Bilkent University

Computer Engineering

CS 461 Artificial Intelligence

Term Project

X-WORD *NYTimes Crossword Clue Generator*

Fuad Aghazada

Can Özgürel

Kaan Kıranbay

Çağatay Sel

Utku Mert Topçuoğlu

Section: 1

04.05.2019

Table of Contents

1.0 INTRODUCTION **3**

2.0 PROBLEM APPROACH **3**

3.0 RESOURCES **4**

4.0 IMPLEMENTATION STEPS **5**

4.1 Scraping & Parsing the Puzzle 5

4.2 Graphical User Interface (GUI) for Displaying the Puzzle 5

4.3 Clue Generation 6

4.3.1 Wordnet 6

4.3.2 Google "Did You Mean" Check 7

4.3.3 Merriam Webster Dictionary 7

4.3.4 Oxford Dictionary 7

4.3.5 Google Dictionary 8

4.3.6 Wikipedia 8

4.3.7 Urban Dictionary 8

Appendix **10**

The source code of the program 10

Package: puzzle 10

scraper.py 10

gui.py 19

extract_answer.py 25

main.py 27

Package: clue_gen 28

wordnet.py 28

wikipediaParser.py 30

urbanParser.py 32

oxford.py 34

merriamParser.py 36

googleDictionary.py 39

didyoumean.py 41

clueGenerator.py 42

clueChanger.py 45

A sample trace from the program: 49

Some test runs: 60

References **73**

1.0 INTRODUCTION

Puzzles are approached in different manners by a variety of people of different ages; some enjoy solving them, while some hate them and never even attempt to get close to them. Nevertheless, a number of puzzle lovers, especially NYTimes puzzle solvers suffer from having only one puzzle for solving each day. After solving the puzzle they have to wait for a whole day to have the next day's puzzle to solve. This is where our Artificial Intelligence (AI) project comes in. In the term project of our AI course, we, as a group named X-WORD, were assigned to release a program for generating brand new clues for the 5x5 puzzle of NYTimes designed by Joel Fagliano [1]. By using this program, the user will be able to obtain different clues for the puzzle of either today or previous days. The geometry and official solutions for the puzzle will be the same, but the clues will be new ones generated from a variety of sources over the Internet. By having access to a variety of clues, we hope that users will be able to enlarge their knowledge by solving the same puzzle with different clues.

2.0 PROBLEM APPROACH

As stated earlier, our goal for this project was to be able to generate different clues from the given puzzle using its current clues. In order to come up with solutions and strategies to solve the task, we started doing research and thinking about possible ways to approach the problem. Primarily, our way of coming up with a solution was to analyze each word of the official solution to the given puzzle. Basically, we needed to generate new clues using the current solutions. In order to determine a strategy, we searched for some words from the official solutions on the web. Our main source were some online dictionaries, especially Wordnet [2]. After some observations, we decided to use 4 properties for a given word; definitions, synonyms, antonyms and example sentences. Definitions are mostly used to explain a word directly but from different perspectives depending on the word, so they can be very useful for having a new clue for the given word. A word can have plenty of synonyms, some of which can replace the word directly, eventually leading to a variety of clues. Antonyms of the word can also be used

depending on the given word. Finally, the usage of the given word in a sentence - example sentences are a choice for generating the new clue by replacing the searched word with an empty pattern. Nevertheless, this property has a high chance of failing to carry out its purpose of being a clue; during the observation stage, we realized that the example sentences that some dictionaries provide do not have the given word in a highly connected manner in terms of the context. For example, looking through the word 'apple', an example sentence 'She bought an apple' would come out, and when the word 'apple' is taken out the sentence 'She bought an ___' can accept plenty of possible answers because it is not a sentence highly related to 'apple' in terms of the context. Therefore, we decided to use the last property as a backup plan for generating the new clue for the puzzles.

3.0 RESOURCES

As mentioned in the previous section, our strategy of the solution was to search through some dictionaries to find the 4 properties of a word. Our first resource is Wordnet [2] by Princeton University. According to our manual test of searching the words from the puzzle, this resource was pretty good in terms of finding most of the properties. The next resources are popular English dictionaries named Oxford [3] and Merriam Webster [4] dictionaries. These dictionaries are also useful in terms of finding the 4 properties. For the sake of having a different resource, we also decided to use Urban [5] dictionary. Nonetheless, the results coming from Urban dictionary have a high chance of being offensive or jargon. Therefore, it is used as a backup resource. Additionally, we used the Google dictionary [6] because of its high chance of defining the given word when the other dictionaries failed to return a result. Finally, we thought that Wikipedia [7] could be a good resource for having better example sentences from the summary of the searched word, especially when the word is a proper noun. Numerically, we have used 6 different resources for generating new clues for the puzzles.

4.0 IMPLEMENTATION STEPS

4.1 Scraping & Parsing the Puzzle

For implementing the project we chose the Python programming language because of the power of its automated modules, which were suitable for such a project. To be able to generate new clues for a puzzle, firstly, we needed to have the puzzle from the website of NYTimes. In order to implement this part of the project, we used the Selenium [8] framework to be able to control the browser operations using the automated scripts. The reason behind the need of browser operations was not being able to reveal the official solution of the puzzle by just sending a request to the URL of the website; the listener for the 'Reveal Puzzle' button in the website was constructed with a Javascript method, therefore, handling the revealing process through the browser was a necessary approach. Scraping the puzzle from the website consists of a few straightforward steps; the browser is opened and request is sent to the URL of NYTimes Crossword [1]; the script waits until the page is fully loaded; the 'Reveal Puzzle' button is clicked; the HTML code of the puzzle is parsed by taking the geometry, clues, and date of the puzzle; the parsed data is written into a JSON file for having a local copy of the puzzle data; the browser is closed.

4.2 Graphical User Interface (GUI) for Displaying the Puzzle

After scraping the data from the website, we needed to display the puzzle together with the original clues to the user. Using Tkinter - a built-in module of Python [9], we designed a simple UI for the program. It consists of a canvas (for displaying the puzzle geometry with solution) and a few labels for displaying the title and the clues (across and down separately). Considering the fact that the main goal of the project is having new clues for the puzzle, we put a purposely blank space for the labels of the generated clues. In addition, having the JSON files of the old puzzle, the user is able to choose the desired puzzle by simply selecting the date of the puzzle from the checkbox in the startup of the program.

4.3 Clue Generation

Obtaining the puzzle data and a GUI for display, we were ready to move to the core part of the project - clue generation. Our puzzle data consists of the following attributes: date of the puzzle, its geometry, across and down clues with their original clues, their answers, and numbers. Having this information in mind, we separated this part of the project from the previously mentioned parts of scraping and GUI. By this way, our plan was to generate this core part of the project separately and use its core method (clueGenerate) in the previous part of the project as a black box to have a high coherence in the organization of project structure.

4.3.1 Wordnet

We started the clue generation part of the project by creating a function for searching the given solution of the puzzle in the dictionary of Wordnet. Although Princeton University provides the build files for using the dictionary database in an offline manner, we determined to use the NLTK (Natural Language Toolkit) [10] module which contains the dictionary of Wordnet. Nowadays NLTK is one of the leading Natural Language Processing (NLP) platforms, which is mainly used for building programs to work with human language. It has a very useful and handy documentation which was beneficial in terms of finding how to handle the necessary parts of the functionality. As mentioned in the problem approach, we needed definitions, synonyms, antonyms and example sentences for the given word. Wordnet module of the NLTK has all 4 attributes for the given word in its database. Basically, the function takes the word as an input and returns the 4 attributes as lists inside a dictionary data type of Python - similar to JSON format. If any property of the given word is not found in the dictionary, it returns that property as an empty list.

4.3.2 Google "Did You Mean" Check

Sometimes Joel Fagliano used some phrases in shortened forms (by removing some characters) in the puzzles to be compatible with the geometry of the puzzle. 'ISNT' (isn't) and 'ILLBE' (I'll be) are examples of such phrases. In such cases, most of the time, dictionaries do not provide an auto correction for such a phrase (especially when requesting from the dictionary API), so it results in having no property found for the given word. To solve this issue, we decided to check the word by simply googling it and check whether there is a "Did You Mean" suggestion for the word. If there is, we assume that the word is in the special form of Joel Fagliano's determination, so we replace the word with its corrected version by Google and continue to search this word over the dictionaries.

4.3.3 Merriam Webster Dictionary

To search over the dictionary through a script, we needed to access Merriam Webster dictionary API . After having the credentials of API key of their website, we were ready to send a request to their API with the searched word attached to the URL of the API. If the request is sent successfully, it returns the available data related to the searched word in JSON format. Parsing through the obtained JSON data, we extract what we need from the whole data and return it in a proper format to 4 properties of the word; a dictionary data type containing definitions, synonyms, antonyms, and example sentences.

4.3.4 Oxford Dictionary

Nearly the same procedure above is applied to get the results from the Oxford Dictionary. Again a request is sent with API credentials and the searched word and the results are obtained in JSON format. However, the different thing is the content and format of JSON data. Therefore, after the proper parsing for the data, the results are returned from the function.

4.3.5 Google Dictionary

Google Dictionary does not provide an API for the public usage, therefore, we needed to use an unofficial API for searching over Google Dictionary. Again, by sending a request to that API, we obtained a JSON data which we parsed to extract the needed result out of it. Finally, the extracted result was returned from the function.

4.3.6 Wikipedia

Since we are using Python programming language, we decided to use some modules which can help us obtain searched data without directly sending the request by writing the code for it. For the purpose of using Wikipedia data, wikipedia module of Python has been used. For having the summary about the given word in the Wikipedia, summary function of the module has been used by passing the word as an argument to the function. Having the paragraph about the given word returned from the function, using the sentence tokenizer of NLTK module, we separated the paragraph to sentences and returned all the sentences as a result of the function.

4.3.7 Urban Dictionary

The same request procedure is applied for obtaining and parsing the obtained JSON data and returning the 4 properties of the result.

After having different functions with the same prototypes and return values, we can search the given word in the corresponding dictionaries or sources using these functions as black boxes. As it is mentioned, our functions are searching for 4 properties for the given word: definitions, synonyms, antonyms and, example sentences. Therefore, we were in need of creating a generic function which takes a word and returns a new clue according to a replacement policy. Our replacement policy is as follows: we are searching properties of the given word in the order of definitions, synonyms, antonyms and, example sentences. Iterating over the resources in a predetermined order, the function returns a result containing the new clue for the given word and its source as soon as it finds a proper one. To have proper results, our program checks

some conditions; while searching over the definitions, if the definition that is found from the dictionary is exactly the same as the original clue for the given word from the puzzle, the program simply ignores it and takes another one if there exists any. Another rule is about having an example sentence for the given word; if the example sentence does not contain the given word, the function ignores it too because, in order to generate a new clue from the example sentence, the function needs to replace the given word in the sentence with '___'. Having passed all the steps and rules of the function, in case no result is found from all of the resources, the function simply returns the original clue. When a list of results is found from a resource for the word, to have variability in generating the clues, the function randomly chooses a result from that list. By this way, each time the function is invoked, it can possibly return a different clue for each call.

Finally, a core function for generating a new clue for each word in the given puzzle is written to be called in the first part of the project. This function basically takes the puzzle data as an input and returns the newly generated clues in the same format of input clue data (across and down clues). After having this function implemented, the only thing necessary was just to call this function from the GUI part of the project.

Word count: 2334

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

Appendix

The source code of the program

Package: puzzle

scraper.py

```
import os
import sys
import math
import json

import bs4
from selenium import webdriver
from selenium.common.exceptions import TimeoutException
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By

from puzzle.extract_answer import extract_answers_from_clues

'''
   Puzzle Scraper
'''
class PuzzleScraper:

   # NYTimes Crossword URL
   URL = 'https://www.nytimes.com/crosswords/game/mini'
```

```
    '''
        Constructor
    '''
    def __init__(self, log_gen):
        # Log generator obj
        self.log_gen = log_gen

    '''
        Fetching the puzzle
    '''
    def fetch_puzzle(self):
        # Sending request to the URL
        req = self.__reveal_answer()

        if req is None:
            return None

        # Scraper
        soup = bs4.BeautifulSoup(req, 'html.parser')

        date = soup.find('div', attrs={'class' :
'PuzzleDetails-date--1HNzj'}).text
        puzzle = soup.find('g', attrs={'data-group' : 'cells'})
        clues = soup.find('section', attrs={'class' :
'Layout-clueLists--10_Xl'})

        # Getting clues and solution
        clues = self.__fetch_clues(clues)
        puzzle = self.__fetch_cells(puzzle)
```

```
        # Data for return
        data = {
            'date' : date,
            'puzzle' : puzzle,
            'clues' : extract_answers_from_clues({'puzzle': puzzle, 'clues':
clues})
        }

        # Writing data in JSON format to the file with date name in 'data'
folder or project dir
        filename = os.getcwd() + '/data/' + str(date).replace(" ", "") +
'.json'

        if '/src' in filename:
            filename = filename.replace("/src", '')

        w_data = json.dumps(data)

        with open(filename, 'w') as outfile:
            outfile.write(w_data)

        # LOG
        self.log_gen.write_to_file('\tPuzzle data is written to the JSON file')
        print('\tPuzzle data is written to the JSON file\n\n')

        return data


    '''
        Opening browser -> Revealing answer -> Returning page source
```

```
    '''
    def __reveal_answer(self):

        timeout = 5
        try:
            # Opening browser
            driver = webdriver.Chrome()

            # LOG
            print("---Getting today's puzzle---\n")
            self.log_gen.write_to_file('\tBrowser is opened and request is sent
to the URL')
            self.log_gen.write_to_file('\tPage is loading')
            print('\tBrowser is opened and request is sent to the URL')
            print('\tPage is loading')

            driver.get(self.URL)

            try:
                # Waiting for modal fully loaded
                modal = EC.presence_of_element_located((By.CLASS_NAME,
"ModalBody-body--3PkKz"))
                WebDriverWait(driver, timeout).until(modal)

                # LOG
                self.log_gen.write_to_file('\tPage is loaded')
                print('\tPage is loaded')

                # CLicking to start btn
                start_btn =
driver.find_element_by_css_selector(".buttons-modalButton--1REsR")
```

```
                start_btn.click()


                # Waiting for toolbar fully loaded

                toolbar = EC.presence_of_element_located((By.CSS_SELECTOR,
".Tool-button--39W4J.Tool-tool--Fiz94.Tool-texty--2w4Br"))

                WebDriverWait(driver, timeout).until(toolbar)


                # 'Reveal'

                reveal_btn =
driver.find_element_by_css_selector(".Toolbar-expandedMenu--2s4M4").find_eleme
nts_by_css_selector(".Tool-button--39W4J.Tool-tool--Fiz94.Tool-texty--2w4Br")[
1]

                reveal_btn.click()


                # 'Puzzle'

                puzzle_btn =
reveal_btn.find_elements_by_css_selector(".HelpMenu-item--1xl0_")[2]

                puzzle_btn.click()


                # LOG

                self.log_gen.write_to_file('\tReveal Puzzle is clicked!')

                print('\tReveal Puzzle is clicked!')


            except TimeoutException:

                print("TIMEOUT")

                sys.exit()


            # CONGRATULATIONS!!!

            driver.find_element_by_css_selector('html').send_keys(u'\ue007')

            driver.find_element_by_css_selector('html').send_keys(u'\ue00c')


            # It will be scraped, so will be returned
```

```
            p_source = driver.page_source


            # Closing browser
            driver.close()


            # LOG
            self.log_gen.write_to_file('\tBrowser is closed and content of the
page is returned')
            print('\tBrowser is closed and content of the page is returned')


        except Exception as e:
            p_source = None
            raise


        return p_source




    '''
        Fetching the clues
    '''
    def __fetch_clues(self, clues):
        # None clues: PROBLEM
        if clues is None:
            print("ERROR: No clues are found\n")
            return None


        # Across : [0] & Down : [1]
        divs = clues.findChildren('div', attrs={'class' :
'ClueList-wrapper--3m-kd'})
```

```
        if len(divs) != 2:
            return None

        # Across
        across = divs[0].find('ol').find_all('li')
        across_data = []
        for li in across:
            across_data.append({
                'number' : li.findChildren()[0].text,
                'clue' : li.findChildren()[1].text
            })

        # LOG
        self.log_gen.write_to_file('\tAcross clues are parsed from the page
content')
        print('\tAcross clues are parsed from the page content')

        # Down
        down = divs[1].find('ol').find_all('li')
        down_data = []
        for li in down:
            down_data.append({
                'number' : li.findChildren()[0].text,
                'clue' : li.findChildren()[1].text
            })

        # LOG
        self.log_gen.write_to_file('\tDown clues are parsed from the page
content')
        print('\tDown clues are parsed from the page content')
```

```
        # Returning whole data
        return {
            'across' : across_data ,
            'down' : down_data
        }



    '''
        Fetching the puzzle cells
    '''
    def __fetch_cells(self, puzzle):
        # None puzzle: PROBLEM
        if puzzle is None:
            print("ERROR: No puzzle is found\n")
            return None

        cells = puzzle.find_all('g')

        # If no cells: smth wrong
        if len(cells) == 0:
            return None

        # 2D array for keeping puzzle geometry
        dimension = int(math.sqrt(len(cells)))        # in our case it is 5
        puzzle_data = [[None for x in range(dimension)] for y in
range(dimension)]

        i = 0
        j = 0
```

```
        for cell in cells:
            if i == dimension:
                break

            if j == dimension:
                i += 1
                j = 0

            # Putting puzzle data into the 2D array
            puzzle_data[i][j] = self.__process_cell_data(cell)
            j += 1

        # LOG
        self.log_gen.write_to_file('\tPuzzle cells are parsed from the page content')
        print('\tPuzzle cells are parsed from the page content')

        return puzzle_data



    '''
        Processing the cell data
    '''
    def __process_cell_data(self, cell):

        components = cell.findChildren('text')
        size = len(components)

        '''
            3 Cases:
```

```
            components with size 2:
                cell has both 'number' and 'letter'
            components with size 1:
                cell has only a 'letter'
            components with size 0:
                cell is black (empty)
    '''

    if size == 2:
        number = components[0].text
        letter = components[1].text
        return [number, letter]
    elif size == 1:
        letter = components[0].text
        return [letter]
    elif size == 0:
        return None
```

---

gui.py

```
import tkinter as tk

'''
   GUI for displaying Puzzle
'''

class GUI:
```

```
    '''
        Constructor
    '''
    def __init__(self, log_gen):
        # Log generator obj
        self.log_gen = log_gen

        self.__root = None

    '''
        Renders the title
    '''
    def __render_title(self, date):

        # Panel
        pane = tk.PanedWindow()
        pane.pack(fill = tk.X, expand = 1, side = tk.TOP, pady = (20, 20))

        font1 = 'Inherit 20 bold'
        font2 = 'Inherit 16 italic'

        tk.Label(pane, text = "X-WORD NYTime Crossword Clue Generator",
font=font1).pack()
        tk.Label(pane, text = date, font = font2).pack()

        # LOG
        self.log_gen.write_to_file('Title is rendered')
        print('\tTitle is rendered')
```

```
'''
    Renders the puzzle grid
'''
def __render_grid(self, puzzle):

    # Panel
    canvas = tk.Canvas(self.__root, width = 505, height = 505)
    canvas.pack(fill = tk.BOTH, side = tk.LEFT)

    # Gap from left edge of screen
    GAP = 5
    tmp = (int(canvas['width']) - GAP * 2)

    # Dimension of the puzzle
    dimension = len(puzzle)

    letter_font = ('Arial', str(tmp // 11))
    number_font = ('Arial', str(tmp // 20))

    # Width of the cell
    w = tmp / dimension

    for i in range(dimension):
        for j in range(dimension):

            number = ''
            letter = ''

            if puzzle[i][j] is None:
```

```
                bg = 'black'
            else:
                bg = 'white'
                letter = '\n' + str(puzzle[i][j][len(puzzle[i][j]) - 1])

                if len(puzzle[i][j]) == 2:
                    number = str(puzzle[i][j][0])

            # Grid components
            canvas.create_rectangle(j * w + GAP, i * w + GAP, j * w + w + GAP, i * w + w + GAP, fill = bg, outline = 'gray', width = 1.5)
            canvas.create_text(j * w + w / 2 + GAP, i * w + w / 3 + GAP, text = letter, width = 100, font = letter_font, fill = "#2860d8")
            canvas.create_text(j * w + 15 + GAP, i * w + 25 + GAP, text = number, width = 100, font = number_font)

        # LOG
        self.log_gen.write_to_file('Grids are rendered')
        print('\tGrids are rendered')


    '''
        Renders the clues
    '''
    def __render_clues(self, clues, title):

        # Fonts
        font = 'Inherit 14 normal'
        font_b = 'Inherit 14 bold'

        # Wrapper panel
```

```
        pane = tk.PanedWindow()

        pane.pack(fill = tk.X, side = tk.TOP)


        # Title

        tk.Label(pane, text = title).pack(fill = tk.BOTH)


        # Across Pane & its title

        across_pane = tk.PanedWindow(pane)

        across_pane.pack(fill = tk.X, side = tk.LEFT, padx = (20, 20), pady =
(20, 20))

        tk.Label(across_pane, text = 'ACROSS', font = font_b,
anchor='w').pack(fill = tk.BOTH)


        clues_str = ""

        for clue in clues['across']:

            clues_str += clue['number'] + '. ' + clue['clue'] + "\n\n"

        tk.Label(across_pane, text = clues_str, font = font, wraplength = 450,
justify=tk.LEFT).pack(fill = tk.BOTH)


        # Down Pane & its title

        down_pane = tk.PanedWindow(pane)

        down_pane.pack(fill = tk.X, side = tk.RIGHT, padx = (20, 20), pady =
(20, 20))

        tk.Label(down_pane, text = 'DOWN', font = font_b, anchor='w').pack(fill
= tk.BOTH)


        clues_str = ""

        for clue in clues['down']:

            clues_str += clue['number'] + '. ' + clue['clue'] + "\n\n"

        tk.Label(down_pane, text = clues_str, font = font, wraplength = 450,
justify=tk.LEFT).pack(fill = tk.BOTH)
```

```
        # LOG
        self.log_gen.write_to_file('Clues are rendered')
        print('\tClues are rendered')



    '''
        Renders all elements
    '''
    def render(self, data):

        # ROOT
        self.__root = tk.Tk()
        self.__root.title('NY Times Crossword')
        # self.__root.geometry('1000x1000')

        # LOG
        print("---Showing the puzzle----\n")
        self.log_gen.write_to_file('Window is constructed')
        print('\tWindow is constructed')

        if data is None or data['date'] is None or data['puzzle'] is None or
data['clues'] is None:
            print("ERROR: Data is not in proper format!")
            return

        self.__render_title(data['date'])
        self.__render_grid(data['puzzle'])
        self.__render_clues(data['clues'], 'Original Clues')
        self.__render_clues(data['generated_clues'], 'Generated Clues')
```

```
        print("---------------------------")

        self.__root.resizable(False, False)
        self.__root.mainloop()
```

---

extract_answer.py

```
import numpy as np

'''
   Extracts the answer from the puzzle according to the given number
   for across or down clue

   @param: puzzle - given puzzle
   @param: num - number for clue
   @type: type of clue: 0 for across | 1 for down
'''
def find_answer_with_clue_num(puzzle, num, type = 0):
   grid = np.array(puzzle)

   a, b = -1, -1
   for i in range(grid.shape[1]):
       for j in range(grid.shape[0]):
           if grid[i][j] is not None and str(num) in grid[i][j]:
               a, b = i, j
               break
```

```
    word = None
    if (a, b) != (-1, -1):
        if type == 0:
            word = grid[a, :]
        else:
            word = grid[:, b]

    answer = ""
    if word is not None:
        for el in word:
            answer += el[-1] if el is not None else ""

    return answer

'''
    Extract all answers from clues according
    to the given crossword data

    @param: data - given crossword data
'''
def extract_answers_from_clues(data):

    puzzle, clues = data['puzzle'], data['clues']

    across = []
    for clue in clues['across']:
        answer = find_answer_with_clue_num(puzzle, clue['number'], 0)
        across.append({'clue': clue['clue'], 'answer': answer, 'number':
clue['number']})
```

```
    down = []
    for clue in clues['down']:
        answer = find_answer_with_clue_num(puzzle, clue['number'], 1)
        down.append({'clue': clue['clue'], 'answer': answer, 'number':
clue['number']})

    return {"across": across, "down": down}
```

---

main.py

```
import sys
import json

from puzzle.scraper import PuzzleScraper
from puzzle.gui import GUI
from util.log_generator import LogGenerator

from clue_gen.clueGenerator import changePuzzle

# Log generator
l_gen = LogGenerator()

# Loading data
data = None
if len(sys.argv) == 2:
    try:
        filename = str(sys.argv[1])
```

```
        with open(filename) as f:
            data = json.load(f)
    except Exception as e:
        print('ERROR: ', e)
        sys.exit()
elif len(sys.argv) == 1:
    p_scrp = PuzzleScraper(l_gen)
    data = p_scrp.fetch_puzzle()
else:
    sys.exit()

# NEW CLUES
print("Generating clues for puzzle of ", data['date'])
data['generated_clues'] = changePuzzle(data['clues'], trace=True)

print("\nClue generation for puzzle of " + data['date'] + " has been
finished!")
print('---------------------------------\n')

# GUI with data
gui = GUI(l_gen)
gui.render(data)
```

Package: clue_gen

wordnet.py

```python
from nltk.corpus import wordnet

'''
    For a given word function returns its (if exists):
        * synonym set
        * antonyms
        * definition
        * example sentences

    :param str word - the given word
    :return dict result - result object
'''
def search_wordnet(word, trace = False):
    # Trace
    if trace is True:
        print("\t\tChecking Wordnet for the word: \"" + word + "\"")

    # Synonym set of the word)
    synset = wordnet.synsets(str(word))

    synonyms, antonyms = [], []
    definitions, examples = [], []

    for syn in synset:
        for l in syn.lemmas():
            synonyms.append(l.name())
            if l.antonyms():
                antonyms.append(l.antonyms()[0].name())
```

```
        definitions.append(syn.definition())
        examples += syn.examples()

    # Removing the duplicates and the word itself
    synonyms = list(set(synonyms))
    if len(synonyms) != 0 and word.lower() in synonyms:
        synonyms.remove(word.lower())

    # Trace
    if trace is True and len(synonyms) == 0 and len(antonyms) == 0 and
len(definitions) == 0 and len(examples) == 0:
        print("\t\tNo results from Wordnet for the word: \"" + word + "\"\n")

    return {
        "word": word,
        "syn": synonyms,
        "ant": antonyms,
        "def": definitions,
        "examples": examples
    }
```

---

wikipediaParser.py

```
import warnings
import wikipedia
from nltk.tokenize import sent_tokenize

'''
    Searching the given word in the Wikipedia
```

```
    :param str word -  the given word
'''
def search_wikipedia(word, trace = False):
    # To ignore module warnings
    warnings.filterwarnings('ignore')

    # Trace
    if trace is True:
        print("\t\tChecking Wikipedia for the word: \"" + word + "\"")

    # Example sentences
    sentences = []

    try:
        # Summary of the given word in Wikipedia
        summary_sentences = sent_tokenize(wikipedia.summary(word))

        # Looking for the sentences containing the given word
        for sentence in summary_sentences:
            if word.lower() in sentence.lower():
                sentences.append(sentence)

    except Exception as e:
        pass

    # Trace
    if trace is True and len(sentences) == 0:
        print("\t\tNo results from Wikipedia for the word: \"" + word +
"\"\n")
```

```
    return {
        "word": word,
        "syn": [],
        "ant": [],
        "def": [],
        "examples": sentences
    }
```

---

urbanParser.py

```
import urllib.request
import json

URL = "http://api.urbandictionary.com/v0/define?term="

'''
   Parsing the given word from Urban Dictionary API

   :param word - the given word
'''
def getWordFromUrban(word, trace = False):
   # Trace
   if trace is True:
       print("\t\tChecking Urban dictionary for the word: \"" + word + "\"")

   url = URL + word
```

```
    defsMixed = []
    exMixed = []

    try:

        request = urllib.request.Request(url)
        response = urllib.request.urlopen(request)
        content = response.read().decode('utf-8');
        contentJson = json.loads(content)



        list = contentJson["list"]

        for i in list:
            if(i["word"] == word):
                definition = i["definition"]
                defsMixed.append(definition)
                example = i["example"]
                exMixed.append(example)

    except Exception as e:
        pass

    # Trace
    if trace is True and len(defsMixed) == 0 and len(exMixed) == 0:
        print("\t\tNo results from Urban dictionary for the word: \"" + word +
"\"\n")

    word = {
```

```
        "word": word,
        "syn": [],
        "ant": [],
        "def": defsMixed,
        "examples": exMixed
    }
    return word
```

---

oxford.py

```
import requests
import json


APP_ID = 'dfe4face'
APP_KEY = 'c2df8098d116ba55a543ac4fc5f194a0'

LANGUAGE = 'en'
URL = 'https://od-api.oxforddictionaries.com:443/api/v2/entries/' + LANGUAGE +
'/'


def getWordFromOxford(word, trace = False):
    # Trace
    if trace is True:
        print("\t\tChecking Oxford dictionary for the word: \"" + word + "\"")

    url = URL + word.lower()
```

```
    synsMixed = []
    antsMixed = []
    exMixed = []
    defsMixed = []

    try:
        r = requests.get(url, headers = {'app_id' : APP_ID, 'app_key' :
APP_KEY})
        contentJson = json.loads(r.text)

        for result in contentJson['results']:
            lexical_entries = result['lexicalEntries']

            for lex_entr in lexical_entries:
                entries = lex_entr['entries']

                for ent in entries:
                    senses = ent['senses']

                    for sense in senses:
                        defsMixed += sense['definitions']
                        defsMixed += sense['shortDefinitions']
                        defsMixed += sense['examples']

    except Exception as e:
        synsMixed = []
        antsMixed = []
        exMixed = []
        defsMixed = []
```

```
    # Trace
    if trace is True and len(defsMixed) == 0 and len(exMixed) == 0:
        print("\t\tNo results from Oxford dictionary for the word: \"" + word +
"\"\n")

    # Returning word
    word = {
        "word": word,
        "syn": synsMixed,
        "ant": antsMixed,
        "def": defsMixed,
        "examples": exMixed
    }

    return word
```

---

merriamParser.py

```
import urllib.request
import json

merriam_app_key = "5a8472e0-77c3-4e2f-867d-b1fa6f4965b0"

'''
    Parsing the given word from Merriam Dictionary API

    :param word - the given word
'''
```

```
def getWordFromMerriam(word, trace = False):
    # Trace
    if trace is True:
        print("\t\tChecking Merriam dictionary for the word: \"" + word + "\"")


    url =
"https://dictionaryapi.com/api/v3/references/thesaurus/json/"+word+"?key="+mer
riam_app_key
    synsMixed = []
    antsMixed = []
    exMixed = []
    defsMixed = []
    try:
        request = urllib.request.Request(url)
        response = urllib.request.urlopen(request)
        content = response.read().decode('utf-8')
        contentJson = json.loads(content)
        for i in contentJson:


                meta = i["meta"]
                if(meta["id"] == word):
                    # Extracting syns and ants
                    syns = meta["syns"]
                    for j in syns:
                            synsMixed.extend(j)
                    ants = meta["ants"]
                    for j in ants:
                            antsMixed.extend(j)
                    # Extracting example sentences
```

```
                    definition = i["def"]
                    sseq = definition[0]["sseq"]
                    dt=sseq[0][0][1]["dt"]
                    example = dt[1][1][0]["t"]
                    example = example.replace('{it}', '').replace('{/it}', '')
                    exMixed.append(example)
                # Extracting definitions
                shortDef = i["shortdef"]
                defsMixed.extend(shortDef)
    except Exception as e:
        synsMixed = []
        antsMixed = []
        exMixed = []
        defsMixed = []

    # Trace
    if trace is True and len(synsMixed) == 0  and len(antsMixed) == 0 and
len(defsMixed) == 0 and len(exMixed) == 0:
        print("\t\tNo results from Merriam dictionary for the word: \"" + word
+ "\"\n")

    # Returning word
    word = {
        "word": word,
        "syn": synsMixed,
        "ant": antsMixed,
        "def": defsMixed,
        "examples": exMixed
    }
```

```
    return word
```

---

googleDictionary.py

```
import urllib.request
import json
import unicodedata

URL = 'https://googledictionaryapi.eu-gb.mybluemix.net/'

'''
    Searching the word in Google Dictionary API (unofficial)

    :param word - the given word
'''


def getWordFromGoogleDictionary(word, trace = False):
    # Trace
    if trace is True:
        print("\t\tChecking Google dictionary for the word: \"" + word + "\"")

    # Replacing non-eng chars
    word = unicodedata.normalize('NFKD', word).encode('ascii',
'ignore').decode('utf8')

    url = URL + '?define=' + word + '&lang=en'

    meanings = []
```

```
    try:
        request = urllib.request.Request(url)
        response = urllib.request.urlopen(request)
        content = response.read().decode('utf-8')
        contentJson = json.loads(content)

        for res in contentJson:
            meaning = res['meaning']

            for key in meaning.keys():
                for definition in meaning[key]:
                    meanings.append(definition['definition'])

    except Exception as e:
        pass

    # Trace
    if trace is True and len(meanings) == 0:
        print("\t\tNo results from Google dictionary for the word: \"" + word
+ "\"\n")

    # Returning word
    word = {
        "word": word,
        "syn": [],
        "ant": [],
        "def": meanings,
        "examples": []
    }
```

```
    return word
```

---

didyoumean.py

```
from bs4 import BeautifulSoup
from googlesearch import get_page


'''
   Parsing for 'Did You Mean'

   :param str query - the given query
   :return str or None result - the corrected (meant) version of query
'''


def did_you_mean(query, trace = False):
   try:
       # Trace
       if trace is True:
           print("\t\t\"Did You Mean (from Google) check\" for the query: \""
+ query + "\"")

       # query = str(query).strip()
       html = get_page('http://www.google.com/search?q=' + query)
       soup = BeautifulSoup(html, 'html.parser')

       answer = soup.find('a', attrs={'class': 'gL9Hy'})
```

```
        result = answer.find('i') if answer is not None else None
        result = result.text if result is not None else None

        # Trace
        if trace is True and result is None:
            print("\t\tNo correction from \"Did You Mean check\" for the query: \"" + query + "\"\n")
        else:
            print("\t\t\"" + query + "\" is replaced with \"" + result +"\"\n")

        return result
    except Exception as e:
        pass

    # Trace
    if trace is True:
        print("\t\tPassed \"Did You Mean check\" for the query: \"" + query + "\"\n")

    return None
```

---

clueGenerator.py

```
from clue_gen.clueChanger import changeClue

'''
    Changing original clue of the whole Puzzle
```

```
    :param clues - the original clues of the puzzle
'''
def changePuzzle(clues, trace = False):
    across_clues = clues['across']
    down_clues = clues['down']

    print("\n----Generating across clues----\n")
    changed_across_clues = []
    for ac_clue in across_clues:
        clue = ac_clue['clue']
        word = ac_clue['answer']
        num = ac_clue['number']

        print("\n----------------------------------")
        print("*** Searching clue for \"" + str(word) + "\" ***\n")
        found = False
        for i in range(4):
            if i == 0: print("\t---Looking for definitions---\n")
            elif i == 1: print("\t---Looking for synonyms---\n")
            elif i == 2: print("\t---Looking for antonyms---\n")
            elif i == 3: print("\t---Looking for examples---\n")

            new_clue = changeClue(word, clue, i, trace)
            if new_clue['new_clue'] is not None:
                print("\n*** New clue for \"" + str(word) + "\" is found from
" + str(new_clue['source']) + " ***")
                changed_across_clues.append({'clue': new_clue['new_clue'],
'answer': word, 'number': num})
                print("----------------------------------\n")
                found = True
```

```
                break

        # No new clue cannot be found
        if found is False:
            print("\n*** No new clue can be found for \"" + str(word) + "\"
***")
            print("----------------------------------\n")
            changed_across_clues.append(ac_clue)

    print("\n----Generating down clues----\n")
    changed_down_clues = []
    for down_clue in down_clues:
        clue = down_clue['clue']
        word = down_clue['answer']
        num = down_clue['number']

        print("\n----------------------------------")
        print("*** Searching clue for \"" + str(word) + "\" ***\n")
        found = False
        for i in range(4):
            if i == 0: print("\t---Looking for definitions---\n")
            elif i == 1: print("\t---Looking for synonyms---\n")
            elif i == 2: print("\t---Looking for antonyms---\n")
            elif i == 3: print("\t---Looking for examples---\n")

            new_clue = changeClue(word, clue, i, trace)
            if new_clue['new_clue'] is not None:
                print("\n*** New clue for \"" + str(word) + "\" is found from
" + str(new_clue['source']) + " ***")
                print("----------------------------------\n")
```

```
                changed_down_clues.append({'clue': new_clue['new_clue'],
'answer': word, 'number': num})
                found = True
                break

        # No new clue cannot be found
        if found is False:
            print("\n*** No new clue can be found for \"" + str(word) + "\"
***")
            print("----------------------------------\n")
            changed_down_clues.append(down_clue)

    # Returning
    return {
            'across': changed_across_clues,
            'down': changed_down_clues
            }
```

---

clueChanger.py

```
from random import shuffle

from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

# Resources
from clue_gen.wordnet import search_wordnet
from clue_gen.merriamParser import getWordFromMerriam
from clue_gen.oxford import getWordFromOxford
```

```python
from clue_gen.urbanParser import getWordFromUrban
from clue_gen.didyoumean import did_you_mean
from clue_gen.wikipediaParser import search_wikipedia
from clue_gen.googleDictionary import getWordFromGoogleDictionary

# All resources to be searched
RESOURCES = [{'name': 'Wikipedia', 'func': search_wikipedia},
             {'name': 'Wordnet', 'func': search_wordnet},
             {'name': 'Merriam', 'func': getWordFromMerriam},
             {'name': 'Oxford', 'func': getWordFromOxford},
             {'name': 'Urban', 'func': getWordFromUrban},
             {'name': 'Google Dict', 'func': getWordFromGoogleDictionary}]

# For finding the stem (root) of the word
ps = PorterStemmer()

'''
    Changing the given word with its original clue
    according to the given replace policy:

    :param str word - the given word
    :param str clue - the original clue for the given word
    :param int replace_policy:
                0 -> definition
                1 -> synonym
                2 -> antonym
                3 -> example sentence
    :return the given word + the new clue + source of new clue
'''
```

```python
def changeClue(word, clue, replace_policy, trace = False):

    # Checking the word in Google's 'Did You Mean'
    did_you_mean_result = did_you_mean(word, trace)
    if did_you_mean_result is not None:
        word = did_you_mean_result

    # Getting the stem of the word
    # word = ps.stem(word)

    # Replace policy
    if replace_policy == 0:
        replaced = 'def'
    elif replace_policy == 1:
        replaced = 'syn'
    elif replace_policy == 2:
        replaced = 'ant'
    elif replace_policy == 3:
        replaced = 'examples'

    # New Clue
    new_clue = None
    is_found = False
    source = None

    # Checking the resources
    for resource in RESOURCES:
        results = resource['func'](word, trace)[replaced]
```

```
        if results is not None and len(results) != 0:
            # If the original clue returned as a new clue
            if clue in results:
                results.remove(clue)

            # Shuffling for randomness
            shuffle(results)

            for result in results:
                # If definition contains the word itself
                if replaced == 'def' and (word.lower() in
word_tokenize(result.lower())):
                    continue
                # If policy wants example sentence
                elif replaced == 'examples':
                    if word.lower() not in result.lower():
                        continue
                    else:
                        new_clue = result.lower().replace(word.lower(),
'___').strip()
                        is_found = True
                        break
                else:
                    new_clue = result.strip()
                    is_found = True
                    break

        # Found
        if is_found is True:
            source = resource['name']
```

```
            break

    # Returning
    return {
        'word': word,
        'new_clue': new_clue,
        'source': source
    }
```

A sample trace from the program:

--Available resources for clue generation:

-Wordnet

-Merriam Dictionary

-Oxford Dictionary

-Urban Dictionary

-Google Dictionary

-Wikipedia

Old puzzles are loading...

Puzzle: April28,2019 has been loaded.

Puzzle: February28,2019 has been loaded.

Puzzle: April7,2019 has been loaded.

Puzzle: April11,2019 has been loaded.

Puzzle: February7,2019 has been loaded.

Puzzle: March8,2019 has been loaded.

Puzzle: February17,2019 has been loaded.

Puzzle: March17,2019 has been loaded.

Puzzle: March13,2019 has been loaded.

Puzzle: April29,2019 has been loaded.

Puzzle: March20,2019 has been loaded.

All old puzzles have been loaded

---Getting today's puzzle---

Browser is opened and request is sent to the URL

Page is loading

Page is loaded

Reveal Puzzle is clicked!

Browser is closed and content of the page is returned

Across clues are parsed from the page content

Down clues are parsed from the page content

Puzzle cells are parsed from the page content

Puzzle data is written to the JSON file

Generating clues for puzzle of Tuesday May 7, 2019

----Generating across clues----

*** Searching clue for "RAM" ***

---Looking for definitions---

"Did You Mean (from Google) check" for the query: "RAM"

No correction from "Did You Mean check" for the query: "RAM"

Checking Wikipedia for the word: "RAM"

No results from Wikipedia for the word: "RAM"

Checking Wordnet for the word: "RAM"

*** New clue for "RAM" is found from Wordnet ***

*** Searching clue for "MOTOR" ***

---Looking for definitions---

"Did You Mean (from Google) check" for the query: "MOTOR"

No correction from "Did You Mean check" for the query: "MOTOR"

Checking Wikipedia for the word: "MOTOR"

Checking Wordnet for the word: "MOTOR"

*** New clue for "MOTOR" is found from Wordnet ***

*** Searching clue for "ISSUE" ***

---Looking for definitions---

"Did You Mean (from Google) check" for the query: "ISSUE"

No correction from "Did You Mean check" for the query: "ISSUE"

Checking Wikipedia for the word: "ISSUE"

No results from Wikipedia for the word: "ISSUE"

Checking Wordnet for the word: "ISSUE"

*** New clue for "ISSUE" is found from Wordnet ***

*** Searching clue for "DIETS" ***

---Looking for definitions---

"Did You Mean (from Google) check" for the query: "DIETS"

No correction from "Did You Mean check" for the query: "DIETS"

Checking Wikipedia for the word: "DIETS"

Checking Wordnet for the word: "DIETS"

*** New clue for "DIETS" is found from Wordnet ***

*** Searching clue for "NAH" ***

---Looking for definitions---

"Did You Mean (from Google) check" for the query: "NAH"

No correction from "Did You Mean check" for the query: "NAH"

Checking Wikipedia for the word: "NAH"

No results from Wikipedia for the word: "NAH"

Checking Wordnet for the word: "NAH"

No results from Wordnet for the word: "NAH"

Checking Merriam dictionary for the word: "NAH"

No results from Merriam dictionary for the word: "NAH"

Checking Oxford dictionary for the word: "NAH"

No results from Oxford dictionary for the word: "NAH"

Checking Urban dictionary for the word: "NAH"

No results from Urban dictionary for the word: "NAH"

Checking Google dictionary for the word: "NAH"

*** New clue for "NAH" is found from Google Dict ***

----Generating down clues----

*** Searching clue for "ROSIN" ***

---Looking for definitions---

"Did You Mean (from Google) check" for the query: "ROSIN"

No correction from "Did You Mean check" for the query: "ROSIN"

Checking Wikipedia for the word: "ROSIN"

Checking Wordnet for the word: "ROSIN"

*** New clue for "ROSIN" is found from Wordnet ***

*** Searching clue for "ATSEA" ***

---Looking for definitions---

"Did You Mean (from Google) check" for the query: "ATSEA"

No correction from "Did You Mean check" for the query: "ATSEA"

Checking Wikipedia for the word: "ATSEA"

No results from Wikipedia for the word: "ATSEA"

Checking Wordnet for the word: "ATSEA"

No results from Wordnet for the word: "ATSEA"

Checking Merriam dictionary for the word: "ATSEA"

No results from Merriam dictionary for the word: "ATSEA"

Checking Oxford dictionary for the word: "ATSEA"

No results from Oxford dictionary for the word: "ATSEA"

Checking Urban dictionary for the word: "ATSEA"

No results from Urban dictionary for the word: "ATSEA"

Checking Google dictionary for the word: "ATSEA"

No results from Google dictionary for the word: "ATSEA"

---Looking for synonyms---

"Did You Mean (from Google) check" for the query: "ATSEA"

No correction from "Did You Mean check" for the query: "ATSEA"

Checking Wikipedia for the word: "ATSEA"

No results from Wikipedia for the word: "ATSEA"

Checking Wordnet for the word: "ATSEA"

No results from Wordnet for the word: "ATSEA"

Checking Merriam dictionary for the word: "ATSEA"

No results from Merriam dictionary for the word: "ATSEA"

Checking Oxford dictionary for the word: "ATSEA"

No results from Oxford dictionary for the word: "ATSEA"

Checking Urban dictionary for the word: "ATSEA"

No results from Urban dictionary for the word: "ATSEA"

Checking Google dictionary for the word: "ATSEA"

No results from Google dictionary for the word: "ATSEA"

---Looking for antonyms---

"Did You Mean (from Google) check" for the query: "ATSEA"

No correction from "Did You Mean check" for the query: "ATSEA"

Checking Wikipedia for the word: "ATSEA"

No results from Wikipedia for the word: "ATSEA"

Checking Wordnet for the word: "ATSEA"

No results from Wordnet for the word: "ATSEA"

Checking Merriam dictionary for the word: "ATSEA"

No results from Merriam dictionary for the word: "ATSEA"

Checking Oxford dictionary for the word: "ATSEA"

No results from Oxford dictionary for the word: "ATSEA"

Checking Urban dictionary for the word: "ATSEA"

No results from Urban dictionary for the word: "ATSEA"

Checking Google dictionary for the word: "ATSEA"

No results from Google dictionary for the word: "ATSEA"

---Looking for examples---

"Did You Mean (from Google) check" for the query: "ATSEA"

No correction from "Did You Mean check" for the query: "ATSEA"

Checking Wikipedia for the word: "ATSEA"

No results from Wikipedia for the word: "ATSEA"

Checking Wordnet for the word: "ATSEA"

No results from Wordnet for the word: "ATSEA"

Checking Merriam dictionary for the word: "ATSEA"

No results from Merriam dictionary for the word: "ATSEA"

Checking Oxford dictionary for the word: "ATSEA"

No results from Oxford dictionary for the word: "ATSEA"

Checking Urban dictionary for the word: "ATSEA"

No results from Urban dictionary for the word: "ATSEA"

Checking Google dictionary for the word: "ATSEA"

No results from Google dictionary for the word: "ATSEA"

*** No new clue can be found for "ATSEA" ***

*** Searching clue for "MOUTH" ***

---Looking for definitions---

"Did You Mean (from Google) check" for the query: "MOUTH"

No correction from "Did You Mean check" for the query: "MOUTH"

Checking Wikipedia for the word: "MOUTH"

Checking Wordnet for the word: "MOUTH"

*** New clue for "MOUTH" is found from Wordnet ***

*** Searching clue for "MID" ***

---Looking for definitions---

"Did You Mean (from Google) check" for the query: "MID"

No correction from "Did You Mean check" for the query: "MID"

Checking Wikipedia for the word: "MID"

No results from Wikipedia for the word: "MID"

Checking Wordnet for the word: "MID"

*** New clue for "MID" is found from Wordnet ***

*** Searching clue for "RES" ***

---Looking for definitions---

"Did You Mean (from Google) check" for the query: "RES"

No correction from "Did You Mean check" for the query: "RES"

Checking Wikipedia for the word: "RES"

No results from Wikipedia for the word: "RES"

Checking Wordnet for the word: "RES"

*** New clue for "RES" is found from Wordnet ***

Clue generation for puzzle of Tuesday May 7, 2019 has been finished!

---Showing the puzzle----

Window is constructed

Title is rendered

Grids are rendered

Clues are rendered

Clues are rendered

Some test runs:

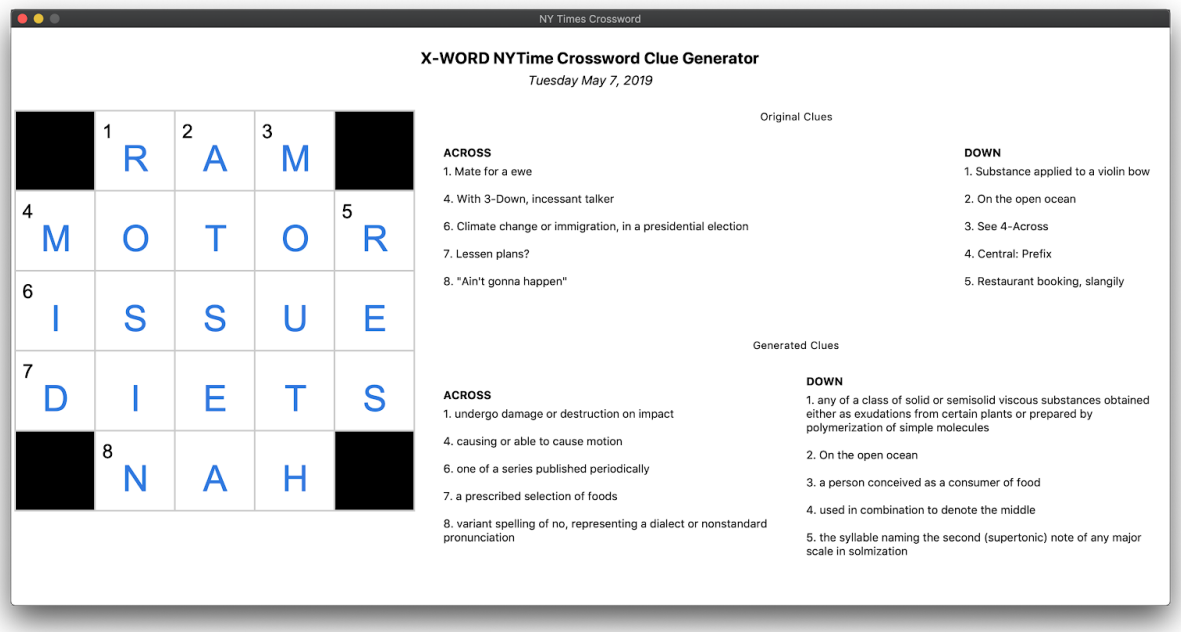

Figure 1. Test run for May 7, 2019.

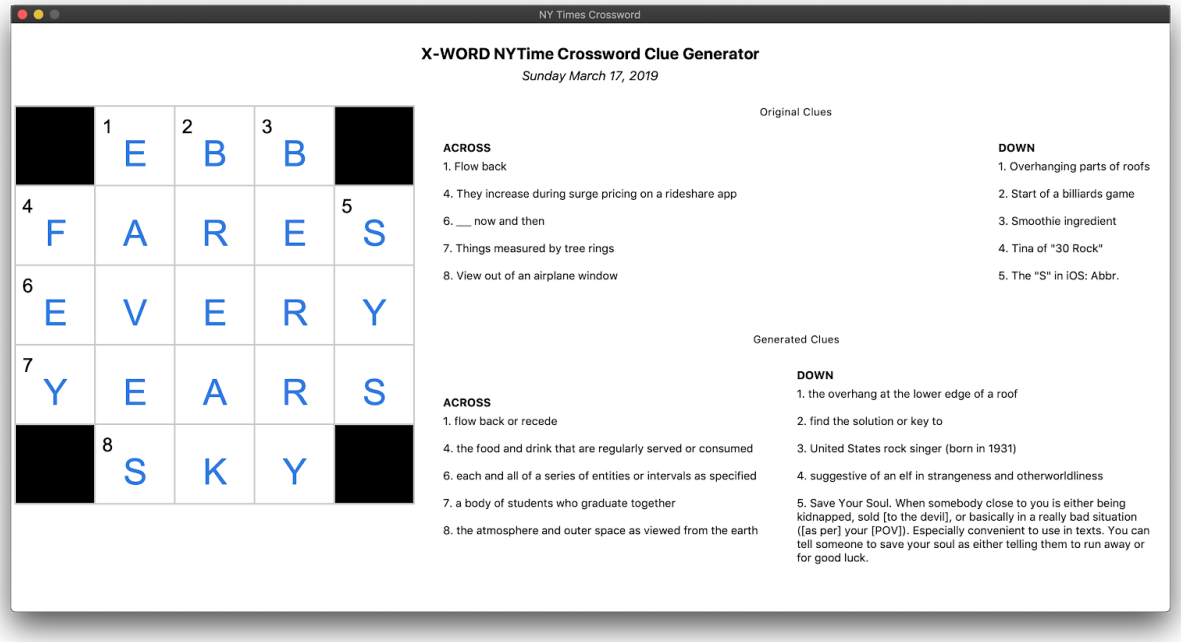

Figure 2. Test run for March 17, 2019.

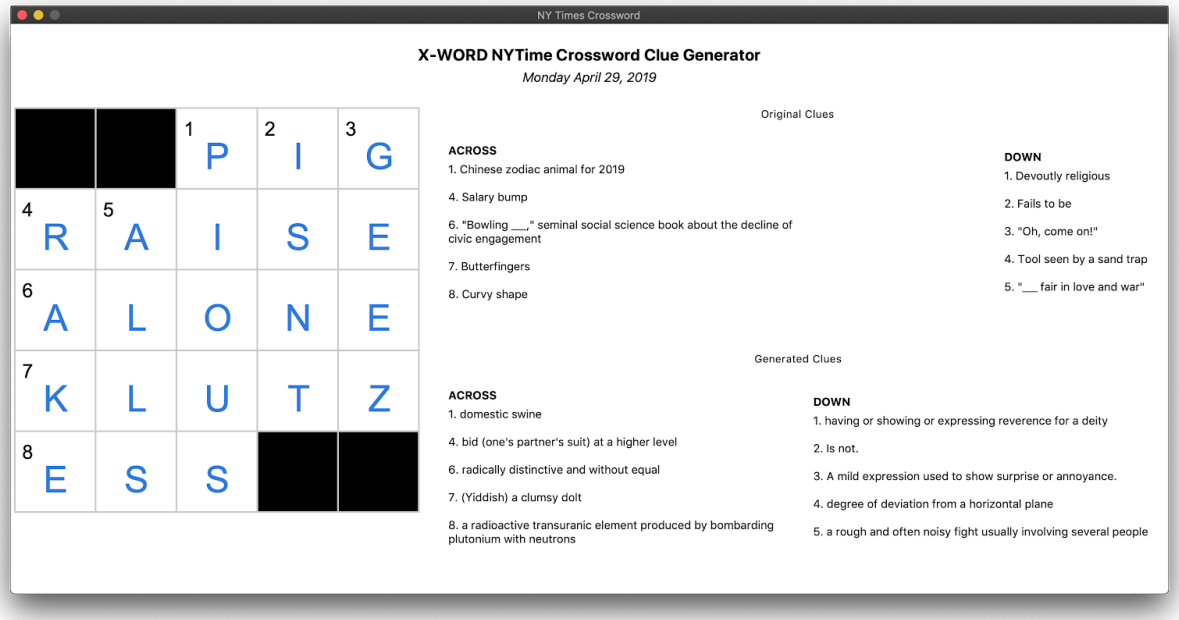

Figure 3. Test run for April 29, 2019.

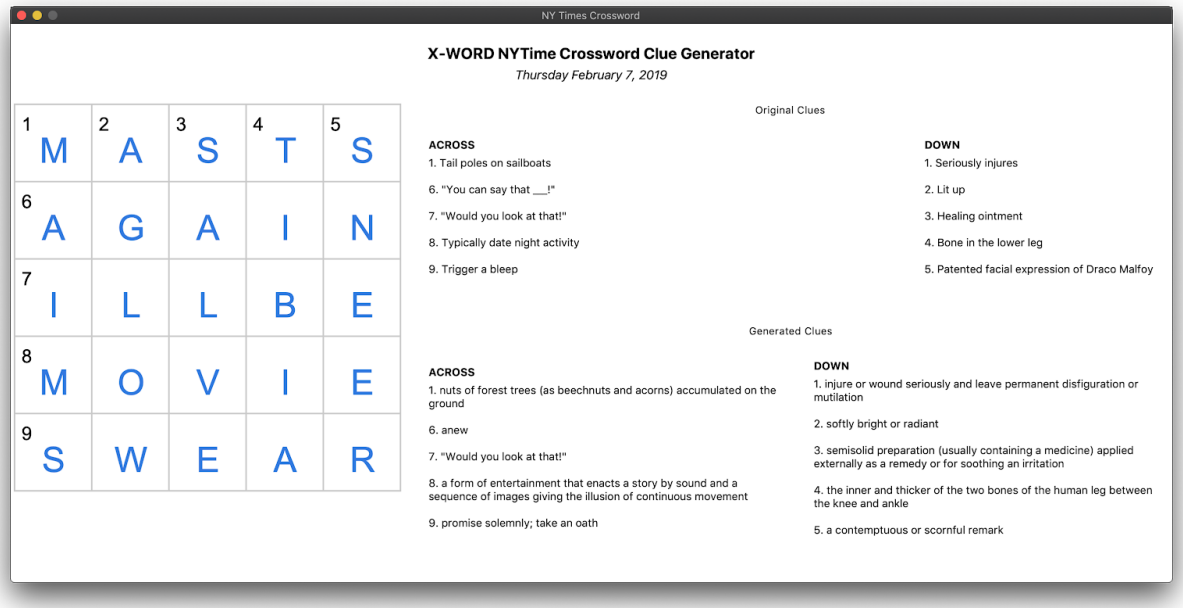

Figure 4. Test run for February 7, 2019.

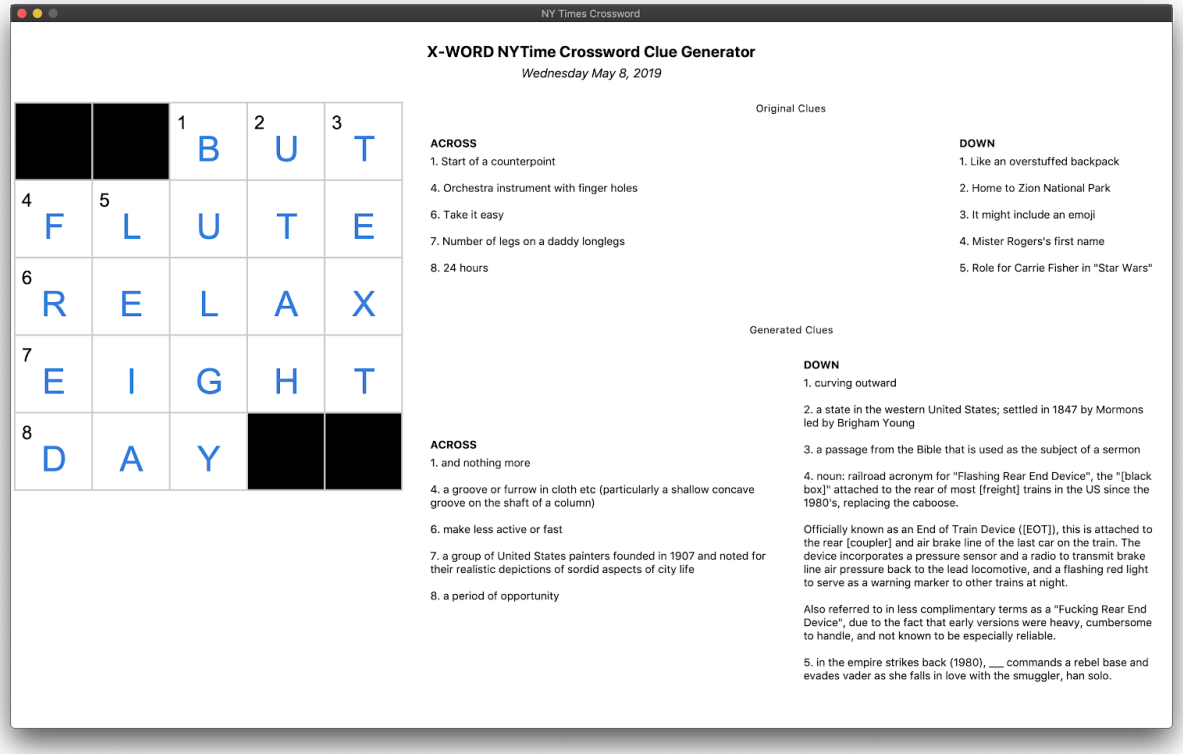

Figure 5. Test run for May 8, 2019.

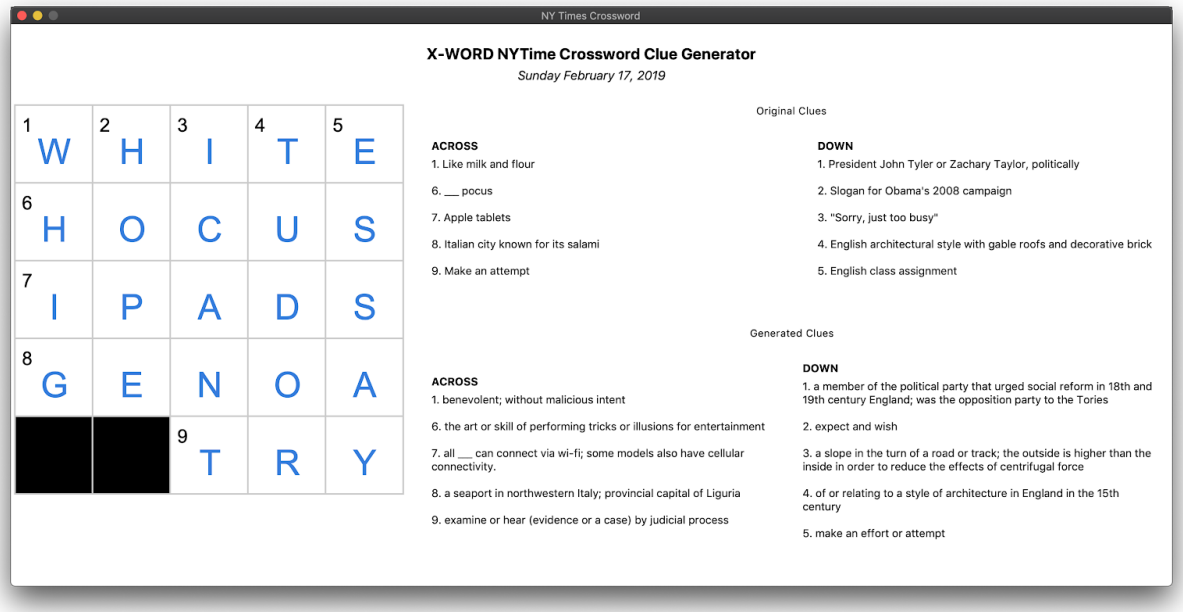

Figure 6. Test run for February 17, 2019.

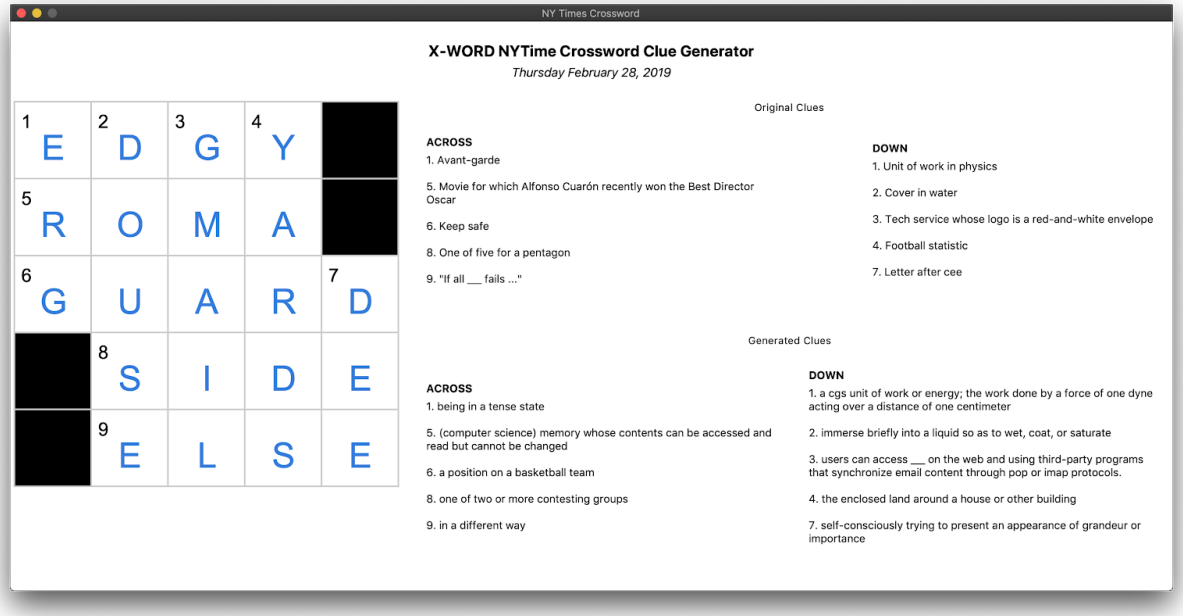

Figure 7. Test run for February 28, 2019.

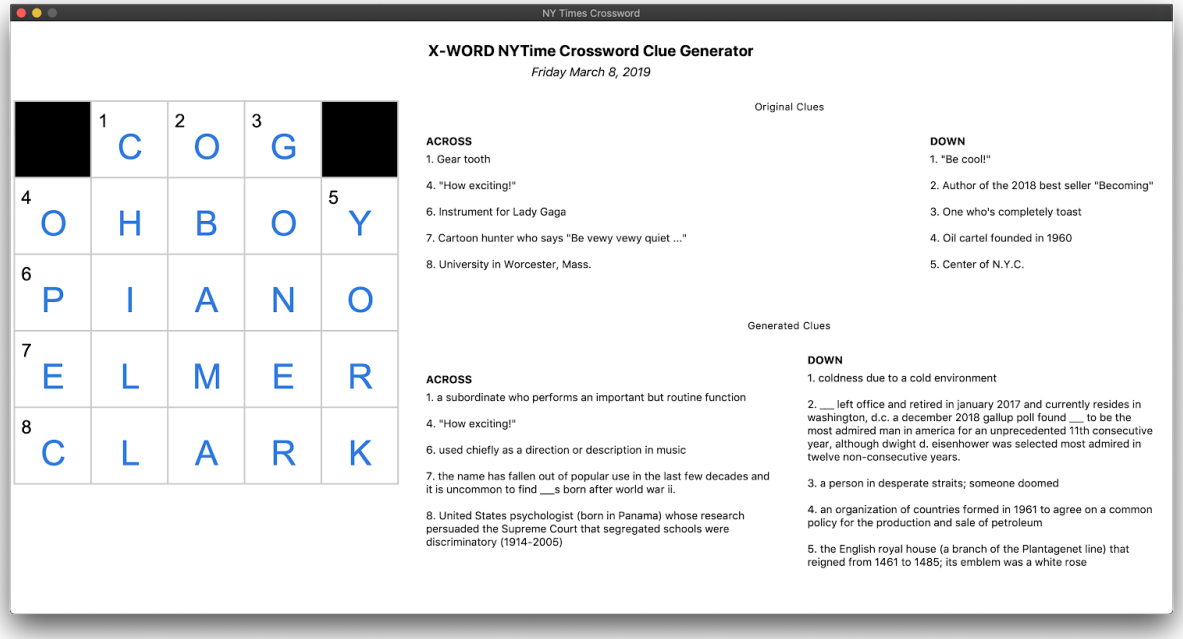

Figure 8. Test run for March 8, 2019.

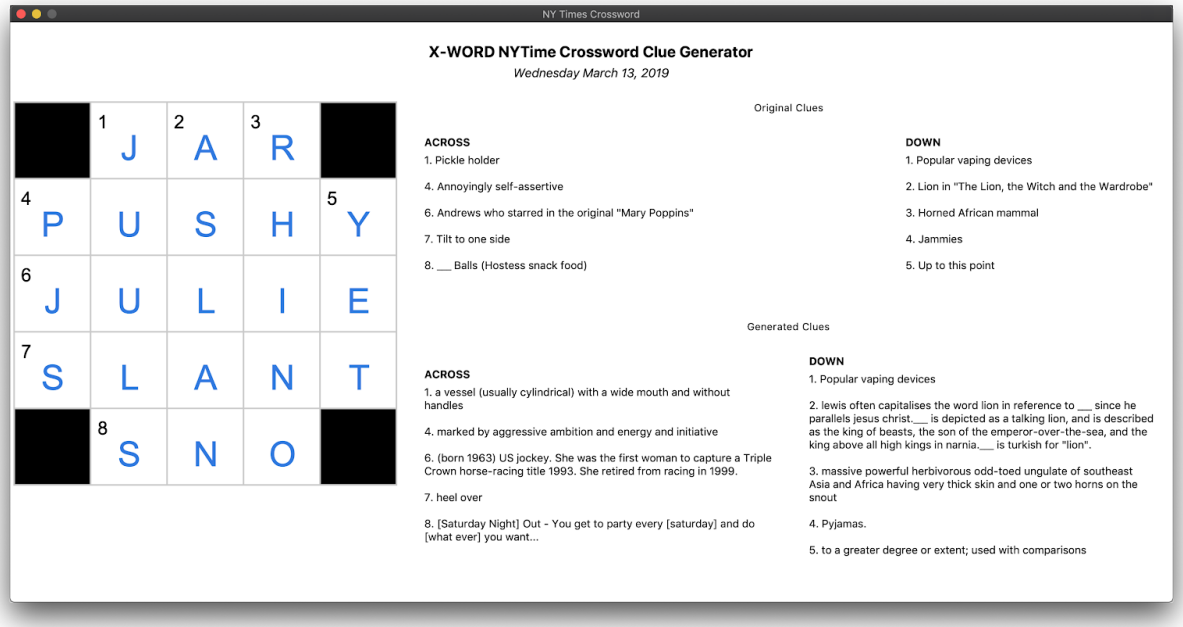

Figure 9. Test run for March 13, 2019.

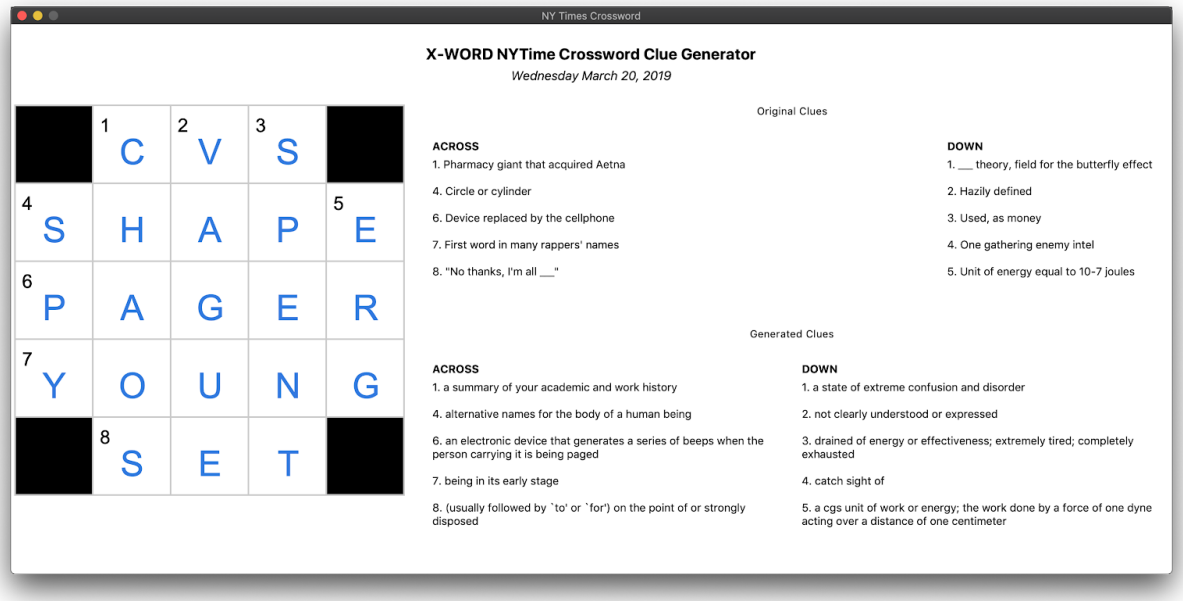

Figure 10. Test run for March 20, 2019.

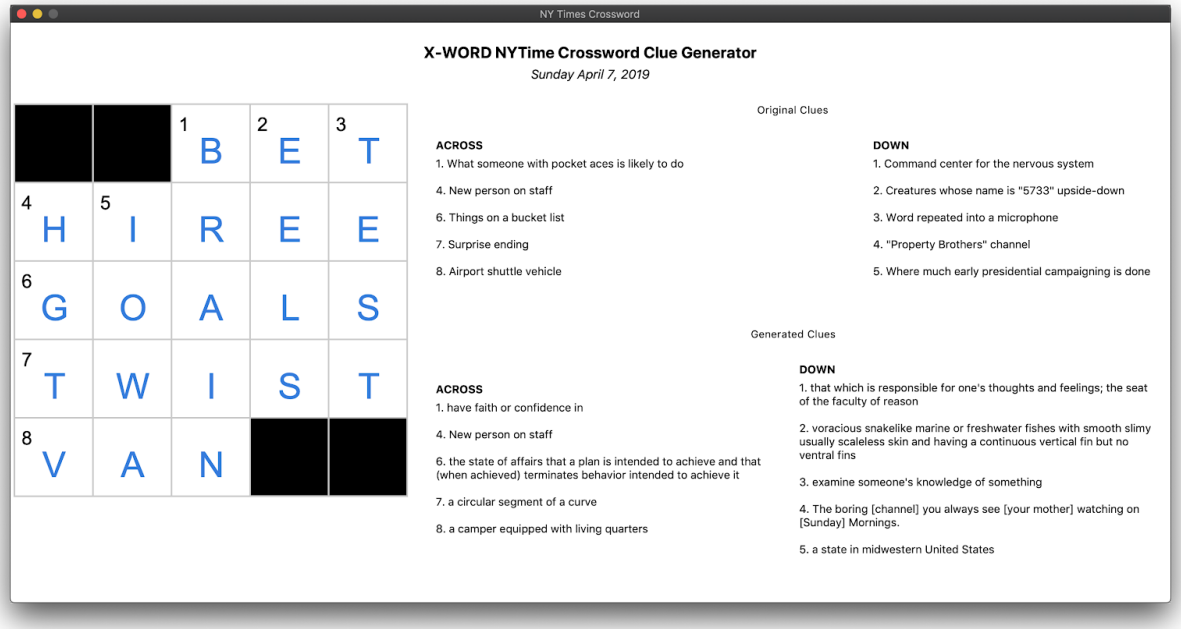

Figure 11. Test run for April 7, 2019.

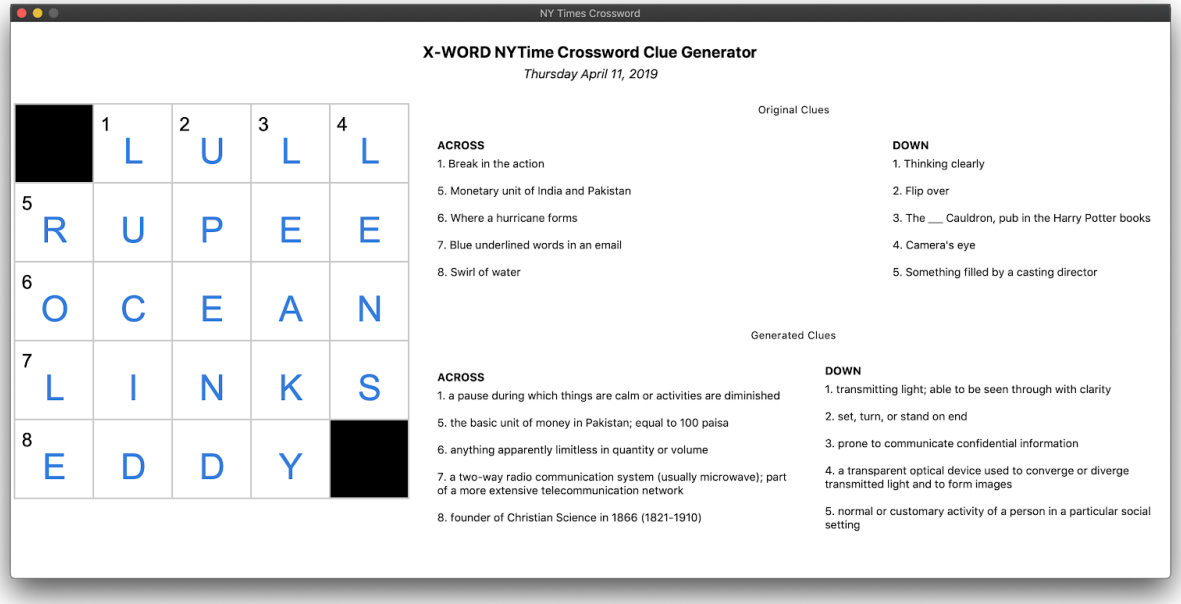

Figure 12. Test run for April 11, 2019.

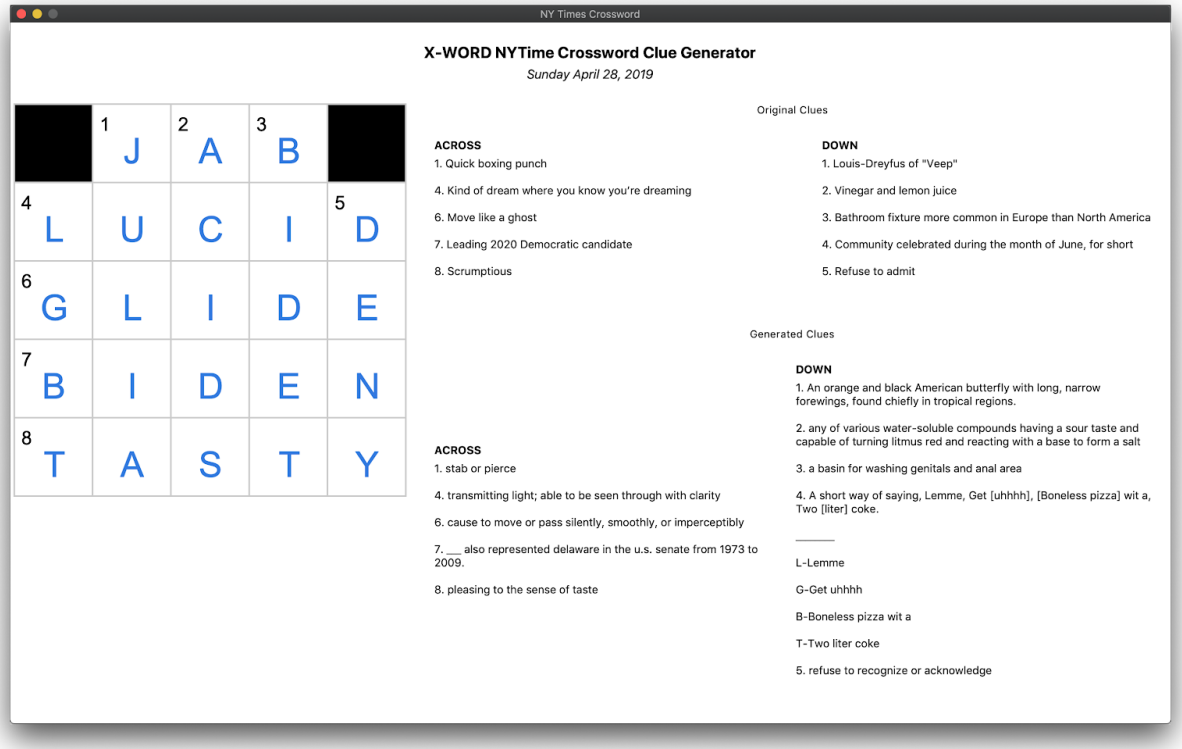

Figure 13. Test run for April 28, 2019.

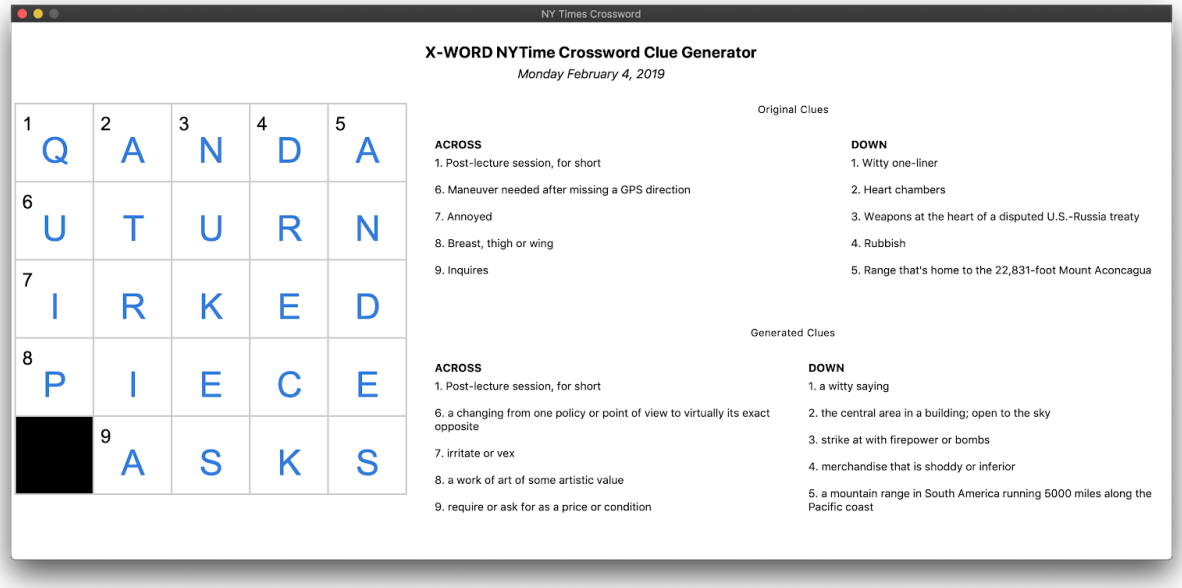

Figure 14. Test run for February 4, 2019.

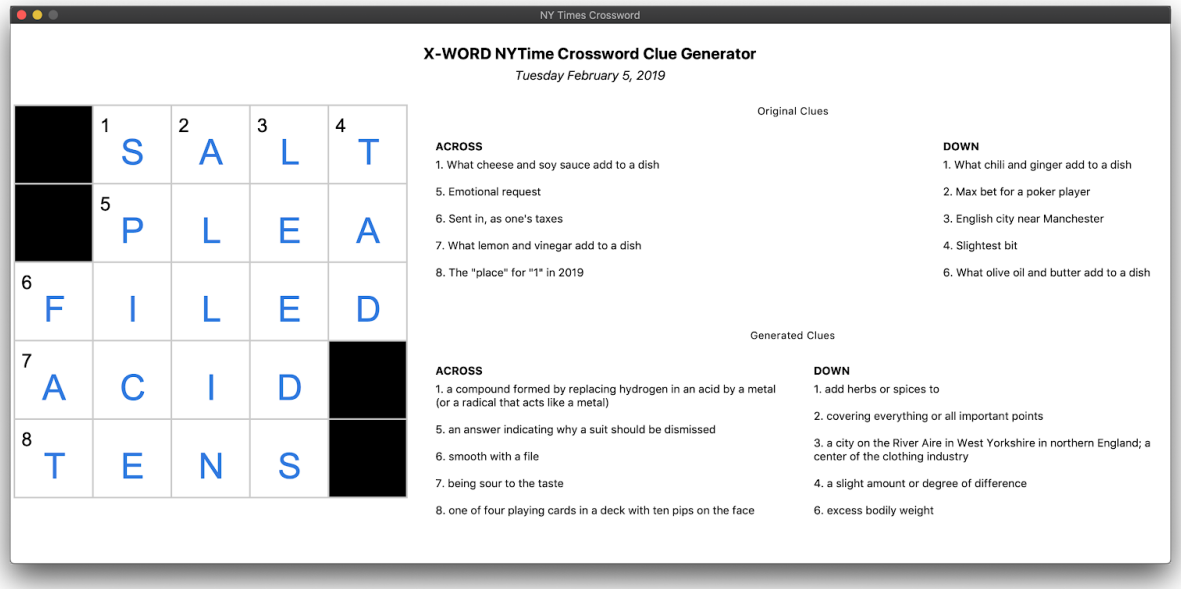

Figure 15. Test run for February 5, 2019.

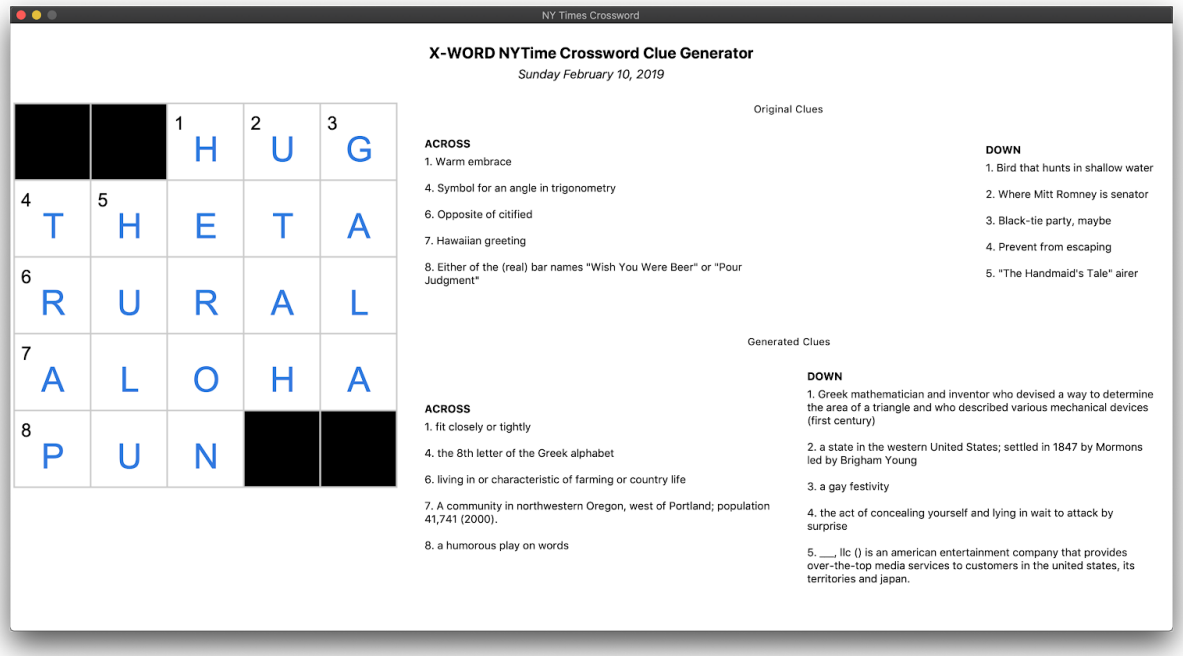

Figure 16. Test run for February 10, 2019.

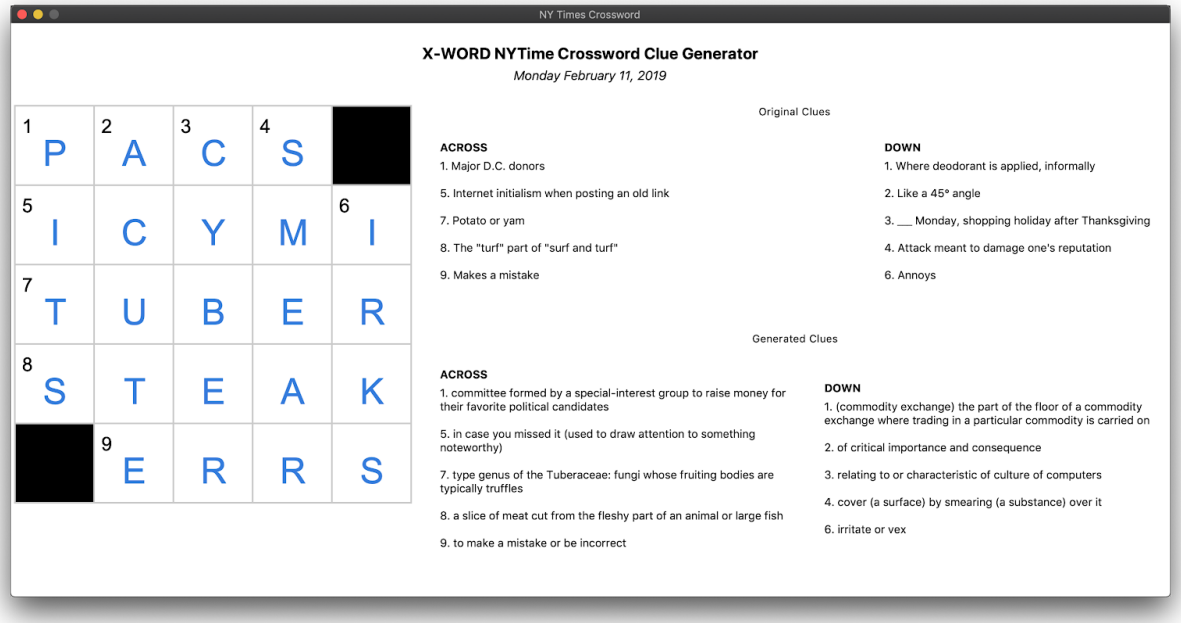

Figure 17. Test run for February 11, 2019.

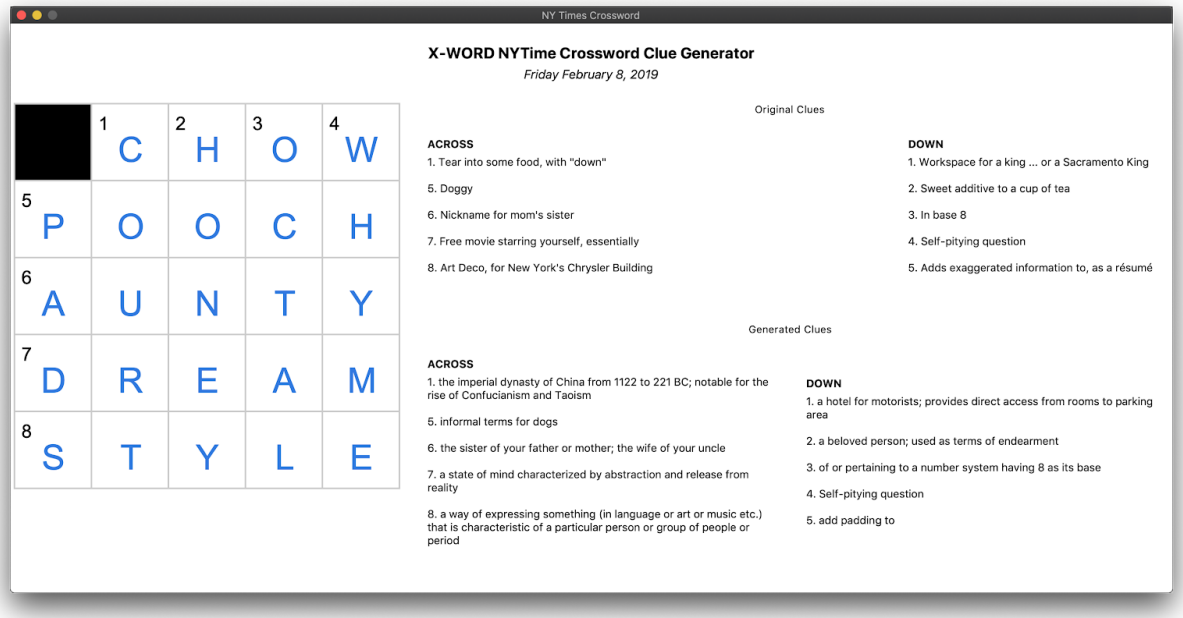

Figure 18. Test run for February 8, 2019.

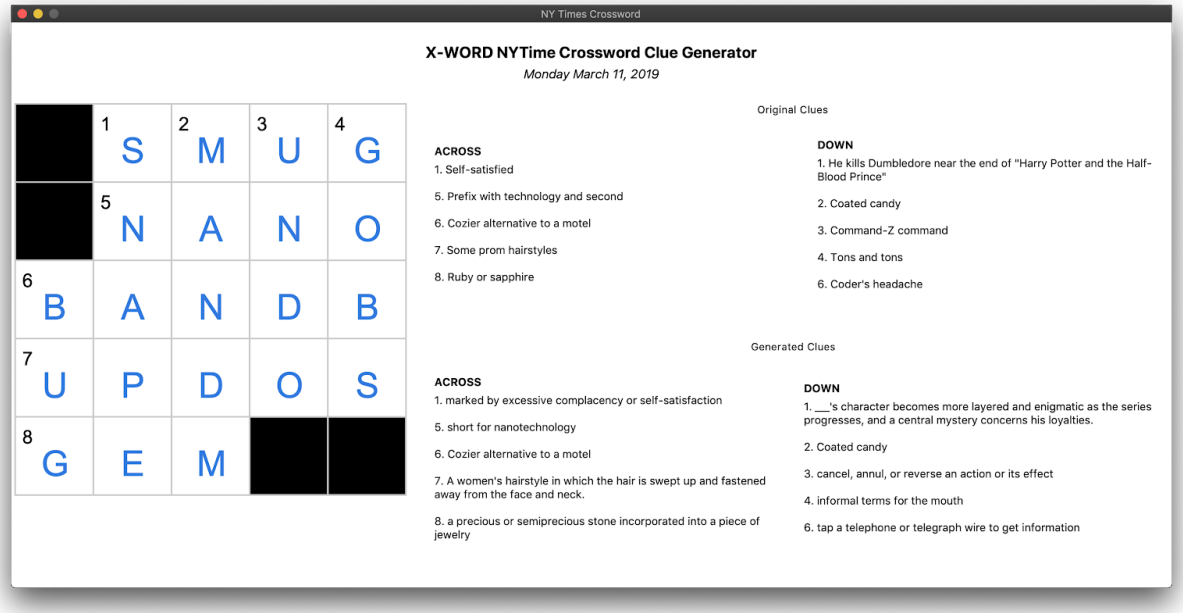

Figure 19. Test run for March 11, 2019.

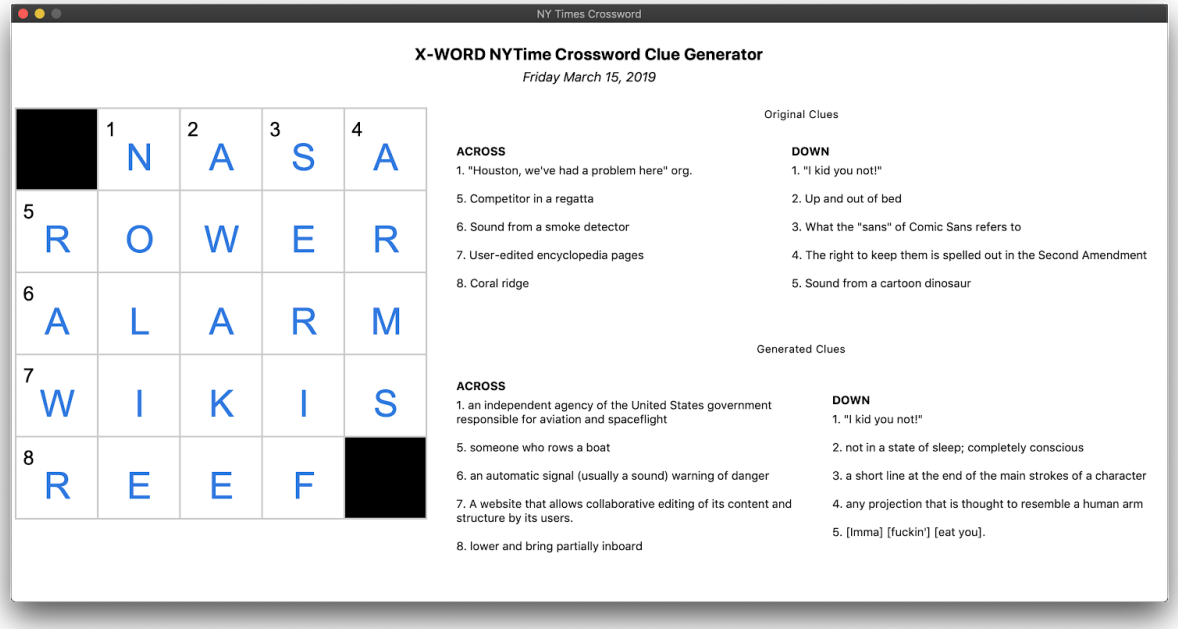

Figure 20. Test run for March 15, 2019.

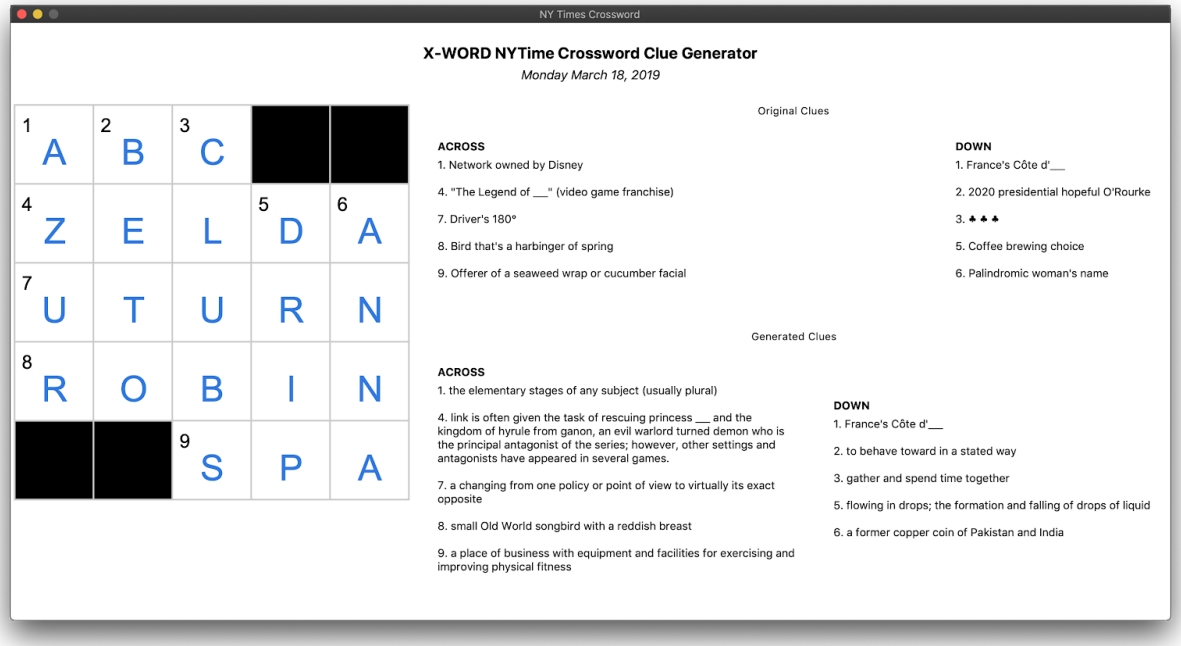

Figure 21. Test run for March 18, 2019.

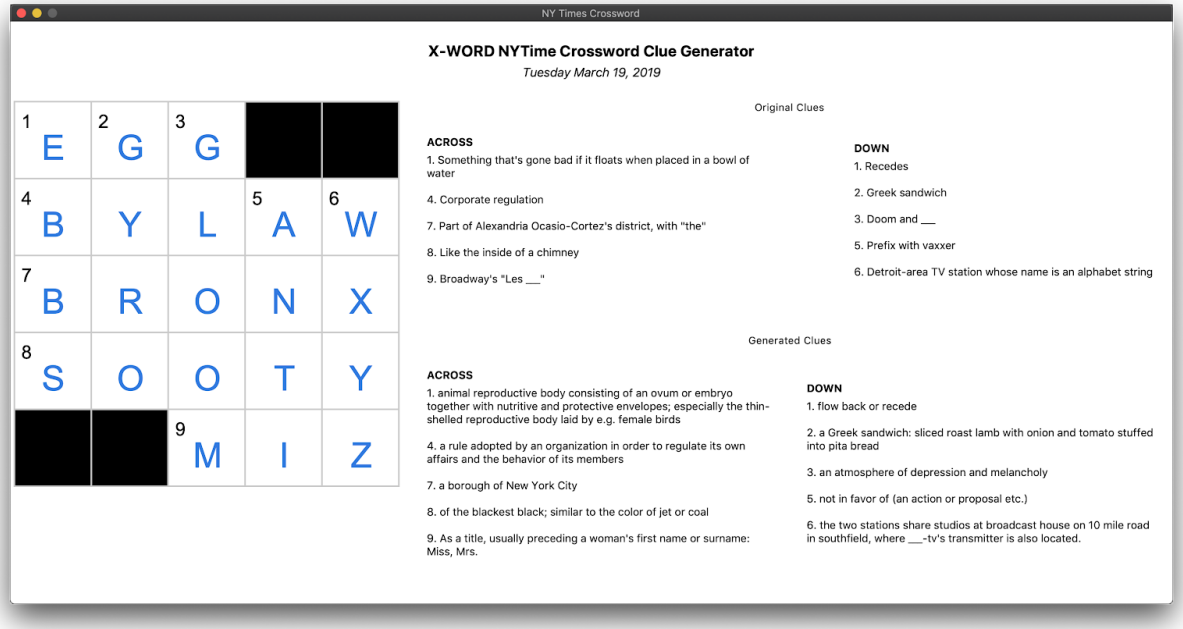

Figure 22. Test run for March 19, 2019.

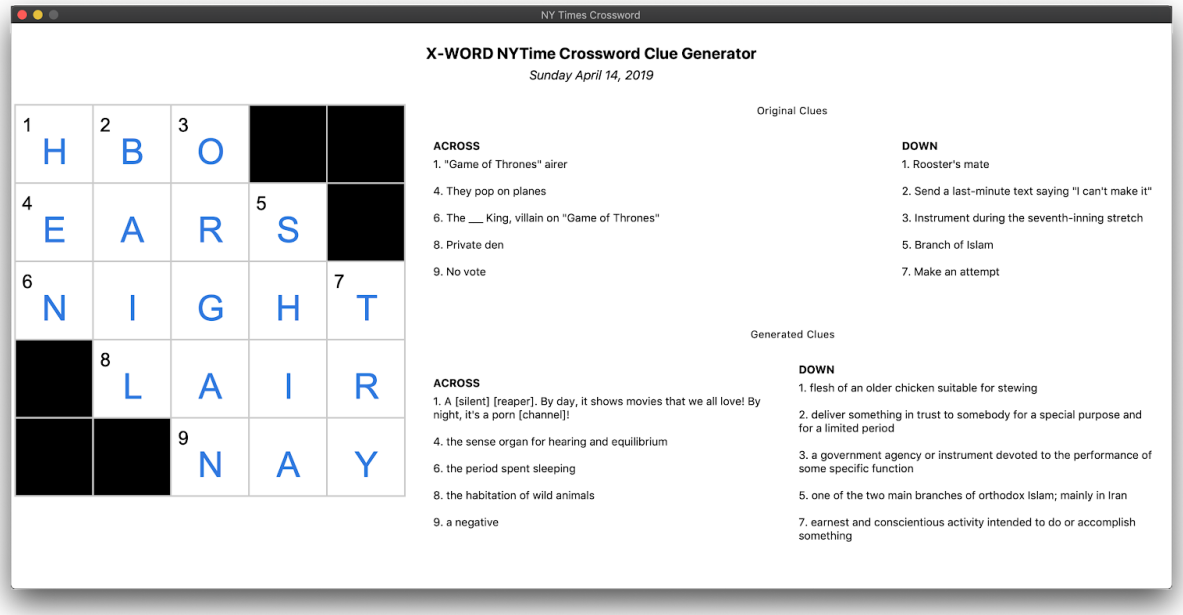

Figure 23. Test run for April 14, 2019.

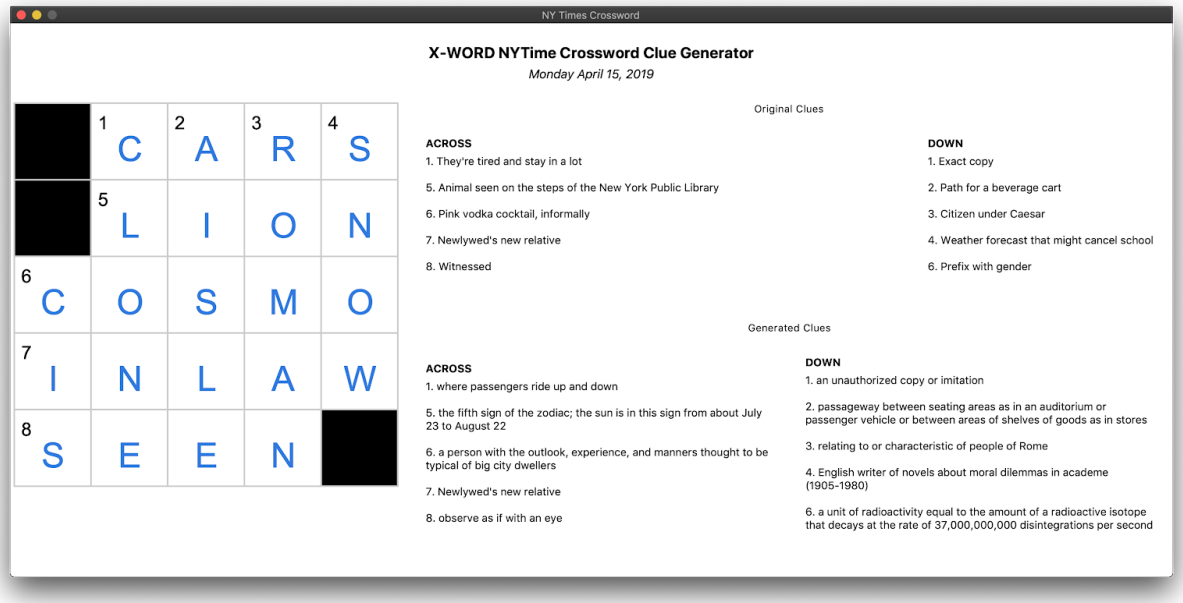

Figure 24. Test run for April 15, 2019.

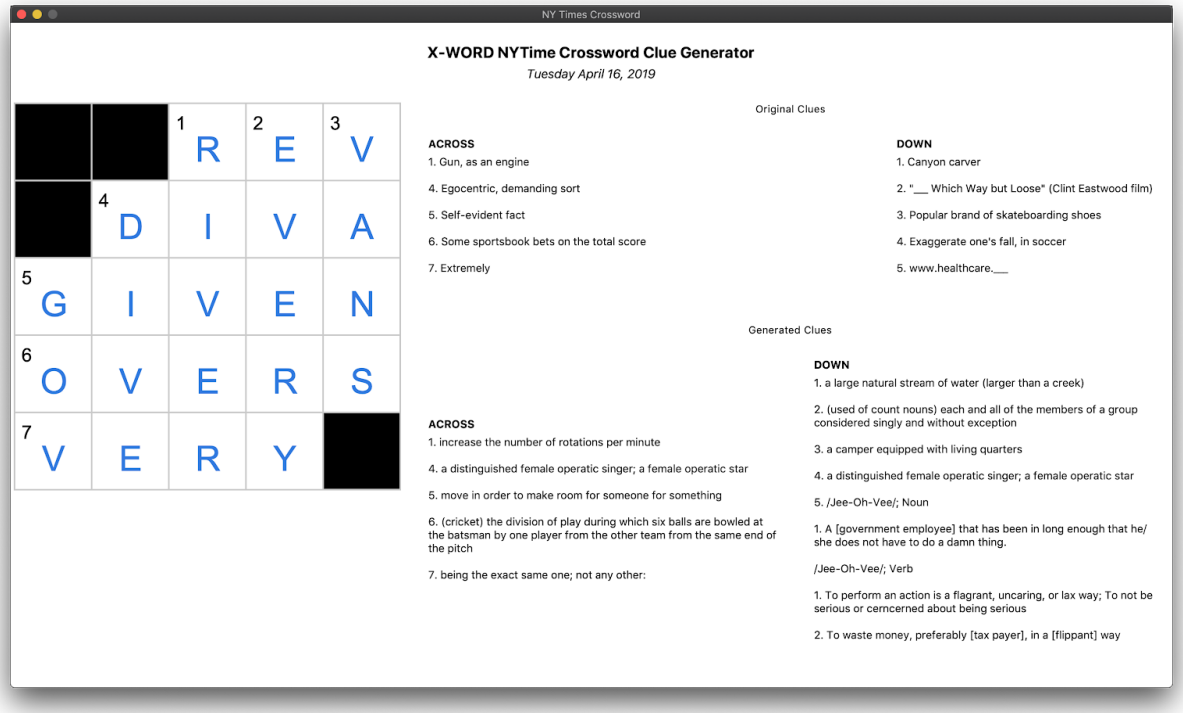

Figure 25. Test run for April 16, 2019.

References

[1] NYTimes, "NYTimes Crossword," 7 May 2019. [Online]. Available: https://www.nytimes.com/crosswords/game/mini.

[2] P. University, "WordNet: A Lexical Database for English," Princeton University, [Online]. Available: https://wordnet.princeton.edu/. [Accessed 7 May 2019].

[3] Oxford, "Oxford Dictionaries," Oxford Dictionary, [Online]. Available: https://developer.oxforddictionaries.com/. [Accessed 7 May 2019].

[4] M. Webster, "Merriam Webster Developer Center," Merriam Webster, [Online]. Available: https://dictionaryapi.com/. [Accessed 7 May 2019].

[5] U. Dictionary, "Rapid API," Rapid API, [Online]. Available: https://rapidapi.com/community/api/urban-dictionary. [Accessed 7 May 2019].

[6] Dictionary.com, "(unofficial) Google Dictionary API," [Online]. Available: https://googledictionaryapi.eu-gb.mybluemix.net/. [Accessed 7 May 2019].

[7] Wikipedia, "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Wikipedia. [Accessed 7 May 2019].

[8] M. Baiju, "Selenium with Python," Selenium, [Online]. Available: https://selenium-python.readthedocs.io/. [Accessed 7 May 2019].

[9] "Graphical User Interfaces with Tk," Python, [Online]. Available: https://docs.python.org/3/library/tk.html. [Accessed 7 May 2019].

[10] "NLTK: Natural Language Toolkit," NLTK, [Online]. Available: https://www.nltk.org/. [Accessed 7 May 2019].