010) A Statistical Study of Failures in Solving Crossword Puzzles, Journal of Quantitative Linguistics, 17:3, 191-211, DOI: 10.1080/09296174.2010.485445

[3] Schuessler, J. (2017, March 22). A Journey Into the Merriam-Webster Word Factory. Retrieved December 25, 2020, from https://www.nytimes.com/2017/03/22/books/merriam-webster-dictionary-kory-stamper.html

[4] How to Solve The New York Times Crossword. (n.d.). Retrieved December 25, 2020, from https://www.nytimes.com/guides/crosswords/how-to-solve-a-crossword-puzzle

[5] Ho SB., Liausvia F. (2013) Knowledge Representation, Learning, and Problem Solving for General Intelligence. In: Kühnberger KU., Rudolph S., Wang P. (eds) Artificial General Intelligence. AGI 2013. Lecture Notes in Computer Science, vol 7999. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-39521-5_7

*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 **ASOFT**.*

Word Count: 2042

Appendix

main.py

```
from urllib.request import urlopen as urlRequest
import bs4
from bs4 import BeautifulSoup as soup
import scraper
import search
import requests
from datetime import date
from datetime import datetime
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from tkinter import *
#import Tkinter as Tk
import copy

class matrix_cell:
    def __init__(self, number, letter):
        self.number = number
        self.letter = letter

# old https://www.nytimes.com/crosswords/game/mini/2016/06/01
# uptodate https://www.nytimes.com/crosswords/game/mini
PUZZLE_URL = "https://www.nytimes.com/crosswords/game/mini"
PUZZLE_SIDE_LENGTH = 100

def showAcrossClues(acClues,acIndexes):
    print("Printing across clues")
    for i in range(len(acClues)):
        print("", acIndexes[i].text, " ", acClues[i].text)
    print()

def showDownClues(dwClues,dwIndexes):
    print("Printing down clues")
    for i in range(len(dwClues)):
        print("", dwIndexes[i].text, " ", dwClues[i].text)
    print()

def getDate():
    print("Retrieving current date and time of the system ")
    today = str(datetime.now().strftime('%Y-%m-%d %H:%M:%S'))
```

```
    return today

def addAcrossCluesCnv(canvas, acClues, acIndexes):

    L = Label(canvas, text="Across Clues",fg="red",font = "Verdana 10
bold").pack(fill='both')


    for i in range(len(acClues)):
        L = Label(canvas, text=acIndexes[i].text + ") " + acClues[i].text,
anchor='w').pack(fill='both')
    print()

def addDownCluesCnv(canvas,dwClues,dwIndexes):
    L = Label(canvas, text="Down Clues",fg="red",font = "Verdana 10
bold").pack(fill='both')

    for i in range(len(dwClues)):
        L = Label(canvas, text=dwIndexes[i].text + ") " + dwClues[i].text,
anchor='w').pack(fill='both')
    print()

def findCellWithNoReturnCol( matrix, x ):

    for i in range(0,5):
        for j in range (0,5):
            if matrix[i,j].number == x:
                return j

def findCellWithNoReturnRow(matrix,  x ):

    for i in range(0,5):
        for j in range (0,5):
            if matrix[i,j].number == x:
                return i

def findCellInfoWithNumber(matrix, x):

    for i in range(0,5):
        for j in range (0,5):
            if matrix[i,j].number == x:
                return i,j,int(x)

def callSearch(isAcross,lst):
```

```
    word = ""

    for element in lst:
        word = word + element[0]

    if(isAcross):
        resDataMuse2 = search.detailedSearchDataMuse(word)

    else:
        resDataMuse2 = search.detailedSearchDataMuse(word)

    return resDataMuse2

# finding intersection of answers to compare
def findIntersectedCells(matrix, acrossBeginningInfo, downBeginningInfo):

    result = {}

    for i in range(0,5):
        for j in range(0,5):
            if(int(matrix[i,j].number) < 0):    # if current cell is black
                result[i,j] = (-1,-1,-1,-1,"-1")
            else:
                for p in range(len(acrossBeginningInfo)):
                    if(acrossBeginningInfo[p][0] == i):
                        for q in range(len(downBeginningInfo)):
                            if(downBeginningInfo[q][1] == j):
                                # storing results in tuples with 4 elements
                                # (0) across answer number that cell belongs to
                                # (1) across distance that cell is far from beginning of word
                                # (2) down answer number that cell belongs to
                                # (3) down distance that cell is far from beginning of word
                                result[i,j] = (acrossBeginningInfo[p][2], (j -
acrossBeginningInfo[p][1]), downBeginningInfo[q][2], (i - downBeginningInfo[q][0]),
"-1")

    return result

def compareAnswerCandidatesFirstIteration(resDataMuse, intersectionInfo):

    across = resDataMuse[0:5]
    down = resDataMuse[5:10]
```

```
    # first getting rid of answer candidates that are not in the same size with answers
(bigger than length 5 gotten rid of before)

    # index of those words are not in the same length of the answers
    acrossDeleteIndexes = []
    downDeleteIndexes = []

    # finding indexes for answer candidates that are not same length with answer
    for i in range(len(across)):
        for j in range(len(across[i])):
            if(acrossLengths[i] != len(across[i][j])):
                acrossDeleteIndexes.append((i,j))
    # indexes are chosen and deleted from search list
    for i in reversed(range(len(acrossDeleteIndexes))):
        p = acrossDeleteIndexes[i][0]
        q = acrossDeleteIndexes[i][1]
        del across[p][q]

    # same process for down
    for i in range(len(down)):
        for j in range(len(down[i])):
            if(downLengths[i] != len(down[i][j])):
                downDeleteIndexes.append((i,j))
    # indexes are chosen and deleted from search list
    for i in reversed(range(len(downDeleteIndexes))):
        p = downDeleteIndexes[i][0]
        q = downDeleteIndexes[i][1]
        del down[p][q]

    # storing answers with index of clues instead of 0,1,2,3,4
    newAcross = [[]]*10
    newDown = [[]]*10
    for i in range(0,10):
        for j in range(len(acrossBeginningInfo)):
            if(acrossBeginningInfo[j][2] == i):
                newAcross[i] = across[j]

    for i in range(0,10):
        for j in range(len(downBeginningInfo)):
            if(downBeginningInfo[j][2] == i):
                newDown[i] = down[j]

    possibleMatches = []
    #return newAcross, newDown
```

```
  for k in range(len(newAcross)):
    if(len(newAcross[k]) > 0 ):
      for t in range(0,5):
        if(acrossBeginningInfo[t][2] == k):
          p = acrossBeginningInfo[t][0]
          q = acrossBeginningInfo[t][1]
      temp = "-1"
      for x in range(q,5):
        acrossBeginIndex, acrossDistance, downBeginIndex, downDistance, temp
= intersectionInfo[p, x]

        for i in range(len(newAcross[acrossBeginIndex])):
          for j in range(len(newDown[downBeginIndex])):
            if(newAcross[acrossBeginIndex][i][acrossDistance] ==
newDown[downBeginIndex][j][downDistance]):
              print(newAcross[acrossBeginIndex][i], " ",
newDown[downBeginIndex][j])
              possibleMatches.append((newAcross[acrossBeginIndex][i],
newDown[downBeginIndex][j]))
              for y in range(0,5):
                if(downBeginningInfo[y][2] == downBeginIndex):
                  a = downBeginningInfo[y][0]
                  b = downBeginningInfo[y][1]

                print()

              newMatchPossibility = ((p,q), newAcross[acrossBeginIndex][i],
(a,b), newDown[downBeginIndex][j], (acrossBeginIndex,downBeginIndex))




              compareAnswerCandidatesNewIteration(newAcross, newDown,
intersectionInfo, newMatchPossibility)



  print()
  return


def compareAnswerCandidatesNewIteration(newAcross, newDown, intersectionInfo,
newMatchPossibility):
```

```
    print(newMatchPossibility)
    # newIntersectionInfo = copy.deepcopy(intersectionInfo)
    # for i in range(newMatchPossibility[0][1],len(newMatchPossibility[1])):
    #     x = newMatchPossibility[0][0]
    #     newIntersectionInfo[x,i][4] = newMatchPossibility[1][i]

    # for i in range(len(newAcross)):
    #     if(len(newAcross[i]) > 0):
    #         for j in range(len(newAcross[i])):
    #             #print(newAcross[i][j])
    #             for k in range(len(newAcross[i][j])):
    #                 x = newMatchPossibility[0][0]
    #                 y = newMatchPossibility[0][1]
    #                 nfinAcross.append((newAcross[i][j][k], (j,k)))
    #
    # for i in range(len(newDown)):
    #     if(len(newDown[i]) > 0):
    #         for j in range(len(newDown[i])):
    #             #print(newAcross[i][j])
    #             for k in range(len(newDown[i][j])):
    #                 x = newMatchPossibility[0][0]
    #                 y = newMatchPossibility[0][1]
    #                 nfinDown.append((newDown[i][j][k], (j,k)))

    nfinAcross.append((newMatchPossibility[4][0],newMatchPossibility[1]))
    nfinDown.append((newMatchPossibility[4][1], newMatchPossibility[3]))

    print()


    return

# list to store answers to check length
acrossBeginningInfo = []
downBeginningInfo = []

finAcross = [[]] * 10
finDown = [[]] * 10
nfinAcross = []
nfinDown = []

acrossLengths = [0,0,0,0,0]
downLengths = [0,0,0,0,0]
```

```
def main():

  pageContent = scraper.getPageContent(PUZZLE_URL) # getting everything from
website
  pageParsed = scraper.parseContent(pageContent) # page content is parsed

  #get clues and indexes
  clues = scraper.getAllOl(pageParsed,"ClueList-list--2dD5-") # found all clue
container
  acrossClues = scraper.getAllSpan(clues[0],"Clue-text--3lZl7") # found all across
clues
  print("Detected all across clues ")
  acrossIndexes = scraper.getAllSpan(clues[0],"Clue-label--2IdMY")  # found all
across clue indexes
  print("Detected all across clue indexes ")
  downClues = scraper.getAllSpan(clues[1],"Clue-text--3lZl7") # found all down
clues
  print("Detected all down clue indexes ")
  downIndexes = scraper.getAllSpan(clues[1],"Clue-label--2IdMY")  # found all
down clue indexes
  print("Detected all down clue indexes ")
  print()
  showAcrossClues(acrossClues,acrossIndexes)
  showDownClues(downClues,downIndexes)
  print("Retrieved date and time is " + getDate())

  #scraper.closeDriver()


  print()

  #resRevDict = search.searchRevDict(acrossClues,downClues)
  #resGoogle =
search.searchGoogle(acrossClues,acrossIndexes,downClues,downIndexes)

  resDataMuse =
search.searchDataMuse(acrossClues,acrossIndexes,downClues,downIndexes)
  #resMerriam =
search.searchMerriam(acrossClues,acrossIndexes,downClues,downIndexes)

  #print("Result RevDict",resRevDict,"\n\n")
  #print("Result Google",resGoogle,"\n\n")
```

```
print("Result DataMuse",resDataMuse,"\n\n")
#print("Result Merriam",resMerriam,"\n\n")



# print("google")
# for i in range(len(resGoogle)):
#     print(resGoogle[i])


#print("\n\ndatamuse")
#for i in range(len(resDataMuse)):
#    print(resDataMuse[i])

# print("\n\nmerriam")
# for i in range(len(resMerriam)):
#     print(resMerriam[i])

#print("\n\nrevDict")
# for i in range(len(resRevDict)):
#     print(resMerriam[i])

print("datamuse ", resDataMuse[0] )


finList = [[]]*(len(acrossClues)+len(downClues))
#
for i in range (len(acrossClues)+len(downClues)):
    finList[i] = resDataMuse[i] #+ resRevDict[i] + resMerriam[i]
print(finList)



##########################################################################
#############################################
print("Drawing the GUI")
#GUI part
master = Tk()
master.title("ASOFT PUZZLE SOLVER")
master.configure()


#puzzle canvas
```

```
    print("Creating the puzzle canvas")

    #title
    titleGrid = Canvas(master, width=6*PUZZLE_SIDE_LENGTH,
height=PUZZLE_SIDE_LENGTH)
    titleGrid.pack(side='top',padx=15,pady=15)
    tittleLabel = Label(titleGrid, text="NYT Mini Crossword Puzzle",font = "Times 36
bold").place(x = 0,  y = 0)


    puzzleGrid = Canvas(master, width=5*PUZZLE_SIDE_LENGTH,
height=6*PUZZLE_SIDE_LENGTH)
    puzzleGrid.pack(side="left",padx=15,pady=15)
    puzzleGrid.outline='black'
    puzzleGrid.width=2
    answers =
[['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']]
    puzzle = [[11, 12, 5, 2, 0], [0, 15, 6, 10, 0], [10, 0, 12, 5, 2], [12, 15, 8, 6, 1], [5, 0, 9,
5, 2]]

    newPuzzleGrid = Canvas(master, width=5*PUZZLE_SIDE_LENGTH,
height=6*PUZZLE_SIDE_LENGTH)
    newPuzzleGrid.pack(side='right',padx=15,pady=15)
    #newPuzzleGrid.outline='black'
    #newPuzzleGrid.width=2

    # clues canvas
    print("Creating the clue canvas")
    cluesCnv = Canvas(master, width=4*PUZZLE_SIDE_LENGTH,
height=4*PUZZLE_SIDE_LENGTH)
    cluesCnv.pack(side = "top",padx=15,pady=15)

    print()
    print("Starting the process of retrieving answers !")
    matrix, across_clue, horiz_clue = scraper.getAnswers(PUZZLE_URL)

    # getting indexes of answer beginnings and storing them
    for i in range(len(acrossIndexes)):
        acrossBeginningInfo.append(findCellInfoWithNumber(matrix,
acrossIndexes[i].text))
    for i in range(len(downIndexes)):
        downBeginningInfo.append(findCellInfoWithNumber(matrix,downIndexes[i].text))

    # getting lengths of across answers
```

```
  # selecting each answer beginning (first letter) info and going across counting
every white cell
  for i in range(len(acrossBeginningInfo)):
      rowNo= acrossBeginningInfo[i][0]
      colNo = acrossBeginningInfo[i][1]
      for j in range(colNo, 5):
          if(int(matrix[rowNo,j].number) >= 0):
              acrossLengths[i] += 1

  # getting lengths of down answers
  # selecting each answer beginning (first letter) info and going down counting every
white cell
  for i in range(len(downBeginningInfo)):
      rowNo = downBeginningInfo[i][0]
      colNo = downBeginningInfo[i][1]
      for j in range(rowNo, 5):
          if (int(matrix[j, colNo].number) >= 0):
              downLengths[i] += 1

  # intersectionInfo[i,j] = (leftMostNumber, distanceFromLeft, topMostNumber,
distanceFromTop) (explained in detail in findIntersectedCells function)
  intersectionInfo = findIntersectedCells(matrix, acrossBeginningInfo,
downBeginningInfo)
  # for i in range(0,5):
  #     for j in range (0,5):
  #         print(intersectionInfo[i,j])

  #na,nd =
  compareAnswerCandidatesFirstIteration(resDataMuse, intersectionInfo)


  print()
  #drawing the grid
  print("Drawing the grid of the puzzle and filling the black squares")
  for i in range(0, 5):

      for j in range(0, 5):

          #add label inside boxes
          if(matrix[i,j].number != 0 and matrix[i,j].number != -8):
              L = Label(puzzleGrid, text=str(matrix[i,j].number), font = "Times
12").place(x = j*PUZZLE_SIDE_LENGTH+5,  y = i*PUZZLE_SIDE_LENGTH+5)
              L = Label(newPuzzleGrid, text=str(matrix[i,j].number), font = "Times
12").place(x = j*PUZZLE_SIDE_LENGTH+5,  y = i*PUZZLE_SIDE_LENGTH+5)
```

```
        #if(matrix[i,j].number != -8):
        #    L = Label(puzzleGrid, text=str(answers[j][i]),font = "Times").place(x =
i*100+50,  y = j*100+50)

        if (matrix[i,j].number==-8):
            puzzleGrid.create_rectangle(j*PUZZLE_SIDE_LENGTH,
i*PUZZLE_SIDE_LENGTH, j*PUZZLE_SIDE_LENGTH+PUZZLE_SIDE_LENGTH,
i*PUZZLE_SIDE_LENGTH+PUZZLE_SIDE_LENGTH, fill="black")
            newPuzzleGrid.create_rectangle(j*PUZZLE_SIDE_LENGTH,
i*PUZZLE_SIDE_LENGTH, j*PUZZLE_SIDE_LENGTH+PUZZLE_SIDE_LENGTH,
i*PUZZLE_SIDE_LENGTH+PUZZLE_SIDE_LENGTH, fill="black")



  puzzleGrid.create_line(2, 0, 2, 5*PUZZLE_SIDE_LENGTH, fill="black", width=2)
  puzzleGrid.create_line(0, 2, 5*PUZZLE_SIDE_LENGTH, 2, fill="black", width=2)

  newPuzzleGrid.create_line(2, 0, 2, 5*PUZZLE_SIDE_LENGTH, fill="black",
width=2)
  newPuzzleGrid.create_line(0, 2, 5*PUZZLE_SIDE_LENGTH, 2, fill="black",
width=2)


  #Creates all vertical lines at intervals of PUZZLE_SIDE_LENGTH
  for i in range(0, 6, 1):
      puzzleGrid.create_line([(i*PUZZLE_SIDE_LENGTH, 0),
(i*PUZZLE_SIDE_LENGTH, 5*PUZZLE_SIDE_LENGTH)], tag='table_line')
      newPuzzleGrid.create_line([(i*PUZZLE_SIDE_LENGTH, 0),
(i*PUZZLE_SIDE_LENGTH, 5*PUZZLE_SIDE_LENGTH)], tag='table_line')


  # Creates all horizontal lines at intervals of PUZZLE_SIDE_LENGTH
  for i in range(0, 6, 1):
      puzzleGrid.create_line([(0, i*PUZZLE_SIDE_LENGTH),
(5*PUZZLE_SIDE_LENGTH, i*PUZZLE_SIDE_LENGTH)], tag='table_line')
      newPuzzleGrid.create_line([(0, i*PUZZLE_SIDE_LENGTH),
(5*PUZZLE_SIDE_LENGTH, i*PUZZLE_SIDE_LENGTH)], tag='table_line')

  print("Printing the time label to the gui")
  timeLabel = Label(newPuzzleGrid, text= "Group Name : ASOFT\nDate and Time
: " + getDate()).place(x = 260,  y = 560)
  #timeLabel.pack()
```

```
print("Adding clues to the canvas")
#add clues to the canvas
addAcrossCluesCnv(cluesCnv,acrossClues,acrossIndexes)
addDownCluesCnv(cluesCnv,downClues,downIndexes)



print("Printing answers to the both gui and terminal")


across = [0]*10
down = [0]*10

Nacross = [0] * 10
Ndown = [0] * 10

#print(na,"\n\n\n",nd,"\n\n\n")
print(nfinAcross,nfinDown)

for i in range(len(down)):
    for j in range(len(nfinDown)):
        if(nfinDown[j][0] == i):
            down[i] = nfinDown[j][1]

for i in range(len(across)):
    for j in range(len(nfinAcross)):
        if(nfinAcross[j][0] == i):
            across[i] = nfinAcross[j][1]

for i in range(len(acrossBeginningInfo)):
    print()

print("Across info",acrossBeginningInfo)
print("Down info",downBeginningInfo)

print("across")
print(across)

print("down")
print(down)

print()
```

```
    for i in range(0, 5):
        ind = 0
        for j in range(0, 5):

            #add label inside boxes
            print(matrix[j,i].number)
            if(matrix[j,i].number != -8):
                word = down[downBeginningInfo[i][2]]
                print(word)
                if(word != 0 and (ind < len(word))):

                    L = Label(newPuzzleGrid, text=str(word[ind]).upper(),font = "Times 42
bold").place(x = i*PUZZLE_SIDE_LENGTH+25,  y =
j*PUZZLE_SIDE_LENGTH+20)
                    ind = ind + 1

    for i in range(0, 5):
        ind = 0
        for j in range(0, 5):

            #add label inside boxes
            if(matrix[i,j].number != -8):
                word = across[acrossBeginningInfo[i][2]]

                if(word != 0):

                    L = Label(newPuzzleGrid, text=str(word[ind].upper()),font = "Times 42
bold").place(x = j*PUZZLE_SIDE_LENGTH+25,  y =
i*PUZZLE_SIDE_LENGTH+20)
                    ind = ind + 1
```

```
    # across[1] = ('C',(0,1) ), ('L',(0,2) ), ('U',(0,3) ), ('B',(0,4) )
    # across[5] = ('L',(1,1) ), ('A',(1,2) ), ('N',(1,3) ), ('E',(1,4) )
    # across[6] = ('M',(2,0) ), ('A',(2,1) ), ('Y',(2,2) ), ('B',(2,3) ), ('E',(2,4) )
    # across[7] = ('O',(3,0) ), ('R',(3,1) ), ('E',(3,2) ), ('O',(3,3) )
    # across[8] = ('M',(4,0) ), ('A',(4,1) ), ('R',(4,2) ), ('X',(4,3) )
```

```
    #
    # down[1] = ('C',(0,0) ), ('L',(1,0) ), ('A',(2,0) ), ('R',(3,0) ), ('A',(4,0) )
    # down[2] = ('L',(0,1) ), ('A',(1,1) ), ('Y',(2,1) ), ('E',(3,1) ), ('R',(4,1) )
    # down[3] = ('U',(0,2) ), ('N',(1,2) ), ('B',(2,2) ), ('O',(3,2) ), ('X',(4,2) )
    # down[4] = ('B',(0,3) ), ('E',(1,3) ), ('E',(2,3) )
    # down[6] = ('M',(2,4) ), ('O',(3,4) ), ('M',(4,4) )

    print("Down")
    for i in range (len(across_clue)):
        print(across_clue[i].getText(), " ---> ")
        colNo = findCellWithNoReturnCol(matrix,across_clue[i].getText()[0])
        for j in range(0,5):
            #L = Label(puzzleGrid, text=str(matrix[j,colNo].letter),font = "Times").place(x =
i*100+50,  y = j*100+50)
            print( matrix[j,colNo].letter, end = "" )
        print("")

    # print(across[5][2])
    # index = 0
    # for i in range(10):
    #     if(across[i] != 0):
    #         for j in range(5):
    #             try:
    #                 L = Label(newPuzzleGrid, text=str(across[i][j][0]),font = "Times 42
bold").place(x = across[i][j][1][1]*PUZZLE_SIDE_LENGTH+25,  y =
across[i][j][1][0]*PUZZLE_SIDE_LENGTH+20)
    #             except:
    #                 pass
    #             index = index + 1

    print("\nAcross")
    for i in range (len(horiz_clue)):
        print(horiz_clue[i].getText(), " ---> ")
        rowNo = findCellWithNoReturnRow(matrix,horiz_clue[i].getText()[0])
        nl = []

        for j in range(0,5):
            if(str(matrix[rowNo, j].letter) != " "):
                L = Label(puzzleGrid, text=str(matrix[rowNo, j].letter),font = "Times 42
bold").place(x = j*PUZZLE_SIDE_LENGTH+25,  y =
i*PUZZLE_SIDE_LENGTH+20)
                ans = ('',(i,j) )
                nl.append(ans)
            Nacross[rowNo] = nl
```

```
            #Nacross[rowNo] = nl
            print (matrix[rowNo, j].number,  "*********\n")
            print (rowNo,"***********\n")
        print("")
    A = "KEYS" #across 5
    B = "KEYS"

    print(Nacross)


    #resDataMuse2 = search.detailedSearchDataMuse("What a black three-leaf clover represents","?lu?")
    #resDataMuse2 = search.detailedSearchDataMuse("mayb?")
    #print(resDataMuse2)



    # calling detailed search method example
    #callSearch(True,across[6])
    print()
    print()
    print()

    print("across")
    print(across)

    print("down")
    print(down)

    mainloop()


main()
```

scraper.py

```
import requests
from bs4 import BeautifulSoup
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.options import Options
import time
```

```
DRIVER_PATH = 'chromedriver.exe'



class matrix_cell:
    def __init__(self, number, letter):
        self.number = number
        self.letter = letter


def getPageContent(url):
    print("Requesting page content from " + url)
    pageContent = requests.get(url)

    return pageContent

def parseContent(pageContent):
    print("Parsing the retrived page content with BeatifulSoup ")
    page = BeautifulSoup(pageContent.content, "html.parser")

    return page

def getAllSpan(getFrom,className):

    result = getFrom.findAll("span", {"class": className})

    return result

def getAllOl(getFrom,className):
    print("Detecting all clue containers ")
    result = getFrom.findAll("ol", {"class": className})

    return result

def closeDriver():
    #driver.quit()
    return


def getAnswers(PUZZLE_URL):

    options = Options()
    options.headless = True
```

```
    options.add_argument("--window-size=800,600")

    driver = webdriver.Chrome(options=options, executable_path=DRIVER_PATH)
    driver.get(PUZZLE_URL)
    sleepTime = 0.5
    print("Connected to the https://www.nytimes.com/crosswords/game/mini")
    print("Waiting time between requests is " + str(sleepTime) + " seconds")
    print("Waiting " + str(sleepTime) + " seconds")

    time.sleep(sleepTime)

    print("----------Clicking to the OK button----------")

driver.find_element_by_xpath("/html/body/div[1]/div/div/div[4]/div/main/div[2]/div/
div[2]/div[3]/div/article/div[2]/button/div/span").click()

    print("Waiting " + str(sleepTime) + " seconds")
    time.sleep(sleepTime)
    print("----------Clicking to the REVEAL button----------")

driver.find_element_by_xpath("/html/body/div[1]/div/div/div[4]/div/main/div[2]/div/
div/ul/div[2]/li[2]/button").click()

    print("Waiting " + str(sleepTime) + " seconds")
    time.sleep(sleepTime)
    print("----------Clicking to the PUZZLE button----------")

driver.find_element_by_xpath("/html/body/div[1]/div/div/div[4]/div/main/div[2]/div/
div/ul/div[2]/li[2]/ul/li[3]/a").click()

    print("Waiting " + str(sleepTime) + " seconds")
    time.sleep(sleepTime)
    print("----------Clicking to the REVEAL button----------")

driver.find_element_by_xpath("/html/body/div[1]/div/div[2]/div[2]/article/div[2]/but
ton[2]/div/span").click()

    print("Waiting " + str(sleepTime) + " seconds")
    time.sleep(sleepTime*3)
    print("----------Clicking to the CLOSE POP UP button----------")
    driver.find_element_by_xpath("/html/body/div[1]/div/div[2]/div[2]/span").click()

    print("Retrieving and parsing the html including the answers")
    html = driver.page_source
```

```
    soup = BeautifulSoup (html, "html.parser")

    clueler =  soup.find("div", {"class": "layout"}).find("article", {"aria-label" : "Main
Puzzle Layout"}).findAll("section")[2]

    #print(clueler.getText())
    across_clue = clueler.findAll("div")[1].findAll("li")
    horiz_clue = clueler.findAll("div")[0].findAll("li")
    #print(across_clue[0].getText())
    #print("**************************************************************")
    #print(horiz_clue[0].getText())

    gler = soup.find(id = "xwd-board").find(role = "table").findAll("g")



    x = 0
    for i in range (0,5):
        for j in range (0,5):
            #print(gler[x].getText())
            x += 1

    #print("*/*/*/*/*/*/**/*\n",gler,"\n/*/*/*/*/*/*/*/*/*\n")


    x = 0
    matrix = {}

    for i in range (0,5):
        for j in range (0,5):
            if(len(gler[x].getText()) == 0):
                matrix[i,j] = matrix_cell(-8," ")
            elif(len(gler[x].getText()) == 2):
                matrix[i,j] = matrix_cell(0, gler[x].getText()[0])
            elif(len(gler[x].getText()) == 3):
                matrix[i,j] = matrix_cell(gler[x].getText()[0], gler[x].getText()[1])

            x += 1

    driver.quit()
    print("Closed the web browser driver")
    print("Returning the answers")
    return matrix, across_clue, horiz_clue
```

```
def findInputPutKey(getUrl, inputId,acClues,dnClues):

    options2 = Options()
    options2.headless = True
    options2.add_argument("--window-size=800,600")

    driver2 = webdriver.Chrome(options=options2, executable_path=DRIVER_PATH)


    driver2.get(getUrl)
    resList = [[]]*(len(acClues)+len(dnClues))


    ind = 0
    for clue in acClues:
        print("Searching for ==> ", clue.text,"\n")
        driver2.find_element_by_id(inputId).send_keys(clue.text)
        time.sleep(0.3)
        btn = driver2.find_element_by_id("search-button")
        btn.click()
        time.sleep(1.2)
        rList = []

        words = driver2.find_elements_by_class_name("item")
        try:
            for i in range(0,10):
                if(len(words[i].text) <= 5):
                    #print(words[i].text)
                    rList.append(words[i].text)
        except:
            pass
        #print("\n")
        resList[ind] = rList
        ind = ind + 1
        btn = driver2.find_element_by_id("clear-search-button")
        btn.click()
        time.sleep(1)


    for clue in dnClues:
        print("Searching for ==> ",clue.text,"\n")
        driver2.find_element_by_id(inputId).send_keys(clue.text)
        time.sleep(0.3)
        btn = driver2.find_element_by_id("search-button")
```

```
        btn.click()
        time.sleep(1.2)
        rList = []
        words = driver2.find_elements_by_class_name("item")
        try:
            for i in range(0,10):
                if(len(words[i].text) <= 5):
                    #print(words[i].text)
                    rList.append(words[i].text)
        except:
            pass
        resList[ind] = rList
        ind = ind + 1
        #print("\n")
        btn = driver2.find_element_by_id("clear-search-button")
        btn.click()
        time.sleep(1)

    driver2.quit()
    return resList
```

search.py

```
import requests
from bs4 import BeautifulSoup
import urllib, json
import requests
import scraper


# define punctuation
punctuations = '''0123456789çşığ!()-[]{};:'"\,<>./?@#$%^&*_~'''



def searchGoogle(acClues,acIndexes,dnClues,dnIndexes):

    resList = []

    for i in range(len(acClues)):
        google_words = set()
```

```
        clue = acClues[i].text.replace(" ", "+")

        request = requests.get('https://www.google.com/search?q={}&lr=lang_en'
        .format(clue))
        soup = BeautifulSoup(request.content, "html.parser")
        result_elements = soup.text

        for word in result_elements.split():
            google_words.add(word)


        # getting rid of from unnecessary chars
        mylist = []
        for word in google_words:
            no_punct = ""
            for char in word:
                if char not in punctuations:
                    no_punct = no_punct + char
            no_punct.lower()
            mylist.append(no_punct)
        google_words = mylist

        # Process word list
        #google_words = self.clear_google_words(google_words)
        google_words = list(filter(lambda x: len(x) <= 5, google_words))
        print("Google Search \nWord Count: " + str(len(google_words)))

    resList.append(google_words)
    return resList

def searchDataMuse(acClues,acIndexes,dnClues,dnIndexes):

    resList = []


    for i in range(len(acClues)):
        #print("", acIndexes[i].text, " ", acClues[i].text)

        query = acClues[i].text.replace(" ", "+")
        query = query.lower()

        request =
requests.get('https://api.datamuse.com/words?ml={}'.format(query))
```

```
        respond = request.json()

        tList = []
        #for j in range(len(respond)):
        for j in range(min(5,len(respond))):
            tList.append(respond[j]['word'])
        resList.append(tList)


        #resList = list(filter(lambda x: len(x) == length, datamuse_words))
        #print("Datamuse Search \nWord Count: " + str(len(datamuse_words)))
    for i in range(len(dnClues)):
        #print("", acIndexes[i].text, " ", acClues[i].text)

        query = dnClues[i].text.replace(" ", "+")
        query = query.lower()

        request =
requests.get('https://api.datamuse.com/words?ml={}'.format(query))

        respond = request.json()

        tList = []
        for item in respond:
            tList.append(item['word'])
        resList.append(tList)
    for i in range(len(resList)):
        resList[i] = list(filter(lambda x: len(x) <= 5, resList[i]))

    return resList

def detailedSearchDataMuse(heuristic):

    resList = []


    #query = knu.replace(" ", "+")
    #query = query.lower()

    request =
requests.get('https://api.datamuse.com/words?sp={}'.format(heuristic))

    respond = request.json()
```

```
    for j in range(len(respond)):
    #for j in range(5):
        resList.append(respond[j]['word'])

    return resList


def searchMerriam(acClues,acIndexes,dnClues,dnIndexes):

    API_KEY = "dd09241d-bf89-4f78-8834-34dc6e0495c4"

    #for clue in acClues:
    #    print(clue.text.split())

    resList = [[]]*(len(acClues)+len(dnClues))

    ind = 0
    for clue in acClues:

        words = clue.text.split()
        rList = []

        if(len(words) <= 3):
            for word in words:

                URL =
"https://www.dictionaryapi.com/api/v3/references/thesaurus/json/"
                URL = URL + word + "?key="+API_KEY



                try:
                    response = requests.get(URL)
                    #data = json.loads(response.read())
                    #print(word, ind)
                    #print (response.json()[0]['meta']['syns'][0])

                    rList.append(response.json()[0]['meta']['syns'][0])
                    #resList[ind].append(response.json()[0]['meta']['syns'][0])
                    #print(rList,"\n\n")
                except:
                    pass
            resList[ind] = rList
```

```
            #print(rList,"\n",resList,'\n\n\n\n')
        ind = ind + 1


    for clue in dnClues:

        words = clue.text.split()
        rList = []

        if(len(words) <= 3):
            for word in words:

                URL =
"https://www.dictionaryapi.com/api/v3/references/thesaurus/json/"
                URL = URL + word + "?key="+API_KEY



                try:
                    response = requests.get(URL)
                    #data = json.loads(response.read())
                    #print(word, ind)
                    #print (response.json()[0]['meta']['syns'][0])

                    rList.append(response.json()[0]['meta']['syns'][0])
                    #resList[ind].append(response.json()[0]['meta']['syns'][0])
                    #print(rList,"\n\n")
                except:
                    pass
            resList[ind] = rList

            #print(rList,"\n",resList,'\n\n\n\n')
        ind = ind + 1


    return resList



def searchRevDict(acClues,dnClues):
    url = "https://reversedictionary.org/"

    resl = scraper.findInputPutKey(url, "query",acClues,dnClues)
```

```
return resl
```