CS461 Project Report

Group: Terra

Mehmet Bora Kurucu 21703404
Emre Orta 21703335
Ibrahim Eren Tilla 21702537
Göktuğ Öztürkcan 21702290
İsmail Yavuzselim Taşçı 21602936

1. Introduction

In computer science, the artificial intelligence concept has been improving, and based on the recent advances in the field of AI and machine learning, it would not be wrong if one claims that there are no limits for AI. AI has become a promising discipline that develops a variety of methods and application systems to reflect the notion of human intelligence on a specific problem. There are different applications of AI such as speech recognition, natural language processing, automation, and so on. In this project, we have focused on solving a crossword puzzle by developing an AI-based program. The challenging part of the crossword puzzles is that the solver of the puzzle is needed to have extensive knowledge of the language and the reasoning ability to construct a set of possible answers that is parallel with the clues provided by the puzzle. Hence, the program that can solve the crossword puzzle should be able to search for answers to the natural language questions and find an optimal solution for those questions. The reasonable approach to finding the possible answers is to solve the clues semantically by searching. The search on the lexicon dataset involves finding the semantical relatives of the clues. One of the aspects of AI, which is constraint satisfaction, is used where the clues in the crossword puzzle describe the constraint to be satisfied. Once the possible set of solutions for the crossword puzzle clues is found, the AI program should determine the optimal solution by considering the frequency of occurrence of the answers. The program aims to come up with different solutions related to the clues and to rank the candidates to find the optimal solution [1]. Overall, the project expects us to construct and solve the constraint satisfaction problem of the crossword puzzle by using and implementing AI-based methods and algorithms. The methods used throughout the project are described in the upcoming sections.

2. The Scope of The Project

The scope is clear, the project aims to solve the New York Times 5 x 5 crossword puzzle created by Joel Fagliano. The puzzle has across and down clues (5 for across and 5 for down). The actual view of the puzzle and the puzzle generated by us can be seen in Figure 2, and Figure 5 respectively. The clues of the puzzle are related to the answers to the puzzle to some extent and the aim is to make connections between clues and their relatives in terms of their meanings to complete the puzzle. First, the puzzle downloads the clues, the official solution, and solves the puzzle. Since using only reverse dictionaries only produces a limited number of solutions, various algorithms will be used to mimic the human mind. There are also various alternative algorithms, which could also be used [3]. The program will solve the using various algorithms such as AI phase, means-like, topic, lc-rc, general scoring. The puzzles that are on the New York Times on Saturdays are 7 x 7, which is not used by our program.

3. Brief Description of the AI

The AI work of the program relies heavily on the following methods' design. There are a couple of preliminary terms which are going to be used: "Nodes" are the square boxes in the grid of the puzzle, "Known" is the determined letters of a word (e.g "PL?NE" where the word "PLANE" is expected to be constructed), "Score" is the points for words returned by the Datamuse API. The AI of the program works in iterations where in each iteration, the program checks the clues with the known values (which are constructed at the start of each iteration) for each clue and proceeds. The AI returns only if all the clues are marked as solved, which means that they are true according to the dictionary. From now on, we will talk about the detailed inner workings of our AI. We highly suggest you look at the flowchart of our AI that we added into the Appendix section, before diving into our textual explanation. It will give you a good understanding of how our AI works.

All the nodes are held in a 2D node array. Each node holds a string of length 1 (the letter in each square), a number, and booleans to indicate if the node is the starting point of an answer. At the starting point of the AI, all the nodes are initialized with empty strings (""), and their booleans are set for the starting point of the clues.

After initialization, queries are sent to the Datamuse API using the clues with the methods mentioned below in the section for the query methods. The API returns lists consisting of 2 strings containing possible answers for each clue. These lists will be used for the selection of candidate words.

Then, the AI selects the candidate word with the highest ratio which is calculated by dividing the score of the word with the highest score over the score of the word with the second-highest score:

(Ratio = Highest Score Word / Second Highest Score Word)

After finding a new word AI copies the 2D node array and branches [2] then the AI writes the selected word into the 2D node array. During the next iterations, the AI uses certain letters that might be written into the grid while searching for new answers for the clues (for example, the first 2 letters of AURA may be known such as AU??). If the AI gets to a place where it can't find any words matching with the clues using the already known letters, the AI breaks out of that branch. The downside of this operation is that in a scenario where the AI decides to prioritize the wrong words which happened to have the highest scores during the first iterations, it takes a long time for the AI to recover since it goes too deep in the solution tree where going back from a wrong solution is inefficient in regards to time.

Finally, if every square in the grid is filled then the AI does a check to see if each of these are actual words (it is necessary because puzzle can be possibly filled by using less than 10 clues and some words might appear because of the answers to other clues answers) [5] and if each of the words are indeed real words the program ends and the final grid is drawn; if not, the AI breaks out of that branch and proceeds with the algorithm.

An example scenario in the puzzle of 23.12.2020 can be seen in the figure below:

```
Iteration 32

Puzzle:

 snoo
 auru
buddy
edge
tier

Requests:

Requesting https://api.datamuse.com/words?sp=snoo&max=1
Requesting https://api.datamuse.com/words?sp=auru&max=1
```

Figure 1: Example of an iteration

Figure 2: Official solutions by Joel Fagliano

In this example the AI is able to reach 7 of the answers in very early iterations, However, it is not able to reach the actual solution since it can not find the answer to clue 4 down. In iteration 32, since all the boxes are filled; the AI checks if all the words are actual words however since “auru” is not an actual word (its query returns empty) the AI breaks out of this branch of the tree[2]. and it looks like the following picture in the next iteration:

Figure 3: Another example of iteration

4. Query Methods

Throughout the project, we used Datamuse API to get our answers. Datamuse API is a word-finding query engine. However, there were different ways and options to send queries to the Datamuse API and we wanted to be able to utilize the capabilities of this API to get better results. To get the most out of the Datamuse API we used 3 different styles of queries depending on the clue. Following are the descriptions of those methods.

4.1 Means-Like Method

This method is set as the default method in our program to send the queries to the Datamuse API. A means-like method looks at all the words in the clue together and returns the words which are the clues that we sent. So, in our program if we can't see a "()" or a "___" we pass the whole clue into the means-like query. Since the means-like method looks at the words in the clues together and returns a result using all the words, it yielded better results when we sent the whole clue without cutting any parts of it. The only exception to this was if the clue had a "," in it. We realized that most of the times whatever came after the comma would be helpful for a human solving the puzzle but it was causing the AI to get worse results from Datamuse API, thus if a clue had a "," in it everything after the comma was cut from the string and the rest was sent into the query with the means-like method.

4.2 Topic Method

This method is activated if the clue has "()" in it. While we were looking at the puzzles we realized that whenever there was a parenthesis in the clue, it was to provide some context for the answer. "A topic-specific crossword is a crossword having most of the definition/answer pairs belonging to a given topic T" [4]. Luckily the Datamuse API has a setting to send in queries with a context and get results in that context. So, whenever a clue had parenthesis, whatever string was in the parenthesis would be passed on to the query as

the topic and the rest of the string would go into the query. This way we were able to get some answers while providing a context to the database which yielded better results than the plain means-like method.

4.3 Left Context, Right Context Method

These methods are activated whenever there is a "___" in the clue. We realized that whenever Joel Fagliano puts a "___" in the clue it usually means that the word that is supposed to be in the blank part usually goes together with the word on its left or right. Once we realized that we decided to use the left context and right context query types of the Datamuse for clues with "___" in them. We checked the position of the "___" and if it was at the beginning of the string we sent the word on it right as the left context in the query, if the "___" was at the end of the string we sent the word on its left as the right context in the query. One important thing to note is that sometimes there is a "___" in the clue and it's not at the beginning or at the end. In that case, we send queries using both the left and the right context.

4.4 Left Context, Right Context with topic

In some rare cases, the clue will have both "___" and "()". When this happens the same rules with regular left context and right context apply but we also add whatever is in the parenthesis as a topic in the query.

5. Conclusion

In conclusion, we managed to implement a complete AI program that can solve the 5*5 New York Times Puzzle created by Joel Fagliano. The AI program that we implemented is based on sending queries to the word searching engine by three distinct methods. The methods used are means-like, topic, left, and right context methods. Applying different methods with different structures provides higher accuracy in finding the candidate words since the clues in the puzzle may appear in different formats such as blank filling, etc. The applied queries return the estimated words and the corresponding scores which are used for ranking the candidate words. The puzzle is updated by assuming that the highest scored puzzle to be the optimal solution for that clue. Once all the clues are matched with a word from the query results, the puzzle takes its final form which can be seen from the screenshots provided in the Appendix section. The solutions that are found by our program for different puzzles are presented, too. It is important to note that the dates in the screenshots are not the actual date that the puzzles were released because we took the puzzle data on the actual date the puzzles were released but we tried all of them in a single day. Also, there are 11 puzzles in the Appendix section that were taken from the last two week's puzzles. Overall, the term project was quite entertaining and informative to implement since it captures several critical AI concepts. The most challenging part of the project is to solve the puzzles that include metaphorical meanings or puzzles with clues that define the actual word indirectly. However, we are pleased with the performance of our program, "Terra", in terms of the accuracy of it and the AI perspective resides behind it.

6. Appendix

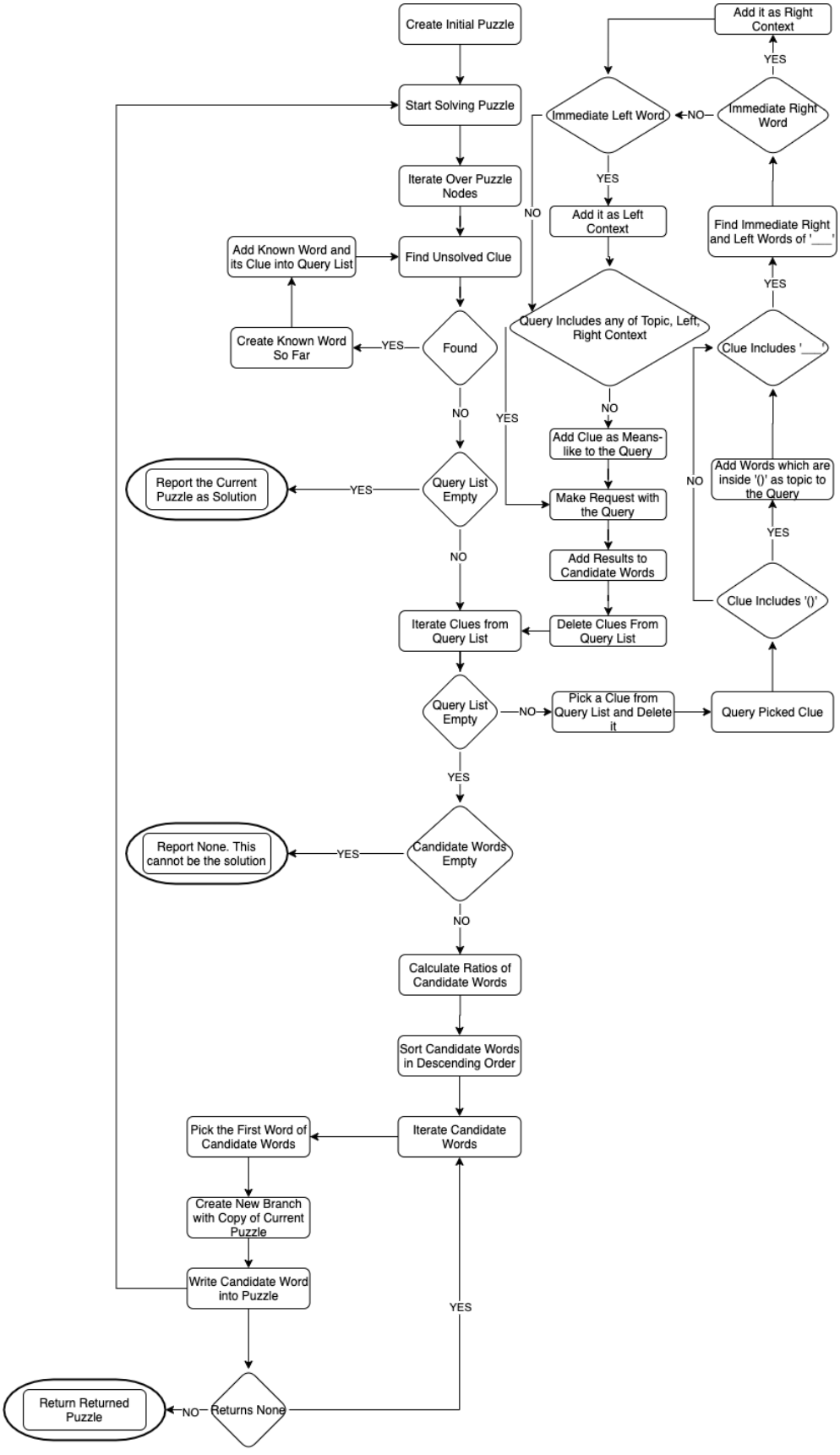

Figure 4: Flowchart of AI

Code

Our code split in 4 files as given below:

DEMO.py

```
"""
Group Name: TERRA

Group Members:  Göktuğ Öztürkcan
                Ibrahim Eren Tilla
                Emre Orta
                Mehmet Bora Kurucu
                İsmail Yavuzselim Taşçı

Programming Language: Python 3

In this demo, we are retrieving the crossword puzzle data from the
nytimes website. Then,
we are processing said data into smaller smaller chunks where we
transform it to a line-by-line
data. We put the data in our GUI, where the user can see the puzzle
clearly. For the answers, we
are using Selenium for the "reveal solutions" use-case. After revealing
the data, we take it and
put in the GUI of the puzzle as well.
"""

"""
====================== HOW TO RUN THE CODE ======================
-Make sure you have python installed in your environment.
-You need to install selenium with following command: pip install
selenium
-You need to have an appropriate chromedriver in the same location as
this file. To download appropriate driver please check out:
https://chromedriver.chromium.org/
-Use the following command into your terminal to run the program:
python demo.py
"""

from gui import CrosswordGUI
from scraper import NYCrossword
from solver import Solver

data = NYCrossword().get_data()
print(data["cells"])
print(data["clues"])
s = Solver()
solution = s.create_puzzle(data['clues'], data['cells'])
gui = CrosswordGUI(data=data, solution=solution)
```

GUI.py

```
import time
from tkinter import *
import datetime
first = 0

################################ CROSSWORD GUI CLASS START
################################
class CrosswordGUI:
    """
  A class used to create the CrosswordGUI and keep its attributes.

  Methods
  ---------
  show_root()
    creates the window for GUI.
  draw_canvas()
    Draws the grid of the puzzle using information taken from the
website.
  draw_clues()
    Gets the clues from the website then writes them on top of the
grid.
  draw_clock(another_puzzle)
    Draws the clock.
  """
    PUZZLE_SIZE = 5
    MARGIN = 10


    def __init__(self, data,solution):
        print("Found the solution")
        print("Constructing user interface...")
        self.solution = solution
        self.data = data
        self.root = Tk()
        self.root.title("The Mini Crossword - NY Times")
        Label(self.root, text='The Mini Crossword', font='Times
48').pack()

        self.draw_canvas(self.data['cells'])
        self.draw_canvas(self.solution)

        #self.draw_canvas2(self.data2)
        self.draw_clues(data)
        self.draw_clock()
        self.show_root()

    def show_root(self):
        self.root.resizable(False, False)
        self.root.mainloop()



    def draw_canvas(self, data):
        """
        Draws the exact square grid in New York Times puzzle using data
from New York Times webpage.
```

```
        Parameters
        ----------
        data : data taken from newyork times website with selenium.
        """
        canvas = Canvas(self.root, width=420, height=420)
        global first
        if first == 0:
            canvas.pack(fill=BOTH, side=LEFT)
            first = 1
        else:
            canvas.pack(fill=BOTH, side=RIGHT)



        usable = int(canvas['width']) - self.MARGIN * 2

        font_size_number = usable//15
        font_number = ('Arial', font_size_number)
        font_number2 = ('Arial', usable//25)
        font_size_letter = font_size_number * 2
        font_letter = ('Arial', 35)

        square_width = usable // self.PUZZLE_SIZE

        for i in range(self.PUZZLE_SIZE):
            for j in range(self.PUZZLE_SIZE):
                cell_id = str(i * self.PUZZLE_SIZE + j)
                cell_data = data[cell_id]

                x1 = j * square_width + self.MARGIN
                y1 = i * square_width + self.MARGIN
                x2 = j * square_width + square_width + self.MARGIN
                y2 = i * square_width + square_width + self.MARGIN
                bg = 'black' if cell_data['block'] else 'white'
                canvas.create_rectangle(x1, y1, x2, y2, fill=bg,
outline='gray', width=1.5)

                x1 = j * square_width + (square_width // 2) +
self.MARGIN
                y1 = i * square_width + (2 * square_width // 3) +
self.MARGIN - 5
                canvas.create_text(x1, y1,
text=cell_data['text'].upper(), font=font_letter)
                x1 = j * square_width + (font_size_number // 2) +
self.MARGIN
                y1 = i * square_width + (2 * font_size_number // 3) +
self.MARGIN
                canvas.create_text(x1, y1, text=cell_data['number'],
font=font_number2)

    def draw_clues(self, data):
        """
        Using selenium gets the clues from the webpage then writes them
to the previously created grid.
        Parameters
        ----------
        data : data taken from New York Times website with selenium.
        """
```

```
        font_main = 'Arial 10 normal'

        font_title = 'Arial 14 bold'

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

        across_pane = PanedWindow(pane)
        across_pane.pack(fill=X, side=LEFT, padx=(self.MARGIN,
self.MARGIN), pady=(self.MARGIN, self.MARGIN))
        Label(across_pane, text='ACROSS', font=font_title,
anchor='center').pack(fill=BOTH)
        for clue in data['clues']['across']:
            text = clue['id'] + '. ' + clue['text'] + '\n'
            Label(across_pane, text=text, font=font_main,
wraplength=7555, anchor='w').pack(fill=BOTH)

        down_pane = PanedWindow(pane)
        down_pane.pack(fill=X, side=RIGHT, padx=(self.MARGIN,
self.MARGIN), pady=(self.MARGIN, self.MARGIN))
        Label(down_pane, text='DOWN', font=font_title,
anchor='center').pack(fill=BOTH)
        for clue in data['clues']['down']:
            text = clue['id'] + '. ' + clue['text'] + '\n'
            Label(down_pane, text=text, font=font_main,
wraplength=7555, anchor='w').pack(fill=BOTH)

    def draw_clock(self):
        """
        draws the clock and prints the group name with it.
        """
        def update_clock():
            """
            Used to update the clock in regular intervals.
            """
            t = time.strftime("%H:%M:%S")
            clock.configure(text=t)
            self.root.after(1000, update_clock)

        pane = PanedWindow()
        pane.pack(fill=X, side=BOTTOM)
        clock = Label(pane, text='', font='Times 14 italic')
        clock.pack(side=LEFT, pady=(0, self.MARGIN))
        date = Label(pane, text=datetime.date.today().strftime('%A, %B
%#d, %Y'), font='Times 14')
        date.pack(side=LEFT, pady=(0, self.MARGIN))
        group_name = Label(pane, text='TERRA', font='Times 14')
        group_name.pack(side=LEFT, pady=(0, self.MARGIN))

        update_clock()
```

SCRAPER.py

```
import time

from selenium import webdriver
from selenium.webdriver.chrome.options import Options


class SeleniumCrosswordHelper:
    """
    A class to determine the web-operations of Selenium.

    Methods
    ---------
    get_clues()
      Returns the data of the clues using the tags that are embedded in
the html file.

    get_cells()
      Returns the data of the cells similarly to get_clues().

    reveal_solutions()
      The processes for the revealment of the solutions using Selenium.

    _click_ok()
      Selenium function to click the OK button.

    _click_reveal_menu_button()
      Selenium function to click the reveal menu button.

    _click_puzzle_reveal_button()
      Selenium function to click the reveal button.

    _click_reveal_confirmation_button()
      Selenium function to click the confirmation button.

    _close_pop_up()
      Selenium function to close pop-ups.
    """

    def __init__(self):
        options = Options()
        options.headless = False
        options.add_argument("--log-level=3")
        options.add_experimental_option('excludeSwitches',
['enable-logging'])
        self.driver = webdriver.Chrome(options=options)
        print("Reaching to
https://www.nytimes.com/crosswords/game/mini...")
        self.driver.get("https://www.nytimes.com/crosswords/game/mini")
        time.sleep(1)

    def get_clues(self):
        """
        This function first finds the class name tag related with the
clues Then,
        it stores the elements which are accessed by the tag names such
as 'div','h3','li','span'
```

```
        Then the function returns the data array that contains the
elements for clues of the puzzle.
        """
        print("Scraping clues...")
        data = {}
        clue_lists =
self.driver.find_element_by_class_name('Layout-clueLists--10_Xl')
        divs = clue_lists.find_elements_by_tag_name('div')
        for div in divs:
            title = div.find_element_by_tag_name('h3').text.lower()
            data[title] = []
            list_items = div.find_elements_by_tag_name('li')
            for list_item in list_items:
                spans = list_item.find_elements_by_tag_name('span')
                data[title].append({'id': spans[0].text, 'text':
spans[1].text})

        return data

    def get_cells(self):

        """
        Creates an empty set at first, then according to the tags of
html, adds cells up to it.
        Returns the set at the end of the function.
        """
        print("Scraping puzzle geometry and solutions...")
        data = {}
        cell_table =
self.driver.find_element_by_css_selector('g[data-group=cells]')
        cells = cell_table.find_elements_by_tag_name('g')
        for cell in cells:
            cell_data = {'block': False, 'text': '', 'number': ''}
            rect = cell.find_element_by_tag_name('rect')
            cell_id = rect.get_attribute('id').split('-')[2]
            if 'Cell-block' in rect.get_attribute('class'):
                cell_data['block'] = True
            text_fields = cell.find_elements_by_tag_name('text')
            for text_field in text_fields:
                if text_field.get_attribute('text-anchor') == 'start':
                    cell_data['number'] = text_field.text
                if text_field.get_attribute('text-anchor') == 'middle':
                    cell_data['text'] = text_field.text
            data[cell_id] = cell_data

        return data

    def reveal_solutions(self):
        """
        Using the simple functions below, this function performs the
process of revealing solutions.
        """
        print("Revealing the solution...")
        self._click_ok()
        self._click_reveal_menu_button()
        self._click_puzzle_reveal_button()
        self._click_reveal_confirmation_button()
        self._close_pop_up()
```

```
    def _click_ok(self):
        """
        Clicks the OK button.
        """
        ok_button =
self.driver.find_element_by_css_selector('button[aria-label="OK"]')
        ok_button.click()

    def _click_reveal_menu_button(self):
        """
        Clicks the reveal menu button.
        """
        reveal_button =
self.driver.find_element_by_css_selector('button[aria-label="reveal"]')
        reveal_button.click()

    def _click_puzzle_reveal_button(self):
        """
        Clicks the reveal button.
        """
        puzzle_reveal_button =
self.driver.find_element_by_link_text('Puzzle')
        puzzle_reveal_button.click()

    def _click_reveal_confirmation_button(self):
        """
        Clicks the confirmation button.
        """
        reveal_button =
self.driver.find_element_by_css_selector('button[aria-label="Reveal"]')
        reveal_button.click()

    def _close_pop_up(self):
        """
        Clicks and closes the pop-up screens.
        """
        spans = self.driver.find_elements_by_tag_name('span')
        for span in spans:
            if 'closeX' in span.get_attribute('class'):
                span.click()
                return


class NYCrossword(SeleniumCrosswordHelper):
    """
    A class to initialize and use the Selenium through NYTimes Mini
Puzzle

    """

    def __init__(self):
        super(NYCrossword, self).__init__()
        self.reveal_solutions()

    def get_data(self):
        return {'clues': self.get_clues(), 'cells': self.get_cells()}
```

SOLVER.py

```
import requests
import math
"""
   Static Functions
   ---------
   query()
     This function queries Datamus API for the given parameters.
     We can query using means-like, left-context, right-context, topic
and known until now.

   prune()
     This function reevaluates results coming from the Datamuse API. We
check the length of the
     results with our puzzle and also we add score fields when we do
not get that field from API.

   make_request()
     This function either sends a request as 'meanslike' 'left-context'
'right-context' depending
     on the structure of the clue.

   get_difference()
     This function is used to calculate the scores of candidate words.
     Divides the scores with the scores to their right and returns the
division array.

   def create_puzzle_from_existing():
     This function simply creates a deepcopy of the given puzzle and
returns it.

   Global Variables
   ----------
   url: Base url of the Datamuse API. Used in query() function.

   iter: Iteration number that we are currently in.
"""

url = 'https://api.datamuse.com/words'
iter = 0


def query(sp, ml=None, lc=None, rc=None, topic=None):
   if sp.find('?') != -1:
       if ml:
           if ml.find(',') != -1:
               ml = ml[:ml.find(',')]

       payload = {
           'ml': ml,
           'sp': sp,
           'max': 2,
           'lc': lc,
           'rc': rc,
           'topics': topic
```

```
        }

        r = requests.get(url, params=payload)
        print('Requesting {}'.format(r.url))

        results = r.json()

        results = prune(results, sp)
        if len(results) != 0:
            return results
    else:
        payload = {'sp': sp, 'max': 1}
        r = requests.get(url, params=payload)
        print('Requesting {}'.format(r.url))
        results = r.json()

        results = prune(results, sp)
        return results


def prune(results, sp):
    for result in results:
        if not 'score' in result.keys():
            result['score'] = 1
        result['word'] = result['word'].replace(" ", "")

    results = [result for result in results if len(result['word']) ==
len(sp)]

    return results


def make_request(known, clue):
    rc = None
    lc = None
    topic = None
    ml = None

    position = clue.find('(')
    if position != -1:
        p2 = clue.find(')')
        topic = clue[position + 1:p2]
        clue = clue[:position] + clue[p2 + 1:]

    position = clue.find('___')
    if position != -1:
        if position == 0:
            rc = clue[4:]
            indexes = [rc.find('.'), rc.find(','), rc.find(' ')]
            indexes = [x for x in indexes if x != -1]
            if len(indexes) != 0:
                rc = rc[:min(indexes)]
        else:
            lc = clue[:position - 1]
            indexes = [lc.find('.'), lc.find(','), lc.find(' ')]
            indexes = [x for x in indexes if x != -1]
            if len(indexes) != 0:
                lc = lc[max(indexes):]
```

```
    else:
        ml = clue

    return query(known, ml, lc, rc, topic)


def get_difference(arr):
    for i in range(len(arr) - 1):
        if arr[i + 1] == 1:
            break
        if 'score' in arr[i].keys() and 'score' in arr[i + 1].keys():
            arr[i]['score'] = arr[i]['score'] / arr[i + 1]['score']


def create_puzzle_from_existing(puzzle_to_copy):
    puzzle = []

    for row in puzzle_to_copy:
        puzzle.append([])
        for node in row:
            new_node = Node(node.text, node.block, node.clue_across,
                            node.clue_down, node.across_candidates,
                            node.down_candidates, node.across_solved,
                            node.down_solved, node.number)
            puzzle[-1].append(new_node)
    return puzzle


class Node:
    """
    A class to represent individual puzzle cells

    Methods
    ---------
    to_string()
      This is the methods we used while printing the puzzle at each
iteration.
      Basically prints the necessary information of the Node class.

    Variables
    ----------
    text: The character that is in a tile of the puzzle

    block: Keeps if tile is writable or blocked

    clue_across: Keeps the number of across clue starting from the tile,
if it exists

    clue_down: Keeps the number of down clues starting from the tile, if
it exists

    across_candidates: Candidate words for the across clue starting from
the tile

    down_candidates: Candidate words for the downwards clue starting
from the tile
```

```
   across_solved: Solved word going across that is starting from the
current tile

   down_solved: Solved word going down that is starting from the
current tile

   number: Clue number of the node

 """

   def __init__(self, text, block, clue_across, clue_down,
across_candidates,
                down_candidates, across_solved, down_solved, number):
       self.clue_across = clue_across
       self.clue_down = clue_down
       self.text = text
       self.block = block
       self.across_candidates = across_candidates
       self.down_candidates = down_candidates
       self.across_solved = across_solved
       self.down_solved = down_solved
       self.number = number

   def to_string(self):
       if self.text == '':
           return ' '
       return self.text


class PuzzleTree:
   """
   A class to represent candidate puzzles in a tree-like manner.

   Methods
   ---------
   solve()
     Recursive function to solve our puzzle. It makes use of query()
and get_difference() functions.
     Basically it creates a new PuzzleTree for each and every candidate
puzzle. Then it deepens the search
     starts from the most promising puzzle.
    Variables
   ---------
   puzzle: Array to keep the current puzzle

   parent: Parent of the current Puzzle

 """

   def __init__(self, puzzle, parent, score):
       self.PUZZLE_LENGTH = 5
       self.parent = parent
       self.puzzle = puzzle
       self.children = []
       self.score = score

   def solve(self):
       global iter
```

```
        iter += 1
        print('\n\n---------------------------------\n\n')
        print('Iteration {}'.format(iter))
        print('\nPuzzle:\n')
        for row in self.puzzle:
            text = ""
            for element in row:
                text = text + element.to_string()
            print(text)

        print('\nRequests:\n')
        for i, row in enumerate(self.puzzle):
            for j, element in enumerate(row):
                if element.clue_across:
                    known = ''
                    for k in range(j, self.PUZZLE_LENGTH):
                        if row[k].block:
                            break
                        if row[k].text == '':
                            known = known + '?'
                        else:
                            known = known + row[k].text
                    if not element.across_solved:
                        element.across_candidates = make_request(
                            known, element.clue_across)
                        if element.across_candidates == None:
                            element.across_candidates = []
                        get_difference(element.across_candidates)
                if element.clue_down:
                    known = ''
                    for k in range(i, self.PUZZLE_LENGTH):
                        if self.puzzle[k][j].block:
                            break
                        if self.puzzle[k][j].text == '':
                            known = known + '?'
                        else:
                            known = known + self.puzzle[k][j].text
                    if not element.down_solved:
                        element.down_candidates = make_request(
                            known, element.clue_down)
                        if element.down_candidates == None:
                            element.down_candidates = []
                        get_difference(element.down_candidates)

        candidates = []
        solved = 0
        for i, row in enumerate(self.puzzle):
            for j, element in enumerate(row):
                if element.across_solved:
                    solved += 1
                elif element.clue_across and len(
                        element.across_candidates) != 0:
                    candidates.append(element.across_candidates[0])
                if element.down_solved:
                    solved += 1
                elif element.clue_down and len(element.down_candidates)
!= 0:
                    candidates.append(element.down_candidates[0])
```

```
        if len(candidates) == 0:
            if solved == self.PUZZLE_LENGTH * 2:
                return self
            return None

        candidates = sorted(
            candidates, key=lambda tup: tup['score'], reverse=True)

        print('\nCandidate Words:\n')
        for candidate in candidates:
            print(candidate)
            new_puzzle = create_puzzle_from_existing(self.puzzle)
            for i, row in enumerate(new_puzzle):
                for j, element in enumerate(row):
                    if element.across_candidates and candidate ==
element.across_candidates[
                            0]:
                        for k, text in enumerate(candidate['word']):
                            row[k + j].text = text
                        element.across_solved = True
                        break
                    if element.down_candidates and candidate ==
element.down_candidates[
                            0]:
                        for k, text in enumerate(candidate['word']):
                            new_puzzle[k + i][j].text = text
                        element.down_solved = True
                        break
            new_tree = PuzzleTree(new_puzzle, self,
                                  self.score + candidate['score'])
            self.children.append(new_tree)

        if len(self.children) == 0:
            return None
        for child in self.children:
            result = child.solve()
            if result != None:
                return result


class Solver:
    """
    Base wrapper class for solving the puzzle. It makes use of
PuzzleTree and provides a simple
    interface to the outside world.

    Methods
    ---------
    create_puzzle()
      This method is the initial point for our solver. Basically it
creates an empty puzzle
      from scraped data to use as a root node in the PuzzleTree.

    create_format()
      This method creates a JSON formatted version of the solution. This
format is same as the
```

```
     scraped data format. Having a single format for both initial
puzzle and solution is very
     convenient when we render both in gui.py

 """

   def __init__(self):
       self.PUZZLE_LENGTH = 5

   def create_puzzle(self, clues, layout):
       puzzle = []
       for i in range(self.PUZZLE_LENGTH):
           puzzle.append([])

       for key in layout.keys():
           text = ''
           block = layout[key]['block']
           clue_across = None
           clue_down = None
           number = layout[key]['number']
           if number != '':
               for clue in clues['across']:
                   if number == clue['id']:
                       clue_across = clue['text']
               for clue in clues['down']:
                   if number == clue['id']:
                       clue_down = clue['text']
           node = Node(text, block, clue_across, clue_down, [], [],
False,
                       False, number)
           puzzle[int(math.floor(int(key) /
self.PUZZLE_LENGTH))].append(node)
       tree = PuzzleTree(puzzle, None, 0)

       return self.create_format(tree.solve())

   def create_format(self, tree):
       puzzle = tree.puzzle

       result = {}

       for i, row in enumerate(puzzle):
           for j, element in enumerate(row):
               sub = {}
               sub['block'] = element.block
               sub['text'] = element.text.upper()
               sub['number'] = element.number
               result['{}'.format(i * self.PUZZLE_LENGTH + j)] = sub
       return result
```

Screenshots

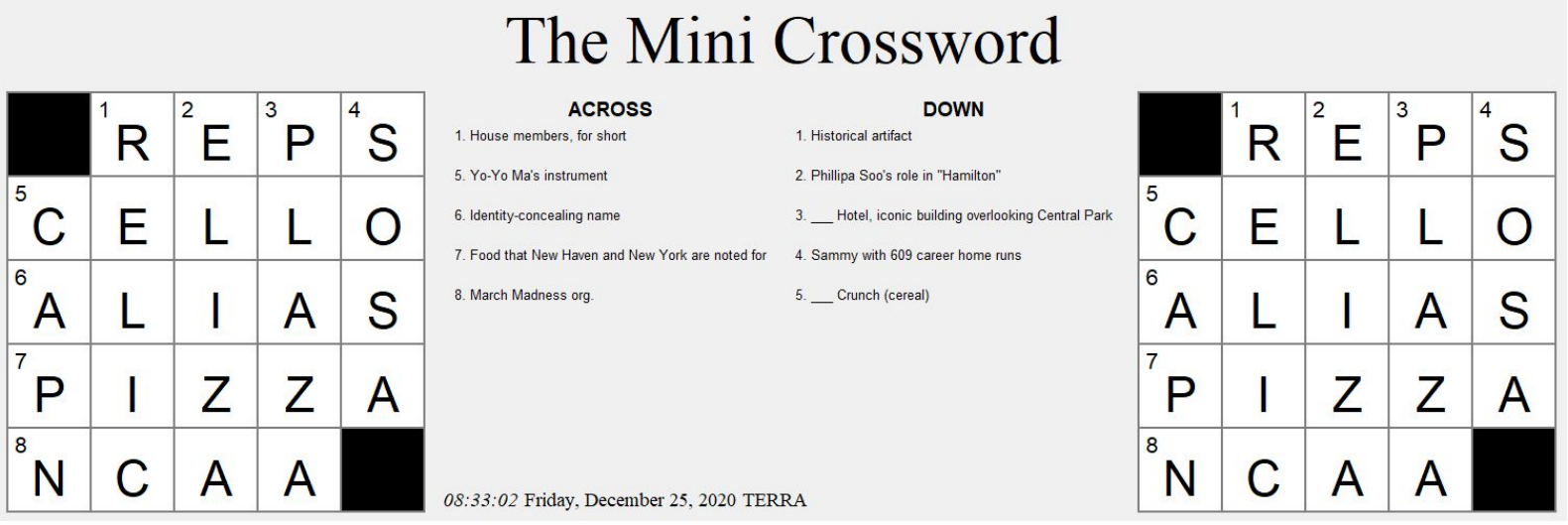

Figure 5: Sample output of our program

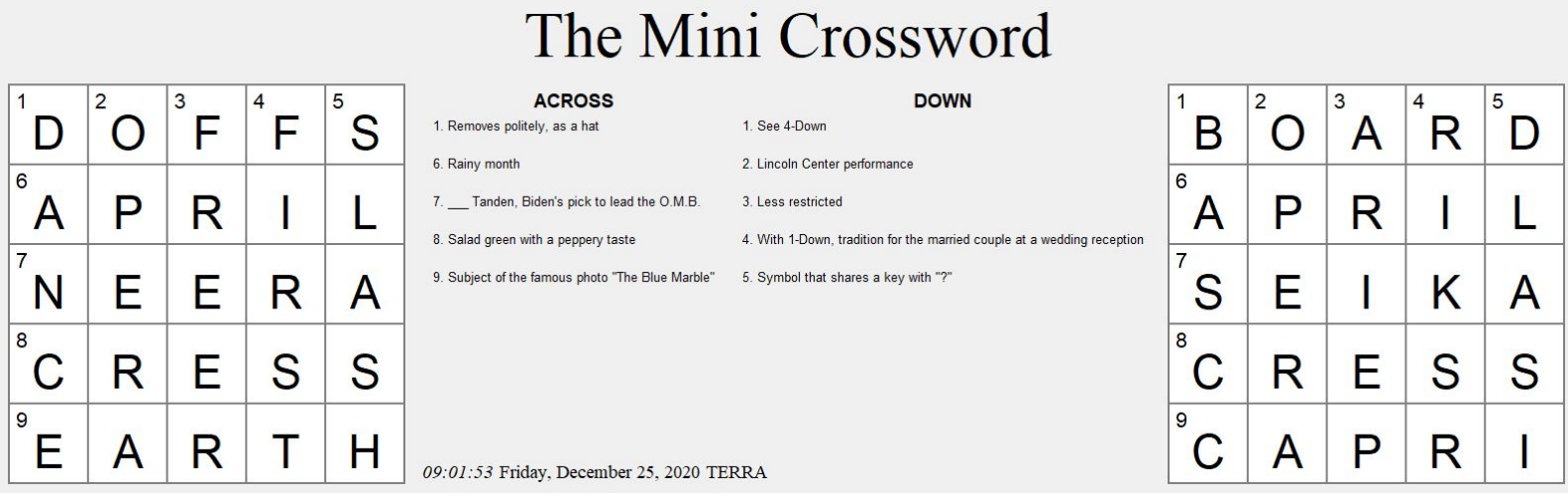

Figure 6: Sample output of our program

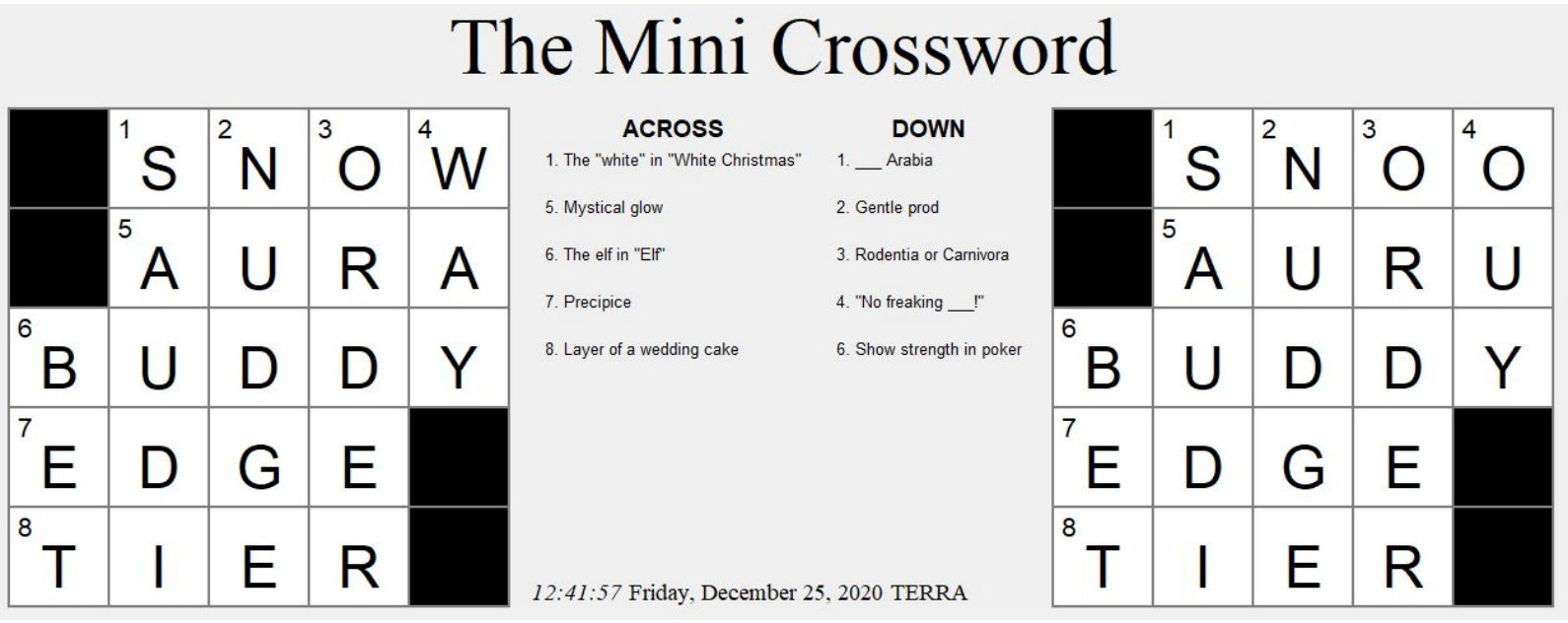

Figure 7: Sample output of our program

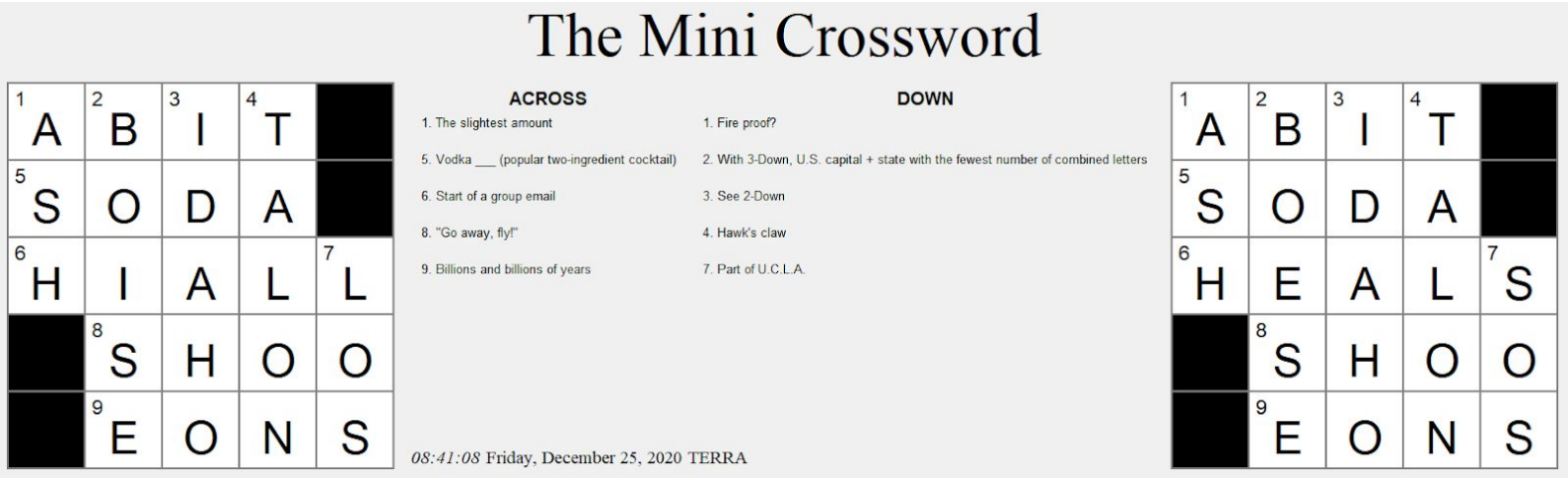

Figure 8: Sample output of our program

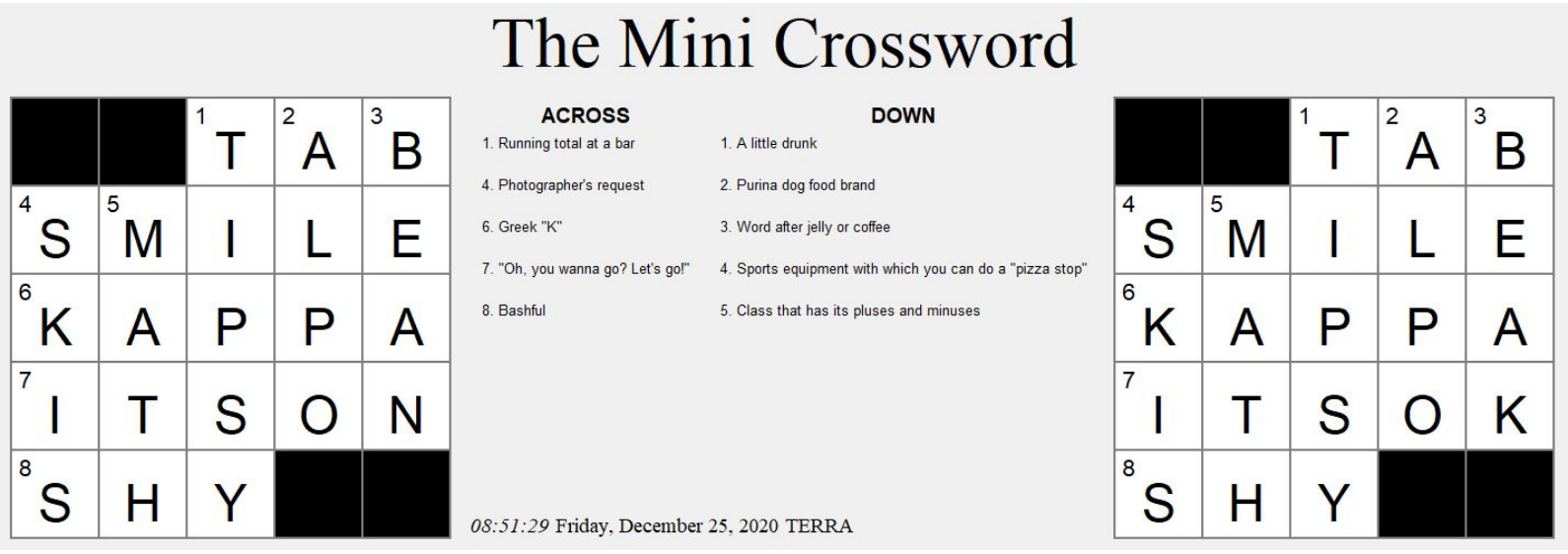

Figure 9: Sample output of our program

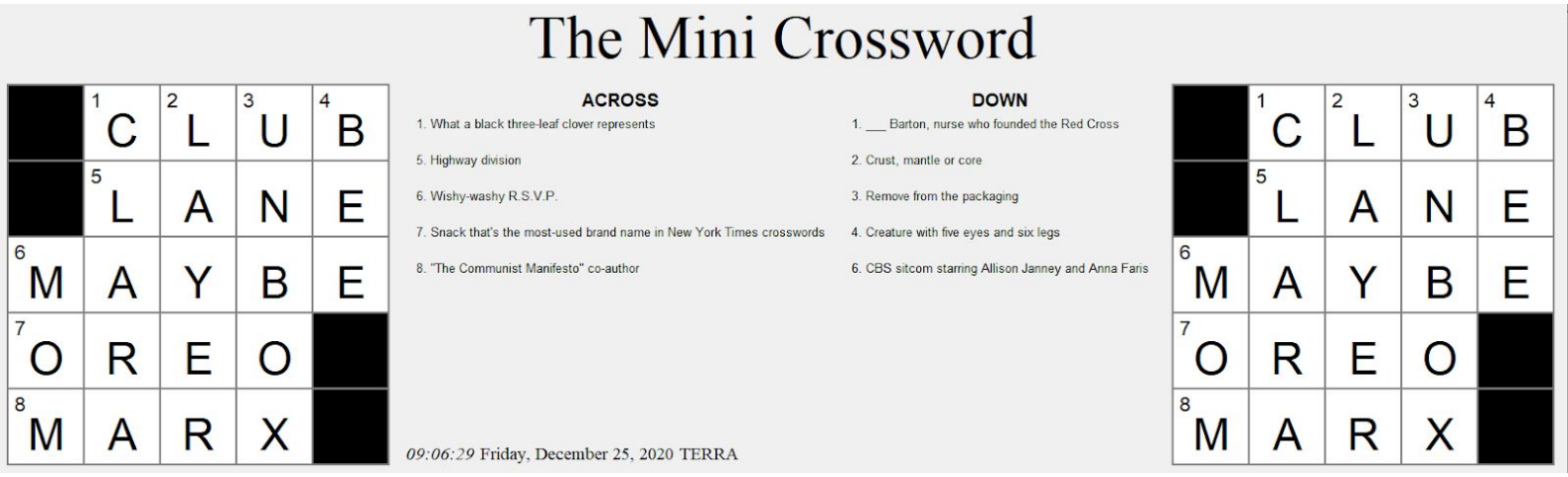

Figure 10: Sample output of our program

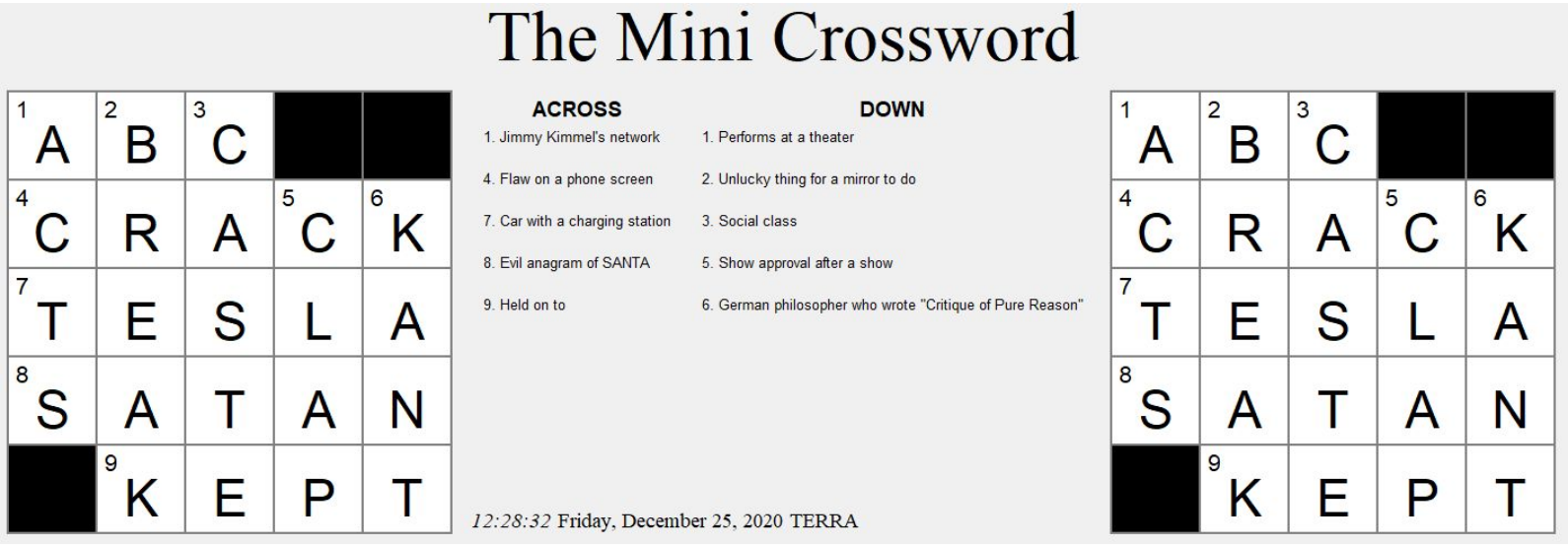

Figure 11: Sample output of our program

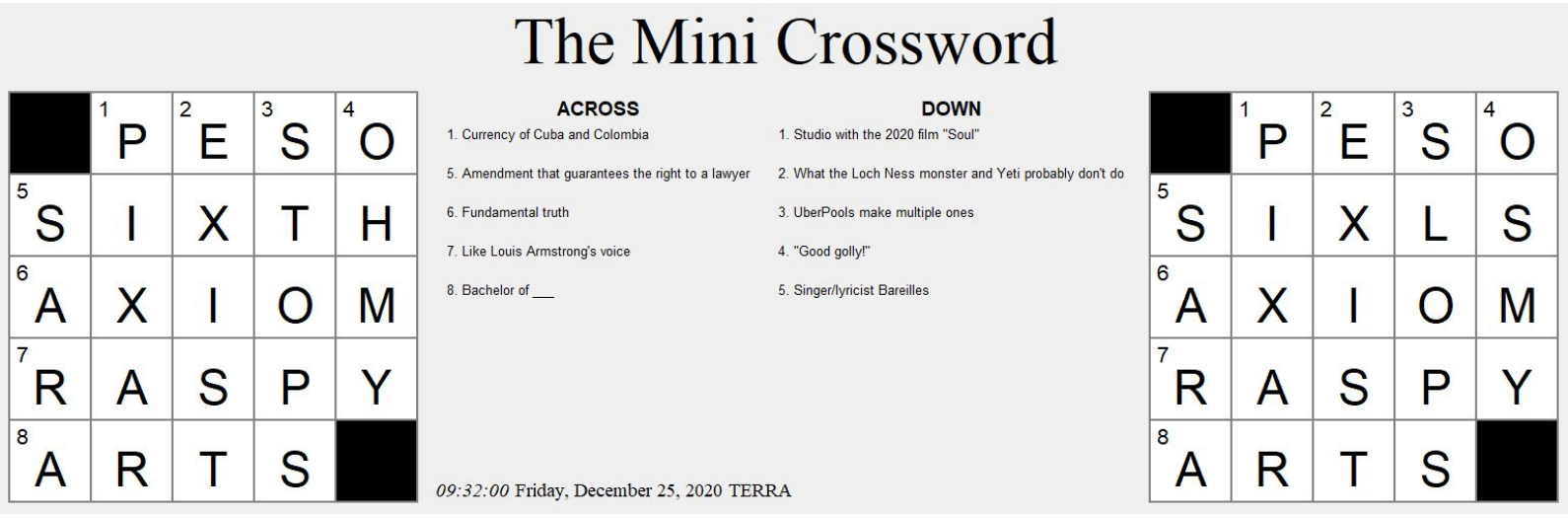

Figure 12: Sample output of our program

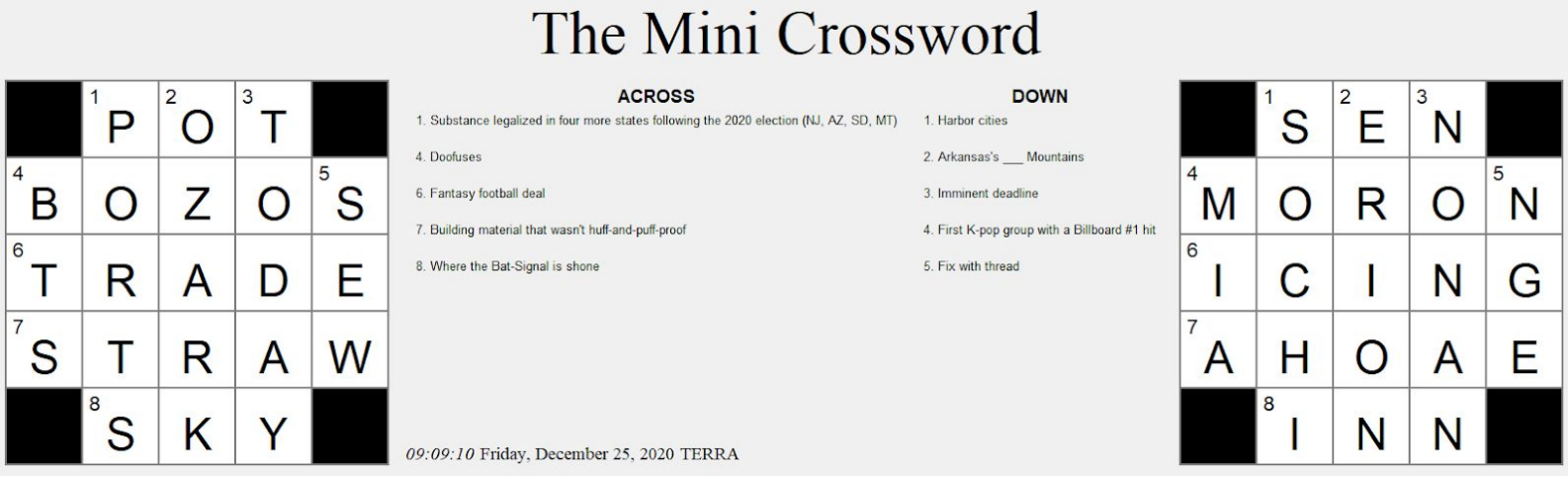

Figure 13: Sample output of our program

Figure 14: Sample output of our program

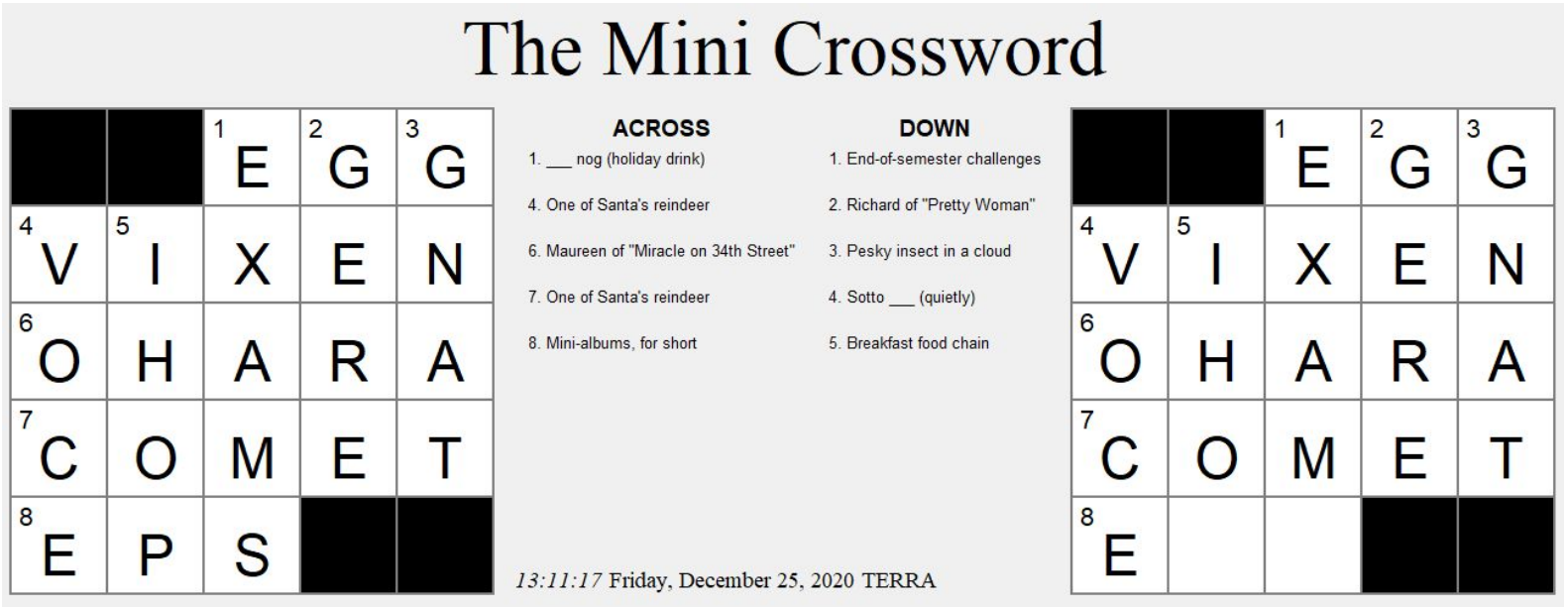

Figure 15: Sample output of our program

7. Bibliography

1. *Artificial Intelligence*, vol. 134, no. 1, Jan. 2002, pp. 23–55. *EBSCOhost*, doi:10.1016/S0004-3702(01)00114-X.

2. Thomas, Anu, and Sangeetha S. “Towards a Semantic Approach for Candidate Answer Generation in Solving Crossword Puzzles.” *Procedia Computer Science*, vol. 171, Jan. 2020, pp. 2310–2315. *EBSCOhost*, doi:10.1016/j.procs.2020.04.250.

3. Kejkaew Thanasuan, and Shane eMueller. “Crossword Expertise as Recognitional Decision Making: An Artificial Intelligence Approach.” *Frontiers in Psychology*, vol. 5, Sept. 2014. *EBSCOhost*, doi:10.3389/fpsyg.2014.01018.

4. RIGUTINI, LEONARDO, et al. “Automatic Generation of Crossword Puzzles.” *International Journal on Artificial Intelligence Tools*, vol. 21, no. 3, June 2012, pp. 1250014-1-1250014–22. *EBSCOhost*, doi:10.1142/S0218213012500145.

5. Nicosia, M. (.1,2)., et al. *Learning to Rank Aggregated Answers for Crossword Puzzles*. Vol. 9022, Springer Verlag. *EBSCOhost*, doi:10.1007/978-3-319-16354-3_61. Accessed 2 Dec. 2020.

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 TERRA.

Word Count = 2115