Term Project Report for CS461

Group Nick: NEMESIS

Group Members:

Melike Arslan - 21601025

Ece Çanga - 21600851

Nursena Kurubaş - 21602965

Selen Uysal - 21702292

Sonat Uzun - 2101857

Bilkent University
Department of Computer Engineering

December 24, 2020

Table of Contents

1. Introduction

Solving puzzles is an enjoyable activity for people of all ages. One specific kind of puzzle is word-based puzzles. Word puzzles are feasible for making use of one's spare time. Especially for the elderly, which has an impact on keeping them away from forms of dementia such as Alzheimer's disease. According to Brooker et al. the middle-aged and elderly individuals who are more familiar with word puzzles, perform better in the tasks that involve cognitive functions [1]. For this reason, solving word puzzles can be considered as an appealing encouragement for a wiser mental state especially for people at the age of 50 and above. In addition, literate people under 50 can also benefit from performing word puzzle activities since such activities build up linguistic skills.

As a pleasant way of pushing limits in terms of language, puzzle-solving leads to a fine grasp of vocabulary. Being more eloquent results in better expression of self and better comprehension and communication skills among people. For similar reasons, solving crossword puzzles can be considered an interesting problem for AI systems too. Various essential subjects of AI can be utilized in solving the puzzles, such as search algorithms, heuristics, constraint satisfaction, etc. [2]

The New York Times provides a way to take one's linguistic achievements to a higher level with its daily crossword puzzles designed by Joel Fagliano [3]. We, as the group NEMESIS, implemented a program under the light of the Artificial Intelligence concepts to solve the 5x5 New York Times mini-puzzle. The program generates candidate answers to the clues respectively. To generate these candidates, online resources such as ConceptNet [4], Datamuse [5], Google [6] and the Wikipedia Python Library [7] are used. Also, the quality of the program is enhanced by the hill climbing algorithm. In the following sections, further details about the implementation of the project will be discussed with the consideration of the AI concepts' impact.

2. AI Aspect

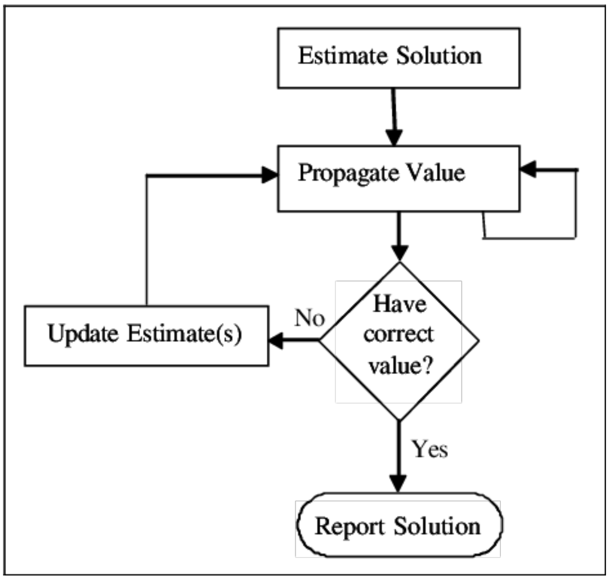


Figure 1: A flowchart model of the generate and test strategy [8].

We got use of several AI algorithms throughout the puzzle-solving process of the program. Our general approach was to use a generate and test method with constraint satisfaction approach where we generate a list composed of words to be strong candidates for the solution of each clue and improve our candidate lists using a hill climbing algorithm as a testing method.

2.1. Collecting Data

The solutions' estimation part of the generate and test method (see Figure 1) takes place in this section. To solve a crossword puzzle, one should analyze the clues in detail. Without the clues, unique solutions would not exist. They would only represent random placements in the grid which indicates the complexity of solving crossword puzzles over other computational problems in AI [9]. Therefore, we needed a list of candidate solutions to try solving the puzzle, and we produced these candidates by searching the fetched clues at different websites (see Appendix B, Figure 11). For this purpose, we used scraping methods on Google and ConceptNet websites, used the Wikipedia Python Library, and API documentations of Datamuse and MoreWords websites. We filtered our candidate lists using several constraints we thought to be appropriate. Before scraping, using the information we obtained from the "How To Solve The New York Times Crossword" article, we classified the clues. In this way, we use the most efficient website for each class of clues and get a more optimized scraping process [10]. This approach was also used by a crossword solver program called PROVERB. In PROVERB, the program uses many modules to generate a list of candidate answers for each clue. These modules consist of database modules such as movie, music and synonyms, syntactic modules such as fill-in-the-blanks and kind-of clues [11]. In our crossword solver, we used similar classes. The classes were as follows: “fill-in-the-blank”, “plural”, “cross-reference” and “abbreviation”. According to this article, each class of clues has relations with the solutions, i.e. the class of clues is also a clue for the solution. We used this classification to increase the efficiency of our data collecting algorithm. For example, if the clue is plural, the solution is plural too, and the same thing applies for the abbreviations. If the clue is “fill-in-the-blank” type, we did not search it in the dictionary websites, instead, we used Wikipedia and Google to get the solution. Moreover, we copied the texts of cross-referenced clues to each other. On the other hand, a clue may not be involved in any of these classes. In that case, we did not take any special action for it and searched it on all the websites. Clue classification was the first constraint we applied to create an effective candidate list.

Regarding the scraping part, firstly, we get clipped clues by excluding the syncategorematic words (e.g. the, a, and, of) and some special characters (e.g. dots, commas, underscores) from the clues to obtain the keywords which will lead us to the solution. We only use the Wikipedia Python Library for its search functionality to look for candidates in the resulting headings from the search. For Google, we use its auto-complete feature (see Appendix A, Figure 3) to find the solutions for the “fill-in-the-blank” type of clues. While scraping ConceptNet, we again use the search box by entering the clipped clues, and we fetch the words listed under the “Related Words” title (see Appendix A, Figure 4, and Figure 5). We retrieve most of our answers from DataMuse API, which queries the clues we provide in OneLook, RhymeZone, Rimar.io, and WikSearch websites, and outputs a JSON list that includes candidate solutions with many features (see Appendix A, Figure 7) like scorings according to the relation [5].

When we fetch the puzzle from the New York Times website, we obtain the letter count of the solution to each clue. For each clue, after scraping the website (see Appendix A, Figure X), we filter the candidate lists we obtained according to this specified letter count constraint. We also give scores to each word in the candidate lists according to their frequency, i.e. the number of websites they are outputted from. This score will then be used to determine the strength of the candidate for the solution.

2.2. Solving the Puzzle

After gathering the candidate words from the web, the algorithm to find an optimal solution for the puzzle starts running. The algorithm works by using states which are a list of selected word indexes from the candidate word lists. These states can be scored in terms of how much conflict occurs between selected words.

2.2.1. Scoring States

Assume that the cell at position (x = 2, y = 3) in the crossword puzzle corresponds to the third letter of the solution to clue 3 and the second letter of the solution to clue 7. If we choose the solution to clue 3 to be “SAND” and the solution to clue 7 “ANKH”, we can see that the third letter of “SAND” is the same as the second letter of “ANKH”. This is desirable, so it will not give the state a score penalty. However, if we select the words “SAND” and “PARK”, then, the third letter of “SAND” is not the same as the second letter of “PARK”. Thus, it will get a score penalty. The state’s score will be calculated after we check each cell in the board for conflicts. Algorithm also slightly prioritizes words with a medium number of occurrences (between 2 and 8) on the web search but this is very trivial since overusing it caused worse board results.

2.2.2. Hill Climbing

To reach the best state possible, a hill climbing algorithm is used in a loop. At each iteration of the loop, possible next states are found by replacing selected words in our current state one by one so that in every possible next state, a single candidate word is different from the previous state, but the possible decisions on which candidate word to change and with which other candidate to replace it with is completely exhausted by the algorithm. For this reason, there is a big (up to 1000s) number of possible next states evaluated in each iteration.

At the end of each iteration, possible next states are sorted by their score. Then, a semi-random beam of the next states is selected to join the next iteration of the loop as current states. This beam is selected in a way that the best 10 states are always selected and then, the worst states are selected by decreasing frequency. We do not simply just take the state with the best score to avoid local maxima.

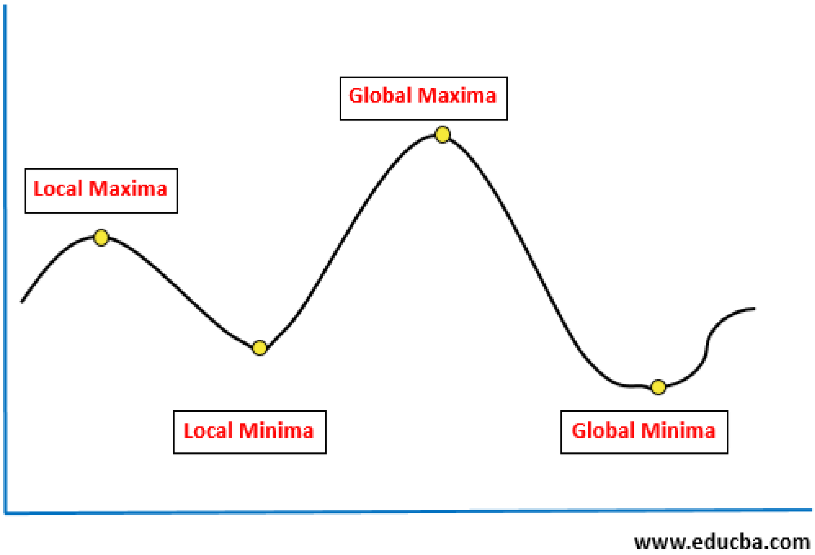


Figure 2: Hill Climbing Logic Representation [12].

2.2.3. Avoiding Local Maxima

If we examine the problem, we will see that even though it is a hill climbing problem, it is a very different and difficult one because we can choose each 10 solution word independently since we are hill climbing in a 10-dimensional space. Observations while writing the algorithm also suggested that the average score is unpleasant compared to the peaks and there are only a couple peaks that the algorithm may reach. To sufficiently solve the problem, using a beam search approach and trying as many next states (moves) as possible at each iteration was necessary. Since it is a different hill climbing problem, usual precautions to avoid local maxima would not be very effective. Instead, we simply run the algorithm a couple of times from scratch and choose the best result.

2.2.4. Finalizing the Results

If there are still conflicts in the best solution after running the algorithm a couple of times, conflicts are resolved by selecting the relevant letter from the word with a more desirable score. Word scores are calculated similarly to candidate scores, the difference is that when there is a conflict the penalty is given to the words with conflicting letters instead of the state. After reducing conflicts, the resulting board is usually very close to the official solution. The algorithm works fine and finds most of the words if most of them are located in the candidate lists that we scraped from the web.

3. Future Work

We had lots of ideas in our minds before implementing the solver, but we could not manage to put all of them into practice. In this section, we will explain some of these methods we thought that we can add to our crossword solver to make it work more efficiently and more effectively as future work.

One clever method that can increase the correctness of the candidate answer we found out is making use of the solution data retrieved from the archived puzzles. In this way, answer-clue pairs of the previous puzzles can be taken into account, and we can update the candidates accordingly. For instance, WebCrow is a web-based system that fixes this issue. In its database, previous solutions of the crossword puzzles are kept [13].

A paper that presents an approach to solving crossword puzzles that uses the Google API to obtain answers to puzzle clues, preprocesses the clues before sending the clues to the API to obtain precise results [14]. Although we follow a similar preprocessing routine by classifying clues, we can do more sensible manipulations to the clue to obtain better candidates. Therefore, the solving time can be reduced. In our case, we could not take any action for the abbreviations and plurals that we mentioned as classifications. We thought we should first search for abbreviations in a clue and replace them with their meanings before we search the whole clue on the internet. For possible plurals, we should not immediately filter the candidate lists according to the letter count of the solution to that clue, first, we should spot the nouns in the candidate list and add their plural forms to the candidate list, then, we should filter the candidate list according to the letter count.

Moreover, there could be other classifications for clues; for instance, according to the verb tenses or parts of speech. Adding these classifications can give us more clues about the solution. For example, if a clue is a verb in the past tense, so should be the solution. Besides, we can add a classification indicating the proper nouns that the clue is involving, if any, and search these proper nouns in a more appropriate website; for instance, IMDB website for TV shows, actors, directors, etc.

There are various ways to improve our system. The first possible solution that comes to mind is that we might scrape more websites to make our frequency score more reliable. We can also get use of the relation scores given by Datamuse -and maybe some other websites we will use that give a scoring- to manipulate our frequency scores. Candidates can be updated with the help of systems similar to WebCrow as well.

4. Conclusion

All in all, solving crossword puzzles is not only a thought-provoking discussion for the human being but also for the AI applications. To illustrate this, we implemented a solver program for the New York Times 5x5 mini-puzzles that scraped the online resources to generate candidate words for the solution. While solving the puzzle, we determined the possible states and scored them according to their relevance with the actual solution. With the guidance of the hill climbing algorithm, we navigated to the best state possible. As a result, we still have more methods to try for the improvement of our system but we believe that we did an excellent job so far.

References

[**1**] H. Brooker, K. A. Wesnes, C. Ballard, A. Hampshire, D. Aarsland, Z. Khan, R. Stenton, L. McCambridge, and A. Corbett, “An online investigation of the relationship between the frequency of word puzzle use and cognitive function in a large sample of older adults,” Wiley Online Library, 16-May-2019. [Online]. Available:
https://onlinelibrary.wiley.com/doi/full/10.1002/gps.5033. [Accessed: 23-Dec-2020].

[**2**] K. Thanasuan and S. T. Mueller, “Crossword expertise as recognitional decision making: an artificial intelligence approach,” Frontiers, 26-Aug-2014. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fpsyg.2014.01018/full. [Accessed: 23-Dec-2020].

[**3**] J. Fagliano, ”The New York Times Mini Crossword”, Nytimes.com. [Online]. Available: https://www.nytimes.com/crosswords/game/mini. [Accessed: 24-Dec-2020].

[**4**] ”ConceptNet”, Conceptnet.io. [Online]. Available: http://conceptnet.io/. [Accessed: 24-Dec-2020].

[**5**] Datamuse API. [Online]. Available: https://www.datamuse.com/api/. [Accessed: 23-Dec-2020].

[**6**] Google. [Online]. Available: https://www.google.com.tr/. [Accessed: 24-Dec-2020].

[**7**] “wikipedia,” PyPI. [Online]. Available: https://pypi.org/project/wikipedia/. [Accessed: 24-Dec-2020].

[**8**] I. R. Katz, A. W. Lipps, and J. G. Trafton, “Factors Affecting Difficulty In The Generating Examples Item Type,” ETS Research Report Series, vol. 2002, no. 1, pp. i-40, 2002.

[**9**] M. Ginsberg, ”Dr.Fill: Crosswords and an Implemented Solver for Singly Weighted CSPs”, J. Artif. Intell. Res., vol. 42, no. 851–886, 2011. Available:
https://arxiv.org/pdf/1401.4597.pdf. [Accessed: 24-Dec-2020].

[**10**] “How to Solve The New York Times Crossword,” The New York Times. [Online]. Available: https://www.nytimes.com/guides/crosswords/how-to-solve-a-crossword-puzzle. [Accessed: 23-Dec-2020].

[11] M. L. Littman, G. A. Keim, and N. Shazeer, "A probabilistic approach to solving crossword puzzles," Artificial Intelligence, 28-Dec-2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S000437020100114X. [Accessed: 23-Dec-2020].

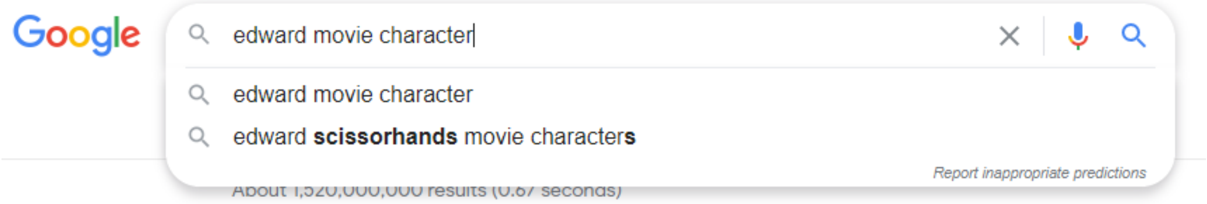
[12] "Hill Climbing in Artificial Intelligence: Types of Hill Climbing Algorithm," EDUCBA, 29-Aug-2020. [Online]. Available: https://www.educba.com/hill-climbing-in-artificial-intelligence/. [Accessed: 23-Dec-2020].

[13] A. Thomas and S. S., "Towards a Semantic Approach for Candidate Answer Generation in Solving Crossword Puzzles," Procedia Computer Science, 04-Jun-2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050920312424. [Accessed: 23-Dec-2020].

[14] D. Goldschmidt and M. Krishnamoorthy, "Solving crossword puzzles via the google api", in IADIS International Conference, Madrid, Spain, 2004, pp. 382–389. [Online]. Available: https://www.researchgate.net/publication/220969144_Solving_Crossword_Puzzles_via_the_Google_API. [Accessed: 23-Dec-2020].

Appendix A

Data Sources



Google

edward movie character

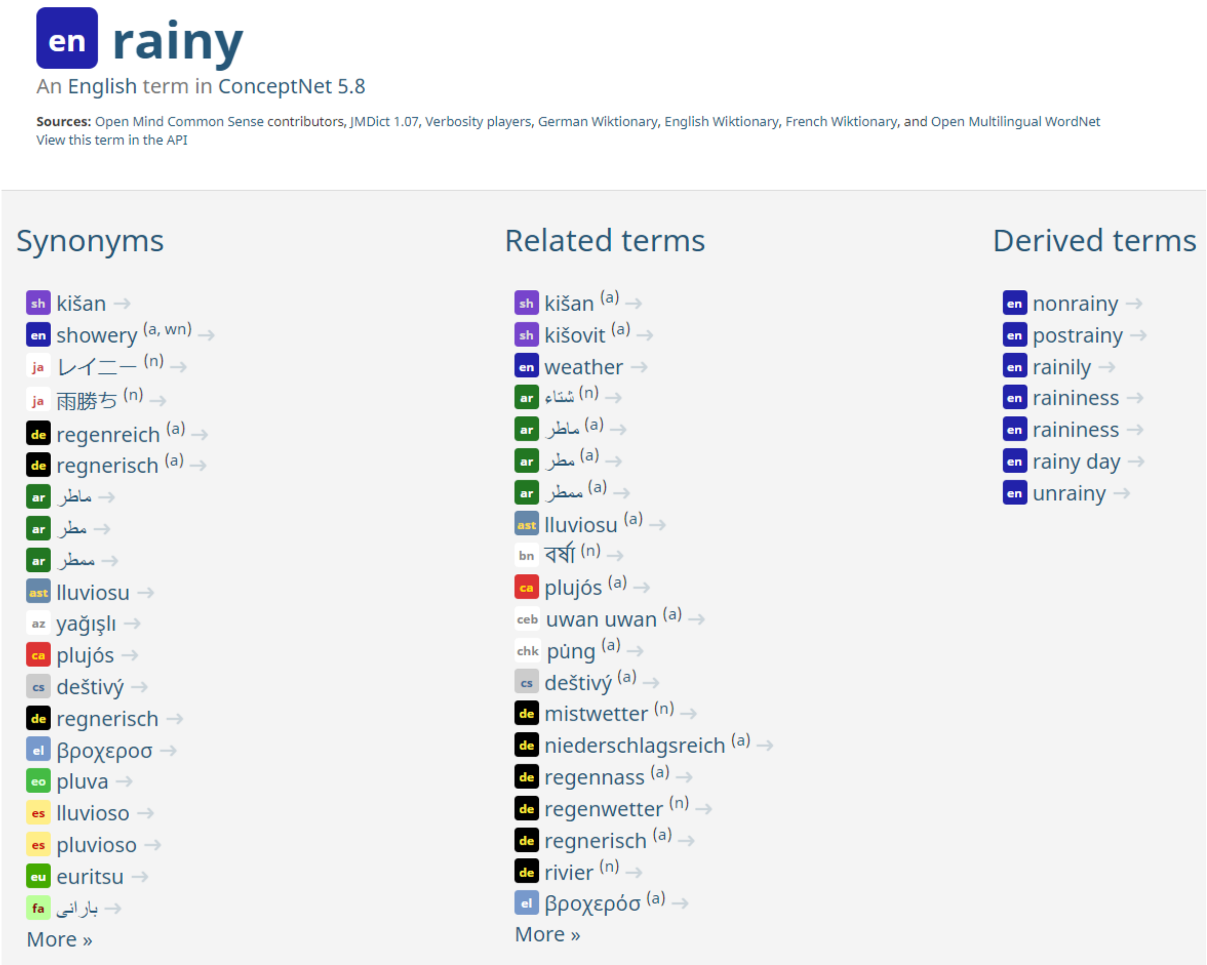
edward movie character

edward **scissorhands** movie characters

Report inappropriate predictions

About 1,520,000,000 results (0.67 seconds)

Figure 3: Google auto-complete function



en **rainy**

An English term in ConceptNet 5.8

Sources: Open Mind Common Sense contributors, JMDict 1.07, Verbosity players, German Wiktionary, English Wiktionary, French Wiktionary, and Open Multilingual WordNet

View this term in the API

Synonyms

- sh kišan →
- en showery (a, wn) →
- ja レイニー (n) →
- ja 雨勝ち (n) →
- de regenreich (a) →
- de regnerisch (a) →
- ar ماطر →
- ar مطر →
- ar ممطر →
- ast lluviosu →
- az yağışlı →
- ca plujós →
- cs deštivý →
- de regnerisch →
- el βροχεροσ →
- eo pluva →
- es lluvioso →
- es pluvioso →
- eu euritsu →
- fa بارانی →

More »

Related terms

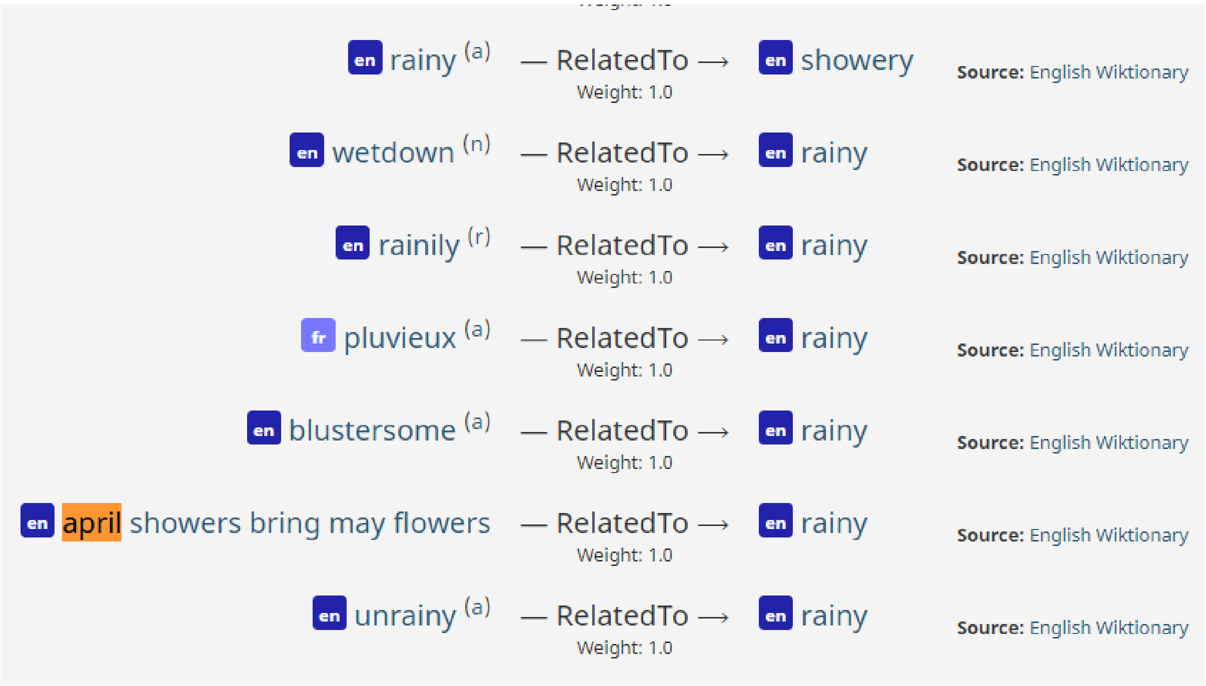
- sh kišan (a) →
- sh kišovit (a) →
- en weather →
- ar شتاء (n) →
- ar ماطر (a) →
- ar مطر (a) →
- ar ممطر (a) →
- ast lluviosu (a) →
- bn বর্ষা (n) →
- ca plujós (a) →
- ceb uwan uwan (a) →
- chk pùng (a) →
- cs deštivý (a) →
- de mistwetter (n) →
- de niederschlagsreich (a) →
- de regennass (a) →
- de regenwetter (n) →
- de regnerisch (a) →
- de rivier (n) →
- el βροχερόσ (a) →

More »

Derived terms

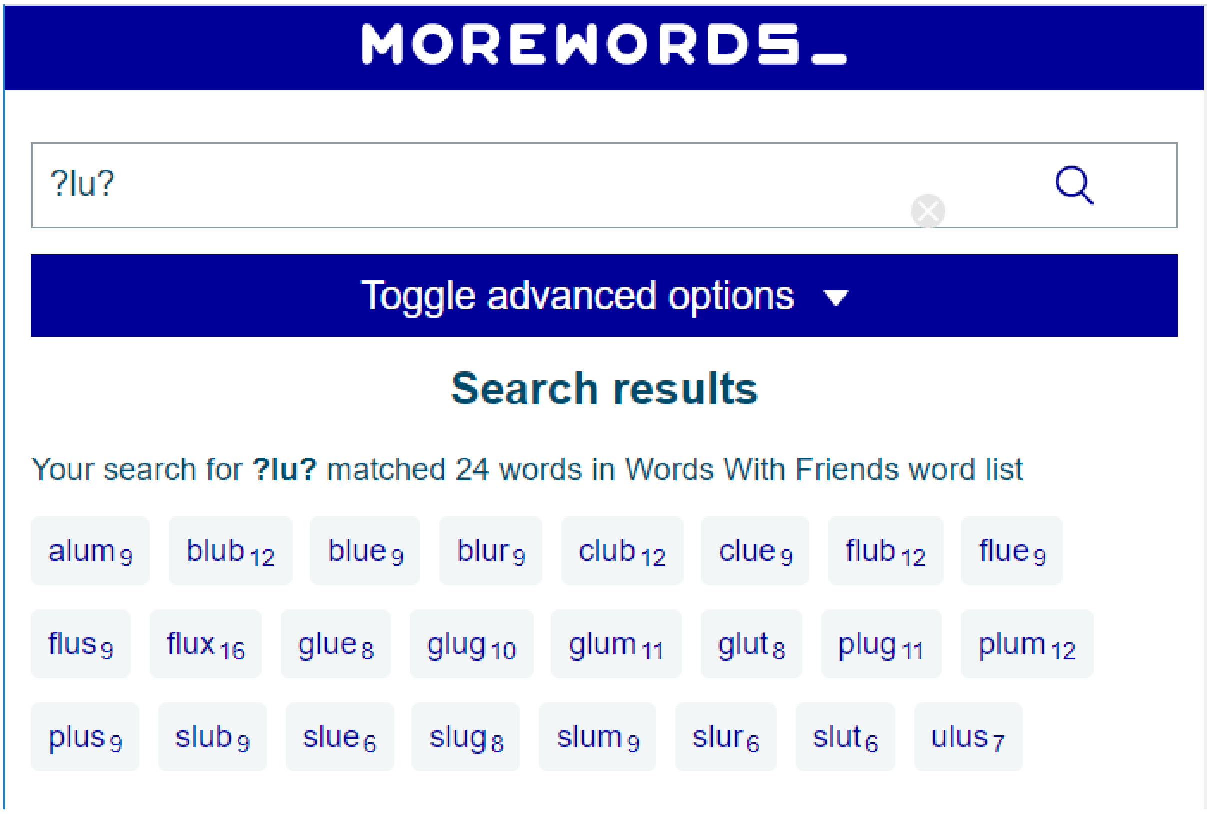
- en nonrainy →
- en postrainy →
- en rainily →
- en raininess →
- en raininess →
- en rainy day →
- en unrainy →

Figure 4: ConceptNet: "Rainy" search results



en rainy (a)	— RelatedTo → Weight: 1.0	en showery	**Source:** English Wiktionary
en wetdown (n)	— RelatedTo → Weight: 1.0	en rainy	**Source:** English Wiktionary
en rainily (r)	— RelatedTo → Weight: 1.0	en rainy	**Source:** English Wiktionary
fr pluvieux (a)	— RelatedTo → Weight: 1.0	en rainy	**Source:** English Wiktionary
en blustersome (a)	— RelatedTo → Weight: 1.0	en rainy	**Source:** English Wiktionary
en april showers bring may flowers	— RelatedTo → Weight: 1.0	en rainy	**Source:** English Wiktionary
en unrainy (a)	— RelatedTo → Weight: 1.0	en rainy	**Source:** English Wiktionary

Figure 5: ConceptNet: "Related Terms" data



MOREWORDS_

?lu?

Toggle advanced options ▼

Search results

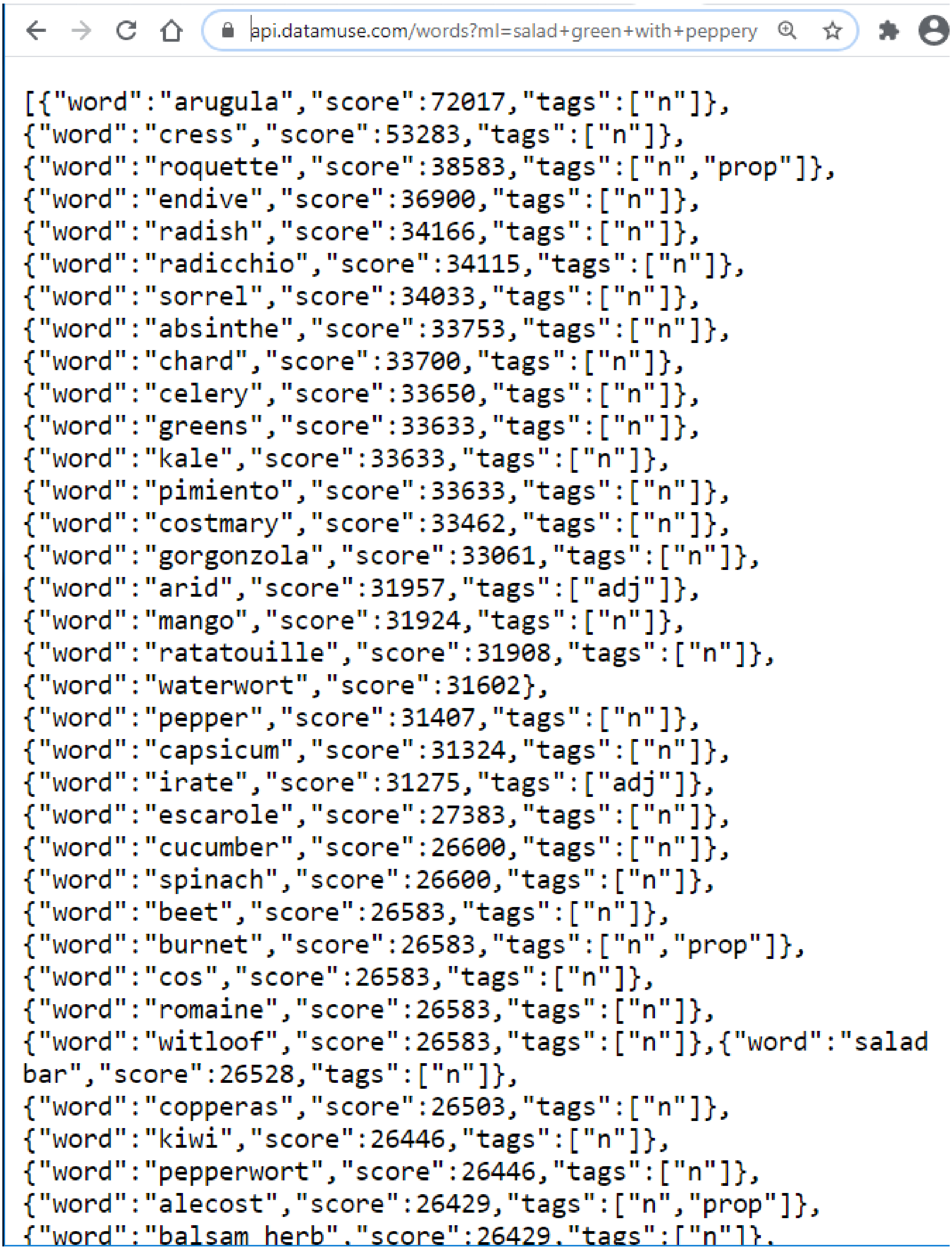
Your search for **?lu?** matched 24 words in Words With Friends word list

alum 9 blub 12 blue 9 blur 9 club 12 clue 9 flub 12 flue 9

flus 9 flux 16 glue 8 glug 10 glum 11 glut 8 plug 11 plum 12

plus 9 slub 9 slue 6 slug 8 slum 9 slur 6 slut 6 ulus 7

Figure 6: Morewords search results for "?lu?"

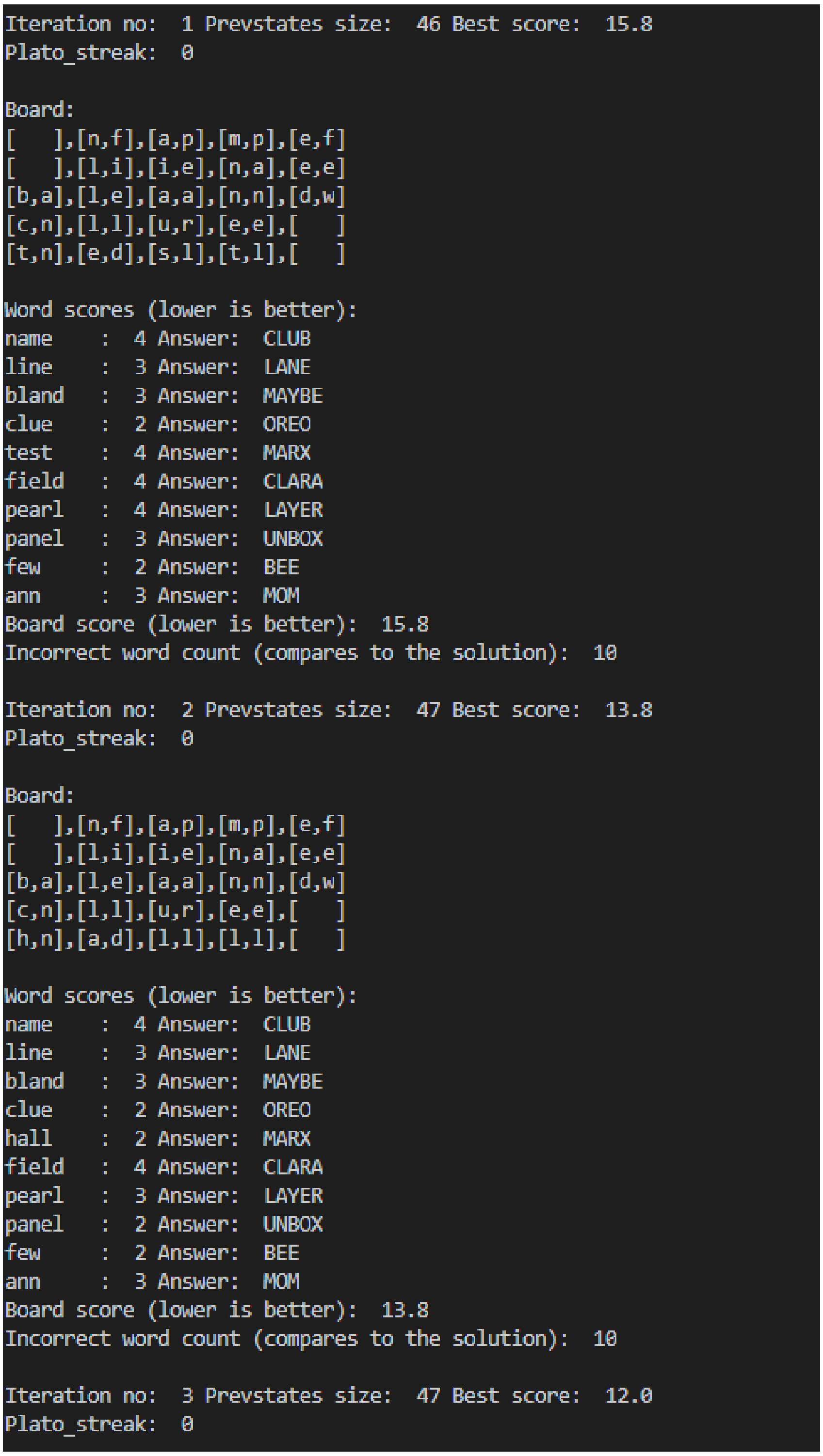


```
[{"word":"arugula","score":72017,"tags":["n"]},
{"word":"cress","score":53283,"tags":["n"]},
{"word":"roquette","score":38583,"tags":["n","prop"]},
{"word":"endive","score":36900,"tags":["n"]},
{"word":"radish","score":34166,"tags":["n"]},
{"word":"radicchio","score":34115,"tags":["n"]},
{"word":"sorrel","score":34033,"tags":["n"]},
{"word":"absinthe","score":33753,"tags":["n"]},
{"word":"chard","score":33700,"tags":["n"]},
{"word":"celery","score":33650,"tags":["n"]},
{"word":"greens","score":33633,"tags":["n"]},
{"word":"kale","score":33633,"tags":["n"]},
{"word":"pimiento","score":33633,"tags":["n"]},
{"word":"costmary","score":33462,"tags":["n"]},
{"word":"gorgonzola","score":33061,"tags":["n"]},
{"word":"arid","score":31957,"tags":["adj"]},
{"word":"mango","score":31924,"tags":["n"]},
{"word":"ratatouille","score":31908,"tags":["n"]},
{"word":"waterwort","score":31602},
{"word":"pepper","score":31407,"tags":["n"]},
{"word":"capsicum","score":31324,"tags":["n"]},
{"word":"irate","score":31275,"tags":["adj"]},
{"word":"escarole","score":27383,"tags":["n"]},
{"word":"cucumber","score":26600,"tags":["n"]},
{"word":"spinach","score":26600,"tags":["n"]},
{"word":"beet","score":26583,"tags":["n"]},
{"word":"burnet","score":26583,"tags":["n","prop"]},
{"word":"cos","score":26583,"tags":["n"]},
{"word":"romaine","score":26583,"tags":["n"]},
{"word":"witloof","score":26583,"tags":["n"]},{"word":"salad
bar","score":26528,"tags":["n"]},
{"word":"copperas","score":26503,"tags":["n"]},
{"word":"kiwi","score":26446,"tags":["n"]},
{"word":"pepperwort","score":26446,"tags":["n"]},
{"word":"alecost","score":26429,"tags":["n","prop"]},
{"word":"balsam herb","score":26429,"tags":["n"]},
```

Figure 7: Datamuse JSON API

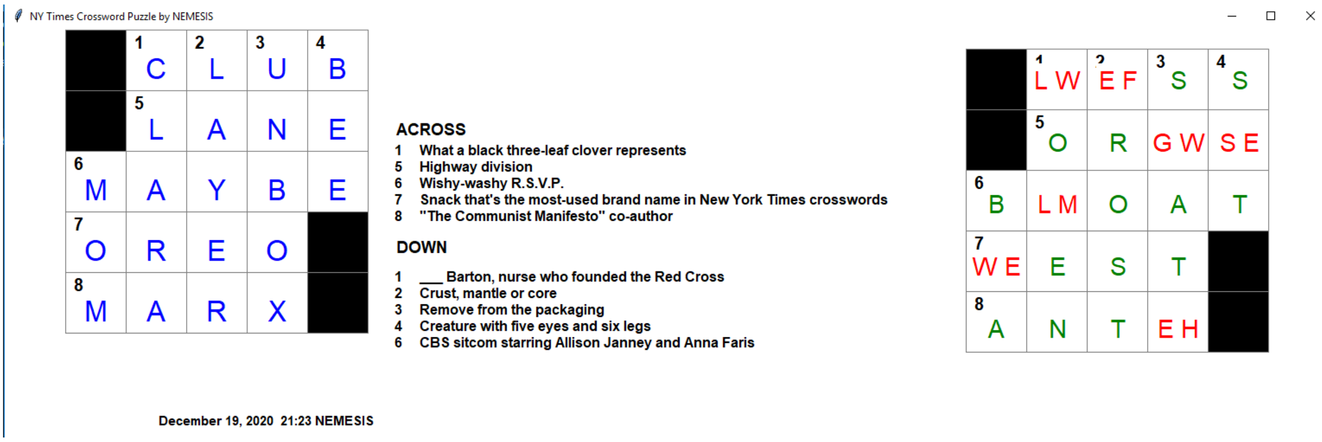
Appendix B

Output



```
Iteration no:  1 Prevstates size:  46 Best score:  15.8
Plato_streak:  0

Board:
[   ],[n,f],[a,p],[m,p],[e,f]
[   ],[l,i],[i,e],[n,a],[e,e]
[b,a],[l,e],[a,a],[n,n],[d,w]
[c,n],[l,l],[u,r],[e,e],[   ]
[t,n],[e,d],[s,l],[t,l],[   ]

Word scores (lower is better):
name    :  4 Answer:  CLUB
line    :  3 Answer:  LANE
bland   :  3 Answer:  MAYBE
clue    :  2 Answer:  OREO
test    :  4 Answer:  MARX
field   :  4 Answer:  CLARA
pearl   :  4 Answer:  LAYER
panel   :  3 Answer:  UNBOX
few     :  2 Answer:  BEE
ann     :  3 Answer:  MOM
Board score (lower is better):  15.8
Incorrect word count (compares to the solution):  10

Iteration no:  2 Prevstates size:  47 Best score:  13.8
Plato_streak:  0

Board:
[   ],[n,f],[a,p],[m,p],[e,f]
[   ],[l,i],[i,e],[n,a],[e,e]
[b,a],[l,e],[a,a],[n,n],[d,w]
[c,n],[l,l],[u,r],[e,e],[   ]
[h,n],[a,d],[l,l],[l,l],[   ]

Word scores (lower is better):
name    :  4 Answer:  CLUB
line    :  3 Answer:  LANE
bland   :  3 Answer:  MAYBE
clue    :  2 Answer:  OREO
hall    :  2 Answer:  MARX
field   :  4 Answer:  CLARA
pearl   :  3 Answer:  LAYER
panel   :  2 Answer:  UNBOX
few     :  2 Answer:  BEE
ann     :  3 Answer:  MOM
Board score (lower is better):  13.8
Incorrect word count (compares to the solution):  10

Iteration no:  3 Prevstates size:  47 Best score:  12.0
Plato_streak:  0
```

Figure 8: Hill-climbing algorithm solution trials: Console output



■	1 C	2 L	3 U	4 B
■	5 L	A	N	E
6 M	A	Y	B	E
7 O	R	E	O	■
8 M	A	R	X	■

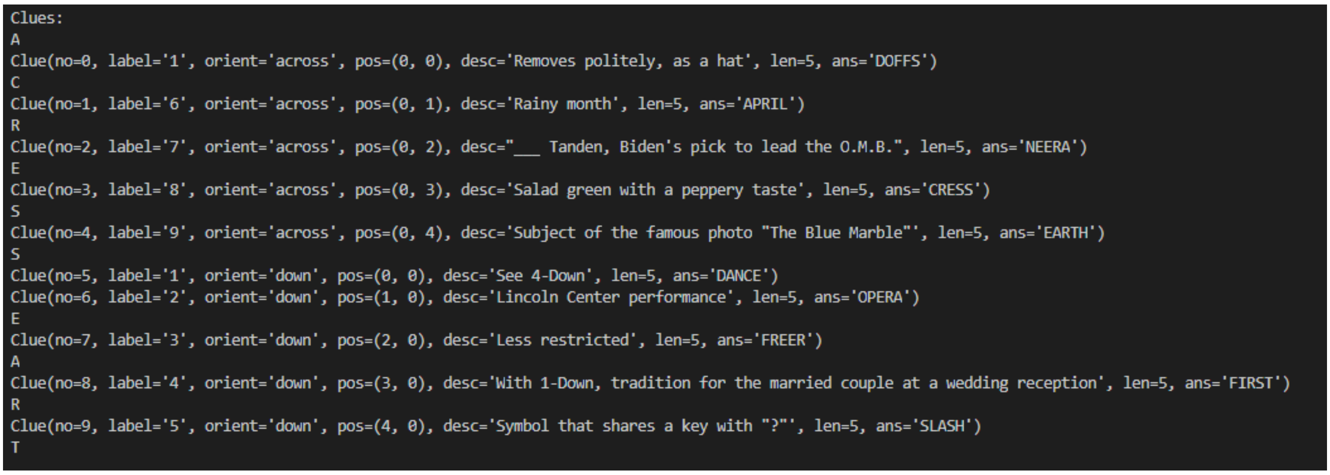
ACROSS
1 What a black three-leaf clover represents
5 Highway division
6 Wishy-washy R.S.V.P.
7 Snack that's the most-used brand name in New York Times crosswords
8 "The Communist Manifesto" co-author

DOWN
1 ___ Barton, nurse who founded the Red Cross
2 Crust, mantle or core
3 Remove from the packaging
4 Creature with five eyes and six legs
6 CBS sitcom starring Allison Janney and Anna Faris

■	1 L W	2 E F	3 S	4 S
■	5 O	R	G W	S E
6 B	L M	O	A	T
7 W E	E	S	T	■
8 A	N	T	E H	■

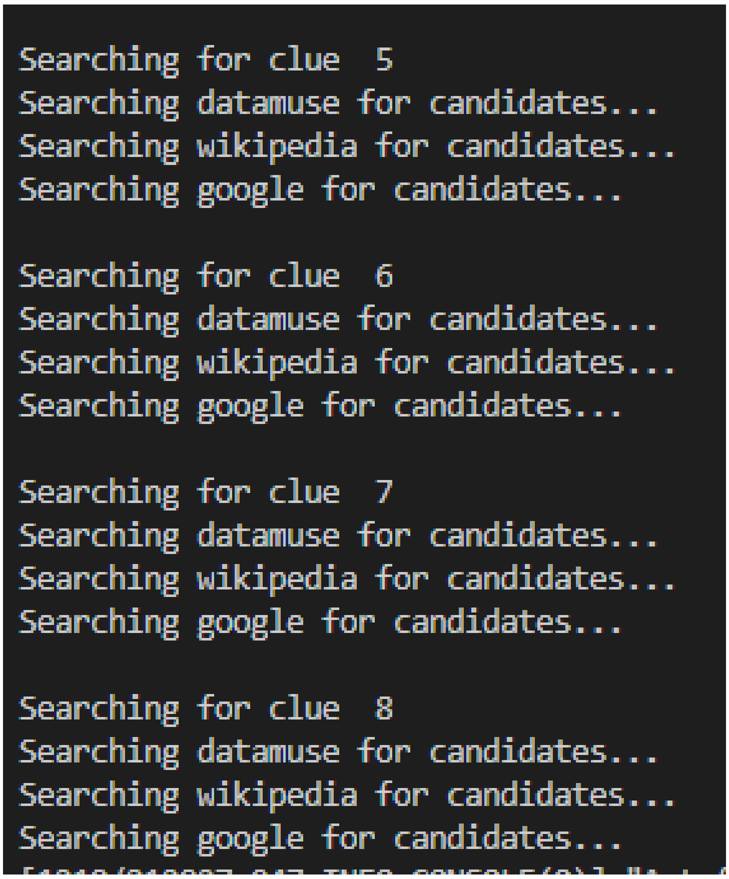
December 19, 2020 21:23 NEMESIS

Figure 9: Hill-climbing algorithm solution trials: Board Display



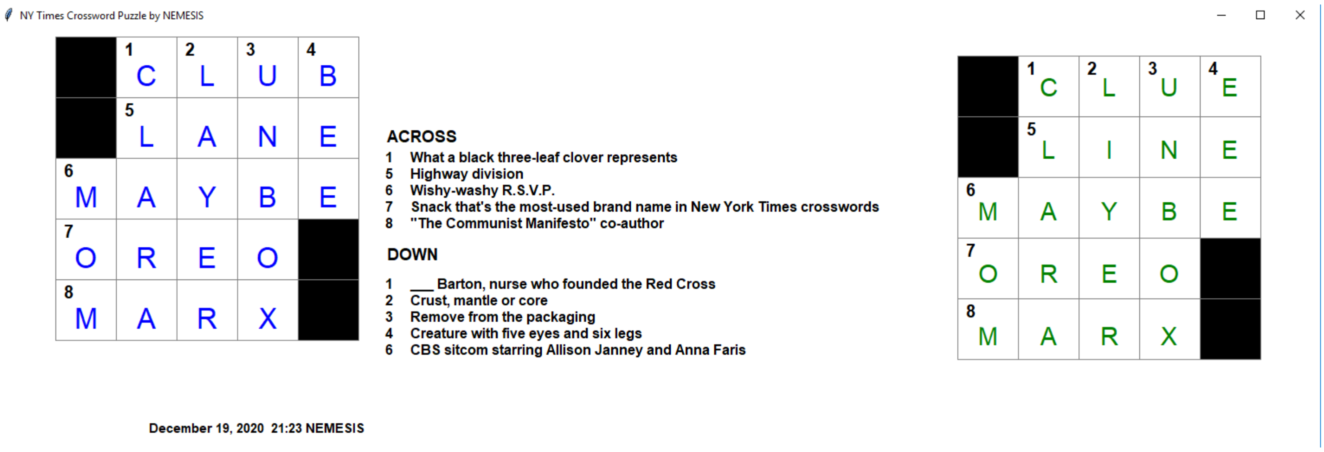
```
Clues:
A
Clue(no=0, label='1', orient='across', pos=(0, 0), desc='Removes politely, as a hat', len=5, ans='DOFFS')
C
Clue(no=1, label='6', orient='across', pos=(0, 1), desc='Rainy month', len=5, ans='APRIL')
R
Clue(no=2, label='7', orient='across', pos=(0, 2), desc="___ Tanden, Biden's pick to lead the O.M.B.", len=5, ans='NEERA')
E
Clue(no=3, label='8', orient='across', pos=(0, 3), desc='Salad green with a peppery taste', len=5, ans='CRESS')
S
Clue(no=4, label='9', orient='across', pos=(0, 4), desc='Subject of the famous photo "The Blue Marble"', len=5, ans='EARTH')
S
Clue(no=5, label='1', orient='down', pos=(0, 0), desc='See 4-Down', len=5, ans='DANCE')
Clue(no=6, label='2', orient='down', pos=(1, 0), desc='Lincoln Center performance', len=5, ans='OPERA')
E
Clue(no=7, label='3', orient='down', pos=(2, 0), desc='Less restricted', len=5, ans='FREER')
A
Clue(no=8, label='4', orient='down', pos=(3, 0), desc='With 1-Down, tradition for the married couple at a wedding reception', len=5, ans='FIRST')
R
Clue(no=9, label='5', orient='down', pos=(4, 0), desc='Symbol that shares a key with "?"', len=5, ans='SLASH')
T
```

Figure 10: Clue object structure



```
Searching for clue  5
Searching datamuse for candidates...
Searching wikipedia for candidates...
Searching google for candidates...

Searching for clue  6
Searching datamuse for candidates...
Searching wikipedia for candidates...
Searching google for candidates...

Searching for clue  7
Searching datamuse for candidates...
Searching wikipedia for candidates...
Searching google for candidates...

Searching for clue  8
Searching datamuse for candidates...
Searching wikipedia for candidates...
Searching google for candidates...
```

Figure 11: Web scraping console output



■	1 C	2 L	3 U	4 B
■	5 L	A	N	E
6 M	A	Y	B	E
7 O	R	E	O	■
8 M	A	R	X	■

ACROSS

1 What a black three-leaf clover represents
5 Highway division
6 Wishy-washy R.S.V.P.
7 Snack that's the most-used brand name in New York Times crosswords
8 "The Communist Manifesto" co-author

DOWN

1 ___ Barton, nurse who founded the Red Cross
2 Crust, mantle or core
3 Remove from the packaging
4 Creature with five eyes and six legs
6 CBS sitcom starring Allison Janney and Anna Faris

■	1 C	2 L	3 U	4 E
■	5 L	I	N	E
6 M	A	Y	B	E
7 O	R	E	O	■
8 M	A	R	X	■

December 19, 2020 21:23 NEMESIS

Figure 12: Final solution display example

Appendix C

Source Code

```python
# CS461 - Artificial Intelligence
# Term project: NYT Crossword Puzzle
# Group: NEMESIS
# Members:
# Melike Arslan 21601025
# Ece Çanga 21600851
# Nursena Kurubaş 21602965
# Selen Uysal 21702292
# Sonat Uzun 2101857
# Date: December 2020
# Note:
# Single stepping approach is available in the console.


import os

from selenium import webdriver
from selenium.common.exceptions import NoSuchElementException
from selenium.webdriver.chrome.options import Options
from datetime import date
import json

# CSS Variables
ok_button_css_list = [
    "#root > div > div > div.app-mainContainer--3CJGG > div > main "
    "> div.layout > div > div.Veil-veil--3oKaF.Veil-stretch--1wgp0 "
    "> div.Veil-veilBody--2x-ZE.Veil-autocheckMessageBody--31wj3 > div "
    "> article > div.buttons-modalButtonContainer--35RTh > button ",

    "#root > div > div > div.app-mainContainer--3CJGG > div > main "
    "> div.layout > div > div.Veil-veil--3oKaF.Veil-stretch--1wgp0 "
    "> div.Veil-veilBody--2x-ZE.Veil-standardMessageBody--1zizj > div "
    "> article > div.buttons-modalButtonContainer--35RTh > button "
]
solve_button_css = "#root > div > div > div.app-mainContainer--3CJGG > div " \
    "> main > div.layout > div > div > ul > div.Toolbar-expandedMenu--2s4M4 " \
    "> li:nth-child(2) > button"
puzzle_button_css = "#root > div > div > div.app-mainContainer--3CJGG > div " \
    "> main > div.layout > div > div > ul > div.Toolbar-expandedMenu--2s4M4 " \
    "> li.Tool-button--39W4J.Tool-tool--Fiz94.Tool-texty--2w4Br.Tool-open--1Moaq " \
    "> ul > li:nth-child(3) "
```

```
reveal_button_css = "#root > div " \
    "> div.ModalWrapper-wrapper--1GgyB.ModalWrapper-stretch--19Bif " \
    "> div.ModalBody-body--3PkKz > article " \
    "> div.buttons-modalButtonContainer--35RTh > button:nth-child(2) "
close_x_css = "#root > div " \
    "> div.ModalWrapper-wrapper--1GgyB.ModalWrapper-stretch--19Bif > span "
clues_css = "#root > div > div > div.app-mainContainer--3CJGG > div > main " \
    "> div.layout > div > article > section.Layout-clueLists--10_Xl > div "
board_css = "[data-group=\"cells\"] > g "

# JSON Variables
ny_times_data = {}
clues = []
board = []


def css_exists(element, css):
    try:
        element.find_element_by_css_selector(css)
    except NoSuchElementException:
        return False
    return True


def tag_exists(element, tag):
    try:
        element.find_element_by_tag_name(tag)
    except NoSuchElementException:
        return False
    return True


def get_clues(browser, css):
    print("Scraping all the clues...")
    clues_content = browser.find_elements_by_css_selector(css)
    for content in clues_content:
        temp = content.text + '\n'
        line, orientation, clue_number, clue_text = "", "", "", ""
        for character in temp:
            if character != '\n':
                line = line + character
            else:
                if line == "ACROSS" or line == "DOWN":
                    orientation = line
                elif line.isnumeric():
                    clue_number = line
                else:
                    clue_text = line
```

```python

                if orientation and clue_number and clue_text:
                    clues.append({
                        'orientation': orientation,
                        'label': clue_number,
                        'clue': clue_text
                    })
                    clue_number, clue_text = "", ""
                line = ""
        print("Scraped {} clues.".format(orientation))


def get_board(browser, css):
    print("\nScraping the board and the answers...")
    cells_content = browser.find_elements_by_css_selector(css)
    x, y = 0, 0
    for content in cells_content:
        rect = content.find_element_by_tag_name("rect")
        fill = rect.value_of_css_property("fill")
        width = rect.value_of_css_property("width")
        height = rect.value_of_css_property("height")
        label = ""
        answer = ""
        if tag_exists(content, "text"):
            label_css = "[text-anchor=\"start\"]"
            answer_css = "[text-anchor=\"middle\"]"
            if css_exists(content, label_css):
                label = content.find_element_by_css_selector(label_css).text

            if css_exists(content, answer_css):
                answer = content.find_element_by_css_selector(answer_css).text

        if y % 5 == 0:
            y = 0
            x += 1
        y += 1

        board.append({
            'coordinate': {'x': x, 'y': y},
            'width': width,
            'height': height,
            'label': label,
            'fill': fill,
            'answer': answer
        })


def scrape():
    # options = Options()
    # options.headless = True
    # options.add_argument("--mute-audio")

    path = os.path.abspath("chromedriver")
    url = "https://www.nytimes.com/crosswords/game/mini"
    chrome_driver = webdriver.Chrome(executable_path=path)      # options=options,
    print("Connecting to https://www.nytimes.com/crosswords/game/mini ...")
    chrome_driver.get(url)
    print("CONNECTED")

    for button in ok_button_css_list:
        if css_exists(chrome_driver, button):
            chrome_driver.find_element_by_css_selector(button).click()

    print("Closed all popups.")

    chrome_driver.find_element_by_css_selector(solve_button_css).click()
    chrome_driver.find_element_by_css_selector(puzzle_button_css).click()
    chrome_driver.find_element_by_css_selector(reveal_button_css).click()
    chrome_driver.find_element_by_css_selector(close_x_css).click()

    print("Clicked on reveal button.")

    get_clues(chrome_driver, clues_css)
    print("Finished scraping the clues!")
    print("\nTHE CLUES:")
    for c in clues:
        print(c)

    get_board(chrome_driver, board_css)
    print("Finished scraping the board and the answers!")
    print("\nTHE BOARD:")
    for b in board:
        print(b)

    ny_times_data['clues'] = clues
    ny_times_data['board'] = board

    date_today = date.today()
    json_path = "puzzles/"
    json_file = "nytimes_puzzle_{}.json".format(date_today)

    print("\nDumping the information to {} ...".format(json_file))
    with open(json_path + json_file, 'w', encoding='utf-8') as outfile:
        json.dump(ny_times_data, outfile, indent=4)

    print("\nSCRAPING DONE!")
```

```python
import itertools
import json
import os

import requests
import time
from collections import namedtuple
import wikipedia as wikipedia
from selenium import webdriver
from selenium.common.exceptions import NoSuchElementException
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.options import Options
import enchant

articles = ("a", "an", "the", "of", "at", "in", "and", "on", "to")
d = enchant.Dict("en_US")

# Constants
Clue = namedtuple('Clue', 'no label orient pos desc len ans clss')
ACROSS = 'across'
DOWN = 'down'
BLOCKED = '-'

# Global Variables
clues = []
board = [[(x, y) for x in range(5)] for y in range(5)]
label_coor = [(0, 0) for x in range(11)]


# Reads the crossword data including the board and the clues and the
# solutions from json file
def read_crossword(json_name):
    with open(json_name, 'r', encoding='utf-8') as json_file:
        data = json.load(json_file)

        for b in data['board']:
            # x and y is swapped because of a mistake in the website
            x = b['coordinate']['y'] - 1
            y = b['coordinate']['x'] - 1
            if b['label'] != '':
                label_coor[int(b['label'])] = (x, y)
            if b['answer'] == '':
                board[y][x] = BLOCKED
            else:
                board[y][x] = b['answer']

        clue_index = 0
        for c in data['clues']:
            answer = ''
            orient = ACROSS if c['orientation'] == "ACROSS" else DOWN
            length = 0
            coor = label_coor[int(c['label'])]
            x = coor[0]
            y = coor[1]
            while not (y >= 5 or x >= 5 or board[y][x] == BLOCKED):
                answer += board[y][x]
                length += 1
                x += int(orient == ACROSS)
                y += int(orient == DOWN)
            coor = label_coor[int(c['label'])]
            classificationlst = classify_clue(c['clue'])
            clues.append(Clue(clue_index, c['label'], orient, coor,
                              c['clue'], length, answer, classificationlst))
            clue_index += 1


def remove_special_characters(in_str):
    s = in_str.replace("___", "")
    s = s.replace("(", "")
    s = s.replace(")", "")
    s = s.replace("!", "")
    # s = s.replace("'", "")
    s = s.replace("-", " ")
    s = s.replace("\"", "")
    # s = s.replace("'s", "")
    s = s.replace(",", "")
    s = s.replace("?", "")
    return s


# Counts the occurences of the items in the total list
def calc_frequency(total):
    freq = {}
    for item in total:
        if item in freq:
            freq[item] += 1
        else:
            freq[item] = 1
    return freq


# Checks whether the css of the element exists
def css_exists(element, css):
    try:
        element.find_element_by_css_selector(css)
    except NoSuchElementException:
        return False
    return True


# Checks whether the clue is a cross reference clue
def is_cross_reference(word):
    ref = word.split("-")
    if len(ref) == 2:
        if ref[0].isnumeric() and (DOWN in ref[1].lower() or
                                   ACROSS in ref[1].lower()):
            return True
    return False


# Classifies the clue according to four types: "fill in the blank",
# "abbreviation", "plural", "cross-reference"
def classify_clue(clue_text):
    classifications = []
    if "___" in clue_text:
        classifications.append("fill in the blank")

    splitted = clue_text.split()
    for w in splitted:
        if w[len(w) - 1] == ".":
            classifications.append("abbreviation")
        if "they" in w.lower() or "and" == w.lower() \
                or w.lower() == "them" or "their" in w.lower():
            classifications.append("plural")

        if "-" in w:
            if is_cross_reference(w):
                classifications.append("cross-reference")

    return classifications

# Scrapes clues from the conceptnet website
def conceptnet(chrome_driver, clue_text, clue):
    print("Searching conceptnet for candidates...")
    candidate_list = []
    word_subsets = []
    splitted = clue_text.split()

    # Finds all the subsets of words of the clue
    for i in range(0, 2):
        word_subsets += list(itertools.combinations(splitted, i + 1))


    no_not_found = 0
    print(word_subsets)
    for subset in word_subsets:
        st = ""
        for s in subset:
            st += s + " "
        if subset[0] in articles or not st:
            continue

        chrome_driver.get("http://conceptnet.io/")
        search = chrome_driver.find_element_by_name("text")
        search.send_keys(st)
        search.send_keys(Keys.ENTER)
        h1 = chrome_driver.find_elements_by_css_selector(
            "#main > div.header > div > div.pure-u-2-3 > h1")

        # If nothing is found on the website, break out of the
        # loop and try the next word
        if h1[0].text == "Not found":
            no_not_found += 1
            if no_not_found == 10:
                no_not_found = 0
                break
            continue
        else:
            # Finding all the web elements according to their css selectors

            categories = chrome_driver.find_elements_by_css_selector(
                "div.rel-grid > div.pure-g > div")
            num_categories = len(categories)
            i = 0
            while i in range(0, num_categories):
                css = "#main > div.content > div.rel-grid " \
                      "> div > div:nth-child({}) > h2".format(i + 1)
                header = chrome_driver.find_element_by_css_selector(css)
                if header.text == "Related terms":
                    more_css = "#main > div.content > div.rel-grid " \
                        "> div > div:nth-child({}) > ul > li.more > a"\
                        .format(i + 1)
                    if css_exists(chrome_driver, more_css):
                        more = chrome_driver.find_element_by_css_selector(
                            more_css)
                        main_window = chrome_driver.current_window_handle
                        link = more.get_attribute("href")
                        chrome_driver.execute_script("window.open();")
                        chrome_driver.switch_to_window(
                            chrome_driver.window_handles[1])
                        chrome_driver.get(link)
```

```python

                        weights = chrome_driver.find_elements_by_css_selector(
                            "div.weight")
                        ind_weight_reached = 1
                        for weight in weights:
                            w = float(weight.text.strip()[8:])
                            if w > 1.0:
                                ind_weight_reached += 1
                            else:
                                break

                        for j in range(1, ind_weight_reached):
                            start_css = "div.edge-list > table > tbody " \
                                        "> tr:nth-child({}) > td.edge-start " \
                                        "> span.term.lang-en > a".format(j)
                            if css_exists(chrome_driver, start_css):
                                start_edge = chrome_driver.\
                                    find_element_by_css_selector(start_css)
                            else:
                                continue

                            candidate0 = start_edge.text
                            for art in articles:
                                if start_edge.text.startswith(art + " "):
                                    candidate0 = start_edge.text.replace(
                                        art + " ", "")

                            if candidate0 == clue_text:
                                continue

                            if len(candidate0) == int(clue.len):
                                candidate_list.append(candidate0.lower())

                        chrome_driver.close()
                        chrome_driver.switch_to_window(main_window)
                    i = num_categories
                else:
                    i += 1

    return list(set(candidate_list))

# Scrapes clues from the datamuse API
def datamuse(clue_text, clue):
    print("Searching datamuse for candidates...")
    response = requests.get("https://api.datamuse.com/words",
                            params={"ml": clue_text})
    json_resp = response.json()
    candidate_list = []
    if len(json_resp) != 0:
        for resp in json_resp:
            if "score" in resp:
                if len(resp["word"]) == clue.len:
                    candidate_list.append(resp["word"].lower())
    return candidate_list


# Scrapes clues from the wikipedia API
def wiki(clue_text, clue):
    print("Searching wikipedia for candidates...")
    search = wikipedia.search(clue_text)
    cand_list = []
    for item in search:
        splitted = item.split()
        for s in splitted:
            if len(s) == clue.len:
                cand_list.append(s.lower())

    return cand_list

# Scrapes clues from the google search engine
def google(clue_text, clue, autoComplete=False):
    print("Searching google for candidates...")
    chrome_driver.get("https://www.google.com/")
    time.sleep(1)
    candidate_list = []
    search = chrome_driver.switch_to.active_element
    search.send_keys(clue_text)
    search.send_keys(Keys.ENTER)
    chrome_driver.get(chrome_driver.current_url + "&lr=lang_en")
    search = chrome_driver.find_element_by_name("q")
    search.clear()

    if autoComplete:
        # Scrapes the autocomplete suggestions
        splitted = clue_text.split()
        word_so_far = ""
        suggestions = None
        for word in splitted:
            word_so_far += word + " "
            search.send_keys(word + " ")
            if css_exists(chrome_driver, "#tsf > div:nth-child(2) "
                                "> div.A8SBwf.emcav > div.UUbT9 > div.aajZCb "
                                "> ul > li > div > div.sbtc > div.sbl1 > span"):
                suggestions = chrome_driver.find_elements_by_css_selector(
                    "#tsf > div:nth-child(2) > div.A8SBwf.emcav > div.UUbT9 "
                    "> div.aajZCb > ul > li > div > div.sbtc > div.sbl1 > span")
            if suggestions:
                for sugg in suggestions:
                    cand = sugg.text
                    if cand in sugg.text and word_so_far in sugg.text:
                        cand = cand[cand.index(word_so_far) + len(word_so_far):]
                        cand = cand.strip()
                    if len(cand) == clue.len:
                        candidate_list.append(cand.lower())

    # Scrapes the search results after the clue has been written
    # to search bar and entered
    search.clear()
    search.send_keys(clue_text)
    search.send_keys(Keys.ENTER)
    chrome_driver.get(chrome_driver.current_url + "&lr=lang_en")
    all = chrome_driver.find_elements_by_id("rso")
    for elem in all:
        lastword = ""
        for word in elem.text.split():
            if len(word) == clue.len:
                candidate_list.append(word.lower())


    return candidate_list

# According to the results coming from the hillclimb algorithm,
# the function searches for the unknown letters that hillclimb
# couldn't find in the board
def morewords(word):
    res = []
    chrome_driver.get("https://www.morewords.com/")
    if css_exists(chrome_driver, "input.mirror"):
        search = chrome_driver.find_element_by_css_selector("input.mirror")
        search.send_keys(word)
        search.send_keys(Keys.ENTER)
        if css_exists(chrome_driver, "#thecontent > div > div.col-md-8 "
                                     "> div > h1"):
            result_word = chrome_driver.find_element_by_css_selector(
                "#thecontent > div > div.col-md-8 > div > h1")
            res.append(result_word)
        if css_exists(chrome_driver, "#thecontent > div.search > div "
                                     "> div.col-md-8 > div > p > a"):
            search_results = chrome_driver.find_elements_by_css_selector(
                "#thecontent > div.search > div > div.col-md-8 > div > p > a")
            for r in search_results:
                ans = ''.join(i for i in r.text if not i.isdigit())
                res.append(ans)

    return res


def get_candidates():
    return candidates_list

with open('date.json', 'r', encoding='utf-8') as json_file:
        date = json.load(json_file)
json_filename = "puzzles/nytimes_puzzle_" + date + ".json"

read_crossword(json_filename)

options = Options()
options.headless = True
# options.add_argument("--mute-audio")

path = os.path.abspath("chromedriver")
chrome_driver = webdriver.Chrome(executable_path=path, options=options)
candidates_list = []

print('\nClues:')
for c in clues:
    print(c)

print('\nBoard:')
for b in board:
    print(b)


for ci in range(len(clues)):
    clue = clues[ci]
    print("\nSearching for clue ", ci)
    clue_text = remove_special_characters(clue.desc)

    datamuse_list = datamuse(clue_text, clue)
    concept_list = conceptnet(chrome_driver, clue_text, clue)
    wiki_list = wiki(clue_text, clue)
    google_list = []
    try:
        google_list = google(clue_text, clue, True)
    except Exception:
        pass

    total = concept_list + datamuse_list + wiki_list + google_list
    result_dict = calc_frequency(total)
    candidates_list.append(result_dict)
```

```python
import json
import copy
import time
import random
from collections import namedtuple

date = input('Input date (yyyy-mm-dd): ')

with open('date.json', 'w') as outfile:
     json.dump(date, outfile)

# from solution_displayer import ret_board, upload_puzzle
from crossword_solver import morewords, get_candidates

#simplify the data
#json_name = input("\nEnter the name of the json file: ")

#a random state to start hill climbing
def random_state():
    res = copy.deepcopy(initial_state)
    for i in range(clue_count):
        res[i] = random.randrange( len(candidates[i]) )
    return res

#calculates the score for a board state
#it mainly takes account if the crossing points of the
# selected candidates are matching
#it also slightly favours words with medium frequencies
def score(state): # lower is better
    res = 0
    curr_words = []
    curr_word_scores = []
    for i in range(clue_count):
        word = candidates[i][state[i]]
        curr_words.append(word)
        if word in candidate_data[i].keys():
            res -= 0.2 * int (  2 < candidate_data[i][word] < 8)
    for by in ind_board:
        for b in by:
            if len(b) == 2:
                l1 = curr_words[b[0][0]][b[0][1]].lower()
                l2 = curr_words[b[1][0]][b[1][1]].lower()
                res += int( l1 != l2 ) #if letters aren't matching
    return res

#a list for scores in selected candidates in a given state
def candidate_score(state):
    res = [ 0 for i in range(clue_count) ]
    curr_words = []
    for i in range(clue_count):
        curr_words.append(candidates[i][state[i]])
    for by in ind_board:
        for b in by:
            if len(b) == 2:
                l1 = curr_words[b[0][0]][b[0][1]].lower()
                l2 = curr_words[b[1][0]][b[1][1]].lower()
                res[b[0][0]] += int( l1 != l2  ) #if letters aren't matching
                res[b[1][0]] += int( l1 != l2  ) #if letters aren't matching
    return res


#number of incorrect words in a given state
def incorrect_words(state):
    res = 0
    for i in range(clue_count):
        if clues[i].ans.lower() != candidates[ i ][ state[i] ].lower():
            res += 1
    return res

#this is a little hill climber brute with beam search
# it tries to minimize the board score
# and works half of the time

def hill_climb(state, climb_length):
    iteration_count = 0
    max_plato = 4
    plato_streak = 0
    prevStates = [state]
    nextStates = []
    while iteration_count < climb_length and score(prevStates[0]) != 0 \
            and plato_streak < max_plato:
        for s in prevStates:
            for i in range(clue_count):
                nextState = copy.deepcopy(s);
                for j in range(len(candidates[i])):
                    nextState[i] = j
                    nextStates.append(copy.deepcopy(nextState))
        nextStates.sort(reverse=False,key=score)
        temp = []

        #remove duplicates
        for s in nextStates:
            if temp[0:len(temp)] == s[0:len(s)]:
                nextStates.remove(s)
            else:
                temp = s

        #a random beam for a higher chance of success
        nslen = len(nextStates) / 20
        if score( nextStates[0] ) == score( prevStates[0] ):
            plato_streak += 1
        else:
            plato_streak = 0
        prevStates = nextStates[0:min(10,len(nextStates))] \
            + nextStates[min(10,len(nextStates)) \
                         + (iteration_count % 10):min(100,len(nextStates)):10] \
            + nextStates[min(100,len(nextStates)) \
                         + (iteration_count % 100):min(1000,len(nextStates)):100] \
            + nextStates[min(1000,len(nextStates)) \
                         - iteration_count:len(nextStates):int(nslen)] \

        #reset
        print("\nIteration no: ", iteration_count,
              "Prevstates size: ", len(prevStates),
              "Best score: ", score( prevStates[0] ) )
        print("Plato_streak: ", plato_streak)
        print_board(prevStates[0])
        nextStates = []
        iteration_count += 1
    return prevStates[0]

#a function that calls hill climbing many times
#this solution was suitable for this particular case of hill climbing
#it decreases our chance of getting stuck in local maximums
def trekking_trip(hill_climb_count = 5, climb_length = 20):
    print("Printing candidates 7: ", candidates[7])
    global state, cand_scores
    min_score = 999
    state = initial_state
    best_state = state
    for i in range(hill_climb_count):
        print("\nHill climb: " , i)
        res = hill_climb(state, climb_length)
        if score(res) <= min_score:
            min_score = score(res)
            best_state = res
        if min_score == 0:
            break;
        state = random_state()

    state = best_state

#a function to print the board
```

```
#and other information
def print_board(state): # lower is better
    res = 0
    global curr_words
    curr_words = []
    for i in range(clue_count):
        curr_words.append(candidates[i][state[i]])
    ind = 0
    print("\nBoard: ")
    for y in range(5):
        pr = ""
        for x in range(5):
            l1 = " "
            l2 = " "
            b = ind_board[y][x]
            pr += "["
            if len(b) >= 1:
                l1 = curr_words[b[0][0]][b[0][1]].lower()
                pr += l1
            else:
                pr += " "
            if len(b) == 2:
                l2 = curr_words[b[1][0]][b[1][1]].lower()
                pr += "," + l2
            else:
                pr += "  "
            pr += '],'
            board[y][x] = (l1, l2)
        print(pr[0:len(pr)-1])

    global cand_scores
    cand_scores = candidate_score(state)
    print("\nWord scores (lower is better): ")
    for i in range(clue_count):
        print( candidates[i][ state[i] ], "\t: ", cand_scores[i],
               "Answer: ", clues[i].ans  )

    print( "Board score (lower is better): ", score(state) )
        #incorrect words function cheats by looking at the answers by the way
    print( "Incorrect word count (compares to the solution): ",
           incorrect_words(state) )
    time.sleep(0.5)
    return res

def get_board():
    return board

def get_curr_words():
    return curr_words


Clue = namedtuple('Clue','no label orient pos desc len ans')
ACROSS = 'across'
DOWN = 'down'
BLOCKED = '-'
clues = []
words = [ None for x in range(11) ]
board = [[ (x,y) for x in range(5) ] for y in range(5)]

#for storing clue letter indexes
ind_board = [[ [] for x in range(5)] for y in range(5)]
clue_coor = [(0,0) for x in range(11)]

clue_count = 10
initial_state = [ 0 for w in words ]
state = initial_state
candidates = [ [] for w in words ]
candidate_date = []
cand_scores = [ 0 for i in range(10) ]
curr_words = [ "" for i in range(10) ]

def main():
    global clues
    global words
    global board
    global ind_board
    global clue_coor

    global clue_count
    global initial_state
    global state
    global candidates
    global candidate_date
    global cand_scores

    #gather board data for the selected date
    with open('puzzles/nytimes_puzzle_' + date + '.json', 'r',
              encoding='utf-8') as json_file:
        data = json.load(json_file)

        for b in data['board']:
            #x and y is swapped because of a mistake in the website
            x = b['coordinate']['y'] - 1
            y = b['coordinate']['x'] - 1
            if b['label'] != '':
                clue_coor[int(b['label'])] = (x,y)
            if b['answer'] == '':
                board[y][x] = BLOCKED
            else:
                board[y][x] = b['answer']

        clue_index = 0
        for c in data['clues']:
            answer = ''
            orient = ACROSS if c['orientation'] == "ACROSS" else DOWN
            length = 0
            coor = clue_coor[int(c['label'])]
            x = coor[0]
            y = coor[1]
            while not (y >= 5 or x >= 5 or board[y][x] == BLOCKED ):
                ind_board[y][x].append( (clue_index, length) )
                answer += board[y][x]
                length += 1
                x += int( orient == ACROSS )
                y += int( orient == DOWN )
            coor = clue_coor[int(c['label'])]
            clues.append( Clue(clue_index, c['label'], orient, coor,
                               c['clue'], length, answer ) )
            clue_index += 1
        for i in range(clue_index):
            words[i] = clues[i].ans

    for i in range(clue_count):
        global candidate_data
        candidate_data = get_candidates()
        keys = candidate_data[i].keys()
        for k in keys:
            if k.isalpha():
                candidates[i].append(k)

    print('\nClues:')
    for c in clues:
        print(c)

    print('\nBoard:')
    for b in board:
        print(b)

    print('\nWord index Board:')
    for b in ind_board:
        print(b)


    state = copy.deepcopy(initial_state)
    trekking_trip()
    print_board(state)

    #this part results in the final board state
    for i in range(1):
        query = []
        second_worst_score = 0
        worst_score = 0
        for j in range(clue_count):
            if cand_scores[j] >= worst_score:
                second_worst_score = worst_score
                worst_score = cand_scores[i]
            elif cand_scores[j] > second_worst_score:
                second_worst_score = cand_scores[j]
        treshold = second_worst_score
        for i in range(clue_count):
            word = candidates[i][state[i]]
            query.append(list(word))
        for by in ind_board:
            for b in by:
                if len(b) == 2:
                    i1 = b[0][0]
                    i2 = b[1][0]
                    li1 = b[0][1]
                    li2 = b[1][1]
                    s1 = cand_scores[i1]
                    s2 = cand_scores[i2]

                    if s2 >= s1:
                        query[i2][li2] = query[i1][li1]
                    else:
                        query[i1][li1] = query[i2][li2]
        for i in range(clue_count):
            query[i] = "".join(query[i])
            candidates[i][0] = query[i]
        state = initial_state
        print_board(state)
        print("\nFinished solving the puzzle.")
```

```
from tkinter import *
import json
from datetime import date, datetime
import crossword_scraper
import os
from hillclimb import main, get_board, curr_words
from threading import Thread
```

```python

# global clues_across
clues_across = []
# global clues_down
clues_down = []
# global board
board = []

# Gets the today answer from NYTimes website
def get_today():
    print("\nGetting today's puzzle...")
    crossword_scraper.scrape()
    today = date.today()
    json_path = "puzzles/"
    json_name = "nytimes_puzzle_{}.json".format(today)
    with open(json_path + json_name, 'r', encoding='utf-8') as json_file:
        data = json.load(json_file)
        for c in data['clues']:
            if c['orientation'] == "ACROSS":
                clues_across.append(c)

            if c['orientation'] == "DOWN":
                clues_down.append(c)

        for b in data['board']:
            board.append(b)



# Uploads old puzzle data from folder
def upload_puzzle():
    global clues_across
    global clues_down
    global board
    with open('date.json', 'r', encoding='utf-8') as json_file:
        date = json.load(json_file)
    json_path = 'puzzles/nytimes_puzzle_' + date + '.json'
    if not os.path.exists(json_path):
        get_today()
        return
    with open(json_path, 'r', encoding='utf-8') as json_file:
        data = json.load(json_file)
        for c in data['clues']:
            if c['orientation'] == "ACROSS":
                clues_across.append(c)

            if c['orientation'] == "DOWN":
                clues_down.append(c)
        for b in data['board']:
            board.append(b)

# Uploads the link of the old puzzle
def upload_puzzle_link(json_name):
    json_path = "puzzles/"
    with open(json_path + json_name, 'r', encoding='utf-8') as json_file:
        data = json.load(json_file)
        for c in data['clues']:
            if c['orientation'] == "ACROSS":
                clues_across.append(c)

            if c['orientation'] == "DOWN":
                clues_down.append(c)

        for b in data['board']:
            board.append(b)


def ret_board():
    return board

def ret_across():
    return clues_across


def ret_down():
    return clues_down

# Call main
def cont(event=None):
    main()

def clock():
    board = get_board()
    for y in range(5):
        for x in range(5):
            try:
                l1 = str(board[y][x][0]).upper()
                l2 = str(board[y][x][1]).upper()
            except Exception:
                print( board[y][x] )
            if l1 == l2:
                board_text[y][x]['text'] = l1
                board_text[y][x]['fg'] = 'green'
            else:
                board_text[y][x]['text'] = l1 + " " + l2
                board_text[y][x]['fg'] = 'red'
    our_main.after(100, clock) # run itself again after 1000 m

upload_puzzle()

## Display of the complete GUI
print("\nDisplaying the puzzle...")
our_main = Tk()
our_main.title("NY Times Crossword Puzzle by NEMESIS")
our_main.config(bg='#FFFFFF')
left = Frame(our_main, width=400, height=400, background='white',
             borderwidth=0, highlightthickness=0)
right = Frame(our_main, width=80, height=400, background='white',
              borderwidth=0, highlightthickness=0)
right_most = Frame(our_main, width=400, height=400, background='white',
                   borderwidth=0, highlightthickness=0)
left.pack(side=LEFT)
right.pack(side=LEFT)
right_most.pack(side=LEFT)

board_canvas = Canvas(left, width=430, height=430, background='white',
                      bd=0, highlightthickness=0)

now = datetime.now()
formatted_now = now.strftime("%B %d, %Y  %H:%M NEMESIS")
bottom_label_frame = Frame(left, width=50, height=50,
                           background='white', pady=10)
bottom_label = Label(bottom_label_frame, text=formatted_now,
                     background='white', font="franklin 11 bold")
bottom_label.pack(side=RIGHT, anchor=E)
bottom_label_frame.pack(anchor=E, side=BOTTOM)

width = 70
height = 70

our_board = board

for b in board:
    y0 = width * (b['coordinate']['x'] - 1)
    x0 = height * b['coordinate']['y']
    x1 = x0 + width
    y1 = y0 + height
    board_canvas.create_line(x0, y0, x0, y1, fill="light grey")
    board_canvas.create_line(x0, y0, x1, y0, fill="light grey")
    if b['fill'] == "rgb(0, 0, 0)":
        board_canvas.create_rectangle(x0, y0, x1, y1,
                                      fill='black', outline='grey')
    else:
        rect = board_canvas.create_rectangle(x0, y0, x1, y1,
                                             fill='white', outline='grey')
        if b['label']:
            board_canvas.create_text(x0 + 15, y0 + 15, text=b['label'],
                                     fill='black', font='arial 15 bold')
        if b['answer']:
            board_canvas.create_text(x0 + width / 2, y0 + height / 2 + 10,
                                text=b['answer'], fill='blue', font='arial 25')

board_canvas.pack(side=LEFT)

clues_frame = Frame(right, width=80, height=700, background='white',
                    highlightthickness=0)

clues_across_frame = Frame(clues_frame, width=50, height=6,
                           background='white', padx=20, highlightthickness=0)
label_across_frame = Frame(clues_frame, width=50, height=6,
                           background='white', pady=0, padx=20)
across_text_area = Text(clues_across_frame, width=80, height=6,
                        background='white', bd=0, highlightthickness=0)
across_label = Label(label_across_frame, text='ACROSS',
                     background='white', font="franklin 14 bold")

label_across_frame.pack(anchor=NW)
clues_across_frame.pack(side=TOP)
across_label.pack(side=LEFT)
across_text_area.pack(fill=BOTH)

clues_down_frame = Frame(clues_frame, width=80, height=6,
                         background='white', padx=20)
label_down_frame = Frame(clues_frame, width=50, height=6,
                         background='white', pady=10, padx=20)
down_text_area = Text(clues_down_frame, width=80, height=6,
                      background='white', bd=0, highlightthickness=0)
down_label = Label(label_down_frame, text='DOWN', background='white',
                   font="franklin 14 bold")

label_down_frame.pack(anchor=NW)
clues_down_frame.pack(side=TOP)
down_label.pack(side=LEFT)
down_text_area.pack(fill=BOTH)

line_ind = 0
label_end = 0
char_ind = 0
for clue in clues_across:
    label_end = 0
    char_ind = 0
    text = clue['label'] + "     " + clue['clue'] + "\n"
    across_text_area.insert(END, text)
    for c in text:
        if c == "     ":
            break
        label_end += 1

    char_ind = label_end + 1
    line_ind += 1

    label_tag_start = str(line_ind) + ".0"
    label_tag_end = str(line_ind) + "." + str(char_ind)
    across_text_area.tag_add("label", label_tag_start, label_tag_end)

    clue_tag_start = str(line_ind) + "." + str(char_ind)
    clue_tag_end = str(line_ind) + "." \
                   + str(len(clue['label'] + " " + clue['clue']))
    across_text_area.tag_add("clue", clue_tag_start, clue_tag_end)
    across_text_area.tag_config("label", font="franklin 12 bold")
    across_text_area.tag_config("clue", font="franklin 11")

line_ind = 0
label_end = 0
char_ind = 0
for clue in clues_down:
    label_end = 0
    char_ind = 0
    text = clue['label'] + "     " + clue['clue'] + "\n"
    down_text_area.insert(END, text)
    for c in text:
        if c == "     ":
            break
        label_end += 1

    char_ind = label_end + 1
    line_ind += 1

    label_tag_start = str(line_ind) + ".0"
    label_tag_end = str(line_ind) + "." + str(char_ind)
    down_text_area.tag_add("label", label_tag_start, label_tag_end)

    clue_tag_start = str(line_ind) + "." + str(char_ind)
    clue_tag_end = str(line_ind) + "." \
                   + str(len(clue['label'] + " " + clue['clue']))
    down_text_area.tag_add("clue", clue_tag_start, clue_tag_end)
    down_text_area.tag_config("label", font="franklin 12 bold")
    down_text_area.tag_config("clue", font="franklin 11")
clues_frame.pack()
```

```python

our_board_canvas = Canvas(right_most, width=500, height=430,
                          background='white', bd=0, highlightthickness=0)

width = 70
height = 70

board_text = [ [ None for x in range(5) ] for y in range(5) ]


for b in our_board:
    x = b['coordinate']['y'] - 1
    y = b['coordinate']['x'] - 1
    y0 = width *  (b['coordinate']['x'] - 1)
    x0 = height * (b['coordinate']['y'] - 1)
    x1 = x0 + width
    y1 = y0 + height
    our_board_canvas.create_line(x0, y0, x0, y1, fill="light grey")
    our_board_canvas.create_line(x0, y0, x1, y0, fill="light grey")
    if b['fill'] == "rgb(0, 0, 0)":
        our_board_canvas.create_rectangle(x0, y0, x1, y1,
                                          fill='black', outline='grey')
    else:
        rect = our_board_canvas.create_rectangle(x0, y0, x1, y1,
                                        fill='white', outline='grey')
        if b['label']:
            our_board_canvas.create_text(x0 + 15, y0 + 15,
                            text=b['label'], fill='black', font='arial 15 bold')
    board_text[y][x] = Label(our_board_canvas, text=" ",
                             fg='blue', bg='white', font='arial 22' )
    if b['fill'] == "rgb(0, 0, 0)":
        board_text[y][x]['bg'] = 'black'
    board_text[y][x].place( relx = (1 + 2*x) / 12.0,
                            rely = (1 + 2*y) / 12.0, anchor = 'center')
our_board_canvas.pack(side=LEFT)
search_thread = Thread(target=main, daemon=True)
search_thread.start()
clock()
our_main.mainloop()
```

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 NEMESIS.

Word Count: **2265**